Michael Fisher
Wiebe van der Hoek
Boris Konev
Alexei Lisitsa (Eds.)

LNAI 4160

Logics in Artificial Intelligence

10th European Conference, JELIA 2006
Liverpool, UK, September 2006
Proceedings

Springer

Lecture Notes in Artificial Intelligence 4160

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Michael Fisher Wiebe van der Hoek
Boris Konev Alexei Lisitsa (Eds.)

Logics in Artificial Intelligence

10th European Conference, JELIA 2006
Liverpool, UK, September 13-15, 2006
Proceedings

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Michael Fisher
Wiebe van der Hoek
Boris Konev
Alexei Lisitsa

University of Liverpool
Department of Computer Science
Liverpool; L69 3BX, UK
E-mail: {M.Fisher,wiebe,B.Konev,alexei}@csc.liv.ac.uk

Library of Congress Control Number: 2006932041

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

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-39625-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-39625-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11853886 06/3142 5 4 3 2 1 0

Preface

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 organised biennially, with English as official language, and with proceedings published in Springer's Lecture Notes in Artificial Intelligence. Previous meetings took place in Roscoff, France (1988), Amsterdam, Netherlands (1990), Berlin, Germany (1992), York, UK (1994), Évora, Portugal (1996), Dagstuhl, Germany (1998), Málaga, Spain (2000), Cosenza, Italy (2002), and Lisbon, Portugal (2004).

The increasing interest in this forum, its international level with growing participation from researchers outside Europe, and the overall technical quality, has turned JELIA into a major forum for the discussion of logic-based approaches to AI. JELIA 2006 constituted the Tenth International Conference on Logics in Artificial Intelligence, and was held in Liverpool (UK) in September 2006. As with previous JELIA conferences, the aim of JELIA 2006 was to bring together active researchers interested in all aspects concerning the use of logics in AI to discuss current research, results, problems and applications of both a theoretical and practical nature.

We received a total of 96 submissions, comprising 77 regular papers and 19 tool descriptions. These submissions represented a wide range of topics throughout Artificial Intelligence and, as well as originating in Europe, we were pleased to receive submissions from a variety of other countries across the world, including Australia, Brazil, China, Sri Lanka, South Korea and USA. We would like to take this opportunity to thank all those who submitted papers and whose contributions have helped make such a strong final programme.

The regular paper submissions were usually evaluated by at least three members of the Programme Committee (see below) and in many cases further discussion on the merits of particular papers was entered into. Tool description papers were each evaluated by two members of the Programme Committee. We would like to thank all the members of the Programme Committee and the additional referees (see below) for the professional way in which they carried out their reviewing and selection duties.

The review process was extremely selective and many good papers could not be accepted for the final program. As a result of the reviewing process 34 regular papers (44% of submissions) were selected for full presentation at JELIA 2006. In addition, 12 tool descriptions (62% of submissions) were selected for presentation and demonstration. The papers appearing in these proceedings cover a range of topics within the scope of the conference, such as logic programming, description logics, non-monotonic reasoning, agent theories, automated reasoning, and machine learning. Together with the programme of technical papers, we are pleased to acknowledge a strong series of

invited talks by leading members of the Logic in AI community: Sašo Džeroski (Jozef Stefan Institute, Slovenia); Ilkka Niemelä (Helsinki University of Technology, Finland); and Andrei Voronkov (University of Manchester, UK). We are confident that you will find the contents of this volume stimulating and enlightening, and that it will provide an invaluable reference to many current research issues in Logics in AI.

Finally, we are indebted to the members of the JELIA Steering Committee (see below) for selecting Liverpool for the tenth JELIA event, to sponsorship from EPSRC, AgentcitiesUK and the University of Liverpool, and to Catherine Atherton and Dave Shield for their invaluable assistance in hosting this conference.

July 2006

Michael Fisher
[Programme Chair]

Wiebe van der Hoek
[General Chair]

Boris Konev
[Tool Session Chair]

Alexei Lisitsa
[Local Organising Chair]

Organization

JELIA Steering Committee:

Gerhard Brewka David Pearce Luís Moniz Pereira

JELIA-06 Programme Committee:

José Júlio Alferes
Franz Baader
Chitta Baral
Peter Baumgartner
Salem Benferhat
Alexander Bochman
Rafael Bordini
Gerhard Brewka
Walter Carnielli
Luis Fariñas del Cerro
Mehdi Dastani
James Delgrande
Jürgen Dix
Clare Dixon
Roy Dyckhoff
Thomas Eiter
Patrice Enjalbert
Michael Fisher
Maria Fox
Enrico Franconi
Ulrich Furbach
Sergio Greco
Lluís Godo
James Harland
Tomi Janhunen
Peter Jonsson
Boris Konev
Manolis Koubarakis
João Leite
Maurizio Lenzerini
Nicola Leone
Gérard Ligozat
John-Jules Meyer
Angelo Montanari
Bernhard Nebel
Manuel Ojeda-Aciego
David Pearce
Charles Pecheur
Luís Moniz Pereira
Henri Prade
Henry Prakken
Francesca Rossi
Ken Satoh
Renate Schmidt
Terry Swift
Francesca Toni
Paolo Torroni
Mirek Truszczynski
Toby Walsh
Mary-Anne Williams
Michael Zakharyaschev

Additional Reviewers

Salvador Abreu
Wolfgang Ahrendt
Alessandro Artale
Pedro Barahona
Bernhard Beckert
Piero Bonatti
Krysia Broda
Diego Calvanese
Iliano Cervesato
Marta Cialdea
Pierangelo Dell'Acqua
Agostino Dovier
Esra Erdem
Michael Fink
Giorgos Flouris
Laura Giordano
Valentin Goranko
Rajeev Goré
Guido Governatori
Gianluigi Greco
Pascal Hitzler
Wiebe van der Hoek
Aaron Hunter
Ullrich Hustadt
Giovambattista Ianni
Wojtek Jamroga
Andrew Jones
Reinhard Kahle
Ralf Küsters
Zhen Li
Thomas Lukasiewicz
Michael Maher
Davide Marchignoli
Wolfgang May
Paola Mello
Thomas Meyer
Maja Milicic
Rafiq Muhammad
Alexander Nittka
Peter Novak
Magdalena Ortiz
Simona Perri

Gerald Pfeifer
Axel Polleres
Helmut Prendinger
Birna van Riemsdijk
Fabrizio Riguzzi
Jussin Rintanen
Rob Rothenberg
Jordi Sabater-Mir
Mehrnoosh Sadrzadeh
Torsten Schaub
Ute Schmid
Steven Shapiro
Tran Cao Son
Giorgos Stamou
Phiniki Stouppa
Thomas Studer
Aaron Stump
Uwe Waldmann
Kewen Wang
Gregory Wheeler
Frank Wolter
Bozena Wozna
Bruno Zanuttini
Hantao Zhang

Table of Contents

I Invited Talks

II Technical Papers

III Tool Descriptions

From Inductive Logic Programming to Relational Data Mining

Sašo Džeroski

Jozef Stefan Institute, Department of Knowledge Technologies,
Jamova 39, 1000 Ljubljana, Slovenija
Saso.Dzeroski@ijs.si

Abstract. Situated at the intersection of machine learning and logic programming, inductive logic programming (ILP) has been concerned with finding patterns expressed as logic programs. While ILP initially focussed on automated program synthesis from examples, it has recently expanded its scope to cover a whole range of data analysis tasks (classification, regression, clustering, association analysis). ILP algorithms can this be used to find patterns in relational data, i.e., for relational data mining (RDM). This paper briefly introduces the basic concepts of ILP and RDM and discusses some recent research trends in these areas.

1 Introduction

Logic programming as a subset of first-order logic is mostly concerned with deductive inference. Inductive logic programming (ILP) [24], on the other hand, is concerned with inductive inference. It generalizes from individual instances/observations in the presence of background knowledge, finding regularities / hypotheses about yet unseen instances.

In its early days, ILP focussed on automated program synthesis from examples, formulated as a binary classification task. In recent years, however, the scope of ILP has broadened to cover a variety of data mining tasks, such as classification, regression, clustering, association analysis. Data mining is concerned with finding patterns in data, the most common types of patterns encountered being classification rules, classification and regression trees, and association rules.

ILP approaches can be used to find patterns in relational data, i.e., for relational data mining (RDM) [12]. The types of patterns encountered in data mining now have relational counterparts, such as relational classification rules, relational regression trees, relational association rules. The major classes of data mining algorithms (such as decision tree induction, distance-based clustering and prediction, etc.) have also been upgraded to relational data mining algorithms.

In this paper we first briefly introduce the task of inductive logic programming. We assume the reader is familiar with basic logic programming notation. We start with logical settings for concept learning and continue with discussing the task of relational rule induction. We next discuss the relational extensions of two major types of patterns considered in data mining: classification and regression

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 1–14, 2006.

trees and association rules. We only discuss the patterns, not the algorithms for finding such patterns from data. We conclude with a discussion of recent trends in ILP and RDM.

2 Inductive Logic Programming: Settings and Approaches

Logic programming as a subset of first-order logic is mostly concerned with deductive inference. Inductive logic programming, on the other hand, is concerned with inductive inference. It generalizes from individual instances/observations in the presence of background knowledge, finding regularities/hypotheses about yet unseen instances.

In this section, we discuss the different ILP settings as well as the different relational learning tasks, starting with the induction of logic programs (sets of relational rules). We also discuss the two major approaches to solving relational learning tasks, namely transforming relational problems to propositional form and upgrading propositional algorithms to a relational setting.

2.1 Logical Settings for Concept Learning

One of the most basic and most often considered tasks in machine learning is the task of inductive concept learning. Given $\mathcal{U}$, a universal set of objects (observations), a *concept* $\mathcal{C}$ is a subset of objects in $\mathcal{U}$, $\mathcal{C} \subseteq \mathcal{U}$. For example, if $\mathcal{U}$ is the set of all patients in a given hospital $\mathcal{C}$ could be the set of all patients diagnosed with Hepatitis A. The task of *inductive concept learning* is defined as follows: Given instances and non-instances of concept $\mathcal{C}$, find a hypothesis (classifier) H able to tell whether $x \in \mathcal{C}$, for each $x \in \mathcal{U}$.

Table 1. The task of inductive concept learning

Given:

- a language of examples L_E
- a language of concept descriptions L_H
- a **covers** relation between L_H and L_E, defining when an example e is *covered* by a hypothesis$H$: $covers(H, e)$
- sets of positive P and negative N examples described in L_E

Find hypothesis H from L_H, such that

- **completeness:** H covers all positive examples $p \in P$
- **consistency:** H does not cover any negative example $n \in N$

To define the task of inductive concept learning more precisely, we need to specify $\mathcal{U}$ the space of instances (examples), as well as the space of hypotheses considered. This is done through specifying the languages of examples (L_E) and concept descriptions (L_H). In addition, a coverage relation $covers(H, e)$ has to

be specified, which tells us when an example e is considered to belong to the concept represented by hypothesis H. Examples that belong to the target concept are termed positive, those that do not are termed negative. Given positive and negative examples, we want hypotheses that are complete (cover all positive examples) and consistent (do not cover negative examples).

Looking at concept learning in a logical framework, De Raedt [9] considers three settings for concept learning. The key aspect that varies in these settings is the notion of coverage, but the languages L_E and L_H vary as well. We characterize these for each of the three settings below.

- In *learning from entailment*, the coverage relation is defined as $covers(H, e)$ iff $H \models e$. The hypothesis logically entails the exampe. Here H is a clausal theory and e is a clause.
- In *learning from interpretations*, we have $covers(H, e)$ iff e is model of H. The example has to be a model of the hypothesis. H is a clausal theory and e is a Herbrand interpretation.
- In *learning from satisfiability*, $covers(H, e)$ iff $H \wedge e \not\models \bot$. The example and the hypothesis taken together have to be satisfiable. Here both H and e are clausal theories.

The setting of learning from entailment, introduced by Muggleton [24], is the one that has received the most attention in the field of ILP. The alternative ILP setting of learning from interpretations was proposed by De Raedt and Džeroski [10]: this setting is a natural generalization of propositional learning. Many learning algorithms for propositional learning have been upgraded to the learning from interpretations ILP setting. Finally, the setting of learning from satisfiability was introduced by Wrobel and Džeroski [36], but has rarely been used in practice due to computational complexity problems.

De Raedt [9] also discusses the relationships among the three settings for concept learning. Learning from finite interpretations reduces to learning from entailment. Learning from entailment reduces to learning from satisfiability. Learning from interpretations is thus the easiest and learning from satisfiability the hardest of the three settings.

As introduced above, the logical settings for concept learning do not take into account background knowledge, one of the essential ingredients of ILP. However, the definitions of the settings are easily extended to take it into account. Given background knowledge B, which in its most general form can be a clausal theory, the definition of coverage should be modified by replacing H with $B \wedge H$ for all three settings.

2.2 The ILP Task of Relational Rule Induction

The most commonly addressed task in ILP is the task of learning logical definitions of relations [30], where tuples that belong or do not belong to the target relation are given as examples. From training examples ILP then induces a logic program (predicate definition) corresponding to a view that defines the target relation in terms of other relations that are given as background knowledge.

This classical ILP task is addressed, for instance, by the seminal MIS system [32] (rightfully considered as one of the most influential ancestors of ILP) and one of the best known ILP systems FOIL [30].

Given is a set of examples, i.e., tuples that belong to the target relation p (positive examples) and tuples that do not belong to p (negative examples). Given are also background relations (or background predicates) q_i that constitute the background knowledge and can be used in the learned definition of p. Finally, a hypothesis language, specifying syntactic restrictions on the definition of p is also given (either explicitly or implicitly). The task is to find a definition of the target relation p that is consistent and complete, i.e., explains all the positive and none of the negative tuples.

Formally, given is a set of examples $E = P \cup N$, where P contains positive and N negative examples, and background knowledge B. The task is to find a hypothesis H such that $\forall e \in P : B \wedge H \models e$ (H is complete) and $\forall e \in N : B \wedge H \not\models e$ (H is consistent), where $\models$ stands for logical implication or entailment. This setting, introduced by Muggleton [24], is thus also called learning from entailment.

In the most general formulation, each e, as well as B and H can be a clausal theory. In practice, each e is most often a ground example (tuple), B is a relational database (which may or may not contain views) and H is a definite logic program. The semantic entailment ($\models$) is in practice replaced with syntactic entailment ($\vdash$) or provability, where the resolution inference rule (as implemented in Prolog) is most often used to prove examples from a hypothesis and the background knowledge. In learning from entailment, a positive fact is explained if it can be found among the answer substitutions for h produced by a query $? - b$ on database B, where $h \leftarrow b$ is a clause in H. In learning from interpretations, a clause $h \leftarrow b$ from H is true in the minimal Herbrand model of B if the query $b \wedge \neg h$ fails on B.

As an illustration, consider the task of defining relation $daughter(X, Y)$, which states that person X is a daughter of person Y, in terms of the background knowledge relations $female$ and $parent$. These relations are given in Table 2. There are two positive and two negative examples of the target relation $daughter$. In the hypothesis language of definite program clauses it is possible to formulate the following definition of the target relation,

$$daughter(X, Y) \leftarrow female(X), parent(Y, X).$$

which is consistent and complete with respect to the background knowledge and the training examples.

In general, depending on the background knowledge, the hypothesis language and the complexity of the target concept, the target predicate definition may consist of a set of clauses, such as

$$daughter(X, Y) \leftarrow female(X), mother(Y, X).$$
$$daughter(X, Y) \leftarrow female(X), father(Y, X).$$

if the relations $mother$ and $father$ were given in the background knowledge instead of the $parent$ relation.

Table 2. A simple ILP problem: learning the *daughter* relation. Positive examples are denoted by $\oplus$ and negative by $\ominus$.

Training examples		Background knowledge	
$daughter(mary, ann).$	$\oplus$	$parent(ann, mary).$	$female(ann).$
$daughter(eve, tom).$	$\oplus$	$parent(ann, tom).$	$female(mary).$
$daughter(tom, ann).$	$\ominus$	$parent(tom, eve).$	$female(eve).$
$daughter(eve, ann).$	$\ominus$	$parent(tom, ian).$	

The hypothesis language is typically a subset of the language of program clauses. As the complexity of learning grows with the expressiveness of the hypothesis language, restrictions have to be imposed on hypothesized clauses. Typical restrictions are the exclusion of recursion and restrictions on variables that appear in the body of the clause but not in its head (so-called new variables).

Declarative bias [28] explicitly specifies the language of hypotheses (clauses) considered by the ILP system at hand. This is input to the learning system (and not hard-wired in the learning algorithm). Various types of declarative bias have been used by different ILP systems, such as argument types and input/output modes, parametrized language bias (e.g., maximum number of variables, literals, depth of variables, arity, etc,), clause templates and grammars. For example, a suitable clause template for learning family relationships would be $P(X, Y) \leftarrow Q(X, Z), R(Z, Y)$. Here P, Q and R are second order variables that can be replaced by predicates, e.g., *grandmother*, *mother* and *parent*. The same template can be used to learn the notions of grandmother and a grandfather.

2.3 Other Tasks of Relational Learning

Initial efforts in ILP focussed on relational rule induction, more precisely on concept learning in first-order logic and synthesis of logic programs, cf. [24]. An overview of early work is given in the textbook on ILP by Lavrač and Džeroski [23]. Representative early ILP systems addressing this task are CIGOL [26], FOIL [30], GOLEM [27] and LINUS [22]. More recent representative ILP systems are PROGOL [25] and ALEPH [33].

State-of-the-art ILP approaches now span most of the spectrum of data mining tasks and use a variety of techniques to address these. The distinguishing features of using multiple relations directly and discovering patterns expressed in first-order logic are present throughout: the ILP approaches can thus be viewed as upgrades of traditional approaches. Van Laer and De Raedt [34] (Chapter 10 of [12]) present a case study of upgrading a propositional approach to classification rule induction to first order logic. Note, however, that upgrading to first-order logic is non-trivial: the expressive power of first-order logic implies computational costs and much work is needed in balancing the expressive power of the pattern languages used and the computational complexity of the data mining algorithm looking for such patterns. This search for a balance between the two has occupied much of the ILP research in the last ten years.

Present ILP approaches to multi-class classification involve the induction of relational classification rules (ICL [34]), as well as first order logical decision trees in TILDE [1] and S-CART [21]. ICL upgrades the propositional rule inducer CN2 [6]. TILDE and S-CART upgrade decision tree induction as implemented in C4.5 [31] and CART [4]. A nearest-neighbor approach to relational classification is implemented in RIBL [16] and its successor RIBL2. [18, 20].

Relational regression approaches upgrade propositional regression tree and rules approaches. TILDE and S-CART, as well as RIBL2 can handle continuous classes. FORS [19] learns decision lists (ordered sets of rules) for relational regression.

The main non-predictive or descriptive data mining tasks are clustering and discovery of association rules. These have been also addressed in a first-order logic setting. The RIBL distance measure has been used to perform hierarchical agglomerative clustering in RDBC [20], as well as k-medoids clustering. Section 4 describes a relational approach to the discovery of frequent queries and query extensions, a first-order version of association rules.

With such a wide arsenal of relational data mining techniques, there is also a variety of practical applications. ILP has been successfully applied to discover knowledge from relational data and background knowledge in the areas of molecular biology (including drug design, protein structure prediction and functional genomics), environmental sciences, traffic control and natural language processing. An overview of applications is given by Džeroski [14] (Chapter 14 in [12]).

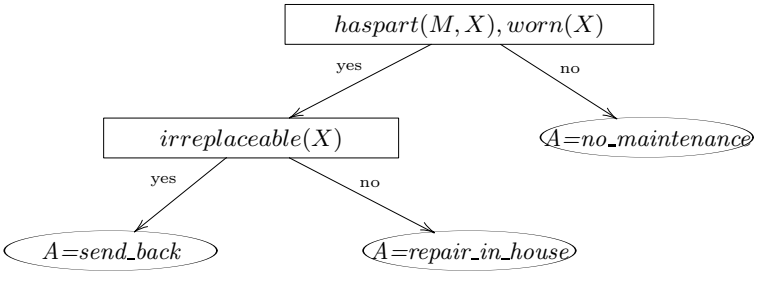


Fig. 1. A relational decision tree, predicting the class variable A in the target predicate $maintenance(M, A)$

3 Relational Decision Trees

Decision tree induction is one of the major approaches to data mining. Upgrading this approach to a relational setting has thus been of great importance. In this section, we look into what relational decision trees are, i.e., how they are defined. We do not discuss how such trees can be induced from multi-relational data: we refer the reader to [21], [1] and [12].

Without loss of generality, we can say the task of relational prediction is defined by a two-place target predicate $target(ExampleID, ClassVar)$, which has as arguments an example ID and the class variable, and a set of background

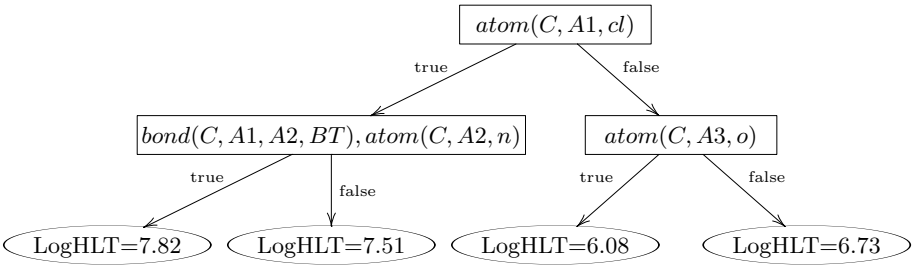


Fig. 2. A relational regression tree for predicting the degradation time $LogHLT$ of a chemical compound C (target predicate $degrades(C, LogHLT)$)

knowledge predicates/relations. Depending on whether the class variable is discrete or continuous, we talk about relational classification or regression. Relational decision trees are one approach to solving this task.

An example relational decision tree is given in Figure 1. It predicts the maintenance action A to be taken on machine M ($maintenance(M, A)$), based on parts the machine contains ($haspart(M, X)$), their condition ($worn(X)$) and ease of replacement ($irreplaceable(X)$). The target predicate here is $maintenance(M, A)$, the class variable is A, and background knowledge predicates are $haspart(M, X)$, $worn(X)$ and $irreplaceable(X)$.

Relational decision trees have much the same structure as propositional decision trees. Internal nodes contain tests, while leaves contain predictions for the class value. If the class variable is discrete/continuous, we talk about relational classification/regression trees. For regression, linear equations may be allowed in the leaves instead of constant class-value predictions: in this case we talk about relational model trees.

The tree in Figure 1 is a relational classification tree, while the tree in Figure 2 is a relational regression tree. The latter predicts the degradation time (the logarithm of the mean half-life time in water [13]) of a chemical compound from its chemical structure, where the latter is represented by the atoms in the compound and the bonds between them. The target predicate is $degrades(C, LogHLT)$, the class variable $LogHLT$, and the background knowledge predicates are $atom(C, AtomID, Element)$ and $bond(C, A_1, A_2, BondType)$. The test at the root of the tree $atom(C, A1, cl)$ asks if the compound C has a chlorine atom $A1$ and the test along the left branch checks whether the chlorine atom $A1$ is connected to a nitrogen atom $A2$.

As can be seen from the above examples, the major difference between propositional and relational decision trees is in the tests that can appear in internal nodes. In the relational case, tests are queries, i.e., conjunctions of literals with existentially quantified variables, e.g., $haspart(M, X), worn(X)$. Relational trees are binary: each internal node has a left (yes) and a right (no) branch. If the query succeeds, i.e., if there exists an answer substitution that makes it true, the yes branch is taken.

It is important to note that variables can be shared among nodes, i.e., a variable introduced in a node can be referred to in the left (yes) subtree of that node. For example, the X in $irreplaceable(X)$ refers to the machine part X introduced in the root node test $haspart(M, X), worn(X)$. Similarly, the $A1$ in $bond(C, A1, A2, BT)$ refers to the chlorine atom introduced in the root node $atom(C, A1, cl)$. One cannot refer to variables introduced in a node in the right (no) subtree of that node. For example, referring to the chlorine atom $A1$ in the right subtree of the tree in Figure 2 makes no sense, as going along the right (no) branch means that the compound contains no chlorine atoms.

The actual test that has to be executed in a node is the conjunction of the literals in the node itself and the literals on the path from the root of the tree to the node in question. For example, the test in the node $irreplaceable(X)$ in Figure 1 is actually $haspart(M, X), worn(X), irreplaceable(X)$. In other words, we need to send the machine back to the manufacturer for maintenance only if it has a part which is both worn and irreplaceable. Similarly, the test in the node $bond(C, A1, A2, BT)$, $atom(C, A2, n)$ in Figure 2 is in fact $atom(C, A1, cl), bond(C, A1, A2, BT), atom(C, A2, n)$. As a consequence, one cannot transform relational decision trees to logic programs in the fashion "one clause per leaf" (unlike propositional decision trees, where a transformation "one rule per leaf" is possible).

Table 3. A decision list representation of the relational decision tree in Figure 1

$maintenance(M, A) \leftarrow haspart(M, X), worn(X),$
$\quad irreplaceable(X)$!, $A = send\_back$
$maintenance(M, A) \leftarrow haspart(M, X), worn(X)$, !,
$\quad A = repair\_in\_house$
$maintenance(M, A) \leftarrow A = no\_maintenance$

Relational decision trees can be easily transformed into first-order decision lists, which are ordered sets of clauses (clauses in logic programs are unordered). When applying a decision list to an example, we always take the first clause that applies and return the answer produced. When applying a logic program, all applicable clauses are used and a set of answers can be produced. First-order decision lists can be represented by Prolog programs with cuts (!) [3]: cuts ensure that only the first applicable clause is used.

Table 4. A decision list representation of the relational regression tree for predicting the biodegradability of a compound, given in Figure 2

$degrades(C, LogHLT) \leftarrow atom(C, A1, cl),$
$\quad bond(C, A1, A2, BT), atom(C, A2, n), LogHLT = 7.82, !$
$degrades(C, LogHLT) \leftarrow atom(C, A1, cl),$
$\quad LogHLT = 7.51, !$
$degrades(C, LogHLT) \leftarrow atom(C, A3, o),$
$\quad LogHLT = 6.08, !$
$degrades(C, LogHLT) \leftarrow LogHLT = 6.73.$

Table 5. A logic program representation of the relational decision tree in Figure 1

$a(M) \leftarrow haspart(M, X), worn(X), irreplaceable(X)$
$b(M) \leftarrow haspart(M, X), worn(X)$
$maintenance(M, A) \leftarrow not\ a(M), A = no\_aintenance$
$maintenance(M, A) \leftarrow b(M), A = repair\_in\_house$
$maintenance(M, A) \leftarrow a(M), not\ b(M),\ A = send\_back$

A decision list is produced by traversing the relational regression tree in a depth-first fashion, going down left branches first. At each leaf, a clause is output that contains the prediction of the leaf and all the conditions along the left (yes) branches leading to that leaf. A decision list obtained from the tree in Figure 1 is given in Table 3. For the first clause (*send_back*), the conditions in both internal nodes are output, as the left branches out of both nodes have been followed to reach the corresponding leaf. For the second clause, only the condition in the root is output: to reach the *repair_in_house* leaf, the left (yes) branch out of the root has been followed, but the right (no) branch out of the $irreplaceable(X)$ node has been followed. A decision list produced from the relational regression tree in Figure 2 is given in Table 4.

Generating a logic program from a relational decision tree is more complicated. It requires the introduction of new predicates. We will not describe the transformation process in detail, but rather give an example. A logic program, corresponding to the tree in Figure 1 is given in Table 5.

4 Relational Association Rules

The discovery of frequent patterns and association rules is one of the most commonly studied tasks in data mining. Here we first describe frequent relational patterns (frequent Datalog patterns). We then discuss relational association rules (query extensions).

Dehaspe and Toivonen [7], [8] (Chapter 8 of [12]) consider patterns in the form of Datalog queries, which reduce to SQL queries. A Datalog query has the form $? - A_1, A_2, \ldots A_n$, where the A_i's are logical atoms.

An example Datalog query is

$$? - person(X), parent(X, Y), hasPet(Y, Z)$$

This query on a Prolog database containing predicates *person*, *parent*, and *hasPet* is equivalent to the SQL query

```
SELECT PERSON.ID, PARENT.KID, HASPET.AID
FROM PERSON, PARENT, HASPET
WHERE PERSON.ID = PARENT.PID
AND PARENT.KID = HASPET.PID
```

on a database containing relations PERSON with argument ID, PARENT with arguments PID and KID, and HASPET with arguments PID and AID. This query finds triples (x, y, z), where child y of person x has pet z.

Datalog queries can be viewed as a relational version of itemsets (which are sets of items occurring together). Consider the itemset $\{person, parent, child, pet\}$. The market-basket interpretation of this pattern is that a person, a parent, a child, and a pet occur together. This is also partly the meaning of the above query. However, the variables X, Y, and Z add extra information: the person and the parent are the same, the parent and the child belong to the same family, and the pet belongs to the child. This illustrates the fact that queries are a more expressive variant of itemsets.

To discover frequent patterns, we need to have a notion of frequency. Given that we consider queries as patterns and that queries can have variables, it is not immediately obvious what the frequency of a given query is. This is resolved by specifying an additional parameter of the pattern discovery task, called the key. The key is an atom which has to be present in all queries considered during the discovery process. It determines what is actually counted. In the above query, if $person(X)$ is the key, we count persons, if $parent(X, Y)$ is the key, we count (parent,child) pairs, and if $hasPet(Y, Z)$ is the key, we count (owner,pet) pairs. This is described more precisely below.

Submitting a query $Q =? - A_1, A_2, \ldots A_n$ with variables $\{X_1, \ldots X_m\}$ to a Datalog database $\mathbf{r}$ corresponds to asking whether a grounding substitution exists (which replaces each of the variables in Q with a constant), such that the conjunction $A_1, A_2, \ldots A_n$ holds in $\mathbf{r}$. The answer to the query produces answering substitutions $\theta = \{X_1/a_1, \ldots X_m/a_m\}$ such that $Q\theta$ succeeds. The set of all answering substitutions obtained by submitting a query Q to a Datalog database $\mathbf{r}$ is denoted $answerset(Q, \mathbf{r})$.

The absolute frequency of a query Q is the number of answer substitutions θ for the variables in the key atom for which the query $Q\theta$ succeeds in the given database, i.e., $a(Q, \mathbf{r}, key) = |\{\theta \in answerset(key, \mathbf{r}) | Q\theta \text{ succeeds w.r.t. } \mathbf{r}\}|$. The relative frequency (support) can be calculated as $f(Q, \mathbf{r}, key) = a(Q, \mathbf{r}, key)/|\{\theta \in answerset(key, \mathbf{r})\}|$. Assuming the key is $person(X)$, the absolute frequency for the above example query can be calculated by the following SQL statement:

```
SELECT count(distinct *)
FROM SELECT PERSON.ID
    FROM PERSON, PARENT, HASPET
    WHERE PERSON.ID = PARENT.PID
    AND PARENT.KID = HASPET.PID
```

Association rules have the form $A \rightarrow C$ and the intuitive market-basket interpretation "customers that buy A typically also buy C". If itemsets A and C have supports f_A and f_C, respectively, the confidence of the association rule is defined to be $c_{A \rightarrow C} = f_C/f_A$. The task of association rule discovery is to find

all association rules $A \rightarrow C$, where f_C and $c_{A \rightarrow C}$ exceed prespecified thresholds (minsup and minconf).

Association rules are typically obtained from frequent itemsets. Suppose we have two frequent itemsets A and C, such that $A \subset C$, where $C = A \cup B$. If the support of A is f_A and the support of C is f_C, we can derive an association rule $A \rightarrow B$, which has confidence f_C/f_A. Treating the arrow as implication, note that we can derive $A \rightarrow C$ from $A \rightarrow B$ ($A \rightarrow A$ and $A \rightarrow B$ implies $A \rightarrow A \cup B$, i.e., $A \rightarrow C$).

Relational association rules can be derived in a similar manner from frequent Datalog queries. From two frequent queries $Q_1 = ? - l_1, \ldots l_m$ and $Q_2 = ? - l_1, \ldots l_m, l_{m+1}, \ldots l_n$, where Q_2 θ-subsumes Q_1, we can derive a relational association rule $Q_1 \rightarrow Q_2$. Since Q_2 extends Q_1, such a relational association rule is named a query extension.

A query extension is thus an existentially quantified implication of the form $? - l_1, \ldots l_m \rightarrow ? - l_1, \ldots l_m, l_{m+1}, \ldots l_n$ (since variables in queries are existentially quantified). A shorthand notation for the above query extension is $? - l_1, \ldots l_m \rightsquigarrow l_{m+1}, \ldots l_n$. We call the query $? - l_1, \ldots l_m$ the body and the sub-query $l_{m+1}, \ldots l_n$ the head of the query extension. Note, however, that the head of the query extension does not correspond to its conclusion (which is $? - l_1, \ldots l_m, l_{m+1}, \ldots l_n$).

Assume the queries $Q_1 = ? - person(X), parent(X, Y)$ and $Q_2 = ? - person(X), parent(X, Y), hasPet(Y, Z)$ are frequent, with absolute frequencies of 40 and 30, respectively. The query extension E, where E is defined as $E = ? - person(X), parent(X, Y) \rightsquigarrow hasPet(Y, Z)$, can be considered a relational association rule with a support of 30 and confidence of $30/40 = 75\%$. Note the difference in meaning between the query extension E and two obvious, but incorrect, attempts at defining relational association rules. The clause $person(X), parent(X, Y) \rightarrow hasPet(Y, Z)$ (which stands for the formula $\forall XYZ : person(X) \wedge parent(X, Y) \rightarrow hasPet(Y, Z)$) would be interpreted as follows: "if a person has a child, then this child has a pet". The implication $? - person(X), parent(X, Y) \rightarrow ? - hasPet(Y, Z)$, which stands for $(\exists XY : person(X) \wedge parent(X, Y)) \rightarrow (\exists YZ : hasPet(Y, Z))$ is trivially true if at least one person in the database has a pet. The correct interpretation of the query extension E is: "if a person has a child, then this person also has a child that has a pet."

5 Recent Trends in ILP and RDM

Hot topics and recent advances in ILP and RDM mirror the hot topics in data mining and machine learning. These include scalability issues, ensemble methods, and kernel methods, as well as relational probabilistic representations and learning methods. The latest developments in ILP and RDM are discussed in a special issue of *SIGKDD Explorations* [15].

Scalability issues do indeed deserve a lot of attention when learning in a relational setting, as the complexity of learning increases with the expressive power of the hypothesis language. Scalability methods for ILP include classical ones, such as sampling or turning the loop of hypothesis evaluation inside out

(going through each example once) in decision tree induction. Methods more specific to ILP, such as query packs, have also been considered. For an overview, we refer the reader to the article of Blockeel and Sebag [2] (in [15]).

Boosting was the first ensemble method to be used on top of a relational learning system [29] (Chapter 11 of [12]). This was followed by bagging [5]. More recently, methods for learning random forests have been adapted to the relational setting [35].

Kernel methods have become the mainstream of research in machine learning and data mining in recent years. The development of kernel methods for learning in a relational setting has thus emerged as as a natural research direction. Significant effort has been devoted to the development of kernels for structured/relational data, such as graphs and sequences. An overview is given by Gaertner [17] (in [15]).

Besides the topics mentioned above, the hottest research topic in ILP and RDM is the study of probabilistic representations and learning methods. A variety of these have been recently considered, e.g., Bayesian logic programs and probabilistic relational models. A comprehensive survey of such representations and methods is presented by De Raedt and Kersting [11] (in [15]).

References

1. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101: 285–297, 1998.
2. H. Blockeel and M. Sebag. Scalability and Efficiency in Multi-Relational Data Mining. *SIGKDD Explorations*, 5(1):17–30, 2003.
3. I. Bratko. *Prolog Programming for Artificial Intelligence*, 3rd edition. Addison-Wesley, Harlow, England, 2001.
4. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
5. de Castro Dutra, I., D. Page, V. Costa, and J. Shavlik. An Empirical Evalutation of Bagging in Inductive Logic Programming. In *Proceedings of the Twelfth International Conference on Inductive Logic Programming*, pages 48–65. Springer, Berlin, 2002.
6. P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In *Proceedings of the Fifth European Working Session on Learning*, pages 151–163. Springer, Berlin, 1991.
7. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1): 7–36, 1999.
8. L. Dehaspe and H. Toivonen. Discovery of Relational Association Rules. In [12], pages 189–212, 2001.
9. L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95: 187–201, 1997.
10. L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70: 375–392, 1994.
11. L. De Raedt and K. Kersting. Probabilistic Logic Learning. *SIGKDD Explorations*, 5(1):31–48, 2003.
12. S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer, Berlin, 2001.

13. S. Džeroski, H. Blockeel, B. Kompare, S. Kramer, B. Pfahringer, and W. Van Laer. Experiments in Predicting Biodegradability. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, pages 80–91. Springer, Berlin, 1999.
14. S. Džeroski. Relational Data Mining Applications: An Overview. In [12], pages 339–364, 2001.
15. S. Džeroski and L. De Raedt, editors. Special Issue on Multi-Relational Data Mining. *SIGKDD Explorations*, 5(1), 2003.
16. W. Emde and D. Wettschereck. Relational instance-based learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 122–130. Morgan Kaufmann, San Mateo, CA, 1996.
17. T. Gaertner. Kernel-based Learning in Multi-Relational Data Mining. *SIGKDD Explorations*, 5(1):49–58, 2003.
18. T. Horváth, S. Wrobel, and U. Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1-2):53–80, 2001.
19. A. Karalič and I. Bratko. First order regression. *Machine Learning* 26: 147-176, 1997.
20. M. Kirsten, S. Wrobel, and T. Horváth. Distance Based Approaches to Relational Learning and Clustering. In [12], pages 213–232, 2001.
21. S. Kramer. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 812–819. MIT Press, Cambridge, MA, 1996.
22. N. Lavrač, S. Džeroski, and M. Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the Fifth European Working Session on Learning*, pages 265–281. Springer, Berlin, 1991.
23. N. Lavrač and S. Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, Chichester, 1994. Freely available at `http://www-ai.ijs.si/SasoDzeroski/ILPBook/`.
24. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4): 295–318, 1991.
25. S. Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13: 245–286, 1995.
26. S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Morgan Kaufmann, San Mateo, CA, 1988.
27. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the First Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsha, Tokyo, 1990.
28. C. Nedellec, C. Rouveirol, H. Ade, F. Bergadano, and B. Tausend. Declarative bias in inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press, Amsterdam, 1996.
29. R. Quinlan. Relational Learning and Boosting. In [12], pages 292–306, 2001.
30. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3): 239–266, 1990.
31. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
32. E. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, 1983.
33. A. Srinivasan. The Aleph Manual. Technical Report, Computing Laboratory, Oxford University, 2000. Available at `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`

34. V. Van Laer and L. De Raedt. How to Upgrade Propositional Learners to First Order Logic: A Case Study. In [12], pages 235–261, 2001.
35. C. Vens, A. Van Assche, H. Blockeel, and S. Dzeroski. First order random forests with complex aggregates. In *Proceedings of the Fourteenth International Conference on Inductive Logic Programming*, pages 323–340. Springer, Berlin, 2004.
36. S. Wrobel and S. Džeroski. The ILP description learning problem: towards a general model-level definition of data mining in ILP. In *Proceedings Fachgruppentreffen Maschinelles Lernen*. University of Dortmund, Germany, 1995.

Answer Set Programming: A Declarative Approach to Solving Search Problems

Ilkka Niemelä

Helsinki University of Technology, Laboratory for Theoretical Computer Science,
P.O. Box 5400, FI-02015 TKK, Finland
Ilkka.Niemela@tkk.fi
http://www.tcs.hut.fi/~ini/

The term *answer set programming* (ASP) was coined by Vladimir Lifschitz to name a new declarative programming paradigm that has its roots in stable model (answer set) semantics of logic programs [16] and implementations of this semantics developed in the late 90's. When working with the implementations it became evident that they are instantiations of a different programming paradigm [5, 8, 21, 23, 24] than that of standard logic programming. This new ASP paradigm can be characterized as follows. In ASP programs are theories of some formal system with a semantics that assigns to a theory a collection of sets (models) referred to as answer sets of the program. In order to solve a problem using ASP a program is devised such that the solutions of the problem can be retrieved from the answer sets of the program. An ASP solver is a system that takes as input a program and computes answer sets for it.

While ASP has its roots in logic programming, it can be based on other formal systems such as propositional or first-order logic, too. In fact, the basic idea of ASP is similar to, e.g., SAT-based planning [19] or constraint satisfaction problems. However, these approaches are basically propositional but in ASP the goal is to provide a more powerful knowledge representation language for effective problem encoding. Typically ASP systems are based on logic program type rules with variables and default negation. In order to address advanced knowledge representation issues rules have been extended with, e.g., disjunctions, cardinality constraints, weight constraints, aggregates, built-in functions and predicates, optimization, and preferences.

Current implementations of ASP systems are typically based on a two-level architecture where the problem of computing answer sets for a program with variables is first reduced to an answer set computation problem for a program without variables using logic programming and (deductive) database techniques. This problem is then solved employing model computation techniques similar to those used in propositional SAT solvers. A number of successful ASP systems have been developed [25, 26, 9, 22, 20] (see the list below for some available implementations).

For an excellent introduction to problem solving using the ASP paradigm see [4]. A number of interesting applications have been developed including planning [6, 21, 24], decision support for the flight controllers of space shut-

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 15–18, 2006.

tles [3], web-based product configuration [29], configuration of a Linux system distribution [28], computer aided verification [17, 13, 18], VLSI routing [10, 7, 12], network management [27], security protocol analysis [1], network inhibition analysis [2], linguistics [11], data and information integration [14], and diagnosis [15]. See also the WASP Showcase Collection (`http://www.kr.tuwien.ac.at/projects/WASP/showcase.html`) compiled by the EU funded Working group on Answer Set Programming (IST project IST-FET-2001-37004).

Available ASP Systems

`Smodels`	`http://www.tcs.hut.fi/Software/smodels/`
`dlv`	`http://www.dbai.tuwien.ac.at/proj/dlv/`
`GnT`	`http://www.tcs.hut.fi/Software/gnt/`
`CMODELS`	`http://www.cs.utexas.edu/users/tag/cmodels.html`
`ASSAT`	`http://assat.cs.ust.hk/`
`nomore++`	`http://www.cs.uni-potsdam.de/nomore/`
`XASP`	distributed with XSB v2.6 `http://xsb.sourceforge.net`
`pbmodels`	`http://www.cs.engr.uky.edu/ai/pbmodels/`
`aspps`	`http://www.cs.engr.uky.edu/ai/aspps/`
`ccalc`	`http://www.cs.utexas.edu/users/tag/cc/`

Acknowledgements

The financial support of Academy of Finland (project 211025) is gratefully acknowledged.

References

1. L.C. Aiello and F. Massacci. Verifying security protocols as planning in logic programming. *ACM Transactions on Computational Logic*, 2(4):542–580, 2001.
2. T. Aura, M. Bishop, and D. Sniegowski. Analyzing single-server network inhibition. In *Proceedings of the IEEE Computer Security Foundations Workshop*, pages 108–117, Cambridge, UK, July 2000. IEEE Computer Society Press.
3. M. Balduccini, M. Barry, M. Gelfond, M. Nogueira, and R. Watson. An A-Prolog decision support system for the space shuttle. In *Proceedings of the Third International Symposium on Practical Aspects of Declarative Languages*, pages 169–183, Las Vegas, Nevada, 2001. Springer-Verlag. Lecture Notes in Computer Science 1990.
4. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
5. F. Buccafurri, N. Leone, and P. Rullo. Strong and weak constraints in disjunctive datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 2–17. Springer-Verlag, 1997.
6. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in nonmonotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pages 169–181, Toulouse, France, September 1997. Springer-Verlag.

7. D. East and M. Truszczyński. More on wire routing with ASP. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 39–44, Stanford, USA, March 2001. AAAI Press.
8. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.
9. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarnello. The KR system dlv: Progress report, comparisons and benchmarks. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417, Trento, Italy, June 1998. Morgan Kaufmann Publishers.
10. E. Erdem, V. Lifschitz, and M.D.F. Wong. Wire routing and satisfiability planning. In *Proceedings of the First International Conference on Computational Logic, Automated Deduction: Putting Theory into Practice*, pages 822–836, London, U.K., July 2000. Springer-Verlag.
11. Esra Erdem, Vladimir Lifschitz, and Don Ringe. Temporal phylogenetic networks and logic programming. *Theory and Practice of Logic Programming*. To appear.
12. Esra Erdem and Martin D. F. Wong. Rectilinear Steiner tree construction using answer set programming. In *Proceedings of the 20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 386–399, 2004.
13. J. Esparza and K. Heljanko. Implementing LTL model checking with net unfoldings. In *Proceedings of the 8th International SPIN Workshop on Model Checking of Software (SPIN'2001)*, pages 37–56, Toronto, Canada, May 2001. Springer-Verlag. Lecture Notes in Computer Science 2057.
14. Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets and their application to data integration. In *Proceedings of the 10th International Conference on Database Theory*, volume 3363 of *Lecture Notes in Computer Science*, pages 306–320, 2005.
15. M. Gelfond and J. Galloway. Diagnosing dynamic systems in A-Prolog. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 160–166, Stanford, USA, March 2001. AAAI Press.
16. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.
17. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.
18. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4&5):519–550, 2003.
19. Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley, 1992.
20. Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2923 of *Lecture Notes in Computer Science*, pages 346–350, 2004.
21. V. Lifschitz. Answer set planning. In *Proceedings of the 16th International Conference on Logic Programming*, pages 25–37, Las Cruces, New Mexico, December 1999. The MIT Press.

22. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 112–117, Edmonton, Alberta, Canada, July/August 2002. The AAAI Press.
23. W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.
24. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
25. I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.
26. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
27. T.C. Son and J. Lobo. Reasoning about policies using logic programs. In *Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, pages 210–216, Stanford, USA, March 2001. AAAI Press.
28. T. Syrjänen. A rule-based formal model for software configuration. Research Report A55, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Helsinki, Finland, December 1999.
29. J. Tiihonen, T. Soininen, I. Niemelä, and R. Sulonen. A practical tool for mass-customising configurable products. In *Proceedings of the 14th International Conference on Engineering Design*, pages 1290–1299, 2003.

Inconsistencies in Ontologies

Andrei Voronkov

Department of Computer Science, University of Manchester, Manchester, UK
voronkov@cs.man.ac.uk
http://www.cs.man.ac.uk/~voronkov

Abstract. Traditionally, theorem provers have been used to prove theorems with relatively small axiomatisations. The recent development of large ontologies poses a non-trivial challenge of reasoning with axiomatisations consisting of hundreds of thousands axioms. In the near future much larger ontologies will be available. These ontologies will be created by large groups of people and by computer programs and will contain knowledge of varying quality.

In the talk we describe an adaptation of the theorem prover Vampire for reasoning with large ontologies using expressive logics. For our experiments we used SUMO and the terrorism ontology. Based on the analysis of inconsistencies found in these ontologies we analyse the quality of information in them. Our research reveals interesting problems in studying the evolution and the quality of formal knowledge.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, p. 19, 2006.

On Arbitrary Selection Strategies for Basic Superposition

Vladimir Aleksic and Anatoli Degtyarev

Department of Computer Science, King's College, Strand, London WC2R 2LS, U.K.
{vladimir, anatoli}@dcs.kcl.ac.uk

Abstract. For first-order Horn clauses without equality, resolution is complete with an arbitrary selection of a single literal in each clause [dN 96]. Here we extend this result to the case of clauses with equality for superposition-based inference systems. Our result is a generalization of the result given in [BG 01]. We answer their question about the completeness of a superposition-based system for general clauses with an arbitrary selection strategy, provided there exists a refutation without applications of the factoring inference rule.

1 Introduction

Since the appearance of paramodulation as a development of resolution for first-order logic with equality, there has been a lot of research in the direction of improving the efficiency of paramodulation-based inference systems. It resulted in numerous refinements of paramodulation, which all aimed at restricting the applicability of the paramodulation inference rule. In this paper, we deal with one such refinement, namely superposition on constrained clauses with constraint inheritance [NR 95], hence with basic superposition.

It is possible to further reduce the search space by applying selection strategies. The key idea is to restrict the application of inference rules by allowing inference only on selected literals. Some of the known complete selection strategies for basic superposition are the maximal strategy (where only maximal literals are selected in each clause) and the positive strategy (where a single negative literal is selected, whenever there is one in a clause).

There has been a few attempts to generalize the completeness results for different selection strategies (for example, see [DKV 95]). The latest result is the one of Bofill and Godoy [BG 01], where they prove that arbitrary selection strategies are complete for a basic superposition calculus on Horn clauses, if it is compatible with the positive strategy. Here we strengthen up their result (and answer a question they posed) by proving that a basic superposition calculus for general first-order clauses is complete with arbitrary selection strategies, provided that there exists a refutation without factoring inferences. A similar result, under the same restriction for factoring inferences, was proved in [dN 96] (Theorem 6.7.4) for resolution calculi, and our result means its generalization to basic superposition calculi.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 20–28, 2006.

2 Preliminaries

Here we present only notions and definitions necessary for understanding the paper. For a more thorough overview, see [NR 01]. It is assumed that the reader has a basic knowledge in substitution and unification.

All formulae are constructed over a fixed signature Σ containing at least one constant and a binary predicate symbol $\approx$. In order to distinguish equality from identity, we use $=$ to denote the latter. By X we denote a set of variables. The set of all *terms* over the signature Σ with variables from X is denoted by $T_\Sigma(X)$ and the set of *ground terms* $T_\Sigma(\emptyset)$ by T_Σ.

An *equation* is an expression denoted by $t_1 \approx t_2$ or equivalently $t_2 \approx t_1$. For dealing with non-equality predicates, atoms $P(t_1, \ldots, t_n)$, where P is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, can be expressed by equations $P(t_1, \ldots, t_n) \approx \mathit{true}$, where *true* is a new symbol. A *literal* is a positive or a negative equation.

The expression $A[s]$ indicates that an expression A contains s as a subexpression. $A[t]$ is a result of replacing the occurrence of s in A by t. An *instance* $A\sigma$ of A is the result of applying the substitution σ to A.

A *clause* is a disjunction of literals, denoted by a formula $L_1, L_2, \ldots, L_m$. This definition allows for multiple occurrences of identical literals, i.e. for treating a clause as a multiset of literals. Sometimes, especially in examples, to improve readability we use the sequent notation by which a clause $\neg A_1, \ldots, \neg A_k, B_1, \ldots B_l$ is represented as $A_1, \ldots, A_k \rightarrow B_1, \ldots, B_l$. A *Horn clause* is a clause that contains only one positive literal.

A *constraint* is a possibly empty conjunction of *atomic equality constraints* $s = t$ or *atomic ordering constraints* $s \succ t$ or $s \succeq t$. The empty constraint is denoted by $\top$.

A *constrained clause* is a pair consisting of a clause C and a constraint T, written as $C \mid T$. The part C will be referred to as the *clause part* and T the *constraint part* of $C \mid T$. A constrained clause $C \mid \top$ will be identified with the unconstrained clause C.

A substitution σ is said to be a *solution of an atomic equality constraint* $s = t$, if $s\sigma$ and $t\sigma$ are syntactically equivalent. It is a *solution of an ordering constraint* $s \succ t$ (with respect to a reduction ordering $>$ which is total on ground terms), if $s\sigma > t\sigma$, and a solution of $s \succeq t$ if it is a solution of $s \succ t$ or $s = t$. Generally, a substitution σ is a solution of a constraint T, if it is a simultaneous solution to all its atomic constraints. A constraint is satisfiable if it has a solution.

A *ground instance* of a constrained clause $C \mid T$ is any ground clause $C\sigma$, such that σ is a ground substitution and σ is a solution to T.

A *contradiction* is a constrained clause $\Box \mid T$, with the empty clause part such that the constraint T is satisfiable. A constrained clause is called *void* if its constraint is unsatisfiable. Void clauses have no ground instances and therefore are redundant.

A set of constrained clauses is *satisfiable* if the set of all its ground instances is satisfiable.

A *derivation* of a constrained clause C from a set of constrained clauses S is a sequence of constrained clauses $C_1, \ldots, C_m$ such that $C = C_m$ and each constrained clause C_i is either an element of S or else the conclusion by an inference rule applied to constrained clauses from premises $C_1, \ldots, C_{i-1}$. A derivation of the contradiction is called a *refutation*.

A *selection strategy* is a function from a set of clauses, that maps each clause to one of its sub-multisets. If a clause is non-empty, then the selected sub-multiset is non-empty too. A derivation is *compatible* with a selection strategy if all the inferences are performed on the selected literals, i.e. all the literals involved in the inferences are selected.

We will often just write "clause" instead of "constrained clause" if it is clear from the context.

3 Completeness for Refutations Without Factoring

In this section we prove that basic superposition is complete with arbitrary selection strategies, provided that there exists a refutation without factoring inferences. Our result is given for the following system **BS** for constrained clauses, which is motivated by strict superposition given in [NR 95]. We drop their factoring inference rule and, for left and right superposition inferences, the literal ordering requirements.

Left superposition

$$\frac{\Gamma_1, l \approx r \mid T_1 \quad \Gamma_2, s[l'] \not\approx t \mid T_2}{\Gamma_1, \Gamma_2, s[r] \not\approx t \mid T_1 \wedge T_2 \wedge l' = l \wedge l \succ r \wedge s \succ t}$$

where l' is not a variable.

Right superposition

$$\frac{\Gamma_1, l \approx r \mid T_1 \quad \Gamma_2, s[l'] \approx t \mid T_2}{\Gamma_1, \Gamma_2, s[r] \approx t \mid T_1 \wedge T_2 \wedge l' = l \wedge l \succ r \wedge s \succ t}$$

where l' is not a variable.

Equality solution

$$\frac{\Gamma, s \not\approx t \mid T}{\Gamma \mid T \wedge s = t}$$

Further in the paper we assume that derivations are *tree-like*, that is, no clause is used more than once as an premise for an inference rule; we may make copies of the clauses in the derivation in order to make it tree-like.

We prove our result by applying so called *permutation rules* to transform derivation trees. A similar approach is used in [dN 96], but for derivations by resolution. For basic superposition calculi, in [BG 01] the authors use a transformation method to prove their result on arbitrary selection on Horn clauses.

However, our transformation is essentially different from the one used in [BG 01], for two reasons. First, we address derivations from general clauses, whereas they restrict themselves to the Horn case. Secondly, their transformation method is constrained by the condition that a superposition-based calculus is complete with the positive selection strategy, while we don't assume any such requirement.

The permutation rules are applied to derivation trees, and their effect is inverting the order of two consecutive inferences. Depending on the inferences involved, they fall into three categories. The permutations we define apply to:

- two superposition inferences,
- two equality solutions,
- a superposition inference and an equality solution.

More in detail, the permutation rules are defined as follows. Wherever the symbol $\dot{\approx}$ is used, it can represent either $\approx$ or $\not\approx$.

s-es rule – Superposition followed by equality solution

$$\dfrac{\dfrac{\Gamma_1, l_1 \approx r_1 \mid T_1 \quad \Gamma_2, s \not\approx t, l_2[l'] \mathrel{\dot{\approx}} r_2 \mid T_2}{\Gamma_1, \Gamma_2, s \not\approx t, l_2[r_1] \mathrel{\dot{\approx}} r_2 \mid T_3}\ (\mathbf{s})}{\Gamma_1, \Gamma_2, l_2[r_1] \mathrel{\dot{\approx}} r_2 \mid T_3 \wedge s = t}\ (\mathbf{es})$$

where T_3 stands for $T_1 \wedge T_2 \wedge l' = l_1 \wedge l_1 \succ r_1 \wedge l_2 \succ r_2$. This sequence of applications of inference rules permutes into:

$$\dfrac{\Gamma_1, l_1 \approx r_1 \mid T_1 \quad \dfrac{\Gamma_2, s \not\approx t, l_2[l'] \mathrel{\dot{\approx}} r_2 \mid T_2}{\Gamma_2, l_2[l'] \mathrel{\dot{\approx}} r_2 \mid T_2 \wedge s = t}\ (\mathbf{es})}{\Gamma_1, \Gamma_2, l_2[r_1] \mathrel{\dot{\approx}} r_2 \mid T_1 \wedge T_2 \wedge s = t \wedge l' = l_1 \wedge l_1 \succ r_1 \wedge l_2 \succ r_2}\ (\mathbf{s})$$

Note that, in order for the permutation to be possible, it is essential that the literals $s \not\approx t$ and $l_2 \mathrel{\dot{\approx}} r_2$ are distinct (in the multiset context). In case they were not, the equality solution in the original derivation would be possible only after the superposition, and therefore the two inferences would never be possible to swap.

es-s rule – Equality solution followed by superposition. This rule is defined as the converse of **s-es**, and its application is always possible.

es-es rule – Two equality solution inferences occur immediately after one another

$$\dfrac{\dfrac{\Gamma, s_1 \not\approx t_1, s_2 \not\approx t_2 \mid T}{\Gamma, s_1 \not\approx t_1 \mid T \wedge s_2 = t_2}\ (\mathbf{es})}{\Gamma \mid T \wedge s_2 = t_2 \wedge s_1 = t_1}\ (\mathbf{es})$$

Since they take place on different literals, they trivially swap.

$$\dfrac{\dfrac{\Gamma, s_1 \not\approx t_1, s_2 \not\approx t_2 \mid T}{\Gamma, s_2 \not\approx t_2 \mid T \wedge s_1 = t_1}\ (\mathbf{es})}{\Gamma \mid T \wedge s_1 = t_1 \wedge s_2 = t_2}\ (\mathbf{es})$$

s-s rule – Two superposition inferences appear one immediately after another

$$\dfrac{\dfrac{\Gamma_1, l_1 \approx r_1 \mid T_1 \quad \Gamma_2, s_2[l'] \mathrel{\dot{\approx}} t_2, l_2 \approx r_2 \mid T_2}{\Gamma_1, \Gamma_2, s_2[r_1] \mathrel{\dot{\approx}} t_2, l_2 \approx r_2 \mid T_4}\,(\mathbf{s}) \quad \Gamma_3, s_3[l''] \mathrel{\dot{\approx}} t_3 \mid T_3}{\Gamma_1, \Gamma_2, \Gamma_3, s_2[r_1] \mathrel{\dot{\approx}} t_2, s_3[r_2] \mathrel{\dot{\approx}} t_3 \mid T_3 \wedge T_4 \wedge l'' = l_2 \wedge l_2 \succ r_2 \wedge s_3 \succ t_3}\,(\mathbf{s})$$

where T_4 represents $T_1 \wedge T_2 \wedge l_1 = l' \wedge l_1 \succ r_1 \wedge s_2 \succ t_2$. Permutation can be done resulting in:

$$\dfrac{\Gamma_1, l_1 \approx r_1 \mid T_1 \quad \dfrac{\Gamma_2, s_2[l'] \not\approx t_2, l_2 \approx r_2 \mid T_2 \quad \Gamma_3, s_3[l''] \not\approx t_3 \mid T_3}{\Gamma_2, \Gamma_3, s_2[l'] \not\approx t_2, s_3[r_2] \not\approx t_3 \mid T_4'}\,(\mathbf{s})}{\Gamma_1, \Gamma_2, \Gamma_3, s_2[r_1] \not\approx t_2, s_3[r_2] \not\approx t_3 \mid T_1 \wedge T_4 \wedge l' = l_1 \wedge l_1 \succ r_1 \wedge s_2 \succ t_2}\,(\mathbf{s})$$

where T_4' is stands for $T_2 \wedge T_3 \wedge l'' = l_2 \wedge l_2 \succ r_2 \wedge s_3 \succ t_3$.

Note that in the above definition of the rule, the conclusion of the first superposition appears as the "from" premise of the superposition inference which follows. This does not restrict the rule, and we assume a definition of its other instance in which the conclusion of the first superposition appears as the "to" premise of the subsequent inference.

Similarly like at the **s-es** rule, it is important to point out scenarios in which this rule can not be applied. A problem would appear if the literal $s_2 \mathrel{\dot{\approx}} t_2$ from the negative premise of the top superposition was used later as the "from" literal, instead of $l_2 \approx r_2$. Luckily, in the consideration below this case will never be met, and we can neglect it at this point.

It could seem that it is necessary to introduce another rule of the type **s-s**, where the superposition inferences to be swapped inferences take place into the same occurence of a literal, but into different positions. However this rule would be redundant in our proof of completeness.

Lemma 1. *The above permutation rules modify* **BS** *derivations into* **BS** *derivations.*

Proof. Every permutation rule defines a way of inverting the order of two adjacent inference rules in a derivation tree. After changing positions, the inferences still take place with the same literals at the same positions in terms as it was in the original derivation. Also, all ordering constraints are kept. Therefore, the resulting derivation is a valid **BS** derivation.

Before proving our main result (see the theorem below), we show by an example the way a refutation can be modified, using the permutation rules, so that it becomes compatible with a chosen selection strategy.

Example 1. Consider the following refutation:

$$\dfrac{\dfrac{\dfrac{a \approx b \quad a \not\approx b, b \approx c}{b \not\approx b, b \approx c}\,(\mathbf{s}_1) \qquad b \not\approx c}{b \not\approx b, c \not\approx c}\,(\mathbf{s}_2)}{\dfrac{b \not\approx b}{\square}\,(\mathbf{es}_1)}\,(\mathbf{es}_2)$$

Assume that a reduction ordering is defined by $a \succ b \succ c$. Lets now "apply" an arbitrary selection strategy to the clauses in the refutation. The selected clauses are underlined, while the framed ones are actually used in the inferences. Note that in unit clauses no literal is boxed nor framed, because by our definition of selection, they are selected by default.

$$\dfrac{\dfrac{\dfrac{a \approx b \quad \boxed{a \not\approx b}, \underline{b \approx c}}{\underline{b \not\approx b}, \boxed{b \approx c}}\,(\mathbf{s}_1) \qquad b \not\approx c}{\dfrac{\underline{b \not\approx b}, \boxed{c \not\approx c}}{\dfrac{b \not\approx b}{\square}\,(\mathbf{es}_2)}\,(\mathbf{es}_1)}\,(\mathbf{s}_2)$$

We modify the proof by "making" the clauses take part in inferences with the selected literals, and we do it from the leaves of the refutation towards the root of the derivation tree. As the first step, we apply the rule **s-s** to the inferences $\mathbf{s}_1$ and $\mathbf{s}_2$.

$$\dfrac{\dfrac{a \approx b \qquad \dfrac{a \not\approx b, \boxed{\underline{b \approx c}} \quad b \not\approx c}{\boxed{a \not\approx b}, \underline{c \not\approx c}}\,(\mathbf{s}'_2)}{\dfrac{\underline{b \not\approx b}, \boxed{c \not\approx c}}{\dfrac{b \not\approx b}{\square}\,(\mathbf{es}_2)}\,(\mathbf{es}_1)}\,(\mathbf{s}'_1)$$

Working further down the refutation tree, we apply the rule **s-es** to the inferences $\mathbf{s}'_1$ and $\mathbf{es}_1$.

$$\dfrac{\dfrac{a \approx b \qquad \dfrac{a \not\approx b, \boxed{\underline{b \approx c}} \quad b \not\approx c}{a \not\approx b, \boxed{\underline{c \not\approx c}}}\,(\mathbf{s}'_2)}{\dfrac{a \not\approx b}{\dfrac{b \not\approx b}{\square}\,(\mathbf{es}_2)}\,(\mathbf{s}''_1)}\,(\mathbf{es}'_1)$$

At this point, it is not necessary to apply permutation rules any further. The refutation is compatible with the chosen selection function.

Theorem 1. *Let S be a set of constrained first-order clauses that has a refutation by* **BS***, which not necessarily employs a selection strategy. Then there exists a refutation compatible with any selection strategy.*

Proof. Let Ω be a refutation from S. Note that, further in the proof, the construction

$$\begin{array}{c}\Omega\\ C\end{array}$$

denotes that the derivation Ω is rooted by the clause C. Consider now a given arbitrary selection, and mark the literals of the clauses of the derivation Ω that are selected. We call *misused* any clause in which the literal that takes part in an inference is not the one selected by the selection function. A clause C is *well-used* if it is not misused and there are no misused clauses in the sub-derivation of Ω rooted by C. We use induction on the number of well-used clauses.

Assume that Ω contains misused clauses and that it is of the form:

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\Omega_1 & \Omega_2\\ C_1 & C_2\end{array}}{C_3}\,(\mathbf{s}_1)\\ \vdots}{\begin{array}{cc}\Omega_5 & \\ C_5 & C_4\end{array}}\,(\mathbf{s}_2)}{C_6}\\ \vdots\\ \square$$

such that C_1 is misused and there are no misused clauses in Ω_1. This is without a loss of generality, and represents only one of a number of essentially similar scenarios in which misused clauses can appear. Assume that the clause C_1 is $\Gamma_1, s_1[l'] \not\approx t_1, l_1 \approx r_1 \mid T_1$, such that the selected literal is $s_1[l'] \not\approx t_1$ and the one used in the inference is $l_1 \approx r_1$. Also assume that the clause C_5 be of the form $\Gamma_5, l_2 \approx r_2 \mid T_5$. Let the inference $\mathbf{s_2}$ take place with the literal $s_1[l'] \not\approx t_1$ and assume that there are no other inferences with the same literal between $\mathbf{s}_1$ and $\mathbf{s}_2$ (therefore there are no inferences into different positions of the same literal). The last assumption makes it possible to apply the permutation rules from the inference $\mathbf{s}_2$ towards the inference $\mathbf{s}_1$, each time moving the application of the clause C_5 one inference up the derivation tree. This way, the derivation Ω transforms to Ω':

$$\dfrac{\dfrac{\dfrac{\begin{array}{cc}\Omega_5 & \Omega_1\\ C_5 & C_1\end{array}}{C'_3}\,(\mathbf{s}'_1) \quad \begin{array}{c}\Omega_2\\ C_2\end{array}}{C'_4}\,(\mathbf{s}'_2)\\ \vdots\\ C'_6\\ \vdots\\ \square$$

where C_6 and C'_6 are variants.

Since no permutation rule is applied to an inference that has a well-used clause as its conclusion, the transformation has not changed the property to be well-used

of any clause from Ω. In addition it has made the clause C_1 well-used. Finally, the transformation has not added to the number of clauses in the refutation and therefore the induction hypothesis applies.

Since in the case of derivations with Horn clauses the factoring inference never appears, the following statement easily follows from the previous theorem.

Corollary 1. *Basic superposition with equality and ordering constraints for Horn clauses is complete with arbitrary selection.*

This result can not be generalized for arbitrary clauses. In the case where all refutations involve factoring, incompleteness for arbitrary selection strategies already appears in the propositional case (see [Ly 97]).

4 Conclusion and Future Work

Our transformations, the same as the transformations in [BG 01], are based on the use of an inference system with inherited constrains. However, there is another representation of the basic strategy introduced in [BGLS 95], which uses *closure substitutions* instead of constraints. Clauses with closure substitutions are called *closures*. The main difference is that the systems of constrained clauses allow for ordering constraints inheritance, but the system of closures do not. Instead of ordering constraints inheritance the rules of left and right superposition are restricted by the ordering condition $l\sigma \succ r\sigma$ for some ground substitution σ which is a solution of the equality constraint in the conclusion.

In [Ly 97] the completeness of arbitrary selection strategy for Horn clauses with closure substitutions was proved using the model generation technique. Unfortunately, as it was noticed in [BG 01], some severe flaws in this completeness proof were discovered. The example below shows that under the weaker ordering inheritance strategy determined by closures, our transformation technique can not be applied, and Theorem 1 does not hold.

Example 2. Let **BS** denote a basic superposition inference system over closures, **s** and **es** denote superposition and equality solution inference rules, respectively.

Consider the following **BS**-derivation over closures: This is a correct **BS**-derivation

$$\dfrac{\dfrac{\dfrac{u \approx g(v) \;\cdot [u \mapsto h(u_1)] \quad p(x,y) \not\approx p(g(z),h(z)),\; h(x) \not\approx g(y) \;\cdot [x \mapsto g(y_1), y \mapsto h(x_1)]}{p(x,y) \not\approx p(g(z),h(z)),\; g(v) \not\approx g(y) \;\cdot [x \mapsto g(y_1), y \mapsto h(x_1)]}\ (\mathbf{s})}{p(x,y) \not\approx p(g(z),h(z)) \;\cdot\; [x \mapsto g(y_1), y \mapsto h(x_1)]}\ (\mathbf{es})}{\square \cdot \varepsilon}\ (\mathbf{es})$$

for every reduction ordering $\succ$.

Let $f(x,y) \not\approx f(g(z),h(z))$ be a selected literal. If we transform this derivations in the style suggested in the previous section, the following derivation is obtained: This is a **BS**-derivation iff $h(g(z) \succ g(h(z))$. If we define $\succ$ to be the

$$\dfrac{u \approx g(v)\ \cdot [u \mapsto h(u_1)] \qquad \dfrac{\dfrac{p(x,y) \not\approx p(g(z),h(z)),\ h(x) \not\approx g(y)\ \cdot [x \mapsto g(y_1), y \mapsto h(x_1)]}{h(x) \not\approx g(y)\ \cdot [x \mapsto g(z), y \mapsto h(z)]}\ (\mathbf{es})}{\dfrac{g(v) \not\approx g(y)\ \cdot\ [y \mapsto h(z)]}{\square \cdot \varepsilon}\ (\mathbf{es})}\ (\mathbf{s})$$

lexicographic path ordering where the precedence is $g > h$, this derivation is not a **BS**-derivation because of the violation of the ordering conditions. However this example is not a counterexample to Lynch's result because we start from closures with non empty substitutions.

References

[BGLS 95] L.Bachmair, H.Ganzinger, C.Lynch and W.Snyder. Basic paramodulation. *Information and Computation*, vol.121, No.2,172–192, 1995.

[BG 01] L. Bofill and G. Godoy. On the completeness of arbitrary selection strategies for paramodulation. In *Proceedings ICALP 2001*, pages 951–962, 2001.

[DKV 95] A. Degtyarev, Y. Koval and A. Voronkov. Handling Equality in Logic Programming via Basic Folding. Technical report 101, Uppsala University, Computing Science Department, 1995.

[dN 96] H. de Nivelle. Ordering refinements of resolution. *Dissertation, Technische Universiteit Delft, Delft*, 1996.

[Ly 97] C. Lynch. Oriented Equational Logic Programming is Complete. *Journal of Symbolic Computations*, 23(1):23–45, 1997.

[NR 95] R. Nieuwenhuis and A. Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computations*, 19:321–351, 1995.

[NR 01] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 3–73, 2001. Elsevier Science Publishers B.V.

An Event-Condition-Action Logic Programming Language*

J.J. Alferes[1], F. Banti[1], and A. Brogi[2]

[1] CENTRIA, Universidade Nova de Lisboa, Portugal
{jja, banti}@di.fct.unl.pt
[2] Dipartimento di Informatica, Università di Pisa, Italy
brogi@di.unipi.it

Abstract. Event-Condition-Action (ECA) languages are an intuitive and powerful paradigm for programming reactive systems. Usually, important features for an ECA language are reactive and reasoning capabilities, the possibility to express complex actions and events, and a declarative semantics. In this paper, we introduce ERA, an ECA language based on, and extending the framework of logic programs updates that, together with these features, also exhibits capabilities to integrate external updates and perform self updates to its knowledge (data and classical rules) and behaviour (reactive rules).

1 Introduction

Event Condition Action (ECA) languages are an intuitive and powerful paradigm for programming reactive systems. The fundamental construct of ECA languages are *reactive rules* of the form **On** $Event$ **If** $Condition$ **Do** $Action$ which mean: when $Event$ occurs, if $Condition$ is verified, then execute $Action$. ECA systems receive inputs (mainly in the form of *events*) from the external environment and react by performing actions that change the stored information (internal actions) or influence the environment itself (external actions). There are many potential and existing areas of applications for ECA languages such as active and distributed database systems [26, 6], Semantic Web applications [21, 24], distributed systems [13], Real-Time Enterprize and Business Activity Management and agents [11].

To be useful in a wide spectrum of applications an ECA language has to satisfy several properties. First of all, events occurring in a reactive rule can be complex, resulting from the occurrence of several basic ones. A widely used way for defining complex events is to rely on some event algebra [10, 1], i.e. to introduce operators that define complex events as the result of compositions of more basic ones that occur at the same or at different instants. Actions that are triggered by reactive rules may also be complex operations involving several (basic) actions that have to be performed concurrently or in a given order and under certain conditions. The possibility to define events and actions in a compositional way (in terms of sub-events and sub-actions), permits a simpler and

* This work has been partly funded by the European Commission under project Rewerse (http://rewerse.net). Thanks are due to Wolfgang May for his comments on previous versions.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 29–42, 2006.

more elegant programming style by breaking complex definitions into simpler ones and by allowing to use the definition of the same entity in different fragments of code.

An ECA language would also benefit from a declarative semantics taking advantage of the simplicity of its the basic concepts. Moreover, an ECA language must in general be coupled with a knowledge base, which, in our opinion, should be richer than a simple set of facts, and allow for the specification of both relational data and classical rules, i.e. rules that specify knowledge about the environment, besides the ECA rules that specify reactions to events. Together with the richer knowledge base, an ECA language should exhibit inference capabilities in order to extract knowledge from such data and rules.

Clearly ECA languages deal with systems that evolve. However, in existing ECA languages this evolution is mostly limited to the evolution of the (extensional) knowledge base. But in a truly evolving system, that is able to adapt to changes in the considered domain, there can be evolution of more than the extensional knowledge base: derivation rules of the knowledge base (intensional knowledge), as well as the reactive rules themselves may change over time. We believe another capability that should be considered is that of *evolving* in this broader sense. Here, by evolving capability we mean that a program should be able to automatically integrate external updates and to autonomously perform self updates. The language should allow updates of both the knowledge (data and classical rules) and the behaviour (reactive rules) of the considered ECA program, due to external and internal changes.

To the best of our knowledge no existing ECA language provides all the above mentioned features (for a detailed discussion see section 5). In particular, none provides the evolving capability, nor it is immediately clear how to incorporate such capability to these languages. The purpose of this paper is to define an ECA language based on logic programming that satisfies all these features. Logic programming (LP) is a flexible and widely studied paradigm for knowledge representation and reasoning based on rules. In the last years, in the area of LP, an amount of effort has been deployed to provide a meaning to updates of logic programs by other logic programs. The output of this research are frameworks that provide meaning to sequence of logic programs, also called Dynamic Logic Programs (DyLPs) [2, 17, 5, 12], and update languages [3, 12, 17, 4] that conjugate a declarative semantics and reasoning capabilities with the possibility to specify (self) evolutions of the program. However, unlike ECA paradigms, these languages do not provide mechanisms for specifying the execution of external actions nor do they provide mechanism for specifying complex events or actions.

To overcome the limitations of both ECA and LP update languages, we present here an ECA language, defined by starting from DyLPs, called ERA (after Evolving Reactive Algebraic programs). This language builds on previous work on the update language Evolp [3], inheriting from it the evolving capabilities, and extending it with the possibility of defining and dealing with complex events and actions, and also considering external actions. The semantics of ERA is defined by means of an *inference system* (specifying what conclusions are derived by a program) and of an *operational semantics* (specifying the effects of actions). The former is derived from the refined semantics for DyLPs [2]. The latter is defined by a transition system inspired by existing work on process algebras. [22, 15].

The rest of the paper is structured as follows: we start in section 2 with an informal introduction to the language introducing its constructs and highlighting its main features. In section 3 we briefly introduce the syntax and semantics of DyLPs, and establish general notation. Section 4 is dedicated to the definition of the syntax and semantics of ERA. The main goals of the paper are the motivation for the language and its formal definition. A study of its properties and formal relation to other systems, cannot be presented here for lack of space. We nevertheless present some comparisons with related work in section 5, where we also draw conclusions and sketch future work.

2 Outline of the Language

Before the formal definition of ERA, which is given in section 4, we start here by informally introducing the various constructs of the language. As stated in the introduction, we aim at defining a language exhibiting both the advantages of ECA languages (with reactive rules, complex events and actions) and of LP updates (with inference rules, possibility of declaratively specifying self-updates). As such, expressions in an ERA program are divided in *rules* (themselves divided into *active, inference and inhibition rules*) and *definitions* (themselves divided into *event* and *action definitions*).

Reactive rules are as usual in ECA languages, and have the form (1), where: $Event$ is a basic or a complex event expressed in an algebra similar to the Snoop algebra [1]; $Condition$ is a conjunction of (positive or negative) literals and $Action$ is a basic or a complex action. *Inference rules* are LP rules with default negation, where default negated heads are allowed [19]. Finally, ERA also includes *inhibition rules* of the form:

When B **Do** $not\ Action$

where B is a conjunction of literals and events. Such an expression intuitively means: when B is true, do not execute $Action$. Inhibition rules are useful for updating the behaviour of reactive rules. If the inhibition rule above is asserted all the rules with $Action$ in the head are updated with the extra condition that B must *not* be satisfied in order to execute $Action$.

ERA allows to combine basic events to obtain complex ones by an event algebra. The operators we use are: $\triangle \mid \bigtriangledown \mid A \mid not$. Intuitively, $e_1 \triangle e_2$ occurs at an instant i iff both e_1 *and* e_2 occur at i; $e_1 \bigtriangledown e_2$ occurs at instant i iff either e_1 *or* e_2 occur at instant i; $not\ e$ occurs at instant i iff e *does not* occur i. $A(e_1, e_2, e_3)$ occurs at the same instant of e_3, in case e_1 occurred before, and e_2 in the middle. This operator is very important since it allows to combine (and reason with) events occurring at different time points.

Actions can also be basic or complex, and they may affect both the stored knowledge (internal actions) or the external environment. Basic external actions are related to the specific application of the language. Basic internal actions are for adding or retracting facts and *rules* (inference, reactive or inhibition rules), of the form $assert(\tau)$ and $retract(\tau)$ respectively, for raising basic events, of the form $raise(e)$. There is also an internal action $define(d)$ for adding new definitions of actions and events (see more on these definitions below). Complex actions are obtained by applying algebraic operators on basic actions. Such operators are: $\triangleright \mid \| \mid IF$, the first for executing actions *sequentially*, and the second for executing them *concurrently*. Executing $IF(C, a_1, a_2)$ amounts to execute a_1 in case C is true, or to execute a_2 otherwise.

For allowing for modularity on the definition of both complex actions and events, ERA allows for *event* and *action definition* expressions. These are of the form, respectively, e_{def} **is** e and a_{def} **is** a where e_{def} (resp. a_{def}) is an atom representing a new event and e (resp. a) is an event (resp. an action) obtained by the event (resp. action) algebra above. It is also possible to use defined events (resp. actions) in the definition of other events (resp. actions).

To better motivate and illustrate these various constructs of the language ERA, including how they concur with the features mentioned in the introduction, we present now an example from the domain of monitoring systems.

Example 1. Consider an (ECA) system for managing electronic devices in a building, viz. the phone lines and the fire security system. The system receives inputs such as signals of sensors and messages from employees and administrators, and can activate devices like electric doors or fireplugs, redirect phone calls and send emails. Sensors alert the system whenever an abnormal quantity of smoke is found. If a (basic) event $(alE(S))$[1], signaling a warning from sensor S occurs, the system opens all the fireplugs Pl in the floor Fl where S is located. This behaviour is encoded by the reactive rule

$$\textbf{On } alE(S) \ \textbf{If } flr(S, Fl), firepl(Pl), flr(Pl, Fl) \textbf{ Do } openA(Pl)$$

The situation is different when the signals are given by several sensors. If two signals from sensors located in different rooms occur without a $stop\_alertE$ event occurring in the meanwhile, the system starts the complex action $fire\_alarmA$, which applies a security protocol: All the doors are unlocked (by the basic action $opendoorsA$) to allow people to leave the building; At the same time, a phone call is sent to a firemen station (by the action $firecallA$); Then the system cuts the electricity in the building (by action $turnA(elect, off)$). $opendoorsA$ and $firecallA$ can be executed simultaneously, but $turnA(elect, off)$ has to be executed after the electric doors have been opened. This behaviour is encoded by following definitions and rules

$$\begin{array}{l} alert2E(S_1, S_2) \ \textbf{ is } A(alE(S_1), alE(S_2), stop\_alertE) \bigtriangledown (alE(S_1) \bigtriangleup alE(S_2)). \\ fire\_alarmA \quad \textbf{ is } (opendoorsA \triangleright turnA(elect, off)) || firecallA. \\ \textbf{On } alert2E(S_1, S_2) \ \textbf{If } not\ same\_room(S_1, S_2) \textbf{ Do } fire\_alarmA. \\ same\_room(S_1, S_2) \leftarrow room(S_1, R_1), room(S_2, R_1). \end{array}$$

The last rule is already a simple example of an inference rule. For another example, suppose that we want to allow the system to be able to notify (by email) all members of a working group in some particular situation. Moreover suppose that working groups are hierarchically defined. Representing in ERA that if an employee belongs to a subgroup she also belongs to its supergroups, can be done by the inference rule[2]:

$$ingroup(Emp, G) \leftarrow ingroup(Emp, S), sub(S, G)$$

We provide now an example of *evolution*. Suppose the administrators decide to update the behaviour of the system such that from then onwards, when a sensor S raises an

[1] In the sequel, we use names of atoms ending in E to represent events, and ending in A to represent actions.

[2] The rules above uses recursion, on the predicate $ingroup/2$, a feature that is beyond the capabilities of many ECA commercial systems, like e.g. SQL-triggers [26].

alarm, only the fireplugs in the room R where S is located is opened. Moreover, each employee can from then onwards command the system to start redirecting phone calls to him (and to stop the previous behaviour of the systems regarding indirections, whatever they were. This behaviour is obtained by updating the system, asserting the following rules and definitions:

R_1 : **When** $alE(S), room(S,R), not\ room(Pl,R)$ **Do** $not\ openA(Pl)$.
R_2 : **On** $redirectE(Emp,Num)$ **If** $true$ **Do** $redirectA(Emp,Num)$.
R_3 : **On** $stop\_redirectE(Emp,Num)$ **If** $true$ **Do** $stop\_redirectA(Emp)$.
$d_1 : redirectA(Emp,Num)$ **is** $assert(\tau_1) \triangleright assert(\tau_2)$.
$d_2 : stop\_redirectA(Emp,Num)$ **is** $retract(\tau_1) \| retract(\tau_2)$.

where τ_1 and τ_2 are the following rules:

τ_1 : **When** $phonE(Call), dest(Call,Emp)$ **Do** $not\ forwA(Call,N)$.
τ_2 : **On** $p\ honE(Call)$ **If** $dest(Call,Emp)$ **Do** $forwA(Call,Num)$.

The formal details of how to update an ERA system are given in section 4.2. Here, when $R1$ is asserted, if $alE(S)$ occurs in room R, any fire plug Pl which is not in R is not opened, even if Pl and S are on the same floor. Reactive rules R_2-R_3 encode the new behaviour of the system when an employee Emp commands the system to start (resp. to stop) redirecting to the phone number Num any phone call $Call$ to him. This is achieved by sequentially asserting (resp. retracting) rules τ_1, τ_2. The former is an inhibition rule that inhibits any previous rule reacting to a phone call for Emp (i.e. to the occurrence of event $phonE(Call)$) by forwarding the call to a number N. The latter is a reactive rule forwarding the call to number Num. Note that τ_1, τ_2 have to be asserted sequentially in order to prevent mutual conflicts. To revert to the previous behaviour it is sufficient to retract τ_1, τ_2 as done by action $stop\_redirectA$.

Such (evolution) changes could alternatively be done by handily modifying the previous rules ie, by *retracting* them and then *asserting* new rules. As with LP updates, also ERA offers the possibility to update reactive rules instead of rewriting. This possibility offered by ERA can be very useful in large systems developed and modified by several programmers and administrators, especially if updates are performed by users that are not aware of the existing rules governing the system, as in the previous example.

Having informally introduced the language, it is now time to start formalizing it. Before that some background on LP updates and notation is required.

3 Background and Notation

In what follows, we use the standard LP notation and, for the knowledge base, *generalized logic programs* (GLP) [19]. Arguments of predicates (here also called atoms) are enclosed within parentheses and separated by commas. Names of arguments with capitalized initials stand for variables, names with uncapitalized initials stand for constants.

A GLP over an alphabet (a set of propositional atoms) $\mathcal{L}$ is a set of rules of the form $L \leftarrow B$, where L (called the head of the rule) is a literal over $\mathcal{L}$, and B (called the body

of the rule) is a set of literals over $\mathcal{L}$. As usual, a literal over $\mathcal{L}$ is either an atom A of $\mathcal{L}$ or the negation of an atom $not\ A$. In the sequel we also use the symbol not to denote complementary default literals, i.e. if $L = not\ A$, by $not\ L$ we denote the atom A.

A (two-valued) *interpretation* I over $\mathcal{L}$ is any set of literals in $\mathcal{L}$ such that, for each atom A, either $A \in I$ or $not\ A \in I$. A set of literals S is true in an interpretation I (or that I satisfies S) iff $S \subseteq I$. In this paper we will use programs containing variables. As usual in these cases a program with variables stands for the propositional program obtained as the set of all possible ground instantiations of its rules. Two rules τ and η are *conflicting* (denoted by $\tau \bowtie \eta$) iff the head of τ is the atom A and the head of η is $not\ A$, or vice versa.

A Dynamic Logic Program $\mathcal{P}$ over an alphabet $\mathcal{L}$ is a sequence $P_1, \ldots, P_m$ where the P_is are GLPs defined over $\mathcal{L}$. Given a DyLP $P_1 \ldots P_n$ and a set of rules R we denote by $\mathcal{P} \setminus R$ the sequence $P_1 \setminus R, \ldots, P_n \setminus R$ where $P_i \setminus R$ is the program obtained by removing all the rules in R from P_i. The *refined stable model semantics* of a DyLP, defined in [2], assigns to each sequence $\mathcal{P}$ a set of refined models (that is proven there to coincide with the set of stable models when the sequence is formed by a single normal or generalized program [19]). The rationale for the definition of a refined model M of a DyLP is made according with the *causal rejection principle* [12, 17]: If the body of a rule in a given update is true in M, then that rule rejects all rules in previous updates that are conflicting with it. Such rejected rules are ignored in the computation of the stable model. In the refined semantics for DyLPs a rule may also reject conflicting rules that belong to the same update. Formally the set of rejected rules of a DyLP $\mathcal{P}$ given an interpretation M is: $Rej^S(\mathcal{P}, M) = \{\tau \in P_i : \exists\, \eta \in P_j\ i \leq j,\ \tau \bowtie \eta\ \wedge\ B(\eta) \subseteq M\}$.

An atom A is false by default if there is no rule, in none of the programs in the DyLP, with head A and a true body in the interpretation M. Formally: $Default(\mathcal{P}, M) = \{not\ A\ :\ \nexists\ A \leftarrow B \in \bigcup P_i\ \ \wedge B \subseteq M\}$. If $\mathcal{P}$ is clear from the context, we omit it as first argument of the above functions.

Definition 1. *Let $\mathcal{P}$ be a DyLP over the alphabet $\mathcal{L}$ and M an interpretation. M is a refined stable model of $\mathcal{P}$ iff $M = least\left(\left(\bigcup P_i \setminus Rej^S(M)\right) \cup Default(M)\right)$, where $least(P)$ denotes the least Herbrand model of the definite program obtained by considering each negative literal $not\ A$ in P as a new atom.*

In the following, a conclusion over an alphabet $\mathcal{L}$ is any set of literals over $\mathcal{L}$. An inference relation $\vdash$ is a relation between a DyLP and a conclusion. Given a DyLP $\mathcal{P}$ with a unique refined model M and a conclusion B, it is natural to define an inference relation $\vdash$ as follows: $P_S \vdash B\ \Leftrightarrow B \subseteq M$ (B is derived iff B is a subset of the unique refined model). However, in the general case of programs with several refined models, there could be several reasonable ways to define such a relation. A possible choice is to derive a conclusion B iff B is a subset of the intersection of all the refined models of the considered program ie, $P_S \vdash B\ \Leftrightarrow\ B \subseteq M\ \forall\ M \in \mathcal{M}(\mathcal{P})$ where $\mathcal{M}(\mathcal{P})$ is the set of all refined models of $\mathcal{P}$. This choice is called *cautious reasoning*. Another possibility is to select one model M (by a selecting function Se) and to derive all the conclusions that are subsets of that model ie, $\mathcal{P} \vdash B\ \Leftrightarrow\ B \subseteq Se(\mathcal{M}(\mathcal{P}))$. This choice is called *brave reasoning*. In the following, in the context of DyLPs, whenever an inference relation $\vdash$ is mentioned , we assume that $\vdash$ is one of the relations defined above.

Let E_S be a sequence of programs (ie, a DyLP) and E_i a GLP, by $E_i.E_S$ we denote the sequence with head E_i and tail E_S. If E_S has length n, by $E_S..E_{n+1}$ we denote the sequence whose first n^{th} elements are those of E_S and whose $(n+1)^{th}$ element is E_{n+1}. For simplicity, we use the notation $E_i.E_{i+1}.E_S$ and $E_S..E_i..E_{i+1}$ in place of $E_i.(E_{i+1}.E_S)$ and $(E_S..E_i)..E_{i+1}$ whenever this creates no confusion. Symbol $null$ denotes the empty sequence. Let E_S be a sequence of n GLPs and $i \leq n$ a natural number, by E_S^i we denote the sequence of the first i^{th} elements of E_S. Let $\mathcal{P} = \mathcal{P}'..P_i$ be a DyLP and E_i a GLP, by $\mathcal{P} \uplus E_i$ we denote the DyLP $\mathcal{P}'..(P_i \cup E_i)$.

4 Formal Definition of ERA

4.1 Syntax of ERA Programs

We start the formal presentation of ERA by defining the syntax introduced in section 2.

Definition 2. *Let $\mathcal{L}$, $\mathcal{E}_B$, $\mathcal{E}_{def}$, $\mathcal{A}_X$ and $\mathcal{A}_{def}$ be sets of atoms respectively called:* condition alphabet*, set of* basic events*, of* event names*, of* external actions*, and of* action names*. Let L, e_b, e_{def}, a_x and a_{def} be generic elements of, respectively, $\mathcal{L}$, $\mathcal{E}_B$, $\mathcal{E}_{def}$, $\mathcal{A}_X$ and $\mathcal{A}_{def}$. The set of* positive events *$\mathcal{E}$ over $\mathcal{E}_B$, and $\mathcal{E}_{def}$ is the set of atoms e_p of the form:*

$$e_p ::= e_b \mid e_1 \triangle e_2 \mid e_1 \bigtriangledown e_2 \mid A(e_1, e_2, e_3) \mid e_{def}$$

where e_1, e_2, e_3 are generic elements of $\mathcal{E}$. An event over *$\mathcal{E}$ is any literal over $\mathcal{E}$. A* negative event *over $\mathcal{E}$ is any literal of the form $not\ e_p$.*

A basic action *a_b over $\mathcal{E}, \mathcal{L}, A_X, \mathcal{A}_{def}$ is any atom of the form:*

$$a_b ::= a_x \mid raise(e_b) \mid assert(\tau) \mid retract(\tau) \mid define(d)$$

where τ (resp. d) is any ERA rule (resp. definition) over $\mathcal{L}^{ERA}$.

The set of actions *$\mathcal{A}$ over $\mathcal{E}, C, A_X, \mathcal{A}_{def}$ is the set of atoms a of the form:*

$$a ::= a_b \mid a_1 \triangleright a_2 \mid a_1 \| a_2 \mid IF(C, a_1, a_2) \mid a_{def}$$

where a_1 and a_2 are arbitrary elements of $\mathcal{A}$ and C is any literal over $\mathcal{E} \cup \mathcal{L}$.

The ERA alphabet *$\mathcal{L}^{ERA}$ over $\mathcal{L}$, $\mathcal{E}_B$, $\mathcal{E}_{def}$, $\mathcal{A}_X$ and $\mathcal{A}_{def}$ is the triple $\mathcal{E}, \mathcal{L}, \mathcal{A}$. Let e and a be arbitrary elements of, respectively, $\mathcal{E}$ and $\mathcal{A}$, B any set of literals over $\mathcal{E} \cup \mathcal{L}$ and $Cond$ any set of literals over $\mathcal{L}$. An* ERA expression *is either an ERA definition or an ERA rule. An* ERA definition *is either an event definition or and action definition. An* event definition *over $\mathcal{L}^{ERA}$ is any expression of the form e_{def} **is** e. An* action definition *over $\mathcal{L}^{ERA}$ is any expression of the form a_{def} **is** a. An* ERA rule *is either an inference, active or inhibition rule over $\mathcal{L}^{ERA}$. An* inference rule *over $\mathcal{L}^{ERA}$ is any rule of the form $L \leftarrow B$. A* reactive rule *over $\mathcal{L}^{ERA}$ is any rule of the form **On** e **If** $Cond$ **Do** a. An* inhibition rule *over $\mathcal{L}^{ERA}$ is any rule of the form **When** B **Do** $not\ a$. An* ERA program *over $\mathcal{L}^{ERA}$ is any set of ERA expressions over $\mathcal{L}^{ERA}$.*

As in DyLPs, ERA considers sequences of programs, each representing an update (with asserted rules or definitions) of the previous ones. Such a sequence is called an *ERA dynamic program*, and determines, at each instant, the behaviour of the system. For this reason the semantics of ERA is given by ERA dynamic programs.

4.2 ERA Systems

The defined syntax allows to program reactive systems, hereafter called *ERA systems*. An ERA system has, at each moment, an ERA dynamic program describing and determining its behaviour, receives input (called *input program*) from the outside, and acts. The actions determine both the evolution of the system (by e.g. adding a new ERA program to the running sequence) and the execution in the external environment. Formally, an input program E_i, over an alphabet $\mathcal{L}^{ERA}$, is any set of either ERA expressions over $\mathcal{L}^{ERA}$ or facts of the form e_b where e_b is an element of $\mathcal{E}_B$ (i.e. a basic event). At any instant i, an ERA systems receives a, possibly empty, input program[3] E_i. The sequence of programs $E_1, \ldots E_n$ denotes the sequence of input programs received at instants $1, \ldots, n$. A basic event e_b *occurs* at instant i iff the fact e_b belongs to E_i. We further assume that every input program contains event $truE$. This allows for defining reactive rules that are always triggered (reacting on event $truE$), or for expressing commands of updates to ERA systems, by having in the input program reactive rules reacting to $truE$ and with empty $true$ condition. For instance, updating the system of example 1 with rule R_1 is done by adding to the input program **On** $truE$ **If** $true$ **Do** $assert(R_1)$.

Since a complex event is obtained by composing basic events that occurred in distinct time instants (viz. when using operator A), for detecting the occurrence of complex events it is necessary to store the sequence of all the received input programs. Formally, an *ERA system* $\mathcal{S}$ is a triple of the form $(\mathcal{P}, E_P, E_i.E_F)$ where $\mathcal{P}$ is an ERA dynamic program, E_P is the sequence of all the previously received input programs and $E_i.E_F$ is the sequence of the current (E_i) and the future (E_F) input programs. As it will be clear from sections 4.3 and 4.4, the sequence E_F does not influence the system at instant i and hence no "look ahead" capability is required. However, since a system is capable (via action $raise$) of autonomously *raising* events in the future, future input programs are included as "passive" elements that are modified as effects of actions (see rule (2)).

The semantics of an ERA system specifies, at each instant, which conclusions are derived, which actions are executed, and what are the effects of those actions. Given a conclusion B, and an ERA system $\mathcal{S}$, notation $S \vdash_e B$ denotes that S derives B (or that B is inferred by S). The definition of $\vdash_e$ is to be found in section 4.3.

At each instant, an ERA system $\mathcal{S}$ *concurrently* executes all the actions a_k such that $\mathcal{S} \vdash_e a_k$. As a result of these actions an ERA system *transits* into another ERA system. While the execution of basic actions is "instantaneous", complex actions may involve the execution of several basic actions in a given order and hence require several transitions to be executed. For this reason, the effects of actions are defined by transitions of the form $\langle \mathcal{S}, A\rangle \mapsto^G \langle \mathcal{S}', A'\rangle$ where $\mathcal{S}, \mathcal{S}'$ are ERA systems, A, A' are sets of actions and G is a set of basic actions. The basic actions in G are the first step of the execution of a set of actions A, while the set of actions A' represents the remaining steps to complete the execution of A. For this reason A' is also called the *set of residual actions* of A. The transition relation $\mapsto$ is defined by a transition system in section 4.4. At each instant an ERA system receives an input program, derives a new set of actions A_N and

[3] ERA adopts a discrete concept of time, any input program is indexed by a natural number representing the instant at which the input program occurs.

starts to execute these actions together with the residual actions not yet executed. As a result, the system evolves according to the transition relation [4] $\rightarrow$. Formally:

$$\frac{A_N = \{a_k \in \mathcal{A} : \ \mathcal{S} \ \vdash_e a_k\} \ \wedge \ \langle \mathcal{S}, (A \cup A_N)\rangle \mapsto^G \langle \mathcal{S}', A'\rangle}{\langle \mathcal{S}, A\rangle \rightarrow^G \langle \mathcal{S}', A'\rangle} \quad (1)$$

4.3 Inferring Conclusions

The inference mechanism of ERA is derived from the inference mechanism for DyLPs. In section 3, we provide two distinct ways (called resp. cautious and brave reasoning) to define an inference relation $\vdash$ between a DyLP and a conclusion on the basis of the refined semantics. From the inference relation $\vdash$, in the following we derive a relation $\vdash_e$ that infers conclusions from an ERA system.

Let $\mathcal{S} = (\mathcal{P}, E_P, E_i.E_F)$ be an ERA system over $\mathcal{L}^{ERA} : \ (\mathcal{E}, \mathcal{L}, \mathcal{A})$, with $E_P = E_1, \ldots E_{i-1}$. For any $m < i$, let $\mathcal{S}^m$ be the ERA system $(\mathcal{P}, E^{m-1}, E^m.null)$. Sequence E_F represents future input programs and is irrelevant for the purpose of inferring conclusions in the present, and sequence E_P stores previous events, and is only used for detecting complex events. The relevant expressions, hence, are those in $\mathcal{P}$ and E_i. As a first step we reduce the expressions of these programs to LP rules. An event definition, associates an event e to a new atom e_{def}. This is encoded by the rule $e_{def} \leftarrow e$. Action definitions, instead, specify what are the effects of actions and hence are not relevant for inferring conclusions. Within ERA, actions are executed iff they are inferred as conclusions. Hence, reactive (resp. inhibition) rules are replaced by LP rules whose heads are actions (resp. negation of actions) and whose bodies are the events and conditions of the rules. Formally: let $\mathcal{P}^R$ and E_i^R be the DyLP and GLP obtained by $\mathcal{P}$ and E_i by deleting every action definition and by replacing:

every rule	**On** e **If** $Condition$ **Do** $Action$.	with $Action \leftarrow Condition, e$.
every rule	**When** B **Do** $not\ Action$	with $not\ Action \leftarrow B$.
every definition	e_{def} **is** e.	with $e_{def} \leftarrow e$.

Basically events are reduced to ordinary literals. Since events are meant to have special meanings, we encode these meanings by extra rules. Intuitively, operators $\triangle$ and $\bigtriangledown$ stands for the logic operators $\wedge$ and $\vee$. This is encoded by the following set of rules

$$ER(\mathcal{E}) : \triangle(e_1, e_2) \leftarrow e_1, e_2. \ \ \bigtriangledown(e_1, e_2) \leftarrow e_1. \ \ \bigtriangledown(e_1, e_2) \leftarrow e_2. \ \forall\, e_1, e_2, e_3 \ \in \mathcal{E}$$

Event $A(e_1, e_2, e_3)$ occurs at instant i iff e_2 occurs at instant i and some conditions on the occurrence of e_1, e_2 and e_3 where satisfied in the previous instants. This is formally encoded by the set of rules $AR(\mathcal{S})$ defined as follows[5]: $AR(\mathcal{S}) =$

$$\left\{ \begin{array}{ll} \forall \ e_1, e_2, e_3 \ \in \mathcal{E} & A(e_1, e_2, e_3) \leftarrow e_2 : \exists\, m < i \ s.t. \\ \mathcal{S}^m \ \vdash_e e_1 \text{ and } \mathcal{S}^m \ \nvdash_e e_3 \text{ and } & (\forall\, j : \ m < j < i : \mathcal{S}^j \ \nvdash_e e_2 \text{ and } \mathcal{S}^j \ \nvdash_e e_3) \end{array} \right\}$$

[4] Transition relation $\mapsto$ defines the effect of the execution of a set of actions, while $\rightarrow$ defines the global evolution of the system.

[5] The definition of $AR(\mathcal{S})$ involves relation $\vdash_e$ which is defined in terms of $AR(\mathcal{S})$ itself . This mutual recursion is well-defined since, at each recursion, $AR(\mathcal{S})$ and $\vdash_e$ are applied on previous instants until eventually reaching the initial instant (i.e. the basic step of the recursion).

The sets of rules E_i^R, $ER(\mathcal{E})$ and $AR(\mathcal{S})$ are added to $\mathcal{P}^R$ and conclusions are derived by the inference relation $\vdash$ applied on the obtained DyLP[6]. Formally:

Definition 3. *Let $\vdash$ be an inference relation defined as in Section 3, and $\mathcal{S}$, $\mathcal{P}^R$, E_i^R, $ER(\mathcal{E})$, $AR(\mathcal{S})$ be as above and K be any conclusion over $\mathcal{E} \cup \mathcal{L} \cup \mathcal{A}$. Then:*

$$(\mathcal{P}, E_P, E_i.E_F) \vdash_e K \quad \Leftrightarrow \quad \mathcal{P}^R \uplus (E_i^R \cup ER(\mathcal{E}) \cup D(\mathcal{P}) \cup AR(\mathcal{S})) \vdash K$$

We specified no rules for operator not . These rules are not needed since event (literal) $not\ e_p$ is inferred by default negation whenever there is no proof for e_p. The following theorem formalizes the intuitive meanings the various operators provided in section 4.1.

Proposition 1. *Let $\mathcal{S}$ be as above, e_b, a basic event, e_p a positive event, e_{def} an event name and e_1, e_2, e_3 three events, the following double implications hold:*

$$\begin{array}{lll}
\mathcal{S} \vdash_e e_1 \triangle e_2 & \Leftrightarrow \mathcal{S} \vdash_e e_1 \ \wedge\ \mathcal{S} \vdash_e e_2. & \mathcal{S} \vdash_e e_b \quad \Leftrightarrow\ e_b \in E_i \\
\mathcal{S} \vdash_e e_1 \bigtriangledown e_2 & \Leftrightarrow \mathcal{S} \vdash_e e_1 \ \vee\ \mathcal{S} \vdash_e e_2. & \mathcal{S} \vdash_e not\ e_p \Leftrightarrow\ \mathcal{S} \nvdash_e e_p. \\
\mathcal{S} \vdash_e A(e_1, e_2, e_3) & \Leftrightarrow\ \exists\, m < i\ s.t.\ \mathcal{S}^m \vdash_e e_1 \ \wedge \mathcal{S}^m \nvdash_e e_3 \ \wedge\ \forall j\ s.t. & \\
 & m < j < i:\ \mathcal{S}^j \nvdash_e e_2 \quad \wedge \mathcal{S}^j \nvdash_e e_3 \ \wedge\ \mathcal{S} \vdash_e e_2. & \\
\mathcal{S} \vdash_e e_{def} & \Leftrightarrow \mathcal{S} \vdash_e e \ \wedge\ e_{def}\ \textbf{\textit{is}}\ e \in \mathcal{P} &
\end{array}$$

4.4 Execution of Actions

We are left with the goal of defining what are the effects of actions. This is accomplished by providing a transition system for the relation $\mapsto$ that completes, together with transition (1) and the definition of $\vdash_e$, the semantics of ERA. As mentioned above, these transitions have the form: $\langle \mathcal{S}, A\rangle \mapsto^G \langle \mathcal{S}', A'\rangle$.

The effects of basic actions on the current ERA program are defined by the *updating function* $up/2$. Let $\mathcal{P}$ be an ERA dynamic program A a set of, either internal or external, basic actions. The output of function $up/2$ is the updated program $up(P, A)$ obtained in the following way: First delete from $\mathcal{P}$ all the rules retracted according to A, and all the (event or action) definitions d_{def} **is** d_{old} such that action $define(d_{def}$ **is** $d_{new})$ belongs to A; then update the obtained ERA dynamic program with the program consisting of all the rules asserted according to A and all the new definitions in A. Formally:

$$\begin{array}{l}
DR(A) = \{d : define(d) \in A\} \ \cup\ \{\tau :\ assert(\tau) \in A\} \cup D(A) \\
R(\mathcal{P}, A) = \{\tau :\ retract(\tau) \in A\} \cup\ \{d_{def}\ \textbf{is}\ d_{old}\ \in \mathcal{P} :\ d_{def}\ \textbf{is}\ d_{new} \in D(A)\} \\
up(\mathcal{P}, A) = (\ \mathcal{P} \setminus R(\mathcal{P}, A)\ ) .. DR(A)
\end{array}$$

Let e_b be any basic event and a_i an external action or an internal action of one of the following forms: $assert(\tau), retract(\tau), define(d)$. On the basis of function $up/2$ above, we define the effects of (internal and external) basic actions. At each transition, the current input program E_i is evaluated and stored in the sequence of past events and the subsequent input program in the sequence E_F becomes the current input program (see

[6] The program transformation above is functional for defining a declarative semantics for ERA, rather than providing an efficient tool for an implementation. Here specific algorithms for event-detection clearly seem to provide a more efficient alternative.

1st and 3rd rules below). The only exception involves action $raise(e_b)$ that adds e_b in the subsequent input program E_{i+1}. When a set of actions A is completely executed its set of residual actions is $\emptyset$. Basic actions (unlike complex ones) are completely executed in one step, hence they have no residual actions. Formally:

$$\begin{array}{rl} \langle(\mathcal{P}, E_P, E_i.E_F), \emptyset\rangle & \mapsto^{\emptyset} \ \langle(\mathcal{P}, E_P..E_i, E_F), \emptyset\rangle \\ \langle(\mathcal{P}, E_P, E_i.E_{i+1}.E_S), \{raise(e_b)\}\rangle & \mapsto^{\emptyset} \ \langle(\mathcal{P}, E_P..E_i, (E_{i+1} \cup \{e_b\}).E_F), \emptyset\rangle \\ \langle(\mathcal{P}, E_P, E_i.E_F), \{a_i\}\rangle & \mapsto^{\{a_i\}} \langle(up(\mathcal{P}, \{a_i\}), E_P..E_i, E_F), \emptyset\rangle \end{array}$$

Note that, although external actions do not affect the ERA system, as they do not affect the result of $up/2$, they are nevertheless *observable*, since they are registered in the set of performed actions (cf. 3rd rule above). Unlike basic actions, generally the execution of a complex action involves several transitions. Action $a_1 \triangleright a_2$, consists into first executing all basic actions for a_1, until the set residual actions is $\emptyset$, then to execute all the basic actions for a_2. We use the notation $A_1 \triangleright a_2$, where A_1 is a set of actions, to denote that action a_2 is executed after all the actions in the set A_1 have no residual actions. Action $a_1 \| a_2$, instead, consists into *concurrently* executing all the basic actions forming both actions, until there are no more of residual actions to execute. Similarly, the execution of a set of actions $A = \{a_1, \ldots, a_n\}$ consists in the concurrent execution of all its actions a_k until the set of residual actions is empty.

The execution of $IF(C, a_1, a_2)$ amounts to the execution of a_1 if the system infers C, or to the execution of e_2 otherwise. Given an ERA system $\mathcal{S} = (\mathcal{P}, E_P, E_i.E_F)$ with $\mathcal{P} : P_1 \ldots P_n$, let $D(\mathcal{S})$ be the set of all the action definitions d such that, for some j, $d \in P_j$ or $d \in E_i$. The execution of action a_{def}, where a_{def} is defined by one or more action definitions, corresponds to the concurrent executions of all the actions a_ks such that a_{def} **is** a_k belongs to $D(\mathcal{S})$. Formally:

$$\frac{\langle \mathcal{S}, \{a_1, a_2\}\rangle \mapsto^{G} \langle \mathcal{S}', A'\rangle}{\langle \mathcal{S}, \{a_1 \| a_2\}\rangle \mapsto^{G} \langle \mathcal{S}', A'\rangle} \quad \frac{\langle \mathcal{S}, \{a_1\}\rangle \mapsto^{G_1} \langle \mathcal{S}', A_1'\rangle}{\langle \mathcal{S}, \{a_1 \triangleright a_2\}\rangle \mapsto^{G_1} \langle \mathcal{S}', \{A_1' \triangleright a_2\}\rangle}$$

$$\frac{\langle \mathcal{S}, A_1\rangle \mapsto^{G_1} \langle \mathcal{S}', A_1'\rangle}{\langle \mathcal{S}, \{A_1 \triangleright a_2\}\rangle \mapsto^{G_1} \langle \mathcal{S}', \{A_1' \triangleright a_2\}\rangle} \quad \frac{\langle \mathcal{S}, \{a_2\}\rangle \mapsto^{G_2} \langle \mathcal{S}'', A_2''\rangle}{\langle \mathcal{S}, \{\emptyset \triangleright a_2\}\rangle \mapsto^{G_2} \langle \mathcal{S}'', A_2''\rangle}$$

$$\frac{\mathcal{S} \vdash_e C \ \wedge \langle \mathcal{S}, \{a_1\}\rangle \mapsto^{G_1} \langle \mathcal{S}', A'\rangle}{\langle \mathcal{S}, \{IF(C, a_1, a_2)\}\rangle \mapsto^{G_1} \langle \mathcal{S}', A_1'\rangle} \quad \frac{\mathcal{S} \nvdash_e C \ \wedge \langle \mathcal{S}, \{a_2\}\rangle \mapsto^{G_2} \langle \mathcal{S}'', A_2''\rangle}{\langle \mathcal{S}, \{IF(C, a_1, a_2)\}\rangle \mapsto^{G_2} \langle \mathcal{S}'', A_2''\rangle}$$

$$\frac{A = \{a_k : \ a_{def} \textbf{ is } a_k. \in D(\mathcal{S})\} \ \wedge \ \langle \mathcal{S}, A\rangle \mapsto^{G} \langle \mathcal{S}', A'\rangle}{\langle \mathcal{S}, \{a_{def}\}\rangle \mapsto^{G} \langle \mathcal{S}', A'\rangle}$$

$$\frac{A = \{a_1, .., a_n\} \ \langle(\mathcal{P}, E_P, E_i.E_{i+1}.E_F), \{a_k\}\rangle \mapsto^{G_k} \langle(\mathcal{P}^k, E_P..E_i, E_{i+1}^k.E_F), A_k'\}\rangle}{\langle(\mathcal{P}, E_P, E_i.E_{i+1}.E_F), A\rangle \mapsto^{\cup G_k} \langle(up(\mathcal{P}, \bigcup G_k), E_P..E_i, \bigcup E_{i+1}^k.E_F), \bigcup A_k'\rangle}$$

As it is clear from this last rule, the definition of concurrent execution of actions in ERA does not rely on any concept of *serialization*. Actions may have, three different effects. Namely: to update the system, to rise new events, or to modify the external environment (by external actions). Semantically, internal updates are defined by function $up/2$ (see section 4.4) which is defined over an ERA dynamic program and a *set* of basic actions, while the raised events are added to the next input program and are then processed concurrently. No serialization is then needed for this kind of actions. Finally, the description and execution of external actions do not belong to the semantics of

ERA, since the meaning and effects of these actions depend on the application domains. Singular applications may require some notion of serialization for external actions (for instance, messages sent over the same communication channel are sent one by one.)

5 Conclusions and Related Work

We identified desirable features for an ECA language, namely: a declarative semantics, the capability to express complex events and actions in a compositional way, and that of receiving external updates, and performing self updates to data, inference rules and reactive rules. For this purpose we defined the logic programming ECA language ERA, and provided it with a declarative semantics based on the refined semantics of DyLPs (for inferring conclusions) and a transition system (for the execution of actions). This new language is close in spirit to LP update languages like EPI [12], LUPS [4], Kabul [17] and, most significantly, Evolp [3]. All these languages have the possibility to update rules (though in EPI and LUPS only derivation rules can be updated). However, none of these supports external nor complex actions or complex events. In [16] Evolp has been extended to consider simple external actions, in the context of an agent architecture. The ERA language goes much beyond in the definition of complex actions and events. A formal comparison with Evolp, clearly showing how ERA is a proper extension of it, cannot be shown here for lack of space.

There exist several alternative proposals of ECA formalisms. Most of these approaches are mainly procedural like, for instance, AMIT [25] and JEDI [13] or at least not fully declarative [26]. A declarative situation calculus-like characterizations of active database systems is given in [6], although the subject of complex actions is not treated there. An example of a Semantic Web-oriented ECA languages is XChange [9], which also has a LP-like semantics, and allows to define reactive rules with complex actions and events. However, it does not support a construct similar to action definitions for defining actions, nor does it consider updates of rules. Updates of rules are also not part of the general framework for reactivity on the semantic web defined in [21]. Defining actions is a possibility allowed by the Agent-Oriented Language DALI [11], which in turn does not support complex events. Another related work is [23] which applies DyLPs to the agent language 3APL. Since 3APL is a language and architecture for programming BDI agents, this work is not directly relatable to ECA paradigms, although future comparisons with ERA could be interesting given the similarity of the semantics for KR. The ideas and methodology for defining complex actions are inspired by works on process algebras like CCS [22] and CSP [15] Rather then proposing high level ECA languages, these works design abstract models for defining programming languages for parallel execution of processes. Other related frameworks are Dynamic Prolog [8] and Transaction Logic Programming (TLP) [7]. These focus on the problem of updating a deductive database by performing transactions. In particular, TLP shares with ERA the possibilities to specify complex (trans)actions in terms of other, simpler, ones. However, TLP (and Dynamic Prolog) does not support complex events, nor does it cope with the possibility of receiving external inputs during the computation of complex actions. Finally, none of these ECA languages show update capabilities analogous to the ones of LP update languages, and that are also in ERA. As such, it is not

obvious how to provide a meaning to inhibition rules or exceptions to rules in those ECA languages.

The language ERA still deserves a significant amount of research. Preliminary investigations evidenced interesting properties of the operators of the action algebra like associativity, commutativity etc, and deserve further study. In this paper we opted for an inference system based on the refined semantics for DyLPs. With limited efforts, it would be possible to define an inference system on the basis of another semantics for DyLPs such as [17, 5, 12]. In particular, we intend to develop a version of ERA based on the well founded semantics of DyLPs [5]. Well founded semantics [14] is a polynomial approximation to the answer set semantics that and is suitable for applications requiring the capability to quickly process vast amount of information. Implementations of the language are subject of ongoing research, where intend to take advantage of existing event-detection algorithms. For simplicity, here we presented a minimal set of operators for the event and action algebras. Specific application domains and confrontations with related languages may suggest eventual extensions of the language. For instance, the language GOLOG [18] presents an operator $\vee$, representing non deterministic choice between two actions which is not expressible in the current definition of ERA. We also plan to provide the possibility to execute ACID transactions in ERA and explore possible relations with Statelog [20].

References

1. Raman Adaikkalavan and Sharma Chakravarthy. Snoopib: Interval-based event specification and detection for active databases. In *ADBIS*, pages 190–204, 2003.
2. J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1), 2005.
3. J. J. Alferes, A. Brogi, J. A. Leite, and L. M. Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *JELIA'02*, LNAI, 2002.
4. J. J. Alferes, L. M. Pereira, H. Przymusinska, and T. Przymusinski. LUPS: A language for updating logic programs. *Artificial Intelligence*, 132(1 & 2), 2002.
5. F. Banti, J. J. Alferes, and A. Brogi. The well founded semantics for dynamic logic programs. In Christian Lemaître, editor, *IBERAMIA-9*, LNAI, 2004.
6. Chitta Baral and Jorge Lobo. Formal characterization of active databases. In *Logic in Databases*, pages 175–195, 1996.
7. A. J. Bonner and M. Kifer. Transaction logic programming. In David S. Warren, editor, *ICLP-93*, pages 257–279, Budapest, Hungary, 1993. The MIT Press.
8. Anthony J. Bonner. A logical semantics for hypothetical rulebases with deletion. *Journal of Logic Programming*, 32(2), 1997.
9. F. Bry, P. Patranjan, and S. Schaffert. Xcerpt and xchange - logic programming languages for querying and evolution on the web. In *ICLP*, pages 450–451, 2004.
10. Jan Carlson and Björn Lisper. An interval-based algebra for restricted event detection. In *FORMATS*, pages 121–133, 2003.
11. Stefania Costantini and Arianna Tocchio. The DALI logic programming agent-oriented language. In *JELIA*, pages 685–688, 2004.
12. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On properties of semantics based on causal rejection. *Theory and Practice of Logic Programming*, 2:711–767, 2002.
13. G.Cugola, E. D. Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *20th Int. Conf. on Software Enginieering*, 1998.

14. A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
15. C.A.R. Hoare. *Communication and Concurrency*. Prentice-Hall, 1985.
16. J. Leite and L. Soares. Enhancing a multi-agent system with evolving logic programs. In K. Satoh K. Inoue and F. Toni, editors, *CLIMA-VII*, 2006.
17. J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.
18. Hector J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 1997.
19. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In B. Nebel, C. Rich, and W. Swartout, editors, *KR-92*, 1992.
20. B. Ludäscher, W. May, and G. Lausen. Nested transactions in a logical language for active rules. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases*, pages 197–222, 1996.
21. W. May, J. Alferes, and R. Amador. Active rules in the Semantic Web: Dealing with language heterogeneity. In *RuleML*, pages 30–44. Springer, 2005.
22. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
23. V. Nigam and J. Leite. Incorporating knowledge updates in 3APL - preliminary report. In R. Bordini, M. Dastani, J. Dix, and A. El F. Seghrouchni, editors, *ProMAS'06*, 2006.
24. S.Abiteboul, C.Culet, L. Mignet, B.Amann, T.Milo, and A. Eyal. Active views for electronic commerce. In *25th Very Large Data Bases Coference Proceedings*, 1999.
25. Vijay A. Saraswat, Radha Jagadeesan, and Vineet Gupta. Amit - the situation manager. *The International Journal on Very Large Data Bases archive*, 13, 2004.
26. J. Widom and S. Ceri, editors. *Active Database Systems – Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann Publishers, 1996.

Distance-Based Repairs of Databases

Ofer Arieli[1], Marc Denecker[2], and Maurice Bruynooghe[2]

[1] Department of Computer Science, The Academic College of Tel-Aviv, Israel
oarieli@mta.ac.il
[2] Department of Computer Science, Katholieke Universiteit Leuven, Belgium
{marcd, maurice}@cs.kuleuven.ac.be

Abstract. We introduce a general framework for repairing inconsistent databases by distance-based considerations. The uniform way of representing repairs and their semantics clarifies the essence behind various approaches to consistency restoration in database systems, helps to compare the underlying formalisms, and relates them to existing methods of defining belief revision operators, merging data sets, and integrating information systems.

1 Introduction and Motivation

Inconsistency of constraint data-sources is a widespread phenomenon. Restoring information consistency (or *repairing* the database) is usually closely related to the *principle of minimal change*, which is the aspiration to reach consistency by a minimal amount of modifications in the 'spoiled' data. To illustrate this, consider the following simple example:

Example 1. Consider a database with two data facts $\mathcal{D} = \{p, r\}$, and an integrity constraint $\mathcal{IC} = p \rightarrow q$. Under the closed world assumption [33], stating that each atomic formula that does not appear in $\mathcal{D}$ is false, this database is clearly inconsistent, as $\mathcal{IC}$ is violated. Two ways of restoring consistency in this case are by inserting q to $\mathcal{D}$ or deleting p from $\mathcal{D}$. Moreover, assuming that integrity constraints cannot be altered, these are the most compact ways of repairing this database, in the sense that any other solution requires a larger amount of changes (i.e., insertions or retractions) in $\mathcal{D}$.

Consistency restoration by minimal change may be traced back to [12] and [35]. In the context of database systems, this notion was introduced by [1], and then considered by many others, including [2, 3, 4, 6, 7, 13, 22, 21, 29, 34]. Some implementations of these methods are reported in [3, 18, 19, 28]. Despite their syntactic and semantic differences, as well as the different notions of repair used by different consistency maintenance formalisms, the rationality behind all these methods is of keeping the 'recovered' data 'as close as possible' to the original (inconsistent) data. This implies that database repairing can be specified in terms of distance semantics, using appropriate metrics.

In this paper, we identify distance-based semantics at the heart of a vast amount of repairing methods, and introduce a corresponding framework for data

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 43–55, 2006.

repair. In this respect, we follow Bertossi's remark [5], that "*identifying general properties of the reasonable repair semantics* [...] *is a very important research direction. Unifying principles seem to be necessary at this stage in order to have a better understanding of consistent query answering*". Indeed, the representation of different repairing methods as distance-based formalisms provides a common ground for relating them. Moreover, we show that the same distance-based considerations are not only the essence of database repairing and consistent query answering, but are also the nucleus of many approaches for belief revision and data integration. In this respect, this work is not restricted to databases only.

The rest of this paper is organized as follows: in Section 2 we give a general representation of consistency restoration in database systems as a distance minimization problem. In Section 3 we consider different distance-based approaches to database repairing, and incorporate the notion of optimal matching (between the spoiled and the recovered data) for generalizing existing repairing methods and defining some new ones. In Section 4 we relate database repairing to different methods of merging independent data-sources. In Section 5 we conclude.

2 Database Repair as a Distance Minimization Problem

Let $\mathcal{L}$ be a first-order language with $\mathcal{P}$ its underlying set of predicates.

Definition 1. A *database* $\mathcal{DB}$ is a pair $(\mathcal{D}, \mathcal{IC})$, where $\mathcal{D}$ is a finite set of ground atomic facts (i.e., atomic formulas without variables) whose predicate names are in $\mathcal{P}$, and $\mathcal{IC}$ is a finite and consistent set of formulae in $\mathcal{L}$.

The set $\mathcal{D}$ in the definition above is called *database instance* and its elements are called *facts*. The meaning of $\mathcal{D}$ is usually determined by the conjunction of its facts augmented with Reiter's closed world assumption [33]. The formulas in $\mathcal{IC}$ are called *integrity constraints*. These formulas specify conditions that should be satisfied, with respect to some underlying semantics $\mathcal{S}$, by all the database facts. We denote this by $\mathcal{D} \models^{\mathcal{S}} \mathcal{IC}$. Common definitions for $\mathcal{S}$ are the standard two-valued semantics, the minimal Herbrand model semantics, the stable model semantics [20], Kleene's three-valued semantics [24], or any other multiple-valued semantics [23]. A semantics $\mathcal{S}$ defines, for a set Γ of formulas in $\mathcal{L}$, the $\mathcal{S}$-*models* of Γ, i.e., a set $mod^{\mathcal{S}}(\Gamma)$ of valuations that satisfy all the formulas in Γ. In this respect $\mathcal{D} \models^{\mathcal{S}} \mathcal{IC}$ means that every element of $mod^{\mathcal{S}}(\mathcal{D})$ is also an $\mathcal{S}$-model of every integrity constraint in $\mathcal{IC}$.

Definition 2. A database $(\mathcal{D}, \mathcal{IC})$ is *consistent* with respect to a semantics $\mathcal{S}$ ($\mathcal{S}$-consistent, for short), if $\mathcal{D} \models^{\mathcal{S}} \mathcal{IC}$.

When a database is not consistent, one or more integrity constraints are violated, and so it is usually required to 'repair' the database, i.e., restore its consistency. For this, given a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$, we denote by $\mathsf{HU}(\mathcal{DB})$ the ground terms in the Herbrand universe of $\mathcal{DB}$, θ_i denotes substitutions of variables by elements in $\mathsf{HU}(\mathcal{DB})$, and $\mathsf{Atoms}(\mathcal{DB})$ denotes the atomic formulas that appear in the formulas of $\mathcal{D} \cup \mathcal{IC}$. For each $P(c_1, \ldots, c_n) \in \mathsf{Atoms}(\mathcal{DB})$, let

$$\mathsf{Upd}\big(P(c_1,\ldots,c_n)\big) = \Big\{P(d_1,\ldots,d_n) \mid \exists 1\leq j\leq n\ d_j \in \mathsf{HU}(\mathcal{DB}) \text{ and } \exists\,\theta_1,\theta_2 \text{ such that } \theta_1\big(P(d_1,\ldots,d_n)\big) = \theta_2\big(P(c_1,\ldots,c_n)\big)\Big\}.$$

An *update* $\mathcal{U}$ of $\mathcal{D}$ is a set that consists of zero or more elements from $\mathsf{Upd}(A)$ for each $A\in\mathsf{Atoms}(\mathcal{DB})$. The set of all the updates of $\mathcal{D}$ is denoted by $\mathsf{Upd}(\mathcal{D})$.

The intuition behind this definition is simple: the predicates in $\mathsf{Atoms}(\mathcal{DB})$ are those that are 'known' to the database, thus they are the relations that potentially appear in a repaired database. Now, the elements in $\mathsf{Upd}\big(P(c_1,\ldots,c_n)\big)$ represent the possible updates of $P(c_1,\ldots,c_n)$. Note that $\{c_i\}$ and $\{d_i\}$ are atomic terms (constants or variables), so the role of the substitutions in the definition of Upd is twofold: to introduce constants instead of variables in predicate tuples (in case that the values are known) and to replace constants by variables (in case that there are wrong values in a tuple and the correct values are unknown. Here, variables may be intuitively regarded as missing (null) values). By this, only erroneous fragments of tuples are modified.[1] The condition that in every tuple at least one component belongs to $\mathsf{HU}(\mathcal{DB})$ is meant to exclude degenerated cases, in which a tuple consists of null values only.

Note 1. Unless all the arguments of an atom A are variables, $A\in\mathsf{Upd}(A)$. Thus, if for some $A\in\mathcal{D}$ and an update $\mathcal{U}$, $\mathcal{U}\cap\mathsf{Upd}(A)=\{A\}$, A remains unchanged. Also, if $\mathcal{U}\cap\mathsf{Upd}(A)=\emptyset$, A is deleted from $\mathcal{D}$, and if $\mathcal{U}\cap\mathsf{Upd}(A)=\{A_1,\ldots,A_n\}$, A is replaced by $n\geq 1$ new facts A_i. Insertions to the database also occur when $\mathcal{U}\cap\mathsf{Upd}(A)$ is not empty for some atom A not in the database instance.

A *potential repair* $\mathcal{R}$ of $\mathcal{DB}$ is an update of $\mathcal{D}$ that preserves $\mathcal{IC}$ with respect to $\mathcal{S}$ ($\mathcal{R}\models^{\mathcal{S}}\mathcal{IC}$). The set of all the potential repairs of $\mathcal{DB}$ is denoted by $\mathsf{Rep}(\mathcal{DB})$.

Example 2. [4] Given the following database instance

$$\left\{\begin{array}{lll} employee(Alice), & salary(Alice,1000), & director(Alice) \\ employee(Bob), & salary(Bob,1000), & \end{array}\right\},$$

and two integrity constraints: one says that every employee has a salary, and the other constraint specifies that a director should earn more money than any other employee. Now, applying here the closed world assumption, we conclude that Bob is not a director. On the other hand, Bob earns the same amount of money as Alice, who *is* a director, so the second integrity constraint is violated. In this case (using some abbreviations with obvious meanings), we have that the updates of the ground facts in the database are the following:

$\mathsf{Upd}(emp(Alice)) = \{emp(Alice)\}$,
$\mathsf{Upd}(emp(Bob)) = \{emp(Bob)\}$,
$\mathsf{Upd}(dir(Alice)) = \{dir(Alice)\}$,
$\mathsf{Upd}(sal(Alice,1000)) = \{sal(Alice,1000), sal(x_A,1000), sal(Alice,val_1)\}$,
$\mathsf{Upd}(sal(Bob,1000)) = \{sal(Bob,1000), sal(x_B,1000), sal(Bob,val_2)\}$. [2]

[1] See also Wijsen's notion of homomorphisms of tableaux [34].

[2] For the other atoms in $\mathsf{Atoms}(\mathcal{DB})$ we have that $\mathsf{Upd}(emp(x)) = \mathsf{Upd}(emp(Alice))\cup\mathsf{Upd}(emp(Bob))$ and $\mathsf{Upd}(sal(x,y)) = \mathsf{Upd}(sal(Alice,1000))\cup\mathsf{Upd}(sal(Bob,1000))$.

Among the possible updates of $\mathcal{D}$ we therefore have the following sets:

$\mathcal{U}_1 = \{emp(Alice), emp(Bob), sal(Alice, 1000), sal(Bob, 1000)\}$,
$\mathcal{U}_2 = \{emp(Alice), emp(Bob), dir(Alice), sal(Alice, val_1), sal(Bob, 1000)\}$,
$\mathcal{U}_3 = \{emp(Alice), emp(Bob), dir(Alice), sal(Alice, 1000), sal(Bob, val_2)\}$.

Note that $\mathcal{U}_1$ is obtained by retracting the fact that Alice is a director (so in this case $\mathsf{Upd}(dir(Alice))$ has no representatives), while $\mathcal{U}_2$ and $\mathcal{U}_3$ cause modifications in the salary of Alice and Bob (respectively). Note also that all these updates are also potential repairs, provided that $val_1 > 1000$ and $val_2 < 1000$.

For selecting the best potential repairs we require that the repaired information should be 'as close as possible' to the original one. Implicitly, then, this criterion involves distance-based considerations and a corresponding metric.

Definition 3. A total function $d : U \times U \to \mathbb{R}^+$ is called *pseudo distance* on U if it is symmetric ($\forall u, v \in U\ d(u, v) = d(v, u)$) and preserves identity ($\forall u, v \in U\ d(u, v) = 0$ iff $u = v$). A *distance function* on U is a pseudo distance on U that satisfies the triangular inequality ($\forall u, v, w \in U\ d(u, v) \leq d(u, w) + d(w, v)$).

Definition 4. A *repair context* for a language $\mathcal{L}$ is a pair $\mathfrak{R} = \langle \models^{\mathcal{S}}, d \rangle$, where $\models^{\mathcal{S}}$ is the entailment relation induced by the underlying semantics $\mathcal{S}$ and d is a pseudo distance on the power set $2^{\mathcal{L}}$ of the well-formed formulae in $\mathcal{L}$.

Repair contexts are parametrized descriptions on how to repair databases. Given a repair context $\mathfrak{R} = \langle \models^{\mathcal{S}}, d \rangle$, the *repairs* of a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ are the instances that $\mathcal{S}$-satisfy $\mathcal{IC}$ and that are d-closest to $\mathcal{D}$. Formally:

Definition 5. The *repairs* of a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ with respect to a repair context $\mathfrak{R} = \langle \models^{\mathcal{S}}, d \rangle$, are the elements of the following set:

$$\Delta_{\mathfrak{R}}(\mathcal{DB}) = \big\{\mathcal{R} \in \mathsf{Rep}(\mathcal{DB}) \mid \forall \mathcal{R}' \in \mathsf{Rep}(\mathcal{DB})\ d(\mathcal{R}, \mathcal{D}) \leq d(\mathcal{R}', \mathcal{D})\big\}.$$

Database repairs induce corresponding notions of query answering:

Definition 6. A *query* $\mathcal{Q}(x_1, \ldots, x_n)$ is a first-order formula with free variables $x_1, \ldots, x_n$. Denote by $\mathcal{Q}[c_1/x_1, \ldots, c_n/x_n]$ the simultaneous substitution in $\mathcal{Q}$ of the variables x_i by the constants c_i ($i = 1, \ldots, n$), respectively. Now, let $\mathfrak{R} = \langle \models^{\mathcal{S}}, d \rangle$ be a repair context, and $\mathcal{Q}(x_1, \ldots, x_n)$ a query on $\mathcal{DB}$.

- A tuple $\langle c_1, \ldots, c_n \rangle$ is a *credulous answer* for $\mathcal{Q}$ if there exists an element $\mathcal{R} \in \Delta_{\mathfrak{R}}(\mathcal{DB})$ s.t. $\mathcal{R} \models^{\mathcal{S}} \mathcal{Q}[c_1/x_1, \ldots, c_n/x_n]$).
- A tuple $\langle c_1, \ldots, c_n \rangle$ is a *conservative answer* (or a *consistent query answer*) for $\mathcal{Q}$ if $\mathcal{R} \models^{\mathcal{S}} \mathcal{Q}[c_1/x_1, \ldots, c_n/x_n])$ for every $\mathcal{R} \in \Delta_{\mathfrak{R}}(\mathcal{DB})$.

3 Distance Semantics for Database Repair

3.1 Distance Functions

The choice of the distance function (and so the metric at hand) plays a crucial role in the repairing process. There are many possibilities to measure distances between the spoiled database instance and its potential repairs. Below we recall two common definitions of such distances:

Definition 7. Let d be a distance function on $\mathcal{L}$. For $A, B \in 2^{\mathcal{L}}$, define:

- *The Hausdorff distance* [15]:

$$d(A,B) = \max\Big(\max_{a\in A}\min_{b\in B} d(a,b),\ \max_{b\in B}\min_{a\in A} d(a,b)\Big).$$

- *Eiter and Mannila's distance* [17]:

$$d(A,B) = \frac{1}{2}\Big(\sum_{a\in A}\min_{b\in B} d(a,b) + \sum_{b\in B}\min_{a\in A} d(a,b)\Big).$$

The following proposition recalls some known facts about these distances:

Proposition 1. *The Hausdorff distance is a distance function on* $2^{\mathcal{L}}$ *and Eiter–Mannila's distance is a pseudo distance on* $2^{\mathcal{L}}$.

In what follows we consider pseudo distances that are defined by matching functions (between the elements of the original database instance and the elements of a potential repair) and by aggregation functions that evaluate the quality of those matchings.

Definition 8. A *numeric aggregation function* f is a total function that accepts multisets of real numbers and returns a real number. Also, f is non-decreasing in the values of its argument,[3] $f(\{x_1,\ldots,x_n\}) = 0$ if $x_1 = \ldots = x_n = 0$, and $\forall x \in \mathbb{R}\ f(\{x\}) = x$.

Definition 9. Let $\mathcal{DB}$ be a database, $A, B \subseteq \mathsf{Atoms}(\mathcal{DB})$, d a (pseudo) distance on the formulae of $\mathcal{L}$, and f a numeric aggregation function.

a) A *matching* m between A and B is a maximal subset of $A\times B$ such that for every $(a_1,b_1),(a_2,b_2) \in m$, $a_1 = a_2$ iff $b_1 = b_2$.

b) For a matching m between A and B, let $m(A) = \{b \mid (a,b) \in m\}$ and $m^{-1}(B) = \{a \mid (a,b) \in m\}$. Denote:

$$d_f(m,A,B) = f\Big(\Big\{d(a,b) \mid (a,b)\in m\Big\} \bigcup \Big\{d(a,B) \mid a \in A\backslash m^{-1}(B)\Big\} \bigcup \Big\{d(b,A) \mid b \in B\backslash m(A)\Big\}\Big),$$

where, for every set S, $d(x,S) = \frac{1}{2}\max\{d(y,z) \mid y,z \in \mathsf{Atoms}(\mathcal{DB})\}$.

Thus, d_f is obtained by applying f on the distances among matched elements and on the distances among non-matched elements and the other set.

c) A matching m between A and B is called $\{d,f\}$-*optimal* if for every matching m' between A and B, $d_f(m,A,B) \le d_f(m',A,B)$.

d) Denote $d_f(A,B) = d_f(m,A,B)$, where m is a $\{d,f\}$-optimal matching between A and B.[4]

[3] That is, the function value is non-decreasing when an element in the multiset is replaced by a larger element.

[4] As all the optimal matchings have the same d_f-value, $d_f(A,B)$ is well-defined.

The aggregation function in Definition 8 may be, e.g., a summation or the average of the distances, the maximum value among those distances (which yields a worst case analysis), a median value (for mean case analysis), and so forth. Such functions are common in data integration systems (see also Section 4 below).

Proposition 2. *The function d_f in Definition 9(d) is a pseudo distance on $2^{\mathcal{L}}$.*[5]

3.2 Aggregation-Based Repairs

Definition 10. An *aggregation-based repair context* is a triple $\mathfrak{R} = \langle \models^{\mathcal{S}}, d, f \rangle$, where $\models^{\mathcal{S}}$ is the entailment relation induced by $\mathcal{S}$, d is a pseudo distance on $\mathcal{L}$, and f is a numeric aggregation function.

Note 2. If $\mathfrak{R} = \langle \models^{\mathcal{S}}, d, f \rangle$ is an aggregation-based repair context in the sense of Definition 10, then $\mathfrak{R} = \langle \models^{\mathcal{S}}, d_f \rangle$, where d_f is obtained from d and f by Definition 9(d), is a repair context in the sense of Definition 4. This is so, since d_f is a pseudo distance on $2^{\mathcal{L}}$ (by Proposition 2).

Definition 11. The *repairs* of a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ with respect to an aggregation-based repair context $\mathfrak{R} = \langle \models^{\mathcal{S}}, d, f \rangle$ are the elements of the set

$$\Delta_{\mathfrak{R}}(\mathcal{DB}) = \big\{ \mathcal{R} \in \mathsf{Rep}(\mathcal{DB}) \mid \forall \mathcal{R}' \in \mathsf{Rep}(\mathcal{DB})\ d_f(\mathcal{R}, \mathcal{D}) \leq d_f(\mathcal{R}', \mathcal{D}) \big\}.$$

By Note 2, Definition 11 is a particular case of Definition 5 for aggregation-based distance functions (and aggregation-based repair contexts).

Example 3. Consider again the database $\mathcal{DB} = (\{p, r\}, \{p \rightarrow q\})$ of Example 1, and let $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$ be an aggregation-based repair context, where d^u is a distance function on the atomic formulas of $\mathcal{L}$, defined by $d^u(s_1, s_2) = 0$ if $s_1 = s_2$, and $d^u(s_1, s_2) = 1$ otherwise. The six potential repairs of $\mathcal{DB}$ and their distances from $\mathcal{D} = \{p, r\}$ are given in the table below.

No.	Potential Repair	$d^u_{\Sigma}(\cdot, \mathcal{D})$	Actions
1	$\{p, q, r\}$	$\frac{1}{2}$	insert q
2	$\{p, q\}$	1	insert q, delete r
3	$\{q, r\}$	1	insert q, delete p
4	$\{q\}$	$1\frac{1}{2}$	insert q, delete p and r
5	$\{r\}$	$\frac{1}{2}$	delete p
6	$\{\}$	1	delete p and r

It follows, then, that the repairs in this case are $\mathcal{R}_1 = \{p, q, r\}$ and $\mathcal{R}_5 = \{r\}$. Among the potential repairs, these repairs require a minimal amount of modifications in $\mathcal{D}$.[6] Thus, e.g., r conservatively (and so credulously) follows from $\mathcal{DB}$, and q credulously (but not conservatively) follows from $\mathcal{DB}$.

[5] Due to lack of space proofs are omitted. Full proofs will appear in an extended version of this paper.

[6] As Proposition 4 below shows, this is not a coincidence.

Definition 12. An aggregation function f such that $f(\{x_1, \ldots, x_n\}) = 0$ only if $x_1 = \ldots = x_n = 0$, is called *strict*. An aggregation-based repair context $\mathfrak{R} = \langle \models^{\mathcal{S}}, d, f \rangle$ is strict if f is strict.

Note that as distances are non-negative, all the aggregation functions on sets of distances considered above (summation, maximum, median, etc.) are strict.[7]

The next proposition shows that, as expected, there is nothing to repair in consistent databases.

Proposition 3. For every strict aggregation-based repair context $\mathfrak{R}$, if $\mathcal{DB}$ is a consistent database, then $\Delta_{\mathfrak{R}}(\mathcal{DB}) = \{\mathcal{D}\}$.

3.3 Domain Independent Repairs

A common definition of an aggregation-based repair context is $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$, where the underlying distance-aggregation function, d^u_Σ, is obtained by a summation of the drastic distances d^u between matched elements (see also Example 3).

Definition 13. The *drastic distance* is $d^u(x, y) = 0$ if $x = y$, else $d^u(x, y) = 1$.

It is easy to verify that d^u and d^u_Σ are distance functions, and both of them are 'blind' to the domain of discourse at hand. Next we show that the metric obtained by d^u_Σ corresponds to the Hamming distance between sets of formulae.[8]

Proposition 4. *Let $|S|$ be the size of S. Then* $d^u_\Sigma(A, B) = \frac{1}{2}(|A \setminus B| + |B \setminus A|)$.

The repair context $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$ corresponds to the repair method introduced in [1], which inspires many other works on (domain independent) database repair (see, e.g., [2, 4, 6, 7, 22, 21, 28]).

Definition 14. [1] A *pairwise*[9] *repair* of $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ is a pair $(\mathsf{Insert}, \mathsf{Retract})$, such that: 1. $\mathsf{Insert} \cap \mathcal{D} = \emptyset$, 2. $\mathsf{Retract} \subseteq \mathcal{D}$, 3. $(\mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}, \mathcal{IC})$ is a consistent database, 4. $(\mathsf{Insert}, \mathsf{Retract})$ is minimal:[10] there is no pair $(\mathsf{Insert}', \mathsf{Retract}')$ that satisfies conditions 1–3 and for which $|\mathsf{Insert}' \cup \mathsf{Retract}'| < |\mathsf{Insert} \cup \mathsf{Retract}|$.

Proposition 5. Consider a database $\mathcal{DB} = (\mathcal{D}, \mathcal{IC})$ and the repair context $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$. Then $(\mathsf{Insert}, \mathsf{Retract})$ is a pairwise repair of $\mathcal{DB}$ iff there is a repair $\mathcal{R} \in \Delta_{\mathfrak{R}}(\mathcal{DB})$ s.t. $\mathsf{Insert} = \mathcal{R} \setminus \mathcal{D}$ and $\mathsf{Retract} = \mathcal{D} \setminus \mathcal{R}$.

It is also interesting to check the distance-based functions of Definition 7 when the domain independent d^u is taken as the basic distance function. In this case the Hausdorff distance is reduced to 0 if $A = B$ and 1 otherwise. While this is still a distance function, it is clearly useless for making subtle preferences among potential repairs. The Eiter–Mannila's distance, on the other hand, is

[7] The minimum function is *not* strict, but it is not useful for repair contexts.
[8] Also known as the symmetric distance, or the Dalal distance [12].
[9] This adjective is added to distinguish this kind of repairs from repairs in our sense.
[10] A different condition may be defined by set inclusion instead of minimal cardinality.

more appropriate in this case, and as in Proposition 5, is it related to pairwise repairing. Indeed, given d^u, the Eiter–Mannila's distance between the original database $\mathcal{D}$ and its repair $\mathcal{R} = \mathcal{D} \cup \mathsf{Insert} \setminus \mathsf{Retract}$ is equal to $\frac{1}{2}(\mathsf{Insert} + \mathsf{Retract})$. In this case we get the Ramon–Bruynooghe matching-based distance [32], which is a distance function (and not only a pseudo distance, cf. Proposition 1).

3.4 Domain-Dependent Repairs

Consider again the database of Example 2. There are several potential repairs in this case. Let's consider two of them:

$\mathcal{R}_1$: remove all the information about Bob from the database,
$\mathcal{R}_2$: change the information about the salary of Bob.

Note that if we use a domain independent repair context with e.g. d^u as the underlying distance function, each potential repair above is as good as the other one, since the cost of the optimal matching between the original database and the repaired database that is obtained by $\mathcal{R}_1$ is the cost of the two retracted elements ($employee(Bob)$ and $salary(Bob, 1000)$) that cannot be matched to an element in the repaired database, which is $\frac{1}{2} + \frac{1}{2} = 1$. Likewise, the optimal matching between the original database and the repaired database that is obtained by $\mathcal{R}_2$ links $employee(Alice)$, $employee(Bob)$, $salary(Alice, 1000)$ and $director(Alice)$ to the same facts in the repaired database, and relates $salary(Bob, 1000)$ to $salary(Bob, x)$ (for some $x < 1000$). The resulting distance is therefore $0 + 0 + 0{+}0{+}1 = 1$. According to the repair context $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$, then, both potential repairs have the same priority. However, in this case, the second repair (salary changes) seems more plausible than the first one (employee removal), as it is more realistic here to assume that the problem is due to a typographic error in the salary information. Moreover, $\mathcal{R}_1$ is more drastic, as it causes information loss (Bob is no longer a reported employee). It is clear, then, that simple cardinality considerations are not useful here, and more delicate considerations, that would yield the preference of $\mathcal{R}_2$ over $\mathcal{R}_1$, are required. [11]

A more subtle preference criterion is obtained by the distance function in [30]:

$$d^1\big(P(t_1,\ldots,t_m), Q(s_1,\ldots,s_n)\big) = \begin{cases} 1 & \text{if } P \neq Q, \\ \dfrac{1}{2n}\displaystyle\sum_{i=1}^{n} d^u(t_i, s_i) & \text{otherwise.} \end{cases}$$

For different predicate symbols the distance d^1 is maximal; however, when the predicate symbols are the same, the distance linearly increases with the number of arguments that have different values, and is at most $\frac{1}{2}$. The intuition behind this is that longer tuples are more error-prone and that multiple errors in the same tuple are less likely.

[11] The need to rectify an error within a tuple without deleting the whole tuple has been acknowledged in [4] (see Example 6.2 of that paper), and is also the main motivation behind the work of Wijsen on database repairing by updates [34].

Proposition 6. *d^1 is a distance function (Definition 3), which is bounded by* 1.

According to d^1, the distance between the database instance $\mathcal{D}$ of Example 2 and $\mathcal{R}_1$ is still 1, while the distance between $\mathcal{D}$ and $\mathcal{R}_2$ is the same as the distance between $salary(Bob, 1000)$ and $salary(Bob, x)$, which is $\frac{1}{4}(0+1) = \frac{1}{4}$. It follows, then, that now $\mathcal{R}_2$ is preferred over $\mathcal{R}_1$, as intuitively expected.

Nienhuys-Cheng's distance d^1 can be further refined to reveal other considerations. For instance, under the assumption that primary keys are less error-prone, one may consider the following variation of d^1:

Definition 15. Below we denote primary key values by underscores, and assume, without loss of generality, that they precede the non-key values. Define:

$$d^2\big(P(\underline{t_1}, \ldots, \underline{t_k}, t_{k+1}, \ldots, t_m), Q(\underline{s_1}, \ldots, \underline{s_l}, t_{l+1}, \ldots, t_n)\big) =$$

$$\begin{cases} 1 & \text{if } P \neq Q \text{ or } \exists 1 \leq i \leq k \text{ s.t. } \underline{t_i} \neq \underline{s_i}, \\ \dfrac{1}{2(m-k)} \displaystyle\sum_{i=k+1}^{m} d^u(t_i, s_i) & \text{otherwise.} \end{cases}$$

Example 4. As noted in Example 2,

$$\mathsf{Upd}(sal(Alice, 1000)) = \{sal(Alice, 1000), sal(x, 1000), sal(Alice, y)\},$$

which means that there are four options regarding the fact $salary(Alice, 1000)$: keeping it unchanged, changing the first argument (employee-name), changing the second argument (salary), or deleting it altogether. Assuming that employee-name is the primary key for the salary relation, according to d^2, the costs of these options are 0, 1, $\frac{1}{2}$ and 1, respectively. Note, also, that in this case, according to the repair context $\mathfrak{R} = \langle \models, d^2, \Sigma \rangle$, the two repairs of the database are:
$\{emp(Alice), emp(Bob), dir(Alice), sal(Alice, v_1), sal(Bob, 1000)\}$ for $v_1 > 1000$,
$\{emp(Alice), emp(Bob), dir(Alice), sal(Alice, 1000), sal(Bob, v_2)\}$ for $v_2 < 1000$.
That is, consistency restoration is obtained here by salary corrections.

3.5 Linking Instead of Matching

The notion of (optimal) matching between the elements of a database instance and its repair may be weakened. Instead of relating each database fact with at most one atomic formula of a repair and vice versa, it is possible to associate a database fact with *several* atoms of a repair. This is called *linking*. Optimal linking and the induced distance between sets are defined just as in Definition 9.

Example 5. Consider a database instance $\mathcal{D} = \{teaches(John, DB)\}$ and integrity constraints that no-one teaches *DB* (since, e.g., this course is cancelled), and that a lecturer must give at least two courses. A repair in this case would be $\mathcal{R} = \{teaches(John, x_1), teaches(John, x_2)\}$ for some $x_1 \neq x_2 \neq DB$. Each one of the two optimal matchings in this case relates the database fact to one of the two elements of $\mathcal{R}$, leaving the other one unmatched. In the notations of the previous section, then, $d^1_\Sigma(\mathcal{D}, \mathcal{R}) = \frac{1}{2} + \frac{1}{4}$. If linking is used instead of matching, there is only one optimal linking between $\mathcal{D}$ and $\mathcal{R}$, which associates the two new facts in $\mathcal{R}$ with the old one in $\mathcal{D}$, hence in this case $d^1_\Sigma(\mathcal{D}, \mathcal{R}) = \frac{1}{4} + \frac{1}{4}$.

3.6 Complexity

Computing all the repairs of a given database is not tractable, as even for propositional databases the number of repairs of a database could be exponential in the database's size. Indeed, the database $(\{p_1, \ldots, p_n\}, \{p_i \rightarrow q_i\}_{i=1}^n)$ has 2^n repairs with respect to $\mathfrak{R} = \langle \models, d^u, \Sigma \rangle$. These repairs correspond to all the combinations of inserting q_i or removing p_i, for $i = 1, \ldots, n$. In an attempt to overcome this problem, most of the existing algorithms for query answering do not compute the repairs themselves, but make inferences using rewriting techniques [1], logic programming paradigms [2, 16, 19, 21, 22], (hyper-)graph computations [10, 11], and proof theoretic methods, such as analytic tableaux [7]. Tractability in such cases is usually reached only for restricted syntactical forms of the integrity constraints. For instance, the technique in [11] is polynomial only for denial integrity constraints[12], and the rewriting technique in [1], which is also tractable, is limited to binary universal constraints. Computational considerations regarding database repairs is beyond the scope of this paper, which is concentrated on the representational aspects of the problem. We note, however, that generally, the distance functions themselves do not add extra computational complexity to the problem. This is demonstrated, for instance, by the following results:

Proposition 7. [32] *Computing $d^u_\Sigma(A, B)$ is polynomial in the size of A and B.*

Proposition 8. *Computing $d^1_\Sigma(A, B)$ and $d^2_\Sigma(A, B)$ is polynomial in the sizes of A, B, and the maximal arity of the predicates in A and B.*

The main computational difficulty remains, therefore, the large amount of potential repairs at hand. Extensive surveys on the computational complexity of existing approaches to database repair and consistent query answering appear in [8, 9, 10] (see also [34] for complexity results regarding update-based repairing).

4 Integration of Constraint Data-Sources

Integration of autonomous data-sources under global integrity constraints (see [26]) is closely related to database repair. The main differences between the two problems is that in contrast to database instances, data-sources may contain negative facts and not only positive ones. Also, the closed world assumption is no longer assumed. In this section we show how our framework may be used for defining operators for the merging problem as well.

Example 6. [26] Four flat co-owners discuss the construction of a swimming pool (s), a tennis-court (t) and a private car-park (p). Building two or more items will increase the rent (r), otherwise the rent will not be changed.

The opinions of the owners are represented by the following four data-sources: $\mathcal{D}_1 = \mathcal{D}_2 = \{s, t, p\}$, $\mathcal{D}_3 = \{\neg s, \neg t, \neg p, \neg r\}$, $\mathcal{D}_4 = \{t, p, \neg r\}$. The impact on the

[12] That is, closed formulae of the form $\forall \overline{x}_1 \ldots \overline{x}_n \neg\big(R_1(\overline{x}_1) \wedge \ldots \wedge R_n(\overline{x}_n) \wedge \phi(\overline{x}_1 \ldots \overline{x}_n)\big)$, where ϕ is a Boolean expression consisting of atomic formulas and built-in predicates.

rent may be represented by the constraint $\mathcal{IC} = \{r \leftrightarrow \big((s \wedge t) \vee (s \wedge p) \vee (t \wedge p)\big)\}$. Note that although the opinion of owner 4 violates the integrity constraint (while the solution must preserve the constraint), it is still taken into account.

In situations such as that of Example 6 it is often required to find a solution that will satisfy the global integrity constraints and will be as close as possible to each data source. This implies that, under the following observations, our framework is adequate for the merging problem as well.

- Instead of database instances, which are sets of atomic facts, data sources are sets of *literals*. Denote by $\mathfrak{D}$ the set of these sources. So, instead of the set of atomic formulas, the following set is considered:
$$\mathsf{Lit}(\mathfrak{D}, \mathcal{IC}) = \bigcup_{\mathcal{D}_i \in \mathfrak{D}} \mathsf{Atoms}(\mathcal{D}_i, \mathcal{IC}) \cup \bigcup_{\mathcal{D}_i \in \mathfrak{D}} \{\neg a \mid a \in \mathsf{Atoms}(\mathcal{D}_i, \mathcal{IC})\}.$$
As before, an update $\mathcal{U}$ of $\mathfrak{D}$ is a consistent set[13] that consists of zero or more elements from $\mathsf{Upd}(L)$ for each $L \in \mathsf{Lit}(\mathfrak{D}, \mathcal{IC})$, and the set $\mathsf{Merge}(\mathfrak{D}, \mathcal{IC})$ of the potential merging of $\mathfrak{D}$ under $\mathcal{IC}$ consists of the updates that satisfy $\mathcal{IC}$.
- A *merging* of data-sources $\mathfrak{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ with respect to the integrity constraints $\mathcal{IC}$ is a straightforward generalization of the notion of database repair (cf. Definitions 10 and 11):
  - A *merging context* is a quadruple $\mathfrak{M} = \langle \models^{\mathcal{S}}, d, f, g \rangle$, where $\models^{\mathcal{S}}$ is the entailment relation induced by the underlying semantics $\mathcal{S}$, d is a pseudo distance function, and f, g are aggregation functions (referring, respectively, to the distances inside a source and among the sources).
  - For a merging context $\mathfrak{M} = \langle \models^{\mathcal{S}}, d, f, g \rangle$, a set $\mathfrak{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ of data-sources, and a potential merging $\mathcal{M} \in \mathsf{Merge}(\mathfrak{D}, \mathcal{IC})$, let
$$d_{g,f}(\mathcal{M}, \mathfrak{D}) = g\big(\{d_f(\mathcal{M}, \mathcal{D}_1), \ldots, d_f(\mathcal{M}, \mathcal{D}_n)\}\big).$$
  - The *mergings* of the data-sources in $\mathfrak{D}$ under $\mathcal{IC}$, and with respect to the merging context $\mathfrak{M} = \langle \models^{\mathcal{S}}, d, f, g \rangle$, are the elements of the set
$$\Delta_{\mathfrak{M}}(\mathcal{DB}) = \big\{\mathcal{M} \in \mathsf{Merge}(\mathfrak{D}, \mathcal{IC}) \mid \forall \mathcal{M}' \in \mathsf{Merge}(\mathfrak{D}, \mathcal{IC})\ d_{g,f}(\mathcal{M}, \mathfrak{D}) \leq d_{g,f}(\mathcal{M}', \mathfrak{D})\ \big\}.$$

Example 7. Consider again Example 6 and two contexts: $\mathfrak{M}_1 = \langle \models, d^u, \Sigma, \Sigma \rangle$, $\mathfrak{M}_2 = \langle \models, d^u, \Sigma, \max \rangle$. According to $\mathfrak{M}_1$ the summation of the distances to the source is minimized, and in $\mathfrak{M}_2$ minimization of maximal distances is used for choosing optimal solutions. The potential mergings in this case are listed below.

No.	Potential merge	$d^u_\Sigma(\cdot, \mathcal{D}_1)$	$d^u_\Sigma(\cdot, \mathcal{D}_2)$	$d^u_\Sigma(\cdot, \mathcal{D}_3)$	$d^u_\Sigma(\cdot, \mathcal{D}_4)$	$d^u_{\Sigma,\Sigma}(\cdot, \mathfrak{D})$	$d^u_{\max,\Sigma}(\cdot, \mathfrak{D})$
$\mathcal{M}_1$	$\{s, t, p, r\}$	$\frac{1}{2}$	$\frac{1}{2}$	4	$1\frac{1}{2}$	$6\frac{1}{2}$	4
$\mathcal{M}_2$	$\{s, t, \neg p, r\}$	$1\frac{1}{2}$	$1\frac{1}{2}$	3	$2\frac{1}{2}$	$8\frac{1}{2}$	3
$\mathcal{M}_3$	$\{s, \neg t, p, r\}$	$1\frac{1}{2}$	$1\frac{1}{2}$	3	$2\frac{1}{2}$	$8\frac{1}{2}$	3
$\mathcal{M}_4$	$\{s, \neg t, \neg p, \neg r\}$	$2\frac{1}{2}$	$2\frac{1}{2}$	1	$2\frac{1}{2}$	$8\frac{1}{2}$	$2\frac{1}{2}$
$\mathcal{M}_5$	$\{\neg s, t, p, r\}$	$1\frac{1}{2}$	$1\frac{1}{2}$	3	$1\frac{1}{2}$	$7\frac{1}{2}$	3
$\mathcal{M}_6$	$\{\neg s, t, \neg p, \neg r\}$	$2\frac{1}{2}$	$2\frac{1}{2}$	1	$1\frac{1}{2}$	$7\frac{1}{2}$	$2\frac{1}{2}$
$\mathcal{M}_7$	$\{\neg s, \neg t, p, \neg r\}$	$2\frac{1}{2}$	$2\frac{1}{2}$	1	$1\frac{1}{2}$	$7\frac{1}{2}$	$2\frac{1}{2}$
$\mathcal{M}_8$	$\{\neg s, \neg t, \neg p, \neg r\}$	$3\frac{1}{2}$	$3\frac{1}{2}$	0	$2\frac{1}{2}$	$9\frac{1}{2}$	$3\frac{1}{2}$

[13] I.e., without complementary literals.

According to $\mathfrak{M}_1$, $\mathcal{M}_1$ is the best potential merging, and so the owners decide to build all the three facilities. As a result, the rent increases. According to $\mathfrak{M}_2$, however, $\mathcal{M}_4$, $\mathcal{M}_6$ and $\mathcal{M}_7$ are the optimal mergings, which implies that only one of the facilities will be built, and so the rent will remain the same.[14] Thus, e.g., r is a consistent answer w.r.t. $\mathfrak{M}_1$ while $\neg r$ is a consistent answer w.r.t. $\mathfrak{M}_2$.

5 Conclusion

Data processing by distance considerations is not a new idea, and it has been used mainly in the context of query answering [1, 2] integration of constraint belief-sets [25, 26] and operators for belief revision [14, 27, 31]. In this paper we introduced a uniform framework for representing, comparing and implementing different approaches for these contexts. Another advantage of our approach is that it opens the door to many new methods that are induced by known distance definitions. This is particularly useful in the context of database repairing, where so far most of the formalisms in the literature that involve distance-based semantics are domain independent while in many practical cases domain dependent repairs are more adequate. The new forms of repairs offered by our framework provide intuitive solutions to such cases, mainly as the notion of closeness can be captured in more subtle ways (most of them are domain dependent), and erroneous components of the data can be detected and updated without violating the valid fragment of the information.

References

1. M. Arenas, L. Bertossi, J. Chomicki. Consistent query answers in inconsistent databases. *Proc. PODS'99*, pp.68–79, 1999.
2. M. Arenas, L. Bertossi, J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Log. Prog.*, 3(4–5):393–424, 2003.
3. O. Arieli, M. Denecker, B. Van Nuffelen, M. Bruynooghe. Coherent integration of databases by abductive logic programming. *Artif. Intell. Res.*, 21:245–286, 2004.
4. O. Arieli, M. Denecker, B. Van Nuffelen, M. Bruynooghe. Computational methods for database repair by signed formulae. *Annals Math. Artif. Intell.*, 46:4–37, 2006.
5. L. Bertossi. Some research directions in consistent query answering: a vision. *Preproc. of EDBT'06 Workshop on Inconsistency in Databases*, pp.109–113, 2006.
6. L. Bertossi, J. Chomicki, A. Cortés, C. Gutierrez. Consistent answers from integrated data sources. *Proc. FQAS'2002*, LNCS 2522, pp.71–85, 2002.
7. L. Bertossi, C. Schwind. Database repairs and analytic tableau. *Annals of Mathematics and Artificial Intelligence*, 40(1–2):5–35, 2004.
8. A. Cali, D. Lembo, R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. *Proc. PODS'03*, 260–271, 2003.
9. J. Chomicki, J. Marchinkowski. A on the computational complexity of minimal-change integrity maintenance in relational databases. In L. Bertossi, A. Hunter, and T. Schaub, editors, *Inconsistency Tolerance*, LNCS 3300, pp.119–150. 2005.

[14] The decision which facility to choose requires further preference criteria. Summation of distances, e.g., prefers $\mathcal{M}_6$ and $\mathcal{M}_7$ over $\mathcal{M}_4$, thus t and p are preferred over s.

10. J. Chomicki, J. Marchinkowski. Minimal-change integrity maintenance using tuple deletion. *Journal of Information and Computation*, 197(1–2):90–121, 2005.
11. J. Chomicki, J. Marchinkowski, and S. Staworko. Computing consistent query answers using conflict hypergraphs. *Proc. CIKM'04*, pp.417–426, 2004.
12. M. Dalal. Investigations into a theory of knowledge base revision. *Proc. AAAI'98*, pp.475–479. AAAI Press, 1988.
13. S. de Amo, W. Carnielli, J. Marcos. A logical framework for integrating inconsistent information in multiple databases. *Proc. FoIKS'02*, LNCS 2284, pp.67–84, 2002.
14. J. Delgrande. Preliminary considerations on the modelling of belief change operators by metric spaces. *Proc. NMR'04*, pp.118–125, 2004.
15. J. Dieudonné, editor. *Foundations of Modern Analysis*. Academic Press, 1969.
16. T. Eiter. Data integration and answer set programming. *Proc. LPNMR'05*, LNCS 3662, pp.13–25. Springer, 2005.
17. T. Eiter, H. Mannila. Distance measure for point sets and their computation. *Acta Informatica*, 34:109–133, 1997.
18. B. Fazzinga, S. Flesca, F. Furfaro, F. Parisi. DART: a data acquisition and repairing tool. *EDBT'06 Workshop on Inconsistency in Databases*, pp.2–16, 2006.
19. E. Franconi, A. Palma, N. Leone, D. Perri, F. Scarcello. Census data repair: A challenging application of disjunctive logic programming. *Proc. LPAR'01*, LNCS 2250, pp.561–578. Springer, 2001.
20. N. Gelfond, V. Lifschitz. The stable model semantics for logic programming. *Proc. 5th Logic Programming Symposium*, pp.1070–1080. MIT Press, 1988.
21. G. Greco, S. Greco, E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. *Proc. ICLP'01*, LNCS 2237, pp.348–363. Springer, 2001.
22. S. Greco, E. Zumpano. Querying inconsistent databases. *Proc. LPAR'2000*, LNAI 1955, pp.308–325. Springer, 2000.
23. M. Kifer, E. L. Lozinskii. A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, 1992.
24. S. C. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1950.
25. S. Konieczny, J. Lang, P. Marquis. Distance-based merging: A general framework and some complexity results. *Proc KR'02*, pp.97–108, 2002.
26. S. Konieczny, R. Pino Pérez. Merging information under constraints: a logical framework. *Journal of Logic and Computation*, 12(5):773–808, 2002.
27. D. Lehmann, M. Magidor, K. Schlechta. Distance semantics for belief revision. *Journal of Symbolic Logic*, 66(1):295–317, 2001.
28. N. Leone, T. Eiter, W. Faber, M. Fink, G. Gottlob, G. Greco. Boosting information integration: The INFOMIX system. *Proc. SEBD'05*, pp.55–66, 2005.
29. P. Liberatore, M. Schaerf. BReLS: A system for the integration of knowledge bases. *Proc. KR'2000*, pp.145–152. Morgan Kaufmann Publishers, 2000.
30. S.H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. *Proc. ILP'97*, LNCS 1297, pp.213–226, 1997.
31. P. Peppas, S. Chopra, N. Foo. Distance semantics for relevance-sensitive belief revision. *Proc. KR'04*, pp.319–328. AAAI Press, 2004.
32. J. Ramon, M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.
33. R. Reiter. On closed world databases. In *Logic and Databases*, pages 55–76. 1978.
34. J. Wijsen. Database repairing using updates. *ACM Transactions on Database Systems*, 30(3):722–768, 2005.
35. M. Winslett. Reasoning about action using a possible models approach. *Proc. AAAI'98*, pp.89–93. AAAI press, 1988.

Natural Deduction Calculus for Linear-Time Temporal Logic

Alexander Bolotov[1], Artie Basukoski[1], Oleg Grigoriev[2,⋆], and Vasilyi Shangin[2]

[1] Harrow School of Computer Science
University of Westminster
Watford Road, Harrow HA1 3TP, UK
A.Bolotov@wmin.ac.uk
http://www2.wmin.ac.uk/bolotoa/index.html
[2] Department of Logic, Faculty of Philosophy, Moscow State University, Moscow, 119899, Russia
{shangin, grig}@philos.msu.ru

Abstract. We present a natural deduction calculus for the propositional linear-time temporal logic and prove its correctness. The system extends the natural deduction construction of the classical propositional logic. This will open the prospect to apply our technique as an automatic reasoning tool in a deliberative decision making framework across various AI applications.

1 Introduction

In this paper we present a natural deduction proof system for the propositional linear-time temporal logic PLTL [7] and establish its correctness. Natural deduction calculi (abbreviated in this paper by 'ND') originally were developed by Gentzen [8] and Jaskowski [9]. Jaskowski-style natural deduction was improved by Fitch [5] and simplified by Quine [14].

It is notable that further development of such systems was controversial. Although there has been an obvious interest in these ND formalisms as representing a 'natural' way of reasoning, ND systems were often considered as inappropriate for an algorithmic representation [6]. This scepticism is not surprising because in general we can have in the proof formulae that violate *the subformula property* (often thought as crucial for automated deduction), which requires that in a proof, any formula which occurs in the conclusion of a rule, is a (negation of) subformula of its premises.

As a consequence, ND systems have been primarily studied within the framework of philosophical logic, being widely used in teaching (but again, mostly in the philosophy curriculum) and have been ignored by the automated theorem-proving community, where research has mostly concentrated on purely *analytic* methods such as *resolution* and *tableau* based approaches [1].

⋆ This research was partially supported by Russian Foundation for Humanities, grant No 06-03-00020a.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 56–68, 2006.

Recently, ND systems have been studied within a wider community. One of the most recent examples of the interest in natural deduction is the area of *logical frameworks* [12], where the notion of *hypothetical* judgements, i.e. reasoning from hypothesis, as in natural deduction, is essential. Here, in particular, ND systems have been developed for intuitionistic linear logic [13].

In this paper we define a natural deduction proof system for the propositional linear-time temporal logic PLTL [7] and establish its correctness. The particular approach to build an ND-calculus we are interested in is described in detail in [2]. It is a modification of Quine's representation of subordinate proof [14] developed for classical propositional and first-order logic. The ND technique initially defined for classical propositional logic was extended to first-order logic [2, 3]. It has also been extended to the non-classical framework of propositional intuitionistic logic [10], where the proof-searching strategies are based upon the proof-searching strategies for classical propositional natural deduction calculus.

We believe that the goal-directed nature of our proof searching technique opens broad prospects for the application of the method in many AI areas, most notably, in agent engineering [16].

The paper is organized as follows. In §2 we review the syntax and semantics of PLTL. In §3 we describe the ND for PLTL henceforth referred to as $PLTL_{ND}$ and give an example of the construction of the proof. Subsequently, in §4, we provide the correctness argument. Finally, in §5, we provide concluding remarks and identify future work.

2 Syntax and Semantics of PLTL

We define the language of PLTL using the following symbols.

- a set, *Prop*, of atomic propositions: $p, q, r, \ldots, p_1, q_1, r_1, \ldots, p_n, q_n, r_n, \ldots$;
- classical operators: $\neg, \wedge, \Rightarrow, \vee$;
- temporal operators:
  $\Box$ – 'always in the future';
  $\Diamond$ – 'at sometime in the future';
  $\bigcirc$ – 'at the next moment in time';
  $\mathcal{U}$ – 'until'.

The set of *well-formed formulae* of PLTL, wff_{PLTL} is defined as follows.

Definition 1 (PLTL syntax).

1. *All atomic propositions (members of Prop) are in* wff_{PLTL}.
2. *If* A *and* B *are in* wff_{PLTL}, *then so are* $A \wedge B$, $\neg A$, $A \vee B$, *and* $A \Rightarrow B$.
3. *If* A *and* B *are in* wff_{PLTL}, *then so are* $\Box A$, $\Diamond A$, $\bigcirc A$, *and* $A \mathcal{U} B$.

For the semantics of PLTL we utilise the notation of [4]. A model for PLTL formulae, is a discrete, linear sequence of states

$$\sigma = s_0, s_1, s_2, \ldots$$

which is isomorphic to the natural numbers, $\mathcal{N}$, and where each state, $s_i,\ 0 \leq i$, consists of the propositions that are true in it at the i-th moment of time. If a well-formed formula A is satisfied in the model σ at the moment i then we abbreviate it by $\langle \sigma, i \rangle \models A$. Below, in Figure 1, we define the relation $\models$, where indices $i, j, k \in \mathcal{N}$.

$\langle \sigma, i \rangle \models p$	iff	$p \in s_i$, for $p \in Prop$
$\langle \sigma, i \rangle \models \neg A$	iff	$\langle \sigma, i \rangle \not\models A$
$\langle \sigma, i \rangle \models A \wedge B$	iff	$\langle \sigma, i \rangle \models A$ and $\langle \sigma, i \rangle \models B$
$\langle \sigma, i \rangle \models A \vee B$	iff	$\langle \sigma, i \rangle \models A$ or $\langle \sigma, i \rangle \models B$
$\langle \sigma, i \rangle \models A \Rightarrow B$	iff	$\langle \sigma, i \rangle \not\models A$ or $\langle \sigma, i \rangle \models B$
$\langle \sigma, i \rangle \models \Box A$	iff	for each j if $i \leq j$ then $\langle \sigma, j \rangle \models A$
$\langle \sigma, i \rangle \models \Diamond A$	iff	there exists j such that $i \leq j$ and $\langle \sigma, j \rangle \models A$
$\langle \sigma, i \rangle \models \bigcirc A$	iff	$\langle \sigma, i+1 \rangle \models A$
$\langle \sigma, i \rangle \models A \mathcal{U} B$	iff	there exists j such that $i \leq j$ and $\langle \sigma, j \rangle \models B$ and for each k if $i \leq k < j$ then $\langle \sigma, k \rangle \models A$

Fig. 1. Semantics for PLTL

Definition 2 (PLTL Satisfiability). *A well-formed formula, A, is satisfiable if, and only if, there exists a model σ such that $\langle \sigma, 0 \rangle \models A$.*

Definition 3 (PLTL Validity). *A well-formed formula, A, is valid if, and only if, A is satisfied in every possible model, i.e. for each σ, $\langle \sigma, 0 \rangle \models A$.*

3 Natural Deduction System $PLTL_{ND}$

3.1 Extended PLTL Syntax and Semantics

To define the rules of the natural system we extend the syntax of PLTL by introducing labelled formulae.

Firstly, we define the set of labels, *Lab*, as a set of variables interpreted over states of σ:

$$Lab : \{x, y, z, x_1, x_2, x_3, \ldots\}.$$

Let g be a function, which maps the set *Lab* to $\mathcal{N}$.

We then define two binary relations '$\preceq$' and 'Next', and the operation $'$ as follows.

Definition 4 (Relations $\prec, \simeq, \preceq$ and $Next$, operation $'$). *For $x,\ y \in Lab$:*

(4.1) $\prec \subset Lab^2 : x \prec y \Leftrightarrow g(x) < g(y)$,
(4.2) $\simeq \subset Lab^2 : x \simeq y \Leftrightarrow g(x) = g(y)$,
(4.3) $\preceq \subset Lab^2 : x \preceq y \Leftrightarrow g(x) \leq g(y)$,
(4.4) $Next \subset Lab^2 : Next(x, y) \Leftrightarrow g(y) = g(x) + 1$, *i.e. it is the 'predecessor-successor' relation such that for any $i \in Lab$, there exists $j \in Lab$ such that $Next(i, j)$ (seriality),*

(4.5) Given a label i, the operation $'$ applied to i gives us the label i' such that $Next(i, i')$.

The following properties follow straightforwardly from Definition 4.

Lemma 1 (Properties $\preceq$ and $Next$).

- *For any $i, j \in Lab$ if $Next(i, j)$ then $i \preceq j$.*
- *For any $i, j \in Lab$ if $i \prec j$ then $i \preceq j$.*
- *Properties of $\preceq$:*
  - *For any $i \in Lab: \ i \preceq i$ (reflexivity),*
  - *For any $i, j, k \in Lab$ if $i \preceq j$ and $j \preceq k$ then $i \preceq k$ (transitivity).*

Following [15], the expressions representing the properties of $\preceq$ and $Next$ are called 'relational judgements'.

Now we are ready to introduce the $PLTL_{ND}$ syntax.

Definition 5 ($PLTL_{ND}$ Syntax).

- *If A is a PLTL formula and $i \in Lab$ then $i{:}A$ is a $PLTL_{ND}$ formula.*
- *Any relational judgement of the type $Next(i, i')$ and $i \preceq j$ is a $PLTL_{ND}$ formula.*

$PLTL_{ND}$ Semantics. For the interpretation of $PLTL_{ND}$ formulae we adapt the semantical constructions defined in §2 for the logic PLTL. In the rest of the paper we will use capital letters $A, B, C, D, \ldots$ as metasymbols for PLTL formulae, and calligraphic letters $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \ldots$ to abbreviate formulae of $PLTL_{ND}$, i.e. either labelled formulae or relational judgements. The intuitive meaning of $i{:}A$ is that A is satisfied at the world i. Thus, based on our observations above, we simply need the following statements.

Let Γ be a set of $PLTL_{ND}$ formulae, let $D_\Gamma = \{x | x : A \in \Gamma\}$, let σ be a model as defined in §2 and let f be a function which maps elements of D_Γ into $\mathcal{N}$ (recall that a PLTL model σ is isomorphic to natural numbers).

Definition 6 (Realisation of $PLTL_{ND}$ formulae in a model). *Model σ* realises *a set, Γ, if there is a mapping, f, which satisfies the following conditions.*

(1) For any $x \in D_\Gamma$, and for any A, if $x{:}A \in \Gamma$ then $\langle \sigma, f(x) \rangle \models A$,
(2) For any x, y, if $x \preceq y \in \Gamma$, and $f(x) = i$, and $f(y) = j$ then $i \leq j$,
(3) For any x, y, if $Next(x, y) \in \Gamma$, and $f(x) = i$, and $f(y) = j$ then $j = i + 1$.

The set Γ in this case is called realisable.

Definition 7 ($\mathbf{PLTL}_{ND}$ Validity). *A well-formed $PLTL_{ND}$ formula, $\mathcal{A} = i{:}B$, is valid (abbreviated as $\models_{ND} \mathcal{A}$) if, and only if, the set $\{\mathcal{A}\}$ is realisable in every possible model, for any function f.*

It is easy to see that if we ignore the labels then the classes of satisfiable and valid formulae introduced by definitions 2 and 6, 3 and 7 respectively, are identical.

3.2 Rules for Boolean Operations

The set of rules is divided into the two classes: *elimination* and *introduction* rules. Rules of the first group allow us to simplify formulae to which they are applied. These are rules for the 'elimination' of logical constants. Rules of the second group are aimed at 'building' formulae, introducing new logical constants.

In Figure 2 we define the sets of elimination and introduction rules, where prefixes '*el*' and '*in*' abbreviate an elimination and an introduction rule, respectively.

Elimination Rules :		Introduction Rules :	
$\wedge\, el_1$	$\dfrac{i{:}A \wedge B}{i{:}A}$	$\wedge\, in$	$\dfrac{i{:}A, \quad i{:}B}{i{:}A \wedge B}$
$\wedge\, el_2$	$\dfrac{i{:}A \wedge B}{i{:}B}$	$\vee\, in_1$	$\dfrac{i{:}A}{i{:}A \vee B}$
$\vee\, el$	$\dfrac{i{:}A \vee B, \quad i{:}\neg A}{i{:}B}$	$\vee\, in_2$	$\dfrac{i{:}B}{i{:}A \vee B}$
$\Rightarrow el$	$\dfrac{i{:}A \Rightarrow B, \quad i{:}A}{i{:}B}$	$\Rightarrow\, in$	$\dfrac{[i{:}C], \quad i{:}B}{i{:}C \Rightarrow B}$
$\neg\, el$	$\dfrac{i{:}\neg\neg A}{i{:}A}$	$\neg\, in$	$\dfrac{[j{:}C], \quad i{:}B, \quad i{:}\neg B}{j{:}\neg C}$

Fig. 2. $PLTL_{ND}$-rules for Booleans

– In the formulation of the rules '$\Rightarrow in$' and '$\neg\ in$' formulae $[i{:}C]$ and $[j{:}C]$ respectively must be the most recent non discarded [3] assumptions occurring in the proof. When we apply one of these rules on step n and discard an assumption on step m, we also discard all formulae from m to $n-1$. We will write $[m-(n-1)]$ to indicate this situation.

3.3 Rules for Temporal Logic

In the formulation of the set of elimination and introduction rules for temporal operators we use the notions of *flagged* and *relatively flagged* label with the meaning similar to the notions of flagged and relatively flagged variable in first order logic [3]. By saying that the label, j, is flagged, abbreviated as $\mapsto j$, we mean that it is bound to a state and, hence, cannot be rebound to some other state. By saying that a variable i is relatively flagged (bound) by j, abbreviated as $j \mapsto i$ we mean that a bounded variable, j, restricts the set of runs for i that is linked to it in the relational judgment, for example $i \preceq j$.

In Figure 3 we define elimination and introduction rules for the temporal logic operators.

> The condition $\forall C(j : C \notin M1)$ in the rules $\Diamond_{el}$, $\mathcal{U}_{el_1}$ means that the label j should not occur in the proof in any formula, C, that is marked by $M1$.

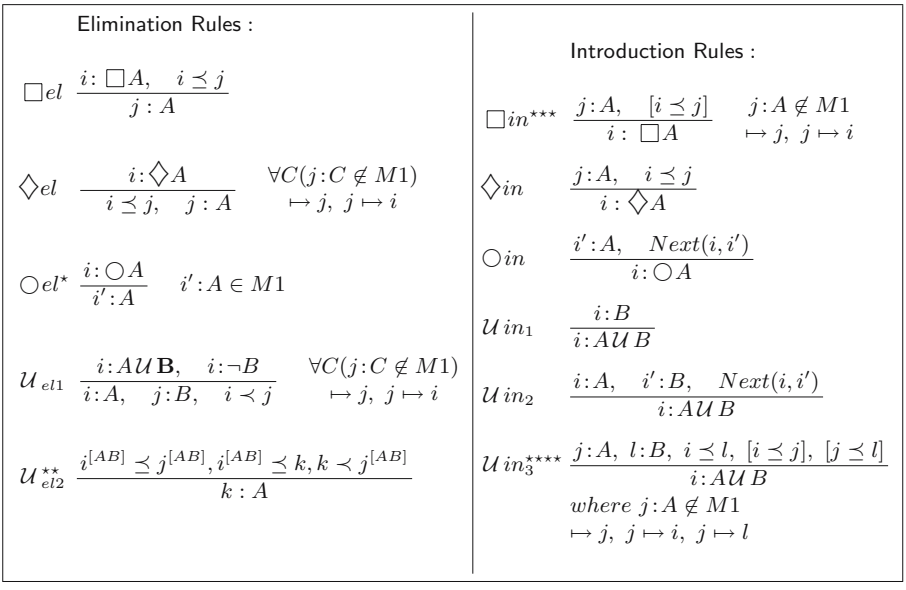


Fig. 3. Temporal ND-rules

The condition $j:A \notin M1$ in the rules $\Box_{in}$ and $\mathcal{U}_{in_3}$ means that $j:A$ is not marked by $M1$.

$\star$ In $\bigcirc_{el}$ the conclusion $i':A$ is marked by M_1.

$\star\star$ In $\mathcal{U}_{el2}$ the expression $i^{[AB]}$ is used with the following meaning: a variable i in the proof can be marked with $[AB]$ if it has been introduced in the proof as a result of the application of the rule $\mathcal{U}_{el1}$ to $i:A\,\mathcal{U}\,B$.

$\star\star\star$ In $\Box_{in}$ formula $i \preceq j$ must be the most recent assumption, applying the rule on the step n of the proof, we discard $i \preceq j$ and all formulae until the step n.

$\star\star\star\star$ Applying the rule $\mathcal{U}_{in3}$ on the step n of the proof, we discard that assumption, $i \preceq j$ or $j \preceq l$, which occurs earlier in the proof and all formulae until the step n.

In addition to these we also require the following Induction Rule:

$$\textit{Induction} \qquad \frac{i:A \quad [i \preceq j] \qquad j:A \Rightarrow \bigcirc A}{i:\Box A}$$

where

- $j:A \notin M1$ and $\mapsto j,\ j \mapsto i$.
- $i \preceq j$ must be the most recent assumption, applying the rule on the step n of the proof, we discard $i \preceq j$ and all formulae until the step n.

We also need the following obvious rules.

$$reflexivity\ \frac{}{i \preceq i} \qquad transitivity\ \frac{i \preceq j,\ j \preceq k}{i \preceq k} \qquad \bigcirc\ seriality\ \frac{}{Next(i, i')}$$

$$\bigcirc / \preceq\ \frac{Next(i, i')}{i \preceq i'} \qquad \prec / \preceq\ \frac{i \prec j}{i \preceq j}$$

Definition 8 (PLTL$_{ND}$ proof). *An ND proof of a PLTL formula B is a finite sequence of PLTL$_{ND}$ formulae $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ which satisfies the following conditions:*

- *every $\mathcal{A}_i$ $(1 \leq i \leq n)$ is either an assumption, in which case it should have been discarded, or the conclusion of one of the ND rules, applied to some foregoing formulae,*
- *the last formula, $\mathcal{A}_n$, is $x : B$, for some label x,*
- *no variable - world label is flagged twice or relatively binds itself.*

When B has a PLTL$_{ND}$ proof we will abbreviate it as $\vdash_{ND} B$.

Now we give an example of the PLTL$_{ND}$ proof establishing that the following formula is a theorem.

$$\Box(p \Rightarrow \bigcirc p) \Rightarrow (p \Rightarrow \Box p) \tag{1}$$

The proof commences by the assumption that the left hand side of the implication of (1), $\Box(p \Rightarrow \bigcirc p)$, is satisfied in some arbitrary world corresponding to x.

1. $x : \Box(p \Rightarrow \bigcirc p)$	*assumption*
2. $x : p$	*assumption*
3. $x \preceq y$	*assumption*
4. $y : p \Rightarrow \bigcirc p$	$\Box_{el}, 1, 3$
5. $x : \Box p$	*Induction* $2, 3, 4, \mapsto y, y \mapsto x, [3-4]$
6. $x : p \Rightarrow \Box p$	$\Rightarrow_{in}\ 5, [2-5]$
7. $x : \Box(p \Rightarrow \bigcirc p) \Rightarrow (p \Rightarrow \Box p)$	$\Rightarrow_{in}, 6, [1-6]$

At steps 2 and 3 we introduce two more assumptions which allows us at step 4 to apply the $\Box_{el}$ rule to formulae 1 and 3. The next step, 5, is the application of the induction rule to formulae 2-4. Recall that applying the induction rule we make the variable y flagged, which, in turn, makes x relatively bound. Also, at this step we discard formulae, 3-4, starting from the most recent assumption, 3. At the next step, 6, we apply the $\Rightarrow_{in}$ rule to 5 discarding formulae 2-5, and the application of the same rule at step 7 gives us the desired proof. At this last step, we discard formulae, this time from the most recent assumption, 1. Since the last formula has the form $x : \Box(p \Rightarrow \bigcirc p) \Rightarrow (p \Rightarrow \Box p)$, and the set of non-discarded assumptions is empty, we have obtained the $PLTL_{ND}$ proof for 1. In the next section we give some more examples of $PLTL_{ND}$ proofs.

4 Correctness

In this section we will establish meta-theoretical properties of the PLTL$_{ND}$ system defined above. Namely, we will show that PLTL$_{ND}$ is sound (§4.1) and complete (§4.2).

4.1 Soundness

Lemma 2. *Let $\Gamma = \{C_1, C_2, \ldots, C_k\}$ be a set of PLTL formulae such that $\hat{\Gamma} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, where each $\mathcal{C}_i$ $(1 \leq i \leq k)$ is $j : C_i$, for some label j, is a set of non-discarded assumptions which are contained in the PLTL$_{ND}$ proof for a PLTL formula B, at some step, m. Let Λ be a set of PLTL$_{ND}$ formulae in the proof at step m such that for any $\mathcal{D}$, $\mathcal{D} \in \Lambda$ if it is obtained by an application of some ND rule, and let Δ be a conclusion of a PLTL$_{ND}$ rule which is applied at step $m+1$. Let $\hat{\Gamma}^\star$ consist of all assumptions from $\hat{\Gamma}$ that have not been discarded by the application of this rule, the same for a set $\Lambda^\star$. Then if $\hat{\Gamma}^\star$ is realisable in a model σ then $\Lambda^\star \cup \Delta$ is also realisable in σ.*

Proof. We prove this lemma by induction on the number of PLTL$_{ND}$ rules applied in the proof. Thus, assuming that lemma is correct for the number, n, of the PLTL$_{ND}$ rules, we must show that it is also correct for the $n+1$-th rule.

The proof is quite obvious for the rules for Booleans. We only show the most interesting case where the rule of $\neg_{in}$ is applied.

Case $\neg_{in}$. Let $x : A$ be an element of $\hat{\Gamma}$ which is the most recent non-discarded assumption in the proof. An application of the rule $\neg_{in}$ at step $m+1$ gives a PLTL$_{ND}$ formula $x : \neg A$ as a conclusion. This means that at some earlier steps of the proof we have $y : C$ and $y : \neg C$. Here we should consider several subcases that depend on the set to which these contradictory PLTL$_{ND}$ formulae belong. We now prove the lemma for some of these cases. *Subcase 1.* Assume that both $y : C$ and $y : \neg C$ are in the set $\hat{\Gamma}^\star$ but nor $y : C$ neither $y : \neg C$ coincides with $x : A$. Then the statement that the realisability of $\hat{\Gamma}^\star$ implies the realisability of $\Lambda \cup \{x : \neg A\}$ is true simply because $\hat{\Gamma}^\star$ is not realisable. *Subcase 2.* Assume that both $y : C$ and $y : \neg C$ are in the set Λ. Then if the set $\hat{\Gamma}$ realisable, the set Λ should be realisable as well. But, as assumed, it is not. So, $\hat{\Gamma}$ also can not be realisable. Note that $\hat{\Gamma} = \hat{\Gamma}^\star \cup \{x : A\}$. It should be clear that if $\hat{\Gamma}^\star$ is realisable then also $\{x : \neg A\}$ is. If we think of the set $\hat{\Gamma}$ as an initial part of the proof, then the set $\Lambda^\star$ is empty after the deletion of the corresponding steps of proof. In this case we are done.

Cases with the rules for temporal operators that do not require restrictions on labels can be shown straightforwardly from the semantics. Let us consider cases with the rules that require restrictions, for example, the rule $\Diamond_{el}$.

Case $\Diamond_{el}$. Let $x : \Diamond A \in \Lambda$. We have to show realisability of $\Lambda^\star \cup \{j : A\}$ provided that realisability of $\hat{\Gamma}^\star$ holds. Actually $\hat{\Gamma}^\star = \hat{\Gamma}$ and $\Lambda^\star = \Lambda$ in this case. By induction hypothesis we know that realisability of Γ implies realisability of Λ. Then for a mapping $f(x) = s_i$, we have $\langle \sigma, s_i \rangle \models \Diamond A$. From the latter, according to the semantics, it follows that there exists a state s_k, $(i \leq k)$,

such that $\langle \sigma, s_k \rangle \models A$. Now we can define a mapping f', the extension of f, as $f' = f \cup \{(j, s_k)\}$. This mapping is correct because the variable j was not used in the proof before, otherwise it should be flagged twice. So, the mapping f is not defined for j. We can see that $\langle \sigma, f(j) \rangle \models A$ and the pair (i, j) satisfies the criteria of Definition 6. Therefore, $\Lambda \cup \{j : A\}$ is realisable.
(End)

Theorem 1 (PLTL$_{ND}$ Soundness). *Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ be a PLTL$_{ND}$ proof of PLTL formula B and let $\Gamma = \{C_1, C_2, \ldots, C_n\}$ be a set of PLTL formulae such that $\hat{\Gamma} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$, where each $\mathcal{C}_i$ $(1 \leq i \leq n)$ is $j{:}C_i$, for some label j, is a set of discarded assumptions which occur in the proof. Then $\models_{ND} B$, i.e. B is a valid formula.*

Proof. Consider the proof $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ for some PLTL formula B. According to Definition 8, $\mathcal{A}_k$ has the form $x : B$, for some label x. In general, $x : B$ belongs to some set, Λ, of non-discarded PLTL$_{ND}$ formulae in the proof. By Lemma 2 we can conclude that realisability of $\hat{\Gamma}$ implies realisability of Λ. But $\hat{\Gamma}$ is empty and, therefore, is realisable in any model and for any function f by Definition 6. So Λ is also realisable in any model and for any function f. That is, any formula that belongs to Λ is valid. In particular $x : B$ is valid. *(End)*

4.2 Completeness

We will prove the completeness of PLTL$_{ND}$ by showing that every theorem of the following axiomatics for PLTL [7, 4] is a theorem of PLTL$_{ND}$.

Axioms for PLTL (schemes).

$A1$. Schemes for classical propositional logic
$A2$. $\Box(A \Rightarrow B) \Rightarrow (\Box A \Rightarrow \Box B)$
$A3$. $\bigcirc \neg A \Rightarrow \neg \bigcirc A$
$A4$. $\neg \bigcirc A \Rightarrow \bigcirc \neg A$
$A5$. $\bigcirc(A \Rightarrow B) \Rightarrow (\bigcirc A \Rightarrow \bigcirc B)$
$A6$. $\Box A \Rightarrow A \wedge \bigcirc \Box A$
$A7$. $\Box(A \Rightarrow \bigcirc A) \Rightarrow (A \Rightarrow \Box A)$
$A8$. $(A \,\mathcal{U}\, B) \Rightarrow \Diamond B$
$A9$. $(A \,\mathcal{U}\, B) \Rightarrow (B \vee (A \wedge \bigcirc(A \,\mathcal{U}\, B)))$
$A10$. $(B \vee (A \wedge \bigcirc(A \,\mathcal{U}\, B))) \Rightarrow (A \,\mathcal{U}\, B)$

Rules:

$$\frac{\vdash A}{\vdash \Box A} \qquad \frac{\vdash A, \quad \vdash A \Rightarrow B}{\vdash B}$$

To prove the completeness of PLTL$_{ND}$ we first show that every instance of the scheme of the above axiomatics is a theorem of PLTL$_{ND}$, and, secondly, that given that the assumptions of the rules of the axiomatics have a PLTL$_{ND}$ proof then so do their conclusions.

Lemma 3. *Every instance of the scheme of the PLTL axiomatics is a theorem of* $PLTL_{ND}$.

Proof. Since PLTL_{ND} extends the natural deduction system for classical propositional logic, all classical schemes are provable in PLTL_{ND} by a simple modification of classical proofs introducing a world label, say x, for any formula of a classical proof [2].

Now we will present proofs for some of the PLTL schemes from Axioms 1-10 above. Note that, demonstrating how the system works, in §3, we have established the $PLTL_{ND}$ proof for the formula (1) which is an instance of Axiom 7. Below we will provide proofs for the instances of Axioms 3 and 9.

Proof for an instance of Axiom 3. $\bigcirc\neg p \Rightarrow \neg\bigcirc p$

1. $x\colon \bigcirc\neg p$	*assumption*
2. $x\colon \bigcirc p$	*assumption*
3. $x'\colon \neg p$	$1, \bigcirc_{el},\ M1(x'\colon\neg p)$
4. $x'\colon p$	$2, \bigcirc_{el},\ M1(x'\colon p)$
5. $x\colon \neg\bigcirc p$	$3, 4, \neg_{in}, [2-4]$
6. $x\colon \bigcirc\neg p \Rightarrow \neg\bigcirc p$	$6, \Rightarrow_{in}, [1-5]$

Proof for an instance of Axiom 9. $(p\,\mathcal{U}\,q) \Rightarrow (q \vee (p \wedge \bigcirc(p\,\mathcal{U}\,q)))$

1. $x : p\,\mathcal{U}\,q$	*assumption*
2. $x : \neg(q \vee (p \wedge \bigcirc(p\,\mathcal{U}\,q)))$	*assumption*
3. $x : \neg q \wedge (\neg p \vee \neg\bigcirc(p\,\mathcal{U}\,q)$	*classical*, 2
4. $x : \neg q$	$\wedge_{el}$ 3
5. $x : \neg p \vee \neg\bigcirc(p\,\mathcal{U}\,q)$	$\wedge_{el}$ 3
6. $x \preceq y$	$\mathcal{U}_{el_1}$ $1, 4, \mapsto y, y \mapsto x$
7. $x : p$	$\mathcal{U}_{el_1}$ $1, 4$
8. $y : q$	$\mathcal{U}_{el_1}$ $1, 4$
9. $y \preceq y$	*reflexivity of* $\preceq$
10. $y : p\,\mathcal{U}\,q$	$\mathcal{U}_{in1}, 8$
11. $x : \bigcirc(p\,\mathcal{U}\,q)$	*subproof*
12. $x : \neg\bigcirc(p\,\mathcal{U}\,q)$	$\vee_{el}, 5, 7$
13. $x : \neg\neg(q \vee (p \wedge \bigcirc(p\,\mathcal{U}\,q)))$	$\neg_{in}, 11, 12, [2-12]$
14. $x : q \vee (p \wedge \bigcirc(p\,\mathcal{U}\,q))$	$\neg_{el}, 13$
15. $x : (p\,\mathcal{U}\,q) \Rightarrow (q \vee (p \wedge \bigcirc(p\,\mathcal{U}\,q)))$	$\Rightarrow_{in}, 14, [1-14]$

It is easy to establish the following proposition.

Proposition 1. *Let* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ *be a* $PLTL_{ND}$ *proof of a PLTL formula* B. *Let* B' *be obtained from* B *by substituting a subformula* C *of* B *by* C'. *Then* $\mathcal{A}'_1, \mathcal{A}'_2, \ldots, \mathcal{A}'_n$, *where any occurrence of* C *is substituted by* C' *is a* $PLTL_{ND}$ *proof of* B'.

Hence by Proposition 1 and the proofs of the instances of PLTL axioms we obtain the proof for Lemma 3.
(End)

Lemma 4. *If A has a $PLTL_{ND}$ proof then $\Box A$ also has a $PLTL_{ND}$ proof.*

Proof. Consider some arbitrarily chosen theorem of PLTL_{ND}, A, and let x and y be the world variables that do not occur in this proof.

Now we start a new proof commencing it with the assumption that $\neg \Box A$ (below, to make the proof more transparent, we will scorn the rigorous presentation of $PLTL_{ND}$ proof, writing metaformulae instead of the PLTL formulae which can be justified based upon Proposition 1):

1. $x : \neg \Box A$	$assumption$
2. $x : \Diamond \neg A$	$1, \neg \Box\ transformation$
3. $x \preceq y$	$2, \Diamond_{el}, \mapsto y, y \mapsto x$
4. $y : \neg A$	$2, \Diamond_{el}$

At this stage we are coming back to the proof of A noticing that at the last step of this proof we have a formula $z : A$ (recall that $z \neq x \neq y$). In this proof of A we do the following: change every occurrence of z to y. Obviously, we still have a proof for A. Take this newly generated proof (say it has n steps) and write it continuing steps 1-4 of the proof above. Thus, we obtain

1. $x : \neg \Box A$	$assumption$
2. $x : \Diamond \neg A$	$1, \neg \Box$
3. $x \preceq y$	$2, \Diamond_{el}, \mapsto y, y \mapsto x$
4. $y : \neg A$	$2, \Diamond_{el}$
5.	$(first\ formula\ of\ the\ proof\ for\ A)$
...	
...	
...	
$n+5$. $y : A$	$(last\ formula\ of\ the\ proof\ for\ A)$

Hence we have a contradiction, steps 4 and $n+5$, which enables us to apply the $\neg_{in}$ rule to obtain $\neg\neg \Box A$ at step n+6 and discard formulae from 1 to n, and to derive $\Box A$ at the next step.

1. $x : \neg \Box A$	$assumption$
2. $x : \Diamond \neg A$	$1, \neg \Box$
3. $x \preceq y$	$2, \Diamond_{el}, \mapsto y, y \mapsto x$
4. $y : \neg A$	$2, \Diamond_{el}$
5.	$(first\ formula\ of\ the\ proof\ for\ A)$
...	
...	
...	
$n+5$. $y : A$	$(last\ formula\ of\ the\ proof\ for\ A)$
$n+6$. $\neg\neg \Box A$	$4, n+5, \neg_{in}, [1-(n+5)]$
$n+7$. $\Box A$	$\neg_{el}, n+6$

Since steps from 5 to $n+5$ satisfy the conditions of the PLTL_{ND} proof (for A) and steps 1-4, $n+6$ satisfy these conditions in the proof above, the whole sequence of formulae from 1 to n+7 represents a proof for A. *(End)*

Lemma 5. *If $A \Rightarrow B$ and A have $PLTL_{ND}$ proofs then B also has an $PLTL_{ND}$ proof.*

Proof. Let the proofs for $A \Rightarrow B$ and A have n and m steps respectively. Since both are PLTL_{ND} proofs we can rewrite these proofs such that they would have completely different sets of the world labels. Let the last formula of the proof for $A \Rightarrow B$ be $x : A \Rightarrow B$. We commence constructing the PLTL_{ND} proof for B as follows:

$$
\begin{array}{ll}
1. & (first\ formula\ of\ the\ proof\ for\ A \Rightarrow B) \\
\ldots & \\
\ldots & \\
\ldots & \\
n.\ x : A \Rightarrow B & (last\ formula\ of\ the\ proof\ for\ A \Rightarrow B)
\end{array}
$$

Now we can change to x the label that occurs at the last step of the proof for A and continue the construction of the proof for B as follows.

$$
\begin{array}{ll}
n+1. & (first\ formula\ of\ the\ proof\ for\ A) \\
\ldots & \\
\ldots & \\
\ldots & \\
n+m.\ x : A & (last\ formula\ of\ the\ proof\ for\ A) \\
n+m+1.\ x : B & n, n+m, \Rightarrow_{el}
\end{array}
$$

It is easy to establish that this proof, by its construction, satisfies the criteria for the proof. *(End)*

Now we are ready to prove the completeness of PLTL_{ND}.

Theorem 2 (PLTL_{ND} Completeness). *For any $PLTL_{ND}$ formula, A, if $\models_{ND} A$ then there exists a $PLTL_{ND}$ proof of A.*

Proof. Consider an arbitrarily chosen theorem, A, of PLTL. By induction on n, the length of the axiomatic proof for A, we now show that A also has a PLTL_{ND} proof.

Base Case. $n = 1$. In this case A is one of the schemes of the PLTL axiomatics, and thus, the base case follows from Lemma 3.

Induction step. If Theorem 2 is correct for the proof of the length m, $(1 \leq m \leq n)$ then it is correct for the proof of the length $m + 1$.

Here the formula at the step $m + 1$ is either an axiom or is obtained from some previous formulae either by generalisation or the modus ponens rules. The proof for these cases follows from Lemma 4 and Lemma 5 respectively.

Therefore, given that A has an axiomatic proof it also has a PLTL_{ND} proof. *(End)*

5 Discussion

We have presented a natural deduction system for propositional linear time temporal logic and established its correctness. To the best of our knowledge,

there is only one other ND construction, in [11] for the full PLTL which is based upon the developments in [15]. In Marchignoli's construction, many rules such as $\vee_{el}, \Rightarrow_{in}, \neg_{in}$ and rules for $\mathcal{U}$, are formulated in so called 'indirect' fashion, i.e. they allow us to transform some given proofs. From our point of view, these rules are much more complex than the rules in our system, and thus would be more difficult for developing a proof-searching procedure.

Although a proof-searching technique for this novel construction is still an open, and far from being trivial, problem, we expect to incorporate many of the methods previously defined for classical propositional and first-order logics.

The study of complexity of the method for both classical and temporal framework, in turn, is another component of future research as well as the extension of the approach to capture the branching-time framework.

References

1. L. Bachmair and H. Ganzinger. A theory of resolution. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter 2. Elsevier, 2001.
2. A. Bolotov, V. Bocharov, A. Gorchakov, V. Makarov, and V. Shangin. *Let Computer Prove It.* Logic and Computer. Nauka, Moscow, 2004. (In Russian).
3. A. Bolotov, V. Bocharov, A. Gorchakov, and V. Shangin. Automated first order natural deduction. In *Proceedings of IICAI*, pages 1292–1311, 2005.
4. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic (TOCL)*, 1(2):12–56, 2001.
5. F. Fitch. *Symbolic Logic.* NY: Roland Press, 1952.
6. M. Fitting. *First-Order Logic and Automated Theorem-Proving.* Springer-Verlag, Berlin, 1996.
7. D. Gabbay, A. Phueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, 1980.
8. G. Gentzen. Investigation into logical deduction. In *The Collected Papers of Gerhard Gentzen*, pages 68–131. Amsterdam: North-Holland, 1969.
9. S. Jaskowski. On the rules of suppositions in formal logic. In *Polish Logic 1920-1939*, pages 232–258. Oxford University Press, 1967.
10. V. Makarov. Automatic theorem-proving in intuitionistic propositional logic. In *Modern Logic: Theory, History and Applications. Proceedings of the 5th Russian Conference*, StPetersburg, 1998. (In Russian).
11. D. Marchignoli. *Natural Deduction Systems for Temporal Logic.* PhD thesis, Department of Informatics, Unviersity of Pisa, 2002.
12. F. Pfenning. Logical frameworks. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, chapter XXI, pages 1063–1147. Elsevier, 2001.
13. J. Polakow and F. Pfenning. Natural deduction for intuitionistic non-commutative linear logic. In *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA'99), Springer-Verlag LNCS, 1581*, L'Aquila, Italy, April 1999.
14. W. Quine. On natural deduction. *Journal of Symbolic Logic*, 15:93–102, 1950.
15. A. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic.* PhD thesis, College of Science and Engineering, School of Informatics, University of Edinburgh, 1994.
16. M. Wooldridge. *Reasoning about Rational Agents.* MIT Press, 2000.

A STIT-Extension of ATL

Jan Broersen[1], Andreas Herzig[2], and Nicolas Troquard[2]

[1] Department of Information and Computing Sciences, Utrecht University
[2] Institut de Recherche en Informatique de Toulouse

Abstract. A problem in many formalisms for reasoning about multi-agent systems, like ATL or PDL, is the inability to express that a certain complex action (as in PDL), choice or strategy (as in ATL) is performed by an agent. However, in so called STIT-logics, this is exactly the main operator: seeing to it that a certain condition is achieved. Here we present an extension of ATL, introducing ideas from STIT-theory, that can express that a group of agents A perform a certain strategy. As a demonstration of the applicability of the formalism, we show how it sheds new light on the problem of modelling 'uniform strategies' in epistemic versions of ATL.

1 Introduction

The present paper introduces a so called 'strategic STIT-operator' in the framework of ATL [1, 2]. For those unfamiliar with the STIT-framework: the characters 'STIT' are an acronym for 'seeing to it that'. STIT logics [3, 4, 5] originate in philosophy, and can be described as endogenous logics of agency, that is, logics of agentive action where actions are not made explicit in the object language. To be more precise, expressions $[A \; stit : \varphi]$ of STIT-logic stand for 'agents A see to it that φ', where φ is a (possibly) temporal formula. The main virtue of STIT logics is that, unlike most (if not all) other logical formalisms, they can express that a choice or action is actually performed / taken / executed by an agent. The aim of the present paper is thus to add this type of expressivity to the ATL-framework. But not only do we want to add the standard STIT expressivity, we intend to define a *strategic* version of STIT as an addition to ATL. This enables us to express what it means that a group of agents performs / takes / executes a certain *strategy*. ATL itself can only talk about the *existence* or 'availability' of certain strategies, not that they are actually being performed.

We consider the definition of a semantics for a strategic version of STIT within the ATL-framework as the main contribution of this paper. Indeed, within the community working on the STIT framework of Belnap [3, 4, 5] and Horty [6], it is perceived as an open problem how to define a suitable notion of *strategic* STIT. As a corollary the semantics shows how we can make the implicit quantifications in the semantics of the ATL operators explicit in the object language: the two central ATL operators will each be decomposed into a strategy quantifier and a strategic STIT operator. To demonstrate the applicability of the formalism, in section 4 we will discuss an extension with epistemic notions, and discuss the problem of 'uniform strategies'. This has also been the subject of [7], but section 4 adds in some new insights. Also the present paper differs

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 69–81, 2006.

from [7] in that we introduce epistemic notions in a STIT-extension of the ATL framework, whereas [7] introduces epistemic notions in the STIT framework. Furthermore, here we deal with general strategies, where [7] only deals with one-step strategies.

Central to our approach will be to evaluate ATL-STIT formulas with respect to strategy / state pairs. Tinkering with the units of evaluation has been suggested before in the literature on ATL and STIT. Horty [6] indeed already suggests it to define a notion of strategic STIT. Although Horty suggests two possible approaches, he circumvents the problem of actually giving definitions for the strategic STIT by syntactically disallowing this operator to occur without an attached operator quantifying over histories. Müller [8] suggests evaluation with respect to strategies to deal with the notion of continuous action within the STIT framework, and Jamroga and gnotes [9] suggest to evaluate with respect to sets of worlds to solve the problem of uniform strategies in epistemic ATL. We will discuss these related approaches in more detail in section 5.

In earlier work [10] we investigated the similarities between the ATL and STIT frameworks. The present paper is a demonstration of our opinion that there can be a fruitful exchange of techniques and ideas between both frameworks. The idea for investigating strategic versions of STIT operators originates from Belnap (Horty [6] mentions an unpublished manuscript) and Horty. Here we show how we can successfully define this concept in the ATL setting. An ensuing next step would then be to transfer these ideas back to the STIT framework.

2 The Meaning of 'Agents a Performing a Strategy'

First we need to explain that we think that 'strategy' seems not the best term for the moment-to-action-functions defined in this paper. We feel it would be more in line with established general AI terminology to call them 'tactics' or 'conditional plans'. Strategies are usually associated with choices for more abstract (sub-)goals, while tactics are indeed more concrete (conditional) plans for reaching these goals. Yet, to adhere to established terminology in both STIT theory and ATL, we will also refer to the conditional plans as 'strategies'.

An important conceptual first question is then what it exactly means to say that 'a group is performing a strategy'. Is whether or not 'a group is performing a strategy' actually a sensible concept amenable to logical truth? For instance, in what sense can it be true that 'agent j, who is still at home, presently performs the strategy of going to the railway station'? A strong intuition is that performing an action / choice is a local matter concerning the present. The problem then seems to be that at any future point j may reconsider his strategy. Half way to the railway station he may decide to go to the cafe and have a beer instead. So how could we ever say that an agent is performing a certain strategy presently if at any future point he may decide to deviate from it? Is it not that all we can say is that an agent is *committed* to a certain strategy, thereby leaving room for the possibility that an agent reconsiders his strategy?

Our answer is that the notion of commitment to a strategy actually presupposes a notion of performing a strategy. How can we say that an agent is committed to going to the railway station (which, one way or the other, expresses a certain preference for some strategies over others) if we cannot say what it means for the agent to actually

perform going to the railway station? The same holds for strategic contents of epistemic notions. For instance, if we want to say that we belief that agent A performs a certain strategy, than first we have to know what it means that A performs a strategy. So, if we do not accept 'agent A is performing a strategy for φ' as a meaningful proposition, we cannot accept 'agent A is *committed* to performing a strategy for φ' and 'agent B *beliefs* that agent A is performing a strategy for φ' as meaningful propositions either. The conclusion then is that although it is maybe strange to think about the truth of propositions talking about performance of strategies as such, it is not at all strange to reason with these propositions within the scope of motivational and epistemic modalities. Human agents do this all the time. Presently we are *committed* to performing the strategy to finish writing this paper (in time), which presupposes that we know what it means to actually perform this strategy. Also, we *believe* that president Bush is performing a strategy of world destruction, which presupposes that it is clear what it means to be performing such a strategy. So the notion of performing a strategy is not inherently problematic. We reason with the notion all the time, and the present proposal defines a semantics for it.

3 ATL-STIT

We present a STIT extension of ATL ([1, 2]) using a non-standard, but concise and intuitive syntax and semantics.

3.1 Core Syntax, Abbreviations and Intended Meanings

Definition 1. *Well-formed formulas of the temporal language $\mathcal{L}_{ATL\text{-}STIT}$ are defined by:*

$$\varphi, \psi, \ldots := p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Diamond_A \varphi \mid \Box_A \varphi \mid [A]\eta \mid \langle A\rangle\eta$$
$$\eta, \theta, \ldots := \phi U^{ee} \psi$$

where $\varphi, \psi, \ldots$ represent arbitrary well-formed formulas, $\eta, \theta, \ldots$ represent temporal path formulas, the p are elements from an infinite set of propositional symbols $\mathcal{P}$, and A is a subset of a finite set of agent names E (we define $\overline{A} \equiv_{def} E \setminus A$). We use the superscript 'ee' for the until operator to denote that this is the version of 'the until' where φ is not required to hold for the present, nor for the point where ψ, i.e., the present and the point where ϕ are *both* excluded. The operators $\Box_A\varphi$ and $\Diamond_A\varphi$ are universal and existential quantifiers over strategies, respectively. The STIT operators $[A]\eta$ and $\langle A\rangle\eta$ are read as 'agents A strategically see to it that η' and 'agents A strategically allow the possibility for η', respectively. The combined operator $\Diamond_A[A]\eta$ is read as 'Agents A have a strategy that ensures η' (this is the 'classical' ATL operator, usually written as $\langle\langle A\rangle\rangle\eta$), and the dual $\Box_A\langle A\rangle\eta$ is read as 'A have no strategy to avoid that possibly η'. A more precise explanation of the intended semantics is as follows:

$\Diamond_A\varphi$: there is a strategy (for the set of agents A, from the current state) such that φ
$\Box_A\varphi$: for all strategies (of the set of agents A, from the current state) φ

The intended interpretations for the new *strategic STIT operators* are:

$[A](\varphi U^{ee}\psi)$: agents A perform a strategy that, whatever strategy is taken by agents $\overline{A}$, ensures that eventually, at some point m, the condition ψ will hold, while φ holds from the next moment until the moment before m

$\langle A\rangle(\varphi U^{ee}\psi)$: Agents A perform a strategy giving agents $\overline{A}$ the possibility to perform a strategy such that eventually, at some point m, the condition ψ will hold, while φ holds from the next moment until the moment before m

We use standard propositional abbreviations, and also define the following operators as abbreviations.

Definition 2.

$[A]X\varphi \equiv_{def} [A](\bot U^{ee}\varphi)$
$[A]F\varphi \equiv_{def} [A](\top U^{ee}\varphi)$
$[A]G\varphi \equiv_{def} \neg\langle A\rangle F\neg\varphi$
$[A](\varphi U^{e}\psi) \equiv_{def} [A](\varphi U^{ee}(\varphi \wedge \psi))$
$[A](\varphi U^{e}_{w}\psi) \equiv_{def} \neg\langle A\rangle(\neg\psi U^{e}\neg\varphi)$

$\langle A\rangle X\varphi \equiv_{def} \langle A\rangle(\bot U^{ee}\varphi)$
$\langle A\rangle F\varphi \equiv_{def} \langle A\rangle(\top U^{ee}\varphi)$
$\langle A\rangle G\varphi \equiv_{def} \neg[A]F\neg\varphi$
$\langle A\rangle(\varphi U^{e}\psi) \equiv_{def} \langle A\rangle(\varphi U^{ee}(\varphi \wedge \psi))$
$\langle A\rangle(\varphi U^{e}_{w}\psi) \equiv_{def} \neg[A](\neg\psi U^{e}\neg\varphi)$

The informal meanings of the formulas are as follows (the informal meanings in combination with the $\langle A\rangle$ operator follow trivially):

$[A]X\varphi$: agents A strategically ensure that at any next moment φ will hold
$[A]F\varphi$: agents A strategically ensure that eventually φ will hold
$[A]G\varphi$: agents A strategically ensure that φ holds henceforth
$[A](\varphi U^{e}\psi)$: agents A strategically ensure that, eventually, at some point the condition ψ will hold, while φ holds from the next moment until then
$[A](\varphi U^{e}_{w}\psi)$: agents A strategically ensure that, if eventually ψ will hold, then φ holds from the next moment until then, or forever otherwise

Note that all STIT formulas refer strictly to the future. Also, for instance, a formula like $[A]G\varphi$ saying that φ holds henceforth, does not imply that φ holds now.

Alternatively, we could have taken $[A]\varphi U^{e}\psi$ and $[A]G\varphi$ as the basic operators of our langauge, which would enable us to define $\langle A\rangle\varphi U^{e}\psi$ in terms of them. A similar choice appears for the definition of related logics like ATL and CTL. However, we prefer the symmetry of the present setup, and we think the semantics of the new weak STIT operator $\langle A\rangle\varphi U^{ee}\psi$ deserves a definition in terms of truth conditions.

3.2 Model Theoretic Semantics

We use alternating transition systems (ATSs) for the semantics. Goranko and Jamroga [11] argue that to define the semantics of ATL, multi-player game models (MGMs) provide more intuitive semantic representations in many examples. However, ATSs are closer to the models used for STIT logics. And actually we do not fully agree that ATSs are better than MGMs as semantic structures for ATL. We believe it is better not

to decorate semantic structures with superfluous information. For instance, in MGMs the actions have explicit names. However ATL is an endogenous temporal formalism where the strategies (which can be seen as conditional plans) are not explicit in the object language. So, ATL is not, so to say, 'aware' of the actions names. We will come back to this point in section 4.2.

The assumption behind ATSs is that agents have choices, such that the non-determinism of each choice is *only* due to the choices other agents have at the same moment. Thus, the simultaneous choice of al agents together, always brings the system to a unique follow-up state. In other words, if an agent would know what the choices of other agents would be, given his own choice, he would know exactly in which state he arrives.

Definition 3. *An ATS* $\mathcal{M} = (S, C, \pi)$*, consists of a non-empty set* S *of states, a total function* $C : E \times S \mapsto 2^{2^S}$ *yielding for each agent and each state a set of choices (informally: 'actions') under the condition that the intersection of each combination of choices for separate agents gives a unique next system state (i.e., for each* s*, the function* $RX(s) = \{\bigcap_{a \in E} Ch_a \mid Ch_a \in C(a, s)\}$ *yields a non-empty set of singleton sets representing the possible follow-up states of* s*), and, finally, an interpretation function* π *for propositional atoms.*

Note that from the condition on the function C it follows that the choices for each individual agent at a certain moment in time are a partitioning of the set of all choices possible for the total system of agents, as embodied by the relation $\mathcal{R}^{sys} = \{(s, s') \mid s \in S \text{ and } \{s'\} \in RX(s)\}$. And, also note that this latter condition does not entail the former. That is, there can be partitions of the choices for the total system that do not correspond to the choices of some agent in the system. Now we are ready to define strategies relative to ATSs.

Definition 4. *Given an ATS, a strategy* α_a *for an agent* a*, is a function* $\alpha_a : S \mapsto 2^S$ *with* $\forall s \in S : \alpha_a(s) \in C(a, s)$*, assigning choices of the agent* a *to states of the ATS.*

In semantics for ATL, strategies are often defined as mappings $\alpha_a : S^+ \mapsto 2^S$, from finite *sequences* of states to choices in the final state of a sequence. However, to interpret ATL, this is not necessary, because ATL is not expressive enough to recognize by which sequence of previous states a certain state is reached (but ATL* is). More in particular, without affecting truth of any ATL formula, we can always transform an ATS into one where $\mathcal{R}^{sys}$ is tree-like. On tree structures it is clear right away that a mapping from states to choices in that state suffices, since any state can only be reached by the actions leading to it. We come back to this point in section 4.

Definition 5. *Strategy functions* α_a *for individual agents* a *are straightforwardly combined to system strategy functions* $\alpha_E : S \times E \mapsto 2^S$ *for the full set of agents* E*. Then* $\alpha_E(s, a)$ *yields the choice of agent* a *in state* s *determined by the system strategy* α_E*. However, central to our semantics will be* partial *strategy functions* $\alpha_A : S \times E \mapsto 2^S$*, where* $A \subseteq E$*. These functions are partial in the sense that no choices are defined for the agents* $\overline{A}$*. For* $B \subseteq A$ *we use the notation* $\alpha_A{\restriction}_B$ *to denote the partial strategy function that is the restriction of the partial strategy function* α_A *to the domain of agents* B *(note*

that $\alpha_A \restriction_A = \alpha_A$*). Furthermore, for* $A \cap B = \emptyset$*, we use* $\alpha_A | \beta_B$ *to denote the minimal joined partial strategy function build from* α_A *and* β_B *such that* $(\alpha_A | \beta_B) \restriction_A = \alpha_A$ *and* $(\alpha_A | \beta_B) \restriction_B = \beta_B$*.*

As said, if in a given state all agents in the system have fixed their choice, a unique next state is determined by the intersection of all choices. Analogously, if all agents in the system have fixed a strategy, from any given point, a unique infinite path into the future is determined by the intersection of all choices in the strategies. We use this in the next definition.

Definition 6. *Given a system strategy* α_E*, we define the follow up function* $F_{\alpha_E} : S \mapsto S$ *as the intersection of all choices for individual agents, that is,* $F_{\alpha_E}(s) = \bigcap_{a \in E} \alpha_E(s, a)$*. Then, by* $(F_{\alpha_E})^n(s)$ *we denote the unique state that results from state* s *by taking* n *steps of the system strategy* α_E

Now we are ready to define the formal semantics of the language $\mathcal{L}_{\text{ATL-STIT}}$. The essential new aspect of this semantics is that it evaluates formulas with respect to strategy / state pairs. For a given fixed ATS, the set of all possible strategies for any group of agents A is well defined. So technically there is no problem with evaluation against strategy / state pairs. The pairs of an ATS form a two-dimensional modal structure, with group strategies and (impersonal) moments constituting the two 'axis'. As is customary for multi-dimensional possible world structures, we have modal operators interpreted on individual dimensions only: the strategy quantification operators $\Diamond_A \varphi$ and $\Box_A \varphi$ are interpreted on the dimension of strategies, relative to a *fixed* moment, and the temporal STIT operators $[A]\phi U^{ee} \psi$ and $\langle A \rangle \phi U^{ee} \psi$ are interpreted on the moments, relative to a *fixed* strategy.

But then the question remains: why should we *want* to evaluate against strategy / state pairs? It is clear that we want to give semantics to the strategic STIT operators. Truth of such operators cannot be determined with respect to states or moments alone, since in general, at the same moment, agents have a choice between several strategies. If we really want to give meaning to an operator that enables us to express that it is *true* that an agent, or group of agents performs a strategy, we have to take the possible strategies as units of evaluation. Then, with group strategies as abstract possible worlds, through evaluation in such worlds we can determine whether or not it is true that a group of agents strategically see to something.

Definition 7. *Validity* $\mathcal{M}, \alpha_A, s \models \varphi$*, of an ATL-STIT-formula* φ *in a strategy / state pair* (α_A, s) *of an ATS* $\mathcal{M} = (S, C, \pi)$ *is defined as:*

$$
\begin{array}{ll}
\mathcal{M}, \alpha_A, s \models p & \Leftrightarrow s \in \pi(p) \\
\mathcal{M}, \alpha_A, s \models \neg\varphi & \Leftrightarrow \textit{not } \mathcal{M}, \alpha_A, s \models \varphi \\
\mathcal{M}, \alpha_A, s \models \varphi \wedge \psi & \Leftrightarrow \mathcal{M}, \alpha_A, s \models \varphi \textit{ and } \mathcal{M}, \alpha_A, s \models \psi \\
\mathcal{M}, \alpha_A, s \models \Diamond_B \varphi & \Leftrightarrow \exists \beta_B \textit{ such that } \mathcal{M}, \beta_B, s \models \varphi \\
\mathcal{M}, \alpha_A, s \models \Box_B \varphi & \Leftrightarrow \forall \beta_B \textit{ it holds that } \mathcal{M}, \beta_B, s \models \varphi \\
\mathcal{M}, \alpha_A, s \models [B]\phi U^{ee} \psi & \Leftrightarrow \forall \beta_{\overline{A \cap B}} \textit{ it holds that } \exists n > 0 \textit{ such that} \\
 & \quad (1)\ \mathcal{M}, \alpha_A, (F_{\alpha_E})^n(s) \models \psi \textit{ and} \\
 & \quad (2)\ \forall i \textit{ with } 0 < i < n \textit{ we have } \mathcal{M}, \alpha_A, (F_{\alpha_E})^i(s) \models \varphi \\
 & \quad \textit{where } \alpha_E \textit{ is defined as: } \alpha_E = \alpha_A \restriction_{A \cap B} | \beta_{\overline{A \cap B}}
\end{array}
$$

$\mathcal{M}, \alpha_A, s \models \langle B\rangle \phi U^{ee} \psi \Leftrightarrow \exists \beta_{\overline{A\cap B}}$ *and* $\exists n > 0$ *such that*
(1) $\mathcal{M}, \alpha_A, (F_{\alpha_E})^n(s) \models \psi$ *and*
(2) $\forall i$ *with* $0 < i < n$ *we have* $\mathcal{M}, \alpha_A, (F_{\alpha_E})^i(s) \models \varphi$
where $\alpha_E = \alpha_A{\restriction}_{A\cap B} \mid \beta_{\overline{A\cap B}}$

Validity on an ATS $\mathcal{M}$ is defined as validity in all strategy / state pairs of the ATS. If φ is valid on an ATS $\mathcal{M}$, we say that $\mathcal{M}$ is a model for φ. General validity of a formula φ is defined as validity on all possible ATSs. The logic ATL-STIT is the subset of all general validities of $\mathcal{L}_{ATL\text{-}STIT}$ over the class of ATSs.

Note that due to the constraints on ATSs, if an atomic proposition is evaluated true on a strategy / state pair, all strategy / state pairs with the same state, will also have to evaluate to true, because for atomic propositions assignment of truth values is independent of the strategy. In Horty and Belnap's STIT formalisms atomic propositions can have different valuations at the same moment, depending on what history they are. In our setting, only formulas referring strictly to the future can evaluate to different values for the same moment, depending on the strategy with respect to which they are evaluated. We might say that in Horty's formalisms choices may affect the present, while our choices may only affect the strict future (both frameworks assume it makes no sense to account for choices affecting the past).

The most important aspect of the above definition is the truth condition for the STIT operators. Note that we evaluate the STIT operator $[B]\eta$ for a group of agents B with respect to a strategy for another group A. The truth condition expresses exactly in what sense the group B may see to it that η in a strategy of group A, namely, exactly if η is guaranteed by the agents in the intersection of both groups. This exploits the intuition that if a subgroup of agents sees to it that η, all supergroups also see to it that η. Now we show that ATL is a fragment of the logic ATL-STIT.

Theorem 1. *The logic ATL is the fragment of the logic ATL-STIT determined by the definitions* $\langle\langle A\rangle\rangle\eta \equiv_{def} \Diamond_A[A]\eta$ *and* $[[A]]\eta \equiv_{def} \Box_A\langle A\rangle\eta$.

Proof. We show that for this fragment, the valuation of formulas becomes 'moment determinate', that is, for all strategy / state pairs with the same state (moment), they evaluate to the same truth value (see Horty [6] for further explanation of this terminology). First note that the truth condition for the combined ('fused', as Horty calls it) operator $\Diamond_A[A]\eta$, reduces to the following moment determinate truth condition.

$\mathcal{M}, \alpha_A, s \models \Diamond_A[A]\phi U^{ee}\psi \Leftrightarrow \exists \beta_A$ such that $\forall \gamma_{\overline{A}}$ it holds that $\exists n > 0$ such that
(1) $\mathcal{M}, \alpha_A, (F_{\beta_A \mid \gamma_{\overline{A}}})^n(s) \models \psi$ and
(2) $\forall i$ with $0 < i < n$ we have $\mathcal{M}, \alpha_A, (F_{\beta_A \mid \gamma_{\overline{A}}})^i(s) \models \varphi$

This truth condition is completely independent of the strategy α_A. For similar reasons the truth condition for the combined operator $\Box_A\langle A\rangle\eta$ is moment determinate. Now notice that also all other formulas of the sub-language determined by $\langle\langle A\rangle\rangle\eta \equiv_{def} \Diamond_A[A]\eta$ and $[[A]]\eta \equiv_{def} \Box_A\langle A\rangle\eta$ are moment determinate. This means the quantification over all strategy / state pairs in the definition of validity gives the same result when performed only with respect to all states (moments). It is not too difficult to see that we thus arrive at a concise, but correct semantics for ATL.

Proposition 1. *The logic of the operators $\Box_A\varphi$ is S5 for every set A.*

This is due to the fact that S5 is sound and complete for equivalence classes. The accessability relation for the modal operator $\Box_A$ is the relation connection alternative A strategies. For any given model the 'alternative relation' forms a fixed equivalence class. As a consequence we have validities such as

$$\models [A]\eta \rightarrow \Diamond_A[A]\eta$$

saying that if agents A strategically see to it that η, indeed they have the ability to do so, and

$$\models \Box_A\langle A\rangle\eta \rightarrow \langle A\rangle\eta$$

saying that if for all strategies it is the case that agents A may encounter η, they currently perform a strategy where they possibly encounter η. It also follows that nesting of operators $\Box_A$ and $\Diamond_A$ is not meaningful, since it is well-known that nested S5 formulas can be replaced by logically equivalent non-nested formulas.

Proposition 2. *The operators $\Box_A\varphi$ obey the interaction axioms:*

$$\models \Box_A\varphi \rightarrow \Box_B\Box_A\varphi$$

$$\models \Diamond_A\varphi \rightarrow \Box_B\Diamond_A\varphi$$

Below we list only a few more validities. Possible complete axiomatizations for the present logic are still under investigation.

Proposition 3. *Additionally, we have the following validities and non-validities.*

$$\models [A]\eta \rightarrow [B]\eta \text{ for } A \subseteq B$$
$$\models \langle A\rangle\eta \rightarrow \langle B\rangle\eta \text{ for } A \supseteq B$$
$$\models [A]X\varphi \wedge [B]X\psi \rightarrow [A \cup B]X(\varphi \wedge \psi)$$
$$\models \langle A \cup B\rangle X(\varphi \wedge \psi) \rightarrow \langle A\rangle X\varphi \vee \langle B\rangle X\psi$$

Note that for the third validity, we do not need the condition of sets A and B being disjoint, as in the axiomatizations of CL [12] and ATL.

4 Epistemic ATL-STIT

As a demonstration of the applicability of the formalism, we extend it with epistemic modalities. We interpret the epistemic modalities using epistemic indistinguishability relations over over strategy / state pairs. The resulting fine-grained epistemic structures enable us to shed new light on the problem of so called 'uniform strategies'.

4.1 Basic Definitions

First we extend the language of ATL-STIT with an operator $K_a\varphi$ for agent a knows φ, an operator $E_A\varphi$ for agents A all know that φ, an operator $D_A\varphi$ for agents A would know that φ if they would exchange all their knowledge, and an operator $C_A\varphi$ for agents A commonly know that φ.

Definition 8. *Well-formed formulas of the temporal language $\mathcal{L}_{E\text{-}ATL\text{-}STIT}$ are defined by:*

$$\varphi, \psi, \ldots := p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_a\varphi \mid E_A\varphi \mid D_A\varphi \mid C_A\varphi \mid \Diamond_A\varphi \mid \Box_A\varphi \mid [A]\eta \mid \langle A\rangle\eta$$
$$\eta, \theta, \ldots := \phi U^{ee} \psi$$

To accommodate epistemic reasoning, we want to define S5 indistinguishability relations over the units of evaluation, that is, strategy / state pairs. However, we have to be careful. As pointed out before, in for instance [13], adding epistemic indistinguishability relations to arbitrary ATSs leaves room for ambiguity: in particular, what is the epistemic status of an action leading from one state to another one that is epistemically indistinguishable? Should we interpret this as the agents not being able to recall the action? Or do they recall the action, but only do not know the resulting and originating state? To avoid this ambiguity, we can better add epistemic relations to ATSs that are trees.

Definition 9. *An ATS $\mathcal{M} = (S, \mathcal{T}, \pi)$ is an ATS where the function $\mathcal{T}$ is such that the system relation $\mathcal{R}^{sys}$ is a tree.*

Now note that on the subclass of tree-ATSs, the definitions of section 3.2 result in exactly the same logic ATL-STIT. This is because any ordinary ATS can be unravelled into a tree-ATS that is modally indistinguishable.

Now we can add the epistemic indistinguishability relations for separate agents. This results in a most general setup for the semantics of E-ATL-STIT, where beforehand nothing is determined about whether agents recall their actions or not: if there is an epistemic indistinguishability relation between two subsequent states of a fixed strategy, the agents cannot recall having done that action, but if there is not such a relation, they can.

Definition 10. *We extend models $\mathcal{M} = (S, \mathcal{T}, \pi)$ to models $\mathcal{M} = (S, \mathcal{R}_A, \mathcal{T}, \pi)$. The relation $\mathcal{R}_a$ for individual agents a is an equivalence relation over strategy / state pairs (α_A, s).*

We can define any of the multi-agent versions of knowledge, that is, distributed (or implicit) knowledge, shared knowledge (everybody knows) and common knowledge (reflexive transitive closure of shared knowledge), in terms of the indistinguishability relations over strategy / state pairs for the individual agents. In the standard way, we extend the truth definitions with the following clauses for the (group) knowledge operators.

Definition 11.

$\mathcal{M}, \alpha_A, s \models K_a\varphi \Leftrightarrow \forall(\beta_B, t)$ *with* $(\alpha_A, s)\mathcal{R}_a(\beta_B, t)$ *it holds that* $\mathcal{M}, \beta_B, t \models \varphi$

$\mathcal{M}, \alpha_A, s \models E_A\varphi \Leftrightarrow \forall(\beta_B, t)$ *with* $(\alpha_A, s)(\bigcup_{a\in A} \mathcal{R}_a)(\beta_B, t)$ *it holds that* $\mathcal{M}, \beta_B, t \models \varphi$

$\mathcal{M}, \alpha_A, s \models D_A\varphi \Leftrightarrow \forall(\beta_B, t)$ *with* $(\alpha_A, s)(\bigcap_{a\in A} \mathcal{R}_a)(\beta_B, t)$ *it holds that* $\mathcal{M}, \beta_B, t \models \varphi$

$\mathcal{M}, \alpha_A, s \models C_A\varphi \Leftrightarrow \forall(\beta_B, t)$ *with* $(\alpha_A, s)((\bigcup_{a\in A} \mathcal{R}_a)^*)(\beta_B, t)$ *it holds that* $\mathcal{M}, \beta_B, t \models \varphi$

The above proposal for adding the epistemic dimension is very general. Clearly it results in an S5 logic for individual agent knowledge, while leaving the sub-logic of ATL-STIT in tact. Of course several intuitive extra relational properties can be considered, leading to specific interaction properties. However, for our discussion on uniform strategies, below, the definitions suffice.

4.2 The Problem of Uniform Strategies

The most discussed problem for epistemic additions to ATL discussed in the literature (ATEL [14]), is the problem of so called 'uniform strategies'. We briefly recall the problem by means of the cards example from [13] (which we slightly adapt). There is a deck of three cards, A, K and Q. There is a somewhat unconventional order on these cards, where A beats K, K beats Q, but Q beats A. Now consider two gambling agents a and b who each get a card from the dealer. Before a showdown occurs, agent a is given the choice to swap his card with the one remaining on the dealers deck. Apparently due to the incompleteness of his knowledge a does not know a winning strategy. He does not know the card still in the deck, but depending on what this card is, he either has to swap or not in order to win. Structures of ATEL equip ATSs with epistemic indistinguishability relations between states (moments). Now it is perceived as counterintuitive that in the ATEL structures we can draw for this little game, at the moment corresponding to the decision point of agent a, it is true that $K_a\langle\langle a\rangle\rangle win$. This holds since the agent cannot distinguish the state where he has the winning card from the state where he has the loosing card, but whichever state he is in, it has a guaranteed possibility to win if it chooses the right strategy in the right state. However, the truth of this formula is perceived as counterintuitive since one is tempted to believe that it expresses that a has a *single* 'uniform strategy' for winning, that is, a strategy that guarantees a win irrespective of the state the agent is in.

But it appears to us that if we stay faithful to the intended meaning of the operators involved, the formula is not counterintuitive: it exactly expresses what is the case, namely that agent a knows that there is a strategy to win. Indeed that does not imply that he knows what strategy to apply, which, in this case, is exactly the only reason why he cannot ensure the win. So, the problem appears to be that one is tempted to read something in the formula that is not there, namely, that the agent knows a uniform strategy for winning. Maybe the present formalism, that decomposes the standard ATL operators in two separate modal operators, enables us to see that more clearly.

However, an ensuing problem is that one indeed would like to have a way of expressing that an agent, or group of agents does not know what the current state is, while at the same time they do know (or do not know) how to win. In the above example, the agent a did not know how to win. We would like to have a formula corresponding to that fact. In ATEL [14] we cannot express that. But the present formalism, with its more fine grained epistemic structures, enables us to express this directly as $\neg\Diamond_a K_a[a]win$, that is, a has no single strategy for which he knows he is guaranteed to win. We cannot find an equivalent formula in ATEL, because ATEL's semantic structures are not fine-grained enough in two respects. First, because in ATEL, evaluation is only with respect to states, it cannot give semantics to the decomposition of the ATL operator $\langle\langle A\rangle\rangle\eta$ into $\Diamond_A[A]\eta$, and second, because epistemic indistinguishability relations are

defined over states, it cannot give semantics to the notion of an agent knowing a strategy.

Then the question is, does this solve the problem of so called 'uniform strategies' as formulated in the literature? That depends on how one looks at it. Actually it is not completely clear to us what in the context of ATSs, should be understood by a 'uniform strategy'. The notion of 'uniform strategy' comes from game theory [15]. But game theory is different from logic in that it studies the properties of game structures as such, that is, independent of a logical language like ATL to be interpreted over them. In game structures the choices have action names. ATL, and also STIT-ATL are endogenous temporal formalisms that cannot express anything related to the action names of game structures. And in particular those action names have been associated to the notion of 'uniform strategies'. Uniform strategies have been described as strategies where the 'same actions' are performed from different states to ensure a certain property. If actions have names, the same actions can be defined as actions having corresponding names. The present proposal does not solve the problem of uniform strategies interpreted in this sense. We believe, solutions would require an exogenous language, where in one way or the other there is reference to the names of actions in the object language. However, in a weaker sense the present proposal does solve the problem. In ATSs actions are identified with what they bring about. Then, typically, single strategies take *different* actions from different states. And it is also the other way around: taking two different strategies in two different states may mean that one performs the same actions. Now, if 'knowing a uniform strategy for φ, without possibly knowing the current state' is defined as 'knowingly seeing to it that φ, without possibly knowing the current state', the present proposal does offer a solution to the problem of uniform strategies.

Generalizing the idea in [7] we can express that there is an A-strategy, where the agents A commonly know that they ensure η as:

$$\Diamond_A C_A[A]\eta$$

Agents A commonly knowing the existence of a strategy (without knowing whether they actually perform the strategy) is expressed as:

$$C_A \Diamond_A[A]\eta$$

Note that in the first of the above formulas, for the concept of 'a group of agents A knowingly performing a strategy', we used that the agents have *common knowledge* that they perform the strategy. We thus defined this concept as $C_A[A]\eta$. In our opinion distributed knowledge or shared knowledge is not enough. For instance, me and a friend can only knowingly follow a strategy of meeting in Paris someday if I know that he knows, and I know that he knows that I know, etc.

5 Related Research

Horty ([6] p. 151) explains that it is not that easy to generalize the standard STIT framework where evaluation is with respect to moment / history pairs, to the strategic case.

In general, more than one strategy may be compatible with the same moment / history pair. Horty's first suggestion is then to implicitly quantify over all strategies that correspond to a given moment / history pair. His second suggestion is much closer to the solution proposed in this paper (note that here we assume the close relatedness between the STIT-framework and the ATL-framework we explored in [10]). Horty suggests to evaluate formulas with respect to 'state / history / history-set' triples (where the history is an element of the history-set), and to define the semantics of his strategic STIT operator $[A\ cstit : \varphi]$ (agents A strategically see to it that φ) as there being a strategy α, such that the history-set equals the histories admitted by the strategy, and φ being true on all these histories. Our proposal differs from this proposal on three points. First, for the present ATL-setting we do not see the need to include the history in the units for evaluation. Second, we think it is better to simply see the strategies themselves as part of the units of evaluation. We explicitly need this in our discussion of uniform strategies in section 4.2. Finally, we believe Horty's definition fails to model the important property that if a set of agents sees to something, any superset also sees to that same something. This property follows from our definition as the result of taking the intersections in the truth conditions for $[A]\varphi$ and $\langle A\rangle\varphi$.

Using ideas similar to ours Müller [8] defines a semantics for the notion of 'continuous action' in the STIT framework. Like us, Müller suggests to take up strategies as elements in the units over which to evaluate formulas. To be more precise, Müller evaluates with respect to 'context-state / state / history / strategy' quadruples. His notion of ISTIT (*is* seeing to it that), is then defined, roughly, as truth on all histories admitted by the strategy. Although the idea to take up strategies in the units of evaluation is similar, other aspects of the approach are quite different. That is not too surprising, since Müller's aim is an ISTIT operator, while we aim at a strategic STIT operator. Also Müller does not aim at defining a multi-agent variant of his operator. More in particular, his strategies are always single agent strategies. In our setting, the problem of dealing with multi-agent strategies is central.

Finally, also Jamroga and gnotes [9] suggest to change the units of evaluation. Aiming at solving the problem of uniform strategies in ATEL, they suggest to evaluate formulas with respect to sets of states. However, their approach is much further removed from our approach than Horty's or Müller's.

6 Conclusion

This paper extends ATL with strategic STIT operators. We argued that the evaluation with respect to strategy / state pairs is essential for a logic that aims to reason about decisions that are fixed for groups of agents. Here the decisions are to take a particular strategy. Also we discussed the problem of uniform strategies, and explained how our formalism can be seen as a partial answer to that problem.

There are many possible applications of this extended formalism. We discussed some preliminary investigations in the epistemic realm. Another route of investigation is the extension with deontic operators. One of the reasons STIT logics are popular in deontic logic is that they are the best formalism around to model the fourth sentence of Chisholm's infamous benchmark scenario for deontic formalizations [16]. To add

deontic expressivity, we may consider several options. For instance, Wansing [17] has suggested to model personal obligations imposed by one agent onto the other by identifying this with 'agent *a* sees to it that agent *b* is punished if he does not comply to his obligations'. This approach can incorporated in the present framework very well. Another option is simply to define a deontic accessibility relation over strategy / state pairs, like we did for the epistemic indistinguishability relation.

References

1. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. In: FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS '97), IEEE Computer Society (1997) 100–109
2. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (2002) 672–713
3. Belnap, N., Perloff, M.: Seeing to it that: A canonical form for agentives. Theoria **54** (1988) 175–199
4. Belnap, N., Perloff, M.: Seeing to it that: A canonical form for agentives. In Kyburg, H.E., Loui, R.P., Carlson, G.N., eds.: Knowledge Representation and Defeasible Reasoning. Kluwer, Boston (1990) 167–190
5. Belnap, N., Perloff, M., Xu, M.: Facing the future: agents and choices in our indeterminist world. Oxford University Press (2001)
6. Horty, J.: Agency and Deontic Logic. Oxford University Press (2001)
7. Herzig, A., Troquard, N.: Knowing How to Play: Uniform Choices in Logics of Agency. In Weiss, G., Stone, P., eds.: 5th International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS-06), Hakodate, Japan, ACM Press (2006) 209–216
8. Müller, T.: On the formal structure of continuous action. In Schmidt, R., Pratt-Hartmann, I., Reynolds, M., Wansing, H., eds.: Advances in Modal Logic. Volume 5., King's College Publications (2005) 191–209
9. Jamroga, W., gotnes, T.: Constructive knowledge: what agents can achieve under incomplete information. Technical Report IfI-05-10, Institute of Computer Science, Clausthal University of Technology, Clausthal-Zellerfeld (2005)
10. Broersen, J., Herzig, A., Troquard, N.: From coalition logic to stit. In: Proceedings LCMAS 2005. Electronic Notes in Theoretical Computer Science, Elsevier (2005)
11. Goranko, V., Jamroga, W.: Comparing semantics of logics for multi-agent systems. Synthese **139**(2) (2004) 241–280
12. Pauly, M.: A modal logic for coalitional power in games. Journal of Logic and Computation **12**(1) (2002) 149–166
13. Jamroga, W., Hoek, W.v.d.: Agents that know how to play. Fundamenta Informaticae **63**(2) (2004)
14. Hoek, W.v.d., Wooldridge, M.: Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. Studia Logica **75**(1) (2003) 125–157
15. Neumann, J.v., Morgenstern, O.: Theory of games and economic behaviour. Princeton University Press (1944)
16. Chisholm, R.: Contrary-to-duty imperatives and deontic logic. Analysis **24** (1963) 33–36
17. Wansing, H.: Obligations, authorities, and history dependence. In Wansing, H., ed.: Essays on Non-classical Logic. World Scientific (2001) 247–258

On the Logic and Computation of Partial Equilibrium Models

Pedro Cabalar[1], Sergei Odintsov[2], David Pearce[3], and Agustín Valverde[4,⋆]

[1] Corunna University (Corunna, Spain)
cabalar@udc.es
[2] Sobolev Institute of Mathematics (Novosibirsk, Russia)
odintsov@math.nsc.ru
[3] Universidad Rey Juan Carlos (Madrid, Spain)
davidandrew.pearce@urjc.es
[4] University of Málaga (Málaga, Spain)
a_valverde@ctima.uma.es

Abstract. The nonmonotonic formalism of *partial equilibrium logic* (PEL) has recently been proposed as a logical foundation for the partial stable and well-founded semantics of logic programs [1, 2]. We study certain logical properties of PEL and some techniques to compute partial equilibrium models.

1 Introduction

The *well-founded semantics* (WFS) of [16] and the closely related semantics of *partial stable models* [12] are among the most established approaches to dealing with default negation in logic programming. Until recently however their logical foundations remained largely undeveloped. Now in [1, 2] a nonmonotonic formalism called *partial equilibrium logic* has been proposed as a foundation for the partial stable (p-stable) and well-founded semantics. The main idea is to identify a (non-model) logic that is adequate for WFS in the sense that its minimal models (appropriately defined) coincide with the p-stable models of a program. The logic in question is based on 6-valued truth matrices and can be considered as a semantic generalisation of the logic HT of here-and-there that has been used to capture the stable model semantics [8]; accordingly we denote it by HT^2. Just as equilibrium models correspond to the stable models of programs, so partial equilibrium models correspond to the p-stable models defined in [12]. While the underlying models of HT and HT^2 are different, in each case the equilibrium construction is similar, based on defining certain total models that are minimal.

In previous work, [1, 2], partial equilibrium logic (PEL) was defined and the correspondence between p-equilibrium and p-stable models was established. The logic HT^2 was axiomatised and completeness proved. Analogous to the case of equilibrium logic, it was shown that the strong equivalence of theories wrt PEL can be captured by equivalence in the logic HT^2. In addition, some properties of nonmonotonic entailment in PEL and its complexity were studied as well as a method for reducing PEL to ordinary equilibrium logic. The aim of this paper is to examine further logical and computational

⋆ This research was partially supported by CICyT project TIC-2003-9001-C02.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 82–94, 2006.

issues associated with p-equilibrium models and their underlying logics: HT^2 and the logic of total HT^2 models, which we denote by HT^*. Specifically we provide a proof theory for PEL by presenting tableau calculi for the logics HT and HT^* as well as for p-equilibrium model checking. The calculus for HT^2 is of independent interest as a means for checking the strong equivalence of theories. We also axiomatise the logic HT^* and discuss its relation to other logics such as Przymusinski's Prz_3 [13]. Lastly we consider the method of *splitting* a logic program, a familiar technique for optimising computation under the stable model semantics [5, 3]. In particular we derive a splitting theorem for disjunctive and nested logic programs under PEL.

2 Logical Preliminaries: The Logics HT^2 and PEL

We introduce the logic HT^2 and its semantics, given in terms of HT^2 *frames*, and we define *partial equilibrium logic* (PEL) in terms of minimal HT^2 models. Formulas of HT^2 are built-up in the usual way using atoms from a given propositional signature At and the standard logical constants: $\wedge$, $\vee$, $\rightarrow$, $\neg$. We write $\mathfrak{L}(At)$ to stand for the set of all well-formed formulae (ie, the language) under signature At. A set of HT^2 formulae is called a *theory*. The axiomatic system for HT^2 is described in two stages. In the first stage we include the following inference rules:

$$\frac{\alpha,\ \alpha \rightarrow \beta}{\beta}\ \text{(Modus Ponens)} \qquad\qquad \frac{\alpha \rightarrow \beta}{\neg\beta \rightarrow \neg\alpha}$$

plus the axiom schemata of *positive logic* together with:

$$A1.\ \neg\alpha \wedge \neg\beta \rightarrow \neg(\alpha \vee \beta) \qquad A2.\ \neg(\alpha \rightarrow \alpha) \rightarrow \beta \qquad A3.\ \neg(\alpha \wedge \beta) \rightarrow \neg\alpha \vee \neg\beta$$

Thus, both De Morgan laws are provable in HT^2. Moreover, axiom A2 allows us to define intuitionistic negation, '$-$', in HT^2 as: $\quad -\alpha := \alpha \rightarrow \neg(p_0 \rightarrow p_0)$.
In a second stage, we further include the rule $\frac{\alpha \vee (\beta \wedge \neg\beta)}{\alpha}$ and the axioms schemata:

A4. $-\alpha \vee - - \alpha$
A5. $-\alpha \vee (\alpha \rightarrow (\beta \vee (\beta \rightarrow (\gamma \vee -\gamma))))$
A6. $\bigwedge_{i=0}^{2}((\alpha_i \rightarrow \bigvee_{j\neq i} \alpha_j) \rightarrow \bigvee_{j\neq i} \alpha_j) \rightarrow \bigvee_{i=0}^{2} \alpha_i$
A7. $\alpha \rightarrow \neg\neg\alpha$
A8. $\alpha \wedge \neg\alpha \rightarrow \neg\beta \vee \neg\neg\beta$
A9. $\neg\alpha \wedge \neg(\alpha \rightarrow \beta) \rightarrow \neg\neg\alpha$
A10. $\neg\neg\alpha \vee \neg\neg\beta \vee \neg(\alpha \rightarrow \beta) \vee \neg\neg(\alpha \rightarrow \beta)$
A11. $\neg\neg\alpha \wedge \neg\neg\beta \rightarrow (\alpha \rightarrow \beta) \vee (\beta \rightarrow \alpha)$

HT^2 is determined by the above inference rules and the schemata A1-A11.

Definition 1. *A (Routley) frame is a triple $\langle W, \leq, * \rangle$, where W is a set, $\leq$ a partial order on W and $* : W \rightarrow W$ is such that $x \leq y$ iff $y^* \leq x^*$. A (Routley) model is a Routley frame together with a valuation V ie. a function from $At \times W \longrightarrow \{0, 1\}$ satisfying (1): $V(p, u) = 1\ \ \&\ \ u \leq w\ \ \Rightarrow\ \ V(p, w) = 1$.*

The valuation V is extended to all formulas via the usual rules for intuitionistic (Kripke) frames for the positive connectives $\wedge$, $\vee$, $\rightarrow$ where the latter is interpreted via the $\leq$ order:

$$V(\varphi \rightarrow \psi, w) = 1 \text{ iff for all } w' \text{ such that } w \leq w', \quad V(\varphi, w') = 1 \Rightarrow V(\psi, w') = 1$$

The main difference with respect to intuitionistic frames is the presence of the $*$ operator that is used for interpreting negation via the following condition:

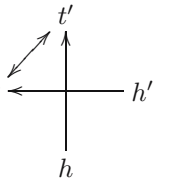
$$V(\neg\varphi, w) = 1 \text{ iff } \quad V(\varphi, w^*) = 0.$$

A proposition φ is said to be *true* in a model $\mathcal{M} = \langle W, \leq, *, V\rangle$, if $V(\varphi, v) = 1$, for all $v \in W$. A formula φ is *valid*, in symbols $\models \varphi$, if it is true in every model. It is easy to prove by induction that condition (1) in Definition 1 above holds for any formula φ, ie

$$V(\varphi, u) = 1 \ \ \& \ \ u \leq w \Rightarrow V(\varphi, w) = 1. \tag{1}$$

Definition 2 (HT^2 model). *An* HT^2 model *is a Routley model* $\mathcal{M} = \langle W, \leq, R, V\rangle$ *such that (i)* W *comprises 4 worlds denoted by* h, h', t, t'*, (ii)* $\leq$ *is a partial ordering on* W *satisfying* $h \leq t$*,* $h \leq h'$*,* $h' \leq t'$ *and* $t \leq t'$*, (iii) the* $*$ *operation is determined by* $h^* = t^* = t'$*,* $(h')^* = (t')^* = t$*, (iv)* V *is a-valuation.*

The diagram on the right depicts the $\leq$-ordering among worlds (a strictly higher location means $\geq$) and the action of the $*$- mapping using arrows.



Truth and validity for HT^2 models are defined analogously to the previous case and from now on we let $\models$ denote the truth (validity) relation for HT^2 models. We have the following completeness theorem[1]:

Theorem 1 ([1]). $\models \varphi$ *iff* φ *is a theorem of* HT^2.

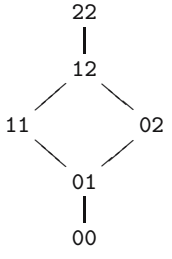
2.1 HT^2 as a 6-Valued Logic

Now, consider an HT^2 model $\mathcal{M} = \langle W, \leq^*, V\rangle$ and let us denote by H, H', T, T' the four sets of atoms respectively verified at each corresponding point or world h, h', t, t'. More succinctly, we can represent $\mathcal{M}$ as the pair $\langle \mathbf{H}, \mathbf{T}\rangle$ so that we group each pair of unprimed/primed world as $\mathbf{H} = (H, H')$ and $\mathbf{T} = (T, T')$. By construction, each of these pairs $\mathbf{I} = (I, I')$ satisfies $I \subseteq I'$, so that $\mathbf{I}$ can be seen as a 3-valued interpretation. Given $\mathbf{I}$ and an atom p, we use the values $\{0, 1, 2\}$ to respectively denote $p \in I$, $p \in I' \setminus I$ and $p \notin I'$. As we have two pairs like this, $\langle \mathbf{H}, \mathbf{T}\rangle$, the possible "situations" of a formula in HT^2 can be defined by a pair of values xy with $x, y \in \{0, 1, 2\}$. Condition (1) restricts the number of these situations to the following six $\texttt{00} := \emptyset$, $\texttt{01} := \{t'\}$, $\texttt{11} := \{h', t'\}$, $\texttt{02} := \{t, t'\}$, $\texttt{12} := \{h', t, t'\}$, $\texttt{22} := W$ where each set shows the worlds at which the formula is satisfied. Thus, an alternative way of describing HT^2 is by providing its logical matrix in terms of a 6-valued logic. As a result, the above

[1] The first stage alone defines a logic complete for the general Routley frames.

setting becomes an algebra of 6 cones: $\mathcal{A}^{HT^2} := \langle\{00, 01, 11, 02, 12, 22\}, \vee, \wedge, \rightarrow, \neg\rangle$ where $\vee$ and $\wedge$ are set theoretical join and meet, whereas $\rightarrow$ and $\neg$ are defined as follows: $x \rightarrow y := \{w : w \leq w' \Rightarrow (w' \in x \Rightarrow w' \in y)\}, \quad \neg\, x := \{w : w^* \notin x\}$.

The only distinguished element is 22. The lattice structure of this algebra can be described by the condition $xy \leq zt \Leftrightarrow x \leq z \;\&\; y \leq t$ and is shown in Figure 1, together with the resulting truth-tables.



φ	$\neg\varphi$
00	22
01	11
11	11
02	00
12	00
22	00

$\rightarrow$	00	01	11	02	12	22
00	22	22	22	22	22	22
01	00	22	22	22	22	22
11	00	02	22	02	22	22
02	00	11	11	22	22	22
12	00	01	11	02	22	22
22	00	01	11	02	12	22

$$V(\phi \wedge \psi) = \min\{V(\phi), V(\psi)\}, V(\phi \vee \psi) = \max\{V(\phi), V(\psi)\}$$

Fig. 1. Lattice structure and truth tables for the 6-valued HT^2 description

2.2 Minimal Models and Relation to Logic Programs

The truth-ordering relation among 3-valued interpretations $\mathbf{I}_1 \leq \mathbf{I}_2$ is defined so that $\mathbf{I}_1$ contains less true atoms and more false ones (wrt set inclusion) than $\mathbf{I}_2$. Note that by the semantics, if $\langle \mathbf{H}, \mathbf{T}\rangle$ is a model then necessarily $\mathbf{H} \leq \mathbf{T}$, since it is easy to check that this condition is equivalent to $H \subseteq T$ and $H' \subseteq T'$. Moreover, for any theory Π note that if $\langle \mathbf{H}, \mathbf{T}\rangle \models \Pi$ then also $\langle \mathbf{T}, \mathbf{T}\rangle \models \Pi$.

The ordering $\leq$ is extended to a partial ordering $\trianglelefteq$ among models as follows. We set $\langle \mathbf{H}_1, \mathbf{T}_1\rangle \trianglelefteq \langle \mathbf{H}_2, \mathbf{T}_2\rangle$ if (i) $\mathbf{T}_1 = \mathbf{T}_2$; (ii) $\mathbf{H}_1 \leq \mathbf{H}_2$. A model $\langle \mathbf{H}, \mathbf{T}\rangle$ in which $\mathbf{H} = \mathbf{T}$ is said to be *total*. Note that the term *total* model does not refer to the absence of undefined atoms. To represent this, we further say that a total partial equilibrium model is *complete* if $\mathbf{T}$ has the form (T, T).

We are interested here in a special kind of minimal model that we call a partial equilibrium (or p-equilibrium) model. Let Π be a theory.

Definition 3 (Partial equilibrium model). *A model $\mathcal{M}$ of Π is said to be a* partial equilibrium *model of Π if (i) $\mathcal{M}$ is total; (ii) $\mathcal{M}$ is minimal among models of Π under the ordering $\trianglelefteq$.*

In other words a p-equilibrium model of Π has the form $\langle \mathbf{T}, \mathbf{T}\rangle$ and is such that if $\langle \mathbf{H}, \mathbf{T}\rangle$ is any model of Π with $\mathbf{H} \leq \mathbf{T}$, then $\mathbf{H} = \mathbf{T}$. We will sometimes use the abbreviation $\mathbf{T} \mid\!\approx \Pi$ to denote that $\langle \mathbf{T}, \mathbf{T}\rangle$ is a p-equilibirum model of theory Π. *Partial equilibrium logic* (PEL) is the logic determined by truth in all p-equilibrium models of a theory.

We turn to the relation between PEL and logic programs. A *disjunctive logic program* is a set of formulas (also called *rules*) of the form

$$a_1 \wedge \ldots \wedge a_m \wedge \neg b_1 \wedge \ldots \wedge \neg b_n \rightarrow c_1 \vee \ldots \vee c_k \tag{2}$$

where the a, b, c with subscripts range over atoms and $m, n, k \geq 0$; for the definition of the p-stable models of a disjunctive logic program Π, see [12].

Theorem 2 ([2]). *A total HT^2 model $\langle \mathbf{T}, \mathbf{T} \rangle$ is a p-equilibrium model of a disjunctive program Π iff the 3-valued interpretation $\mathbf{T}$ is a p-stable model of Π.*

We define a further partial ordering on total models by $\langle \mathbf{T}_1, \mathbf{T}_1 \rangle \preceq \langle \mathbf{T}_2, \mathbf{T}_2 \rangle$ if both $T_1 \subseteq T_2$ and $T_2' \subseteq T_1'$. Then we say that a total HT^2 model that is $\preceq$-minimal among the p-equilibrium models of a theory Γ is a *well-founded model* of Γ. This terminology is justified by the fact that if Π is a normal logic program, the unique $\preceq$-minimal p-equilibrium model of Π coincides with the well-founded model of Π in the sense of [16]. In the general case, however, an arbitrary PEL theory may have several well-founded models, or even no well-founded model at all.

The notion of strong equivalence for logic programs was introduced in [6] and logically characterised for the case of programs under stable model semantics. The study of strong equivalence, its generalisations and computation, has since become a lively research area, with potential for application to program optimisation. Until now there was no analogous research programme for p-stable and WF semantics. A basis is provided however by Theorem 3 below and several extensions proved in [2].

Definition 4 ((strongly) equivalent theories). *Two theories Π, Π' are said to be (PEL)-equivalent or simply* equivalent *(resp.* strongly equivalent*), in symbols $\Pi \equiv \Pi'$ (resp. $\Pi \equiv_s \Pi'$), iff they have the same p-equilibrium models (resp. iff for any Γ, $\Pi \cup \Gamma \equiv \Pi' \cup \Gamma$).*

Theorem 3 ([1]). *Two theories Π, Π' are strongly equivalent iff they are HT^2 equivalent, ie have the same HT^2 models.*

This provides added interest in computational proof systems for HT^2.

2.3 Complexity of Reasoning in HT^2 and PEL

We denote by SAT_{CL} and VAL_{CL} the classes of satisfiable formulas and valid formulas respectively in classical logic, and SAT_{HT^2} and VAL_{HT^2} the classes of satisfiable formulas and valid formulas respectively in HT^2 logic.

Theorem 4 ([2]). *(i) SAT_{HT^2} is NP-complete and VAL_{HT^2} is coNP-complete; (ii) the problem of deciding whether a formula in HT^2 has partial equilibrium models is Σ_2^P-hard.*

Corollary 1 ([2]). *(i) The problem of checking the strong equivalence of theories is coNP-complete. (ii) The decision problem for equilibrium entailment is Π_2^P-hard.*

3 The Logic of Total Models

Total models play an important role in the definition of PEL since p-equilibrium models are a special kind of total model. We describe the logic of total models.

First note that total models can be distinguished among all HT^2-models via the scheme $\neg\neg\varphi \to \varphi$. For an HT^2 model $\mathcal{M} = \langle (H, H'), (T, T') \rangle = \langle \mathcal{W}^{\mathcal{HT}^{\in}}, \mathcal{V} \rangle$, set

$$\Delta_w^{\mathcal{M}} := \{\varphi : V(\varphi, w) = 1\}$$

for $w \in W^{HT^2}$. Obviously, $H \supset \Delta_h^{\mathcal{M}}$, $H' \supset \Delta_{h'}^{\mathcal{M}}$, etc. We omit the superscript $\mathcal{M}$ if it does not lead to confusion.

Proposition 1. *The following items are equivalent:*

1. $\langle \mathbf{H}, \mathbf{T} \rangle \models \neg\neg\varphi \to \varphi$ *for any* φ,
2. $\mathbf{H} = \mathbf{T}$,
3. $\Delta_h = \Delta_t$ *and* $\Delta_{h'} = \Delta_{t'}$.

Let us set $HT^* := HT^2 + \{\neg\neg p \to p\}$. From the last proposition, it follows that the number of possible situations of a formula in a total HT^2-model is reduced to the following three, $\mathtt{00} := \emptyset$, $\mathtt{11} := \{h', t'\}$, $\mathtt{22} := \{h, h', t, t'\}$, where each set shows the worlds at which the formula is satisfied. Thus, logic HT^* can be characterised by the three-element algebra: $\mathcal{A}^{HT^*} := \langle \{\mathtt{00}, \mathtt{11}, \mathtt{22}\}, \vee, \wedge, \to, \neg \rangle$ with the only distinguished element 22 and operations determined as the restrictions of the respective operation of the algebra $\mathcal{A}^{HT^2}$. It is routine to check that the set $\{\mathtt{00}, \mathtt{11}, \mathtt{22}\}$ is closed under $\mathcal{A}^{HT^2}$-operations.

At the same time, HT^* differs from Przymusinski's logic Prz_3 [13] as well as from $\mathbf{N}_3$ [15, 10], classical explosive logic with strong negation. All these logics are three-valued and the operations $\vee$ and $\wedge$ determine the structure of a linearly ordered lattice on the set of truth-values. If we denote the least truth-value in all these logics by 00, the greatest by 22, and the intermediate by 11, we see that all the logics have the same connectives $\neg$, $\vee$, $\wedge$, but different implications:

$\to_{HT^*}$	00	11	22
00	22	22	22
11	00	22	22
22	00	11	22

$\to_{\mathbf{N}_3}$	00	11	22
00	22	22	22
11	22	22	22
22	00	11	22

$\to_{Prz_3}$	00	11	22
00	22	22	22
11	00	22	22
22	00	00	22

Comparing HT^* and $\mathbf{N}_3$ we note the following

Proposition 2. $HT^* \subsetneqq \mathbf{N}_3$, $\neg(p \to q) \leftrightarrow (p \wedge \neg q) \notin HT^*$.

For the comparison of HT^* and Prz_3, recall that the language of Prz_3 contains also the necessity operator l ($l\mathtt{22} = \mathtt{22}$, $l\mathtt{x} = \mathtt{00}$ otherwise) and $\to_{Prz_3}$ can be defined via $\neg$, $\vee$, $\wedge$ and l: $\varphi \to_{Prz_3} \psi := (\neg l\varphi \vee l\psi) \wedge (\neg l\neg\psi \vee l\neg\varphi)$.

At the same time, $l\varphi$ can be defined in HT^* as $\neg(\varphi \to_{HT^*} \neg\varphi)$.

Proposition 3. *Logic* Prz_3 *is definable in* HT^*. □

A simple axiomatisation of HT^* modulo the basic logic N^* is given by the following

Proposition 4. $HT^* = N^* + \{p \vee (p \to q) \vee -q,\ p \leftrightarrow \neg\neg p,\ p \wedge \neg p \to q \vee \neg q\}$.

Proof. In fact, the proof of these statement is a simplified version of the completeness proof for HT^2 in [1].

Thus, we obtain HT^* by extending the intuitionistic fragment to HT and adding the elimination of double negation and the Kleene axiom. Despite the fact that HT^* and HT have the same intuitionistic fragment, they have different negations and $HT^* \neq HT$. We can obtain HT from HT^2 in the following way.

Proposition 5. *The addition to HT^2 of axiom (I) $= \neg\varphi \wedge \varphi \rightarrow \bot$, is equivalent to the condition $T = T'$.* □

Proposition 6. *The addition to HT^2 of De Jongh and Hendrik's axiom (used to obtain HT from intuitionistic logic), (dJH) $= \varphi \vee (\varphi \rightarrow \psi) \vee -\varphi$ is equivalent to the condition: $T, H' \in \{H, T'\}$.*

Proposition 7 (reduction to HT). *$HT = HT^2 \cup$ (I) $\cup$ (dJH).*

4 A Tableau Calculus for PEL

We can describe a tableaux system for HT^2 using the standard methods for finite-valued logics [4, 10]. The formulas in the tableau nodes are labelled with a set of truth-values, named *signs*, and these signs are propagated to the subformulas using the expansion rules. The family of the signs depends on the logic in question and it is possible to describe several tableaux systems for the same logic. For HT^2 we will use the following signs, where $[\geq v] = \{w \in \mathbf{6} \mid w \geq v\}$, and $[\leq v] = \{w \in \mathbf{6} \mid w \leq v\}$:

$$\{00\}, \{01\}, \{11\}, \{02\}, \{22\}, \{01, 11\}, [\leq 01], [\leq 11], [\leq 12], [\geq 01], [\geq 02], [\geq 12]$$

The usual notions of *closed* and *terminated* tableaux can be used in different ways. In the following definition we introduce the concept of *closed tableau* in order to characterise validity in HT^2.

Definition 5. *Let φ be a formula in HT^2:*

1. *The* Initial tableau *to check the validity of φ is: $T_0 = {}_{[\leq 12]:}\varphi$*
2. *If T is a tableau and T' is the tree obtained from T applying one of the expansion rules in figure 2, then T' is tableau for φ.*
3. *A branch B in a tableau T is called* closed *if one of the following conditions hold: (i) it contains the constant $\bot$; (ii) it contains* signed literals*, ${}_{S_1:}p, \ldots, {}_{S_n:}p$, such that $\cap_{i=1}^{n} S_i = \varnothing$. A tableau T is called* closed *if every branch is closed.*

Intuitively, with the initial tableau ${}_{[\leq 12]:}\varphi$ we ask if it is possible to find an assignment for φ that evaluates in $[\leq 12]$, in other words a countermodel. The expansion rules search for ways to evaluate the subformulas so as to define the countermodel.

Theorem 5 (Soundness and completeness of the tableaux system). *The formula φ is valid in HT^2 if and only if there exists a closed tableau for it.*

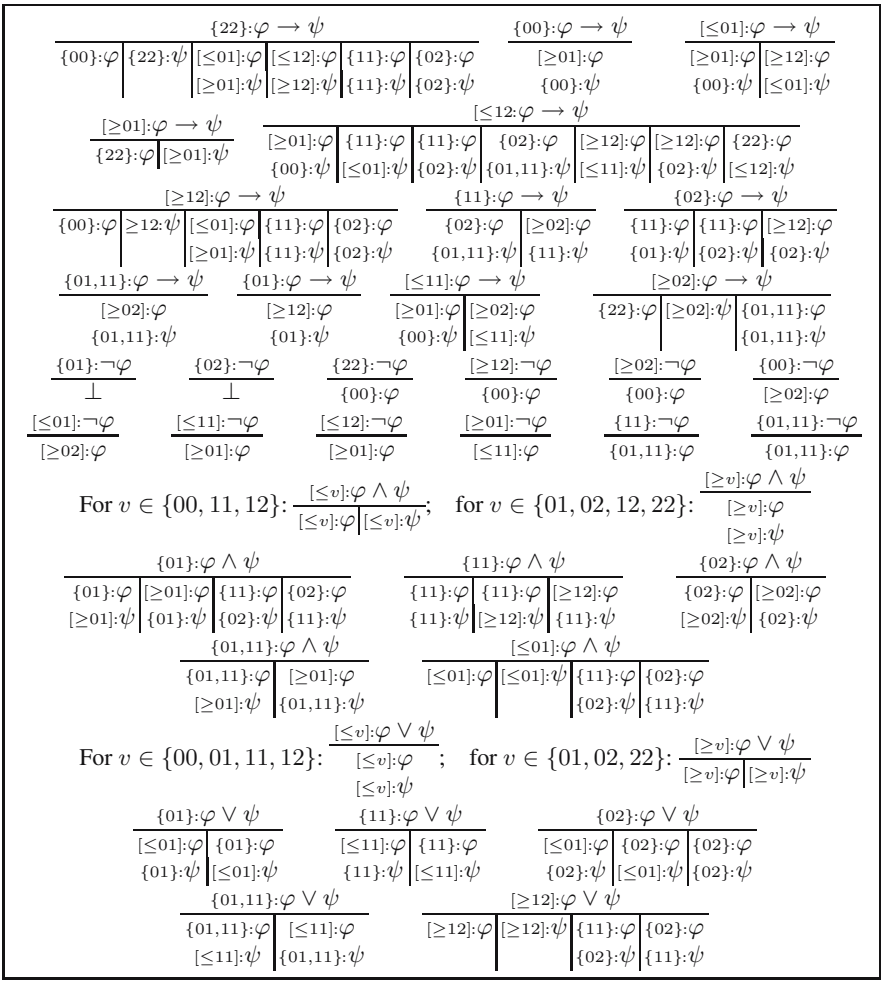


Fig. 2. Expansion rules for HT^2

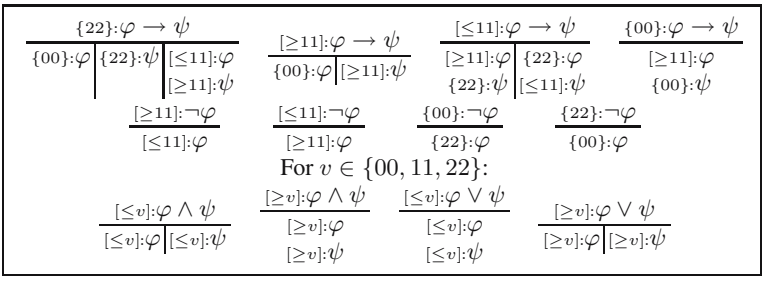


Fig. 3. Expansion rules for total models of HT^2, ie. for HT^*

4.1 Partial Equilibrium Models

Tableaux systems can also be used to study additional properties and relations [10, 11]. In this section we define a system based on auxiliary tableaux in order to generate the partial equilibrium models of a theory. We proceed in two phases. First, we generate the total models of a theory by means of a tableau system in which we search for a *terminated tableau*. Then, for every total model, an auxiliary tableau is constructed to check whether the model in question is in partial equilibrium or not.

The total assignments evaluate formulas in $\{00, 11, 22\}$ and thus we only need to work with the following system of signs: $[\leq 11] = \{00, 11\}$, $[\leq 00] = \{00\}$, $[\geq 11] = \{11, 22\}$, $[\geq 11] = \{22\}$.

Definition 6. *Let $\Pi = \{\varphi_1, \ldots, \varphi_n\}$ a theory in HT^2:*

1. *The* Initial tableau *to generate total models is a single branch tree containing the following signed formulas:* $\{22\}{:}\varphi_1, \ldots, \{22\}{:}\varphi_n$.
2. *If T is a tableau and T' is the tree obtained from T by applying one of the expansion rules in figure 3, then T' is tableau for φ. As usual in tableaux systems for propositional logics, if a formula can be used to expand the tableau, then the tableau is expanded in every branch below the formula using the corresponding rule, and the formula used to expand is marked and is no longer used.*
3. *A branch in a tableau T is called* closed *if the* signed literals *for a variable p, $S_1{:}p, \ldots, S_m{:}p$, verify $\cap_{i=1}^{n} S_i = \varnothing$. It is call* open *otherwise.*
4. *A branch in a tableau T is called* finished *if it doesn't contain non-marked formulas.*
5. *A tableau T is called* closed *if every branch is closed, and it is* terminated *if every branch is either closed or finished.*

In this case the tableau begins with formulas signed with 22, since we are looking for models. The expansion rules guarantee the construction of all possible models in such a way that when all formulas have been expanded, all the models can be determined on the basis of open branches.

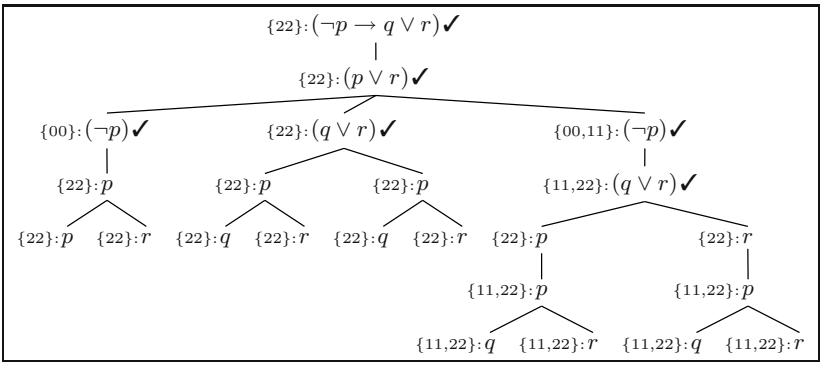


Fig. 4.

Theorem 6. *Let T be a non-closed terminated tableau for Π, and let $\{s_1{:}p_1, \ldots, s_n{:}p_n\}$ be the set of signed literals in an open branch. Then every assignment V verifying $V(p_i) \in S_i$, for all i, is a total model of φ. Moreover, all the total models of Π are generated from T in this way.*

Example: (Taken from [2]) The figure 4 shows the tableau for the theory $\Pi = \{\neg p \rightarrow q \vee r, p \vee r\}$

The tableau is finished and allows us to construct the set of total models of Π, as shown in the following table:

	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}	σ_{11}	σ_{12}	σ_{13}	σ_{14}	σ_{15}
p	22	22	22	22	22	22	22	22	22	11	11	11	00	00	00
q	22	22	22	11	11	11	00	00	00	22	11	00	22	11	00
r	22	11	00	22	11	00	22	11	00	22	22	22	22	22	22

Auxiliary tableau to check the partial equilibrium property. A total model is in partial equilibrium if there is no other model of the theory less than it under the partial ordering $\lhd$. In terms of the many-valued semantics, this ordering is defined between assignments based on the following relations between truth-values: $01 \lhd 11, 02 \lhd 12 \lhd 22$. To look for such a model we construct an initial tableau specifically for each total model by applying the expansion rules in figure 2.

Definition 7. *Let φ be a formula in HT^2 and V a total model of it.*

1. *The* Initial tableau *to check the partial equilibrium property of V for φ is a single branch tree containing the following signed formulas: $\{22\}{:}\varphi$, $\{00\}{:}p$ for every p such that $V(p) = 00$, $\{01,11\}{:}p$ for every p such that $V(p) = 11$, and $\{02,12,22\}{:}p$ for every p such that $V(p) = 22$.*
2. *If T is a tableau and T' is the tree obtained from T applying one of the expansion rules in figure 2, then T' is φ.*
3. *A branch B in a tableau T is called V-*closed *if one of the following condition holds: (i) it contains the constant $\bot$; (ii) it contains* signed literals*, $s_1{:}p, \ldots, s_n{:}p$, such that $\cap_{i=1}^{n} S_i = \varnothing$; (iii) all the formulas in the branch have been expanded and, for every variable p, it contains signed literals, $s_1{:}p, \ldots, s_n{:}p$, such that $\cap_{i=1}^{n} S_i = \{V(p)\}$.*
4. *A tableau T is called V-*closed *if every branch is V-closed.*

Adding literals of the form $\{01,11\}{:}p$, $\{02,12,22\}{:}p$ or $\{00\}{:}p$, depending on the initial tableau, requires that models be evaluated in a particular form; specifically we force models derived from the tableau to be less than V. Nevertheless, we know that one model will always be found, V itself, and therefore we include one more condition on closure: a branch closes if V. is the only model generated.

Theorem 7. *Let V be a total model of φ. V is a partial equilibrium model of Π if and only if there exists a V-closed tableau for φ.*

In the figure 5 we show that, for the previous example, the model σ_9 is a partial equilibrium model; observe that the leftmost branch closes because V is the only model

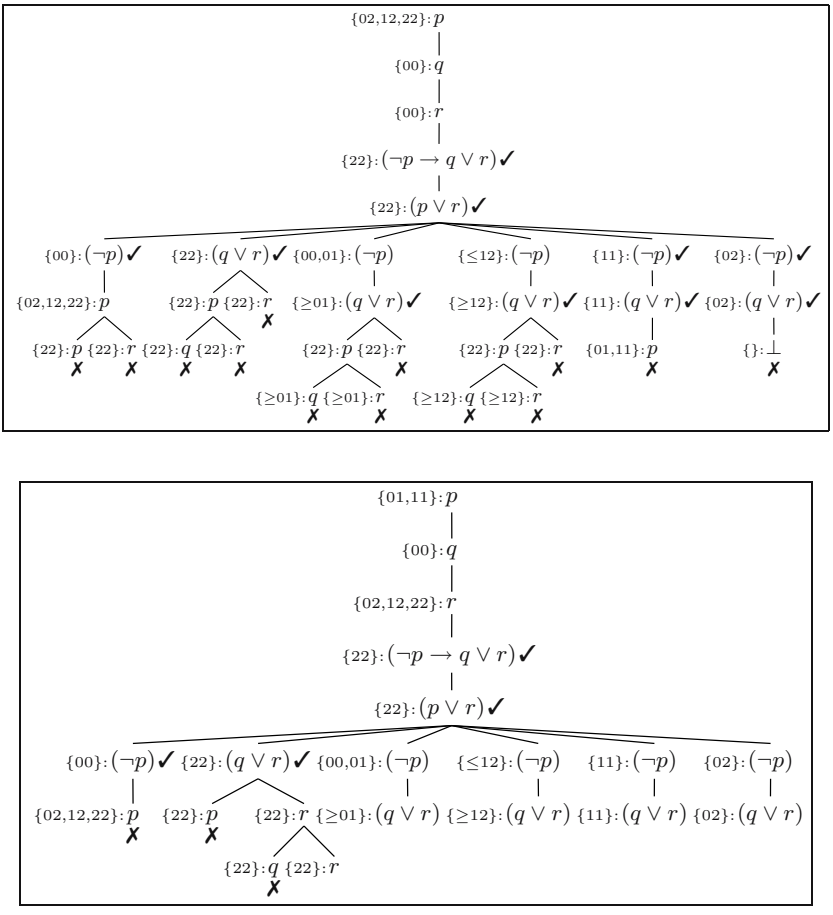


Fig. 5.

generated, while all other branches close due to inconsistencies provoked by the three signed literals added to the initial tableau. In the second tableau in the same figure we check that the model σ_{12} is not a partial equilibrium model.

5 A Splitting Theorem for PEL

The previous tableau calculus offers a general method for satisfiability testing in HT^2 and PEL, given any arbitrary theory. When we restrict the syntax to (some class of) logic programs, we usually expect, however, that simpler computation methods can be applied. Consider for instance the case of disjunctive logic programs. As shown in [2], PEL also coincides with p-stable models for this syntactic class. Maintaining the same minimisation criterion, we may easily get that a disjunctive program yields several well-founded models (even no well-founded model at all), and the typical incremental algorithm for computing WFS for normal programs is not applicable. However, it is

still possible to apply a form of incremental reasoning if we can divide or "split" the program into blocks without cyclic dependences among them. As an example, consider the simple program $\Pi_0 = \{p \vee q\}$ which yields two p-stable models (also well-founded), making p true and q false in one case, and vice versa. Now, assume we have the enlarged program $\Pi_1 = \Pi_0 \cup \{\neg r \wedge p \rightarrow r, q \wedge \neg p \rightarrow s, \neg s \rightarrow s\}$. It seems natural to use this second set of formulas to compute atoms r and s, once p and q are still fixed by the rule in Π_0. This technique is called "splitting" and was first introduced in [5] for the case of stable models. We now establish a similar result for PEL in the more general syntactic case where theories are sets of implications.

Given a pair $\mathbf{T} = (T, T')$ and a set of atoms U, we denote $\mathbf{T}|_U = (T \cap U, T' \cap U)$. We apply a similar notation for theories too. If Π is some theory in language $\mathfrak{L}(V)$, and $U \subseteq V$, then we write $\Pi|_U$ to stand for set of formulas $\Pi \cap \mathfrak{L}(U)$. We respectively call *bottom* and *top* to the subtheories $\Pi|_U$ and $\Pi \backslash \Pi|_U$.

Definition 8 (Splitting set). *Given a set of implications Π on signature V, a subset $U \subseteq V$ is called a* splitting set *for Π if for all $(\varphi \rightarrow \psi) \in \Pi \backslash \Pi|_U$, $\psi \in \mathfrak{L}(V \backslash U)$.* □

Theorem 8 (Splitting theorem). *Let Π be a set of implications, U a splitting set for Π and $\mathbf{T}$ a pair (T, T') of sets of atoms $T \subseteq T'$. Then $\mathbf{T} \approx \Pi$ iff both* (i) $\mathbf{T}|_U \approx \Pi|_U$ *and* (ii) $\mathbf{T} \approx \Pi'$, *being* $\Pi' := (\Pi \backslash \Pi|_U)$

$$\cup\ (T \cap U) \tag{3}$$

$$\cup\ \{\neg p \mid p \in U \backslash T'\} \tag{4}$$

$$\cup\ \{p \leftrightarrow \mathbf{u} \mid p \in (T' \backslash T) \cap U\} \tag{5}$$

The previous theorem is completed with the following result. Let us denote by $\Pi[\varphi/p]$ the replacement in theory Π of any occurrence of atom p by the formula φ.

Theorem 9 (Replacement theorem). *For any theory Π and any model $\mathcal{M}$:*

(i) $\mathcal{M} \models \Pi \cup \{p\}$ *iff* $\mathcal{M} \models \Pi[\top/p] \cup \{p\}$
(ii) $\mathcal{M} \models \Pi \cup \{\neg p\}$ *iff* $\mathcal{M} \models \Pi[\bot/p] \cup \{\neg p\}$
(iii) $\mathcal{M} \models \Pi \cup \{p \leftrightarrow \mathbf{u}\}$ *iff* $\mathcal{M} \models \Pi[\mathbf{u}/p] \cup \{p \leftrightarrow \mathbf{u}\}$

Returning to the example program Π_1, $U = \{p, q\}$ is a splitting set dividing Π_1 into the bottom Π_0 and the top $\Pi_1 \backslash \Pi_0$. As we saw, Π_0 has two p-equilibrium models: $\mathbf{T}_1 = (\{p\}, \{p\})$ and $\mathbf{T}_2 = (\{q\}, \{q\})$. Now, fixing $\mathbf{T}_1$, we consider the theory $\Pi' = \Pi_1 \backslash \Pi_0 \cup \{p\} \cup \{\neg q\}$ which, by the replacement theorem, is equivalent to $\{\neg r \wedge \top \rightarrow r, \bot \wedge \neg\top \rightarrow s, \neg s \rightarrow s, p, \neg q\}$. After some trivial simplifications, this amounts to $\{\neg r \rightarrow r, \neg s \rightarrow s, p, \neg q\}$ whose unique p-equilibrium model is defined by $\mathbf{T}_3 = (\{p\}, \{p, r, s\})$. Following similar steps, when fixing $\mathbf{T}_2$ we finally get the program $\{s, \neg s \rightarrow s, q, \neg p\}$ with the only p-equilibrium model $\mathbf{T}_4 = (\{q, s\}, \{q, s\})$.

6 Concluding Remarks

Partial equilibrium logic (PEL) provides a foundation and generalisation of the p-stable semantics of logic programs and hence is arguably also a suitable framework for studying the well-founded semantics of programs. In this paper we have extended previous

results on PEL by further examining its underlying logics HT^2 and HT^*, and presenting tableaux proof systems for HT^2, HT^* and for PEL itself. As a contribution to the computation of PEL in the case of disjunctive and nested logic programs, we have shown how to apply the splitting method of [5, 3]. Further optimisation of these computational techniques is a topic for future work.

References

1. P. Cabalar, S. Odintsov & D. Pearce. Logical Foundations of Well-Founded Semantics in *Proceedings KR 2006*, AAAI, pp. 25-35.
2. P. Cabalar, S. Odintsov, D. Pearce & A. Valverde. Analysing and Extending Well-Founded and Partial Stable Semantics using Partial Equilibrium Logic. in *Proceedings ICLP 06*, Springer LNAI, to appear.
3. S. T. Erdogan & V. Lifschitz. Definitions in Answer Set Programming. V. Lifschitz & I. Niemela (eds), *Proc. ICLP 04*, Springer, LNAI 2923, 2004, 114-126.
4. R. Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science*. Oxford University Press, 1994.
5. V. Lifschitz & H. Turner. Splitting a Logic Program. in P. van Hentenryck (ed), *Proceedings ICLP 94*, MIT Press, 1994, 23-37.
6. V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, October 2001.
7. V. Lifschitz, L.R. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):369–389, 1999.
8. D. Pearce. A new logical characterisation of stable models and answer sets. In *Proc. of NMELP 96*, LNCS 1216, pp. 57–70. Springer, 1997.
9. D. Pearce. Equilibrium Logic. *Ann. Math & Artificial Int.*, 2006, to appear.
10. D. Pearce, I.P. de Guzmán, and A. Valverde. A tableau calculus for equilibrium entailment. In *Proc. of TABLEAUX 2000*, LNAI 1847, pp. 352–367. Springer, 2000.
11. D. Pearce and A. Valverde. Uniform equivalence for equilibrium logic and logic programs. In *Proc. of LPNMR'04*, LNAI 2923, pp. 194–206. Springer, 2004.
12. Przymusinski, T. Stable semantics for disjunctive programs. *New Generation Computing* 9 (1991), 401-424.
13. Przymusinski, T. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence* 12:141–187, 1994.
14. R. Routley and V. Routley. The Semantics of First Degree Entailment. *Noûs*, 6, 335–359, 1972.
15. D. Vakarelov. Notes on constructive logic with strong negation. *Studia Logica*, 36: 89-107, 1977.
16. A. van Gelder, K.A. Ross, and J.S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *JACM*, 38(3):620–650, 1991

Decidable Fragments of Logic Programming with Value Invention

Francesco Calimeri, Susanna Cozza, and Giovambattista Ianni

Dipartimento di Matematica, Università della Calabria, I-87036 Rende (CS), Italy
{calimeri, cozza, ianni}@mat.unical.it

Abstract. The issue of value invention in logic programming embraces many scenarios, such as logic programming with function symbols, object oriented logic languages, inter-operability with external sources of knowledge, set unification. This paper introduces a framework embedding value invention in a general context. The class of programs having a suitable (but, in general, not decidable) 'finite grounding property' is identified, and the class of 'value invention restricted' programs is introduced. Value invention restricted programs have the finite grounding property and can be decided in polynomial time. They are, in a sense, the broadest polynomially decidable class having this property, whenever no assumption can be made about the nature of invented values (while this latter is the case in the specific literature about logic programming with function symbols). Relationships with existing formalisms are eventually discussed; in particular, value invention restricted programs subsume ω-restricted programs and are incomparable with finitary programs.

1 Introduction

The notion of 'value invention' has been formerly adopted in the database field (see e.g. [1,2]) for denoting those mechanisms aimed at allowing to introduce new domain elements in a logic based query language. Indeed, applications of logic programming often need to deal with a universe of symbols which is not a priori known. We can divide these demands in two main categories: *(i)* 'Constructivist' demands: the majority of logic programming languages has indeed the inherent possibility to build new symbols from pre-existing ones, e.g. by means of traditional constructs like functional terms. Manipulating and creating complex data structures other than simple constant symbols, such as sets, lists, is also a source of value invention. Also, controlled value invention constructs have been proposed in order to deal with the creation of new object identifiers in object oriented deductive databases [3]. *(ii)* 'Externalist' demands: in this setting, non-predictable external sources of knowledge have to be dealt with. For instance, in the Semantic Web area, rule based languages must explicitly embrace the case where ontologies and the universe of individuals is external and not a priori known [4], or is explicitly assumed to be open [5].

Whatever popular semantics is chosen for a rule based logic program (*well-founded, answer set, first order*, etc.), both of the above settings are a source of undecidability difficult to cope with.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 95–110, 2006.

Top down solvers (such as SLD solvers), do not usually address this issue, and leave to the programmer the burden of ensuring termination. Also, the programmer needs a good knowledge of the evaluation strategy implemented in her adopted system, since termination is often algorithm dependent. *Bottom up solvers* (such as DLV or Smodels for the Answer Set Semantics [6,7]), and in general, languages derived from Datalog, are instead conceived for ensuring algorithm independent decidability and full declarativity.

To this aim, the implementation of such languages relies on the explicit choice of computing a ground version of a given program. In a context where value invention is explicitly allowed, grounding a program against an infinite set of symbols leads to an infinite ground program which cannot be built in practice.

The paper adopts the notion of `VI` programs, which are logic programs enriched with the notion of *external predicate* [8]. External predicates model the mechanism of value invention by taking input from a given set of values and returning (possibly newly invented) values. These latter are computed by means of an associated evaluation function (called *oracle*).

In [8] we proved that, although assuming as decidable the external functions defining oracles, the consistency check of `VI` programs is, in general, undecidable.

Thus, it is important to investigate on nontrivial sub-classes of decidable programs. This problem is not addressed satisfactorily in the above paper, which is mainly focused on the operational and declarative properties of the framework and its technical realizability. Indeed, a very strict safety condition for granting decidability of `VI` programs is therein given.

The contributions of the paper are overviewed next:

– We introduce a *safety* condition defining the class of 'value invention restricted' (`VI`-restricted, in the following) programs. This class enjoys the *finite grounding property*, characterizing those programs that can be computed with a finite ground program. Decidability of consistency checking is thus ensured (Section 4).
– The `VI`-restrictedness condition is less restraining than previously introduced syntactic restrictions (such as ω-restricted programs [9] or semi-safe programs [8]). The programmer is thus relieved from the burden of introducing explicit syntactic modifications. However, `VI`-restrictedness can be checked in time polynomial in the size of the non-ground program (Section 5).
– The above condition is generic: no assumption is made on the structure of new invented symbols. Indeed, `VI` programs embed settings such as programs with function symbols, programs with sets (in general logic languages with a generalized notion of unification), or with external constructs (Section 6).
– `VI`-restricted programs subsume the class of ω-restricted programs [9]. Finitary programs [10], a class of programs with answer set semantics and function symbols, are not directly comparable with `VI`-restricted programs. Also, our former definition of semi-safe programs [8] is subsumed (Section 7).
– Our framework relies on the traditional notion of ground program. Thus, results about `VI`-restricted programs can be be adapted to semantics other than Answer Set Programming, such as the Well-Founded Semantics.

2 Motivating Example

The Friend of a Friend (FOAF) [11] project is about creating a Web of machine-readable homepages describing people, the links between them and the things they create and do. It is an RDF/XML Semantic Web vocabulary. Each person P stores its FOAF ontology portion at some url U.

In order to reason on this vocabulary, a rule based logic language would need some special construct for importing this external knowledge. The aim of this language is anyway to keep decidability and declarativity. So it is important not to rely on an operational semantics for the language. In this spirit, [4] introduces a form of *external predicates*, very similar to ours.

Imagine we want to perform the transitive closure of the relation of knowledge among people, starting from the homepage of a given person. Let's suppose to have an external predicate called "*#rdf*" which allows us to access a FOAF ontology located at URL:

$$\#rdf(URL, Object_1, Relation, Object_2).$$

We first collect a set of homepages. In order to avoid wrong information we can accept only a restricted subset of somehow *trusted* urls. Then we simply encode the transitive closure as usual, exploiting the knowledge provided by the collected pages. Let the starting homepage be "myurl"; thus, the following program implements what described above.

$$trusted(X,U) \leftarrow \#rdf(\text{“}myurl\text{”}, X, \text{“}trusts\text{”}, U). \quad (1)$$
$$url(X,U) \leftarrow \#rdf(\text{“}myurl\text{”}, X, \text{“}seealso\text{”}, U),\ trusted(X,U). \quad (2)$$
$$url(X,U) \leftarrow url(\_, U1), \#rdf(U1, X, \text{“}seealso\text{”}, U),\ trusted(X,U). \quad (3)$$
$$connected(X,Y) \leftarrow url(X,U),\ \#rdf(U, X, \text{“}knows\text{”}, Y). \quad (4)$$
$$connected(X,Y) \leftarrow connected(X,Z),\ url(Z,U),\ \#rdf(U, Z, \text{“}knows\text{”}, Y). \quad (5)$$

The above program has two sources of new values: trusted urls, and persons. For instance, in particular the fifth rule may induce a loop, leading to the invention of an infinite number of new symbols. The above program is anyway VI-restricted and can be solved over a finite ground version of it. Intuitively, the number of URLs is finite. Although not explicitly bounded, new persons (coming from the value of Y in the fifth rule) can be extracted only from a finite set of URLs. Observe that rule 1 *invents* new values, but these do not ever propagate through a loop involving an external atom, while this is the case of the Y variable in the fifth rule. The intuition of VI-restricted programs is to investigate how new information propagates in a program, and whether it is bounded in some way. Note that a programmer is not explicitly forced (in order to ensure decidability) to bound variables explicitly such as in this modified version of the fifth rule: { $connected(X,Y) \leftarrow known(Y), connected(X,Z), url(Z,U), \#rdf(U, Z, \text{“}knows\text{”}, Y)$. }.

3 Preliminaries

In this section we briefly recall some notions which we introduced in [8]. Our framework coincides basically with Answer Set Programming, extended with the notion of *external atom*.

Let $\mathcal{U}$, $\mathcal{X}$, $\mathcal{E}$ and $\mathcal{P}$ be mutually disjoint sets whose elements are called *constant names*, *variable names*, *external predicate names*, and *ordinary predicate names*, respectively. Unless explicitly specified, elements from $\mathcal{X}$ (resp., $\mathcal{U}$) are denoted with first letter in upper case (resp., lower case); elements from $\mathcal{E}$ are usually prefixed with '#'. $\mathcal{U}$ constitutes the default *Herbrand Universe*. We assume that any constant appearing in a program or generated by external computation is taken from $\mathcal{U}$, which is possibly infinite[1].

Elements from $\mathcal{U} \cup \mathcal{X}$ are called *terms*. An *atom* is a structure $p(t_1, \ldots, t_n)$, where $t_1, \ldots, t_n$ are terms and $p \in \mathcal{P} \cup \mathcal{E}$; $n \geq 0$ is the *arity* of the atom. p is *the predicate name*. The atom is *ordinary*, if $p \in \mathcal{P}$, otherwise we call it *external atom*. A list of terms $t_1, \ldots, t_n$ is succinctly represented by $\overline{t}$. A positive *literal* is an atom, whereas a *negative literal* is $\texttt{not}\, a$ where a is an atom.

Given a predicate p, $p[i]$ is its i-th argument. A *rule* r is of the form

$$\alpha_1 \vee \cdots \vee \alpha_k \leftarrow \beta_1, \ldots, \beta_n, \texttt{not}\, \beta_{n+1}, \ldots, \texttt{not}\, \beta_m, \tag{6}$$

where $m \geq 0, k \geq 1, \alpha_1, \ldots, \alpha_k$, are ordinary atoms, and $\beta_1, \ldots, \beta_m$ are (ordinary or external) atoms. We define $H(r) = \{\alpha_1, \ldots, \alpha_k\}$ and $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{\beta_1, \ldots, \beta_n\}$ and $B^-(r) = \{\beta_{n+1}, \ldots, \beta_m\}$. $E(r)$ is the set of external atoms of r. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*, and if $B(r) = \emptyset$ and $H(r) \neq \emptyset$, then r is a *fact*; r is *ordinary*, if it contains only ordinary atoms. A *VI program* is a finite set P of rules; it is *ordinary*, if all rules are ordinary. We assume P has no constraints[2], only ground facts, and that each rule is *safe* with respect to negation, i.e. for each rule r, each variable appearing in some negated atom $a \in B^-(r)$ or in the head, appears also in some positive atom $b \in B^+(r)$. Given a set of atoms A and a predicate p, with small abuse of notation we say that $p \in A$ if there is some atom in A with predicate name p. An atom having p as predicate name is usually referred as a_p.

We denote as $Attr(P)$ the set of all arguments of all the predicates appearing in the program P. The *dependency graph* $G(P)$ of P is built in the standard way.

We give the semantics by generalizing the answer-set semantics [12].

In the sequel, we will assume P as a VI program. The *Herbrand base* of P with respect to $\mathcal{U}$, denoted $HB_{\mathcal{U}}(P)$, is the set of all possible ground versions of ordinary atoms and external atoms occurring in P obtained by replacing variables with constants from $\mathcal{U}$. The grounding of a rule r, $grnd_{\mathcal{U}}(r)$, is defined accordingly, and the grounding of program P by $grnd_{\mathcal{U}}(P) = \bigcup_{r \in P} grnd_{\mathcal{U}}(r)$. Note that this ground program can be of infinite size.

An *interpretation* I for P is a pair $\langle S, F \rangle$ where:

– $S \subseteq HB_{\mathcal{U}}(P)$ contains only ordinary atoms; I (or, by small abuse of notation, S) is a *model* of ordinary atom $a \in HB_{\mathcal{U}}(P)$, denoted $I \models a$ ($S \models a$), if $a \in S$.

[1] Also, we assume that constants are encoded using some finite alphabet Σ, i.e. they are finite elements of Σ^*.

[2] Under Answer Set Programming semantics, a constraint $\leftarrow B(r)$ can be easily simulated through the introduction of a corresponding standard rule fail $\leftarrow$ B(r), not fail, where fail is a fresh predicate not occurring elsewhere in the program.

– F is a mapping associating with every external predicate name $\#\mathrm{e} \in \mathcal{E}$, a decidable n-ary function (which we call *oracle*) $F(\#\mathrm{e})$ assigning each tuple $(x_1, \ldots, x_n)$ either 0 or 1, where n is the fixed arity of $\#\mathrm{e}$, and $x_i \in \mathcal{U}$. I (or, by small abuse of notation, F) is a *model* of a ground external atom $a = \#\mathrm{e}(x_1, \ldots, x_n)$, denoted $I \models a$ ($F \models a$), if $F(\#\mathrm{e})(x_1, \ldots, x_n) = 1$.

A positive literal is satisfied if its atom is satisfied, whereas a negated literal is satisfied if its corresponding atom is not satisfied.

Let r be a ground rule. We define:

i. $I \models H(r)$ iff there is some $a \in H(r)$ such that $I \models a$;

ii. $I \models B(r)$ iff $I \models a$ for each atom $a \in B^+(r)$ and $I \not\models a$ for each atom $a \in B^-(r)$;

iii. $I \models r$ (i.e., r is satisfied) iff $I \models H(r)$ whenever $I \models B(r)$.

We say that I is a *model* of a VI program P with respect to a universe $\mathcal{U}$, denoted $I \models_{\mathcal{U}} P$, iff $I \models r$ for all $r \in grnd_{\mathcal{U}}(P)$. For a fixed F, a model $M =< S, F >$ is minimal if there is no model $N =< T, F >$ such that $S \subset T$.

Given a general ground program P, its *Gelfond-Lifschitz reduct* [12] w.r.t. an interpretation I is the positive ground program P^I obtained from P by: *(i)* deleting all rules having a negated literal not satisfied by I; *(ii)* deleting all negated literals from the remaining rules. $I \subseteq HB_{\mathcal{U}}(P)$ is an *answer set* for a program P w.r.t. $\mathcal{U}$ iff I is a minimal model for the positive program $grnd_{\mathcal{U}}(P)^I$. Let $ans_{\mathcal{U}}(P)$ be the set of answer sets of $grnd_{\mathcal{U}}(P)$. We call P *F-satisfiable* if it has some answer set for a fixed function mapping F, i.e. if there is some interpretation $\langle S, F \rangle$ which is an answer set. We will assume in the following to deal with a fixed set F of functions mappings for external predicates. F-satisfiability is undecidable [8].

Given an external predicate name $\#\mathrm{p}$, of arity n and its oracle function $F(\#\mathrm{p})$, a *pattern* is a list of b's and u's, where a b represents a placeholder for a constant (or a bounded variable), and an u is a placeholder for a variable. Given a list of terms, the corresponding pattern is given by replacing each constant with a b, and each variable with a u. Positions where u appears are called *output* positions whereas those denoted with b are called *input* positions. For instance, the pattern related to the list of terms (X, a, Y) is (u, b, u).

Let *pat* be a pattern of length n having k placeholders b (*input positions*), and $n-k$ placeholders of u type (*output positions*). A *functional oracle* $F(\#\mathrm{p})[pat]$ for the pattern *pat*, associated with the external predicate $\#\mathrm{p}$, is a partial function taking k constant arguments from $\mathcal{U}$ and returning a finite relation of arity $n-k$, and such that $d_1, ..., d_{n-k} \in F(\#\mathrm{p})[pat](c_1, ..., c_k)$ iff $F(\#\mathrm{p})(h_1, \ldots, h_n) = 1$, where for each $i (1 \leq i \leq n)$, $h_i = c_j$ if the j-th b value occurs in position i in *pat*, otherwise $h_i = d_j$ if the j-th u value occurs in position i in *pat*.

An external predicate $\#\mathrm{p}$ might be associated to one or more functional oracles 'consistent' with the originating 2-valued one. For instance, consider a binary external predicate #sqr, intuitively associating a natural number to its square value. We can have two functional oracles, $\mathrm{F}(\#\mathrm{sqr})[b, u]$ and $\mathrm{F}(\#\mathrm{sqr})[u, b]$. The

two functional oracles are such that, e.g. F(#sqr)$[b, u](3) = 9$ and F(#sqr)$[u, b]$ $(16) = 4$, consistently with the fact that F(#sqr)$(3, 9) =$ F(#sqr)$(4, 16) = 1$[3].

In the sequel, given an external predicate #e, we will assume that it comes equipped with its oracle $F(\#\text{e})$ (called also *base oracle*) and one functional oracle $F(\#\text{e})[pat_{\#\text{e}}]$, having pattern $pat_{\#\text{e}}$ [4].

We recall now a first condition of safety, which unfortunately does not guarantee finiteness and decidability, but will be exploited in the next Section. Given a rule r, a variable X is *weakly safe in* r if either *(i)* X is safe (i.e. it appears in some positive atom of $B^+(r) \setminus E(r)$; or *(ii)* X appears in some external atom $\#\text{e}(\overline{T}) \in E(r)$, the functional oracle of #e is $F(\#\text{e})[pat]$, X appears in output position with respect to pat and each variable Y appearing in input position in the same atom is weakly safe. A weakly safe variable X is *free* if it appears in $B^+(r)$ only in output position of some external atom. A rule r is weakly safe if each variable X appearing in some atom $a \in B(r)$ is weakly safe. A program P is weakly safe if each rule $r \in P$ is weakly safe.

Example 1. Assume that #sqr is associated to the functional oracle $F(\#\text{sqr})[b, u]$ defined above. The program { square(Y) ← number(X), #sqr(X,Y) } is weakly safe (intuitively the value of Y can be computed once the value of X is known). The same rule is not weakly safe if we consider the functional oracle $F(\#\text{sqr})[u, b]$. □

Definition 1. Let $A = \langle I, F \rangle$ an interpretation. We call $ins(r, A)$ the set of ground instances r_θ of r for which $A \models B^+(r_\theta)$, and such that $A \models E(r_\theta)$. □

Proposition 1. [8] Given an interpretation A and a weakly safe rule r, $ins(r, A)$ is finite. □

Weakly safe rules have the important property of producing a finite set of *relevant* ground instances provided that we know a priori the domain of positive ordinary body atoms. Although desirable, weak safety is intuitively not sufficient in order to guarantee finiteness of answer sets and decidability. For instance, it is easy to see that the program { square(2) ←; square(Y) ← square(X), #sqr(X,Y); } has answer set $\{\text{square}(2), \text{square}(4), \dots\}$.

4 Decidable vi Programs

The introduction of new symbols in a logic program by means of external atoms is a clear source of undecidability. As illustrated in Section 6, value invention is nonetheless desirable in a variety of contexts.

Our approach investigates which programs can be solved by means of a finite ground program having a finite set of models of finite size. This class of programs

[3] Unlike this example, note that in the general case functional oracles might return a set of tuples and are not restricted to single output values.

[4] In [8] we address explicitly the issue of external predicates equipped with multiple functional oracles.

(having the *finite grounding property*) is unluckily not recognizable in finite time. We assume to deal with functional oracles that might have an infinite co-domain. Nonetheless, we will assume also to deal with weakly safe programs and with functional oracles associating to each fixed combination of the values in input always a finite number of combination of values in output.

Definition 2. A class of VI programs $\mathcal{C}$ has the *finite grounding* property if, for each $P \in \mathcal{C}$ there exists a finite set $U \subset \mathcal{U}$ such that $ans_U(P) = ans_{\mathcal{U}}(P)$. □

Theorem 1. Recognizing the class of all the VI programs having the finite grounding property is undecidable.

Proof. (Sketch). Positive logic programs with function symbols can simulate Turing machines. Also weakly safe VI programs can mimic (see section 6 and [8]) programs with function symbols. Given a Turing machine $\mathcal{T}$ and an input string x we can thus build a suitable VI program $P_{\mathcal{T},x}$ encoding $\mathcal{T}$ and x. $\mathcal{T}(x)$ terminates iff $P_{\mathcal{T},x}$ has the finite grounding property. Indeed, if $\mathcal{T}(x)$ terminates, the content of U can be inferred from the finite number of transitions of $\mathcal{T}(x)$. Viceversa, if U is given, the evolution of $\mathcal{T}(x)$ until its termination can be mimicked by looking at the answer sets of $grnd_U(P_{\mathcal{T},x})$. □

4.1 VI-Restricted Programs

The intuition leading to our definition of VI-restrictedness, is based on the idea of controlled propagation of new values throughout a given program. Assume the following VI program is given ($\#b$ has a functional oracle with pattern $[b, u]$): { a(k,c) ←; p(X,Y) ← a(X,Y); p(X,Y) ← s(X,Y), a(Z,Y); s(X,Y) ← p(Z,X), #b(X,Y).}. The last rule of the program generates new symbols by means of the Y variable, which appears in the second attribute of $s(X, Y)$ and in output position of $\#b(X, Y)$. This situation is per se not a problem, but we observe that values of $s[2]$ are propagated to $p[2]$ by means of the last but one rule, and $p[2]$ feeds input values to $\#b(X, Y)$ in the last rule. This occurs by means of the binding given by the X variable. The number of ground instances to be considered for the above program is thus in principle infinite, due to the presence of this kind of cycles between attributes.

We introduce the notion of *dangerous* rule for those rules that propagate new values in recursive cycles, and of *dangerous* attributes for those attributes (e.g. $s[2]$) that carry new information in a cycle.

Actually, the above program can be reconducted to an equivalent finite ground program: we can observe that $p[2]$ takes values from the second and third rule above. In both cases, values are given by bindings to $a[2]$ which has, clearly, a finite domain. So, the number of input values to #b(X,Y) is bounded as well. In some sense, the 'poisoning' effect of the last (dangerous) rule, is canceled by the fact that $p[2]$ limits the number of symbols that can be created.

In order to formalize this type of scenarios we introduce the notion of *savior* and *blocked* attributes. $p[2]$ is *savior* since all the rules where p appears in the head

can be proven to bring values to $p[2]$ from blocked attributes, or from constant values, or from other savior attributes. Also, $s[2]$ is dangerous but *blocked* with respect to the last rule, because of the indirect binding with $p[2]$, which is savior. Note that an attribute is considered blocked with respect to a given rule. Indeed, $s[2]$ might not be blocked in other rules where s appears in the head.

We define an *attribute dependency graph* useful to track how new symbols propagate from an attribute to another by means of bindings of equal variables.

Definition 3. The *attribute dependency graph* $AG(P)$ associated to a weakly safe program P is defined as follows. For each predicate $p \in P$ of arity n, there is a node for each predicate attribute $p[i](1 \leq i \leq n)$, and, looking at each rule $r \in P$, there are the following edges:

– $(q[j], p[i])$, if p appears in some atom $a_p \in H(r)$, q appears in some atom $a_q \in B^+(r) \setminus E(r)$ and $q[j]$ and $p[i]$ share the same variable.
– $(q[j], \#p[i])$, if q appears in some atom $a_q \in B^+(r) \setminus E(r)$, $\#p$ appears in some atom $a_{\#p} \in E(r)$, $q[j]$ and $\#p[i]$ share the same variable, and i is an input position for the functional oracle of $\#p$;
– $(\#q[j], \#p[i])$, if $\#q$ appears in some atom $a_{\#q} \in E(r)$, $\#p$ in some $a_{\#p} \in E(r)$, $\#q[j]$ and $\#p[i]$ share the same variable, j is an output position for the functional oracle of $\#q$, i is an input position for the functional oracle of $\#p$;
– $(\#p[j], \#p[i])$, if $\#p$ appears in some atom $a_{\#p} \in E(r)$, $\#p[j]$ and $\#p[i]$ both have a variable, j is an input position for the functional oracle of $\#p$ and i is an output position for the functional oracle of $\#p$;
– $(\#q[j], p[i])$, if p appears in some atom $a_p \in H(r)$, $\#q$ appears in some atom $a_{\#q} \in E(r)$ and $\#q[j]$ and $p[i]$ share the same variable, and j is an output position for the functional oracle of $\#q$; □

Example 2. The *attribute dependency graph* induced by the *first three* rules of the motivating example in Section 2 is depicted in Figure 1. □

Definition 4. It is given a weakly safe program P. The following definitions are given (all examples refer to the Motivating Example, Section 2, and we assume #rdf has functional oracle with pattern $[b, u, u, u]$):

– A rule r *poisons* an attribute $p[i]$ if some atom $a_p \in H(r)$ has a free variable X in position i. $p[i]$ is said to be *poisoned* by r. For instance, *connected*[2] is poisoned by rule (5).
– A rule r is *dangerous* if it poisons an attribute $p[i]$ $(p \in H(r))$ appearing in a cycle in $AG(P)$. Also, we say that $p[i]$ is *dangerous*. For instance, rule (5) is dangerous since connected[2] is poisoned and appears in a cycle.
– Given a dangerous rule r, a dangerous attribute $p[i]$ (bounded in $H(r)$ to a variable name X), is *blocked* in r if for each atom $a_{\#e} \in E(r)$ where X appears in output position, each variable Y appearing in input position in the same atom is *savior*. Y is *savior* if it appears in some predicate $q \in B^+(r)$ in position i, and $q[i]$ is *savior*.

– An attribute $p[i]$ is *savior* if at least one of the following conditions holds for each rule $r \in P$ where $p \in H(r)$.

- $p[i]$ is bound to a ground value in $H(r)$;
- there is some savior attribute $q[j]$, $q \in B^+(r)$ and $p[i]$ and $q[j]$ are bound to the same variable in r;
- $p[i]$ is blocked in r.

For instance, the dangerous attribute *connected*[2] of rule (5) is blocked since the input variable U is savior (indeed it appears in $url[2]$).
– A rule is VI-restricted if all its dangerous attributes are blocked. P is said to be *VI-restricted* if all its *dangerous* rules are *VI-restricted*. □

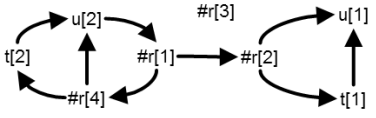


Fig. 1. Attributes Dependency Graph *(Predicate names are shortened to the first letter)*

Theorem 2. VI-restricted programs have the finite grounding property.

Proof. (Sketch). Given a VI-restricted program P, we show how to compute a finite ground program gr_P such that $ans_{\mathcal{U}}(P) = ans_U(gr_P)$, where U is the set of constants appearing in gr_P.

Let's call A the set of *active ground atoms*, initially containing all atoms appearing in some fact of P. gr_P can be constructed by an algorithm $\mathcal{A}$ that repeatedly updates gr_P (initially empty) with the output of $ins(r, I)$ (Definition 1) for each rule $r \in P$, where $I = \langle A, F \rangle$; all atoms belonging to the head of some rule appearing in gr_P are then added to A. The iterative process stops when A is not updated anymore. That is, gr_P is the least fixed point of the operator

$$T_P(Q) = \{\bigcup_{r \in P} ins(r, I) \mid I = \langle A, F \rangle, \text{ and } A = atoms(Q)\}$$

where $atoms(Q)$ is the set of ordinary atoms appearing in Q. $T_P^\infty(\emptyset)$ is finite in case P is VI-restricted. Indeed, gr_P might not cease to grow only in case an infinite number of new constants is generated by the presence of external atoms. This may happen only because of some *dangerous* rule having some 'poisoned' attributes. However, in a *VI-restricted* program all poisoned attributes are *blocked* in dangerous rules where they appear, i.e. they depend from savior attributes. It can be shown that, for a given savior attribute $p[i]$, the number of symbols that appear in position i in an atom a_p such that $a_p \in T_P^\infty(\emptyset)$ is finite. This means that only a finite number of calls to functional oracles is made by $\mathcal{A}$, each of which producing a finite output.

Because of the way it has been constructed, it is easy to see that the set $A = atoms(gr_P)$ is a splitting set [13], for $grnd_{\mathcal{U}}(P)$. Based on this, it is possible to observe that no atom $a \notin A$ can be in any answer set, and to conclude that $ans_U(P) = ans_{\mathcal{U}}(P)$, where U is the set of constants appearing in A. □

5 Recognizing VI-Restricted Programs

An algorithm recognizing VI-restricted programs is depicted in Figure 2. The idea is to iterate through all *dangerous rules* trying to prove that all of them are *VI-restricted*. In order to prove VI-restriction for rules, we incrementally build the set of all *savior attributes*; this set is initially filled with all attributes which can be proven to be savior (i.e. they do not depend from any dangerous attribute). This set is updated with a further attribute $p[i]$ as soon it is proved that each dangerous attribute which $p[i]$ depends on is blocked. The set RTBC of rules to be checked initially consists of all dangerous rules, then the rules which are proven to be VI-restricted are gradually removed from RTBC. If an iteration ends and nothing new can be proved the algorithm stops. The program is VI-restricted if RTBC is empty at the last iteration.

The algorithm execution takes polynomial time in the size of a program P: let m be the total number of rules in P, n the number of different predicates, k the maximum number of attributes over all predicates, and l the maximum number of atoms in a single rule. $O(n * k)$ is an upper bound to the total number of different attributes, while $O(l * k)$ is an upper bound to the number of variables in a rule. A naive implementation of the *isBlocked* function has complexity $O(n * l * k^2)$. The *recognizer* function (Figure 2) iterates $O(n * k)$ times over an inner cycle which performs at most $O(m * k * l)$ steps: each inner step iterates over all rules in $RTBC$, which are at most m; and for each rule all free variables must be checked (this requires $O(k * l)$ checks, in the worst case).

6 Modeling Semantic Extensions by VI Programs

Several semantic extensions contemplating value invention can be mapped to VI programs. We show next how programs with function symbols and with sets can be translated to weakly safe VI programs. When the resulting translation is VI-restricted as well, these semantic extension can be thus evaluated by an answer set solver extended with external predicates.

Functional terms. We consider rule based languages allowing functional terms whose variables appearing in the head appear also in the positive body. A functional term is either a constant, a variable, or $f(X_1, \ldots, X_n)$, where f is a function symbol and $X_1, \ldots, X_n$ are terms.

For each natural number k, we introduce two external predicates $\#function_k$ and $\#function'_k$ of arity $k+2$; they are such that $f_{\#function_k}(F, f, X_1, \ldots, X_k) = f_{\#function'_k}(F, f, X_1, \ldots, X_k) = true$ if and only if the term F is $f(X_1, \ldots, X_k)$. Each $\#function_k$ ($\#function'_k$) predicate is associated to a functional oracle $F(\#function_k)[u, b, b, \ldots, b]$ ($F(\#function'_k)[b, u, u, \ldots, u]$, respectively).

The two families of external predicates are respectively intended in order to construct a functional term if all of its arguments are bounded ($\#function_k$ predicates) or if the whole functional term is grounded and we want to take its arguments ($\#function'_k$ predicates).

```
Bool Function recognizer ( var SA: Set{ Attr };
                               % SA is initialized with provable savior attributes
                               % (i.e. attributes that do not depend from dangerous attributes.
                           var NSA: Set{ pair⟨ Attr, Set{ Attr } ⟩ };
                               % NSA is initialized with attributes which cannot be proven to be
                               % savior, each of which is associated with the set of dangerous
                               % attributes that prevent them to be savior
                           var RTBC : Set{ Rule } ) % Set of dangerous rules to be checked.
    Bool NSA_Updated = true;
    While ( NSA_Updated ) do % Try to prove VI-restriction as far as some change occurs.
        NSA_Updated = false;
        For each Rule r ∈ RTBC do % free(r)=the set of free variables appearing in the rule r.
                Set{Var} varsTBC = free( r );
                Bool allBlocked = true;
                For each Var v ∈ varsTBC do
                    % isBlocked tells if v is blocked in r by means of attributes currently in SA.
                    If ( isBlocked( v, r, SA ) ) then
                        % headAttr returns reference to the head attribute of r containing v
                        Attr p[i] = headAttr( v, r );
                        % update processes the NSA set, deleting p[i] from each set S
                        % such that p[i] ∈ S and ⟨q[j], S⟩ ∈ NSA.
                        % Then each attribute q[j] such that ⟨q[j], S⟩ ∈ NSA
                        % and S = ∅ is moved from NSA to SA.
                        update( NSA, SA, p[i] );
                        % A change occurred, so we have to continue cycling.
                        NSA_Updated = true;
                    Else % At least one free variable can't be proved as blocked.
                        allBlocked = false;
                    EndIf
                EndFor
                If ( allBlocked ) then
                    RTBC.delete(r); %.The rule is VI-restricted, can be deleted from RTBC.
                EndIf
        EndFor
    EndWhile
    If ( RTBC == ∅ ) then
        Return true
    Else % Display the set of rules that can't be proved as VI-restricted.
        printINSAne( RTBC )
        Return false
    EndIf
EndFunction
```

Fig. 2. The VI-Restricted Recognizer Algorithm

Basically, this transformation flattens each rule $r \in P$ containing some functional term $t = f(X_1, \ldots, X_n)$, by replacing it with a fresh variable F, and adding an appropriate atom $\#function_k(F, X_1, \ldots, X_n)$ or $\#function'_k(F, X_1, \ldots, X_n)$ to the body of r. The transformation is continued until a functional term is still in r. We choose $\#function'_k$ if t appears in the body of r, whereas an atom using $\#function_k$ is used if t appears in the head of r.

Example 3. The rule { p(s(X)) ← a(X, f(Y, Z)). } contains two function symbols, s and f. The rewritten rule is { p(F1) ← a(X, F2), $\#function_1(F1, s, X)$, $\#function'_2(F2, f, Y, Z)$. } □

Proposition 2. Given a logic program with functional terms P, $\mathcal{F}(P)$ is the program obtained by applying the above transformation; it is weakly safe. Also, there is a 1-to-1 mapping between the answer sets of P and $ans_{\mathcal{U}}(\mathcal{F}(P))$. □

Set unification. The accommodation of sets in logic programming, often attempted, obliges to reconsider the classic notion of term unification to a generalized one. For instance, the set term $\{X, a, b, c\}$ can be grounded to $\{a, d, b, c\}$. It is possible to embody set constructors and set unification in the context of `VI` programs. Roughly speaking, a logic program with sets replaces the classical notion of term with the notion of *set term*. A set term is either a *(i)* classical term or, *(ii)* $\{X_1, \ldots, X_n\}$ where $X_1, \ldots, X_n$ are set terms, or *(iii)* $X \cup Y$ where X and Y are set terms. Two ground set terms $\{a_1, \ldots, a_n\}$ are equal if they contain the same set of ground terms. For space reasons, we only outline here a method, and refer the reader to [14] for a survey on sets in logic programming and on set unification methods and algorithms.

Remarking that the special symbol {} represents the empty set, the following set of external predicates are introduced: *(i)* A pair of external predicates $\#set_k$, $\#set'_k$ for each natural number k; each of them has $k+1$ arguments such that $f_{\#set_k}(X, Y_1, \ldots, Y_k) = f_{\#set'_k}(X, Y_1, \ldots, Y_k) =$ true if X is the set $\{Y_1, \ldots, Y_k\}$. $\#set_k$ has the functional oracle $F(\#set_k)[u, b, \ldots, b]$ while $\#set'_k$ has the functional oracle $F(\#set_k)[b, u, \ldots, u]$; *(ii)* Two ternary external predicate $\#union$ and $\#union'$; they are such that $f_{\#union}(X, Y, Z) = f_{\#union'}(X, Y, Z) = true$ either if $X = Y \cup Z$, or if X and Y are classical terms, $Z = \emptyset$ and $X = Y$. $\#union$ has the functional oracle $F(\#union)[u, b, b]$ while $\#union'$ has the functional oracle $F(\#union')[b, u, u]$.

A logic program with set terms P is replaced by an equivalent `VI` program by modifying each rule $r \in P$ this way:

– Replacing each set term $\{X_1, \ldots, X_n\}$ appearing in r with a fresh variable T, and adding in the body of r the external atom $\#set_n(T, X_1, \ldots, X_n)$ if the set term appears in the head of r , $\#set'_n(T, X_1, \ldots, X_n)$ otherwise;
– Replacing each set term $X \cup Y$ appearing in r with a fresh variable U, and adding in the body of r the external atom $\#union(U, X, Y)$ if the set term appears in the head of r, $\#union'(U, X, Y)$ otherwise. This and step 6 are applied to r until it contains any set term;
– If a variable X appears in r for m times ($m > 1$), then each occurrence of X is replaced with a fresh variable $X_i (1 \leq i \leq n)$, and for each pair $(X_i, X_j), 1 \leq i < j < m$ the atom $\#union(X_i, X_j, \{\})$ is added to r.

Example 4. Let's consider the rule: { p(X ∪ Y) ← a({a, X}), b({Y}). }; the analogous `VI` rule is: { p(S1) ← a(S2), b(S3), #union(S1, X1, Y1), $\#set'_2(S2, a, X2)$, $\#set'_1(S3, Y2)$, #union(X1,X2,{}), #union(Y1,Y2,{}). } □

Proposition 3. Given a logic program with set terms P, we call $\mathcal{S}(P)$ the `VI` program obtained applying the above transformation. $\mathcal{S}(P)$ is weakly safe. There is a 1-to-1 mapping between the answer sets of P and $ans_{\mathcal{U}}(\mathcal{S}(P))$. □

7 Relationships with Other Classes of Programs

ω-restricted programs. In the same spirit of this paper are ω-stratified programs [9], that allow function symbols under answer set semantics. The introduced restrictions aim at controlling the creation of new functional terms.

Definition 5. [9] An ω-stratification is a traditional stratification extended by the ω-stratum, which contains all predicates depending negatively on each other. ω is conventionally assumed to be uppermost layer of the program. A rule r is ω-restricted iff all variables appearing in r also occur in a positive body literal belonging to a *strictly* lower stratum than the head. A program P is ω-restricted iff all the rules are ω-restricted. □

ω-stratified programs have the finite grounding property: only a finite amount of functional terms can be created since each variable appearing in a rule's head must be bounded to a predicate belonging to a lower layer. `VI`-restricted programs do not introduce special restrictions for non-stratified cycles. Also, it is not necessary to bound each variable to a previous layer explicitly. The class of `VI`-restricted programs contains, in a sense, the class of ω-restricted ones.

Theorem 3. Given an ω-restricted program P, $\mathcal{F}(P)$ is `VI`-restricted.

Proof. We are given an ω-restricted program P. We observe that:

–Attributes belonging to predicates which are not in the ω-stratum can be proven to be savior: the relevant instantiation of these predicates is computable starting from the lowermost layer, and is finite.
–The rewritten rules in $\mathcal{F}(P)$ corresponding to function-free rules cannot be dangerous, since there is no value invention at all.
–Rules with functional terms are rewritten using external atoms; then, all variables occurring in these new external atoms already occur in the original rules, except fresh variables used for substituting functional terms (that we call FTRs, *functional term representations*). Thus, the variables appearing in the poisoned attributes must necessarily appear also in a predicate belonging to a strictly lower stratum than the head (ω-restrictedness). Let's consider an FTR appearing in an external atom $\#function'_k(F1, X_1, \ldots, X_k)$ in first position. If $F1$ is already bound to a positive atom, then there is no value invention; otherwise, it can be shown that all terms $X_1, \ldots, X_k$ are bound either to a positive atom or to another external atom in output position (see Section 6). As stated before, the attributes where $X_1, \ldots, X_k$ appear are savior, and so the FTR $F1$ as well. □

On the other hand, the opposite does not hold.

Theorem 4. It is possible to find non-ω-restricted programs whose transformation $\mathcal{F}$ outputs a `VI`-restricted program.

Proof. The program $P_{n\omega r} = \{p(f(X)) \leftarrow q(X), t(X); q(X) \leftarrow p(X); p(1); t(1)\}$ is not ω-restricted, while $\mathcal{F}(P_{n\omega r})$=$\{p(F1) \leftarrow q(X), t(X), \#function_2(F1, f, X); q(X) \leftarrow p(X); p(1); t(1)\}$ is `VI`-restricted. □

Finitary programs. Finitary programs allow function symbols under answer set semantics [10]. Although they don't have the finite grounding property, brave and cautious ground querying is decidable. A ground program P is finitary iff its

dependency graph $G(P)$ is such that *(i)* any atom p appearing as node in $G(P)$ depends only on a finite set of atoms (through head-body dependency), and *(ii)* $G(P)$ has only a finite number of cycles with an odd number of negated arcs.

Theorem 5. The class of finitary programs is not comparable with the class of `VI`-restricted programs.

Proof. (sketch) A program having rules with free variables is not finitary (eg. $p(X) \leftarrow q(X,Y)$): a ground instance $p(a)$ may depend on infinite ground instances of $q(X,Y)$ $e.g.(q(a,f(a)),q(a,f(f(a)))...)$. In general, the same kind of rules are allowed in `VI`-restricted programs. Vice versa, the class of programs $\{\mathcal{F}(P) \mid P \text{ is finitary}\}$ is not `VI`-restricted: for instance the translation of the finitary program $\{p(0); p(s(X)) \leftarrow p(X)\}$ is not `VI`-restricted. □

Other literature. In the above cited literature, infinite domains are obtained through the introduction of compound functional terms. Thus, the studied theoretical insights are often specialized to this notion of term, and take advantage e.g., of the common unification rules of formal logics over infinite domains. It is, in this setting, possible to study ascending and descending chains of functional terms in order to prove decidability. Similar to our approach is the work on open logic programs, and conceptual logic programs [15]. Such paper addresses the possibility of grounding a logic program, under Answer Set Semantics, over an infinite domain, in a way similar to classical logics and/or description logics. Each constant symbol has no predefined compound structure however. Also similar are [3] and [16], where a special construct, aimed at creating new tuple identifiers in relational databases is introduced.

In [17] and [4] the authors address the issue of implementing generalized quantifiers under Answer Set Semantics, in order to enable Answer Set Solvers to communicate, both directions, with external reasoners. This approach is different from the one considered in this paper since the former is inspired from second order logics and allows bidirectional flow of relational data (to and from external atoms), whereas, in our setting, the information flow is strictly value (first order) based, and allows relational output only in one direction. HEX programs, as defined in [4], do not address explicitly the issue of value invention (although semantics is given in terms of an infinite set of symbols). `VI` programs can simulate external predicates of [4] when relational input is not allowed.

An external predicate *à* la [4] (HEX predicate) is of the form $\#g[Y_1, \ldots, Y_m](X_1, \ldots, X_n)$, where $Y_1, \ldots, Y_n$ are input terms and $X_1, \ldots, X_n$ are output terms. Semantics of these atoms is given by means of a base oracle $f_{\#g}(I, Y_1, \ldots, Y_m, X_1, \ldots, X_m)$ where I is an interpretation. Note that HEX predicates depend on a current interpretation, thus enabling to quantify over predicate extensions. Assuming that for each HEX predicate $f_{\#g}$ do not depend on the current interpretation, and that *higher order atoms* (another special construct featured by HEX programs) are not allowed we can state the following equivalence theorem.

Theorem 6. An HEX program without higher order atoms is equivalent to a `VI` program where each HEX atom $\#g[Y_1, \ldots, Y_m](X_1, \ldots, X_n)$ is replaced

by an atom $\#g'(Y_1, \ldots, Y_m, X_1, \ldots, X_n)$, provided that each evaluation function $f_{\#g'}$ is such that for each I we have that $f_{\#g'}(Y_1, \ldots, Y_m, X_1, \ldots, X_m) = f_{\#g}(I, Y_1, \ldots, Y_m, X_1, \ldots, X_m)$. □

VI-restricted programs overcome the notion of semi-safe programs [8]. These programs have the finite grounding property: a weakly safe program P is *semi-safe* if each cycle in $G(P)$ contains only edges whose label corresponds to a safe rule. Semi-safe programs are strictly contained in the class of VI-restricted programs.

8 Conclusions

VI programs herein presented accommodate several cases where value invention is involved in logic programming. VI-restrictions allow to actually evaluate by means of a finite ground program a variety of programs (such as those with function symbols or set constructors) in many nontrivial cases.

A topic for future work is to investigate to what extent the notion of VI-restrictedness can be relaxed although keeping the complexity of recognizing the class in polynomial time. Intuitively, local analysis techniques can enlarge the class of programs whose finite grounding property is decidable, but this would force to renounce to polynomial complexity. Nonetheless, the spirit of restriction checkers is to keep evaluation times greatly smaller than the overall solving times.

VI programs have been implemented in the DLV system as well as a VI-restriction checker. Further details on the implementation can be found in [8]. A complete toolkit for developing custom external predicates is provided. Specific extensions of the DLV system with function symbols and sets, using VI as underlying framework, are in advanced stage of development and will be dealt with in appropriate papers. The system prototype, examples, manuals and benchmark results are available at `http://www.mat.unical.it/ianni/wiki/dlvex`.

References

1. Abiteboul, S., Vianu, V.: Datalog Extensions for Database Queries and Updates. JCSS **43**(1) (1991) 62–124
2. Cabibbo, L.: Expressiveness of Semipositive Logic Programs with Value Invention. Logic in Databases. (1996) 457–474.
3. Hull, R., Yoshikawa, M.: ILOG: Declarative Creation and Manipulation of Object Identifiers. VLDB 1990. 455–468.
4. Eiter, T., et al.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. IJCAI 2005, 90–96.
5. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. ESWC 2005. 392–407.
6. Leone, N., et al.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL (2006) To appear. `http://www.arxiv.org/ps/cs.AI/0211004`.

7. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. Artificial Intelligence **138** (2002) 181–234.
8. Calimeri, F., Ianni, G.: External sources of computation for Answer Set Solvers. LPNMR 2005, LNCS 3662. 105–118.
9. Syrjänen, T.: Omega-restricted logic programs. LPNMR 2001. 267-279.
10. Bonatti, P.A.: Reasoning with Infinite Stable Models. IJCAI 2001. 603–610.
11. The Friend of a Friend (FOAF) Project. `http://www.foaf-project.org/`.
12. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385.
13. Lifschitz, V., Turner, H.: Splitting a Logic Program. ICLP 1994. 23–37.
14. Dovier, A., Pontelli, E., Rossi, G.: Set unification. TPLP (2006) To appear.
15. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Semantic web reasoning with conceptual logic programs. RuleML 2004. 113–127.
16. Cabibbo, L.: The Expressive Power of Stratified Logic Programs with Value Invention. Inf. and Comp. **147**(1) (1998) 22–56.
17. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Nonmonotonic description logic programs: Implementation and experiments. LPAR 2004. 511–527.

On the Issue of Reinstatement in Argumentation

Martin Caminada

Institute of Information and Computing Sciences, Utrecht University
P.O. Box 80 089 3508 TB Utrecht, The Netherlands
martinc@cs.uu.nl

Abstract. Dung's theory of abstract argumentation frameworks [1] led to the formalization of various argument-based semantics, which are actually particular forms of dealing with the issue of reinstatement. In this paper, we re-examine the issue of semantics from the perspective of postulates. In particular, we ask ourselves the question of which (minimal) requirements have to be fulfilled by any principle for handling reinstatement, and how this relates to Dung's standard semantics. Our purpose is to shed new light on the ongoing discussion on which semantics is most appropriate.

1 Introduction

Dung's abstract theory of formal argumentation [1] has been a guide for researchers in the field of formal argumentation and nonmonotonic logic for more than ten years. During this period, a significant amount of work has been done on proof procedures for Dung's various argument-based semantics [2, 3], as well as on concrete argumentation formalisms (such as [4, 5, 6]) based on Dung's theory.

One specific issue that has received relatively little attention is the nature of reinstatement. Although reinstatement as a principle is not totally uncontroversial [7], the current consensus among many researchers in formal argumentation and nonmonotonic logic is that reinstatement of arguments is an essential feature of defeasible reasoning (as is for instance expressed in [8]). Dung provides several approaches for dealing with reinstatement, like stable semantics, preferred semantics, complete semantics and grounded semantics. Our contribution is not to criticize Dung's theory but rather to strengthen it. In particular, we ask ourselves the question: "Why do these semantics actually make sense?"

In previous work, we have stated a number of postulates which, in our view, every argumentation formalism should satisfy [9]. In the current paper, we will follow the same approach and state some simple and intuitive properties for dealing with the issue of reinstatement We then show how these properties are satisfied by Dung's standard semantics and how the differences between the various semantics could be viewed. We also show that a careful examination of reinstatement postulates reveals a semantics not currently known. Based on this discussion, we then share some thoughts on which type of semantics is most appropriate.

In order to keep things concise, the proofs have been omitted from the current paper. They can be found in a seperate technical report [10].

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 111–123, 2006.

2 Dung's Standard Semantics

A central notion in Dung's theory of abstract argumentation [1] is that of an argumentation framework, which is defined as follows.

Definition 1 (argumentation framework). *An argumentation framework is a pair* (Ar, def) *where* Ar *is a set of arguments and* $def \subseteq Ar \times Ar$.

Definition 2 (defense / conflict-free). *Let* $A \in Ar$ *and* $\mathcal{A}rgs \subseteq Ar$.
We define A^+ *as* $\{B \mid A \; def \; B\}$ *and* $\mathcal{A}rgs^+$ *as* $\{B \mid A \; def \; B \; for \; some \; A \in \mathcal{A}rgs\}$.
We define A^- *as* $\{B \mid B \; def \; A\}$ *and* $\mathcal{A}rgs^-$ *as* $\{B \mid B \; def \; A \; for \; some \; A \in \mathcal{A}rgs\}$.
$\mathcal{A}rgs$ defends *an argument* A *iff* $A^- \subseteq \mathcal{A}rgs^+$.
$\mathcal{A}rgs$ *is* conflict-free *iff* $\mathcal{A}rgs \cap \mathcal{A}rgs^+ = \emptyset$.

In the following definition, $F(\mathcal{A}rgs)$ stands for the set of arguments that are acceptable (in the sense of [1]) with respect to $\mathcal{A}rgs$.

Definition 3 (acceptability semantics). *Let* $\mathcal{A}rgs$ *be a conflict-free set of arguments and* $F : 2^{\mathcal{A}rgs} \rightarrow 2^{\mathcal{A}rgs}$ *be a function with* $F(\mathcal{A}rgs) = \{A \mid A \; is \; defended \; by \; \mathcal{A}rgs\}$.
$\mathcal{A}rgs$ *is* admissible *iff* $\mathcal{A}rgs \subseteq F(\mathcal{A}rgs)$.
$\mathcal{A}rgs$ *is a* complete *extension iff* $\mathcal{A}rgs = F(\mathcal{A}rgs)$.
$\mathcal{A}rgs$ *is a* grounded *extension iff* $\mathcal{A}rgs$ *is the minimal (w.r.t. set-inclusion) complete extension.*
$\mathcal{A}rgs$ *is a* preferred *extension iff* $\mathcal{A}rgs$ *is a maximal (w.r.t. set-inclusion) complete extension.*
$\mathcal{A}rgs$ *is a* stable *extension iff* $\mathcal{A}rgs$ *is a preferred extension that defeats every argument in* $Ar \backslash \mathcal{A}rgs$.

3 Reinstatement Labellings

The issue of quality postulates, or axioms, has recently received some attention in the field of formal argumentation and non-monotonic logic [9, 11]. An interesting question is whether one can also provide quality postulates for dealing with the reinstatement of arguments. Although the reinstatement has to a great extent been studied by Dung [1], the issue of which postulates have to be satisfied in order for a specific criterion for reinstatement to make sense has received relatively little attention.

One possible approach would be to start labelling the arguments in an argumentation framework. We distinguish three labels: "in", "out" and "undec" (undecided).

Definition 4. *Let* (Ar, def) *be a Dung-style argumentation framework. An* AF-labelling *is a (total) function* $\mathcal{L} : Ar \longrightarrow \{\texttt{in}, \texttt{out}, \texttt{undec}\}$. *We define* $\texttt{in}(\mathcal{L})$ *as* $\{A \in Ar \mid \mathcal{L}(A) = \texttt{in}\}$, $\texttt{out}(\mathcal{L})$ *as* $\{A \in Ar \mid \mathcal{L}(A) = \texttt{out}\}$ *and* $\texttt{undec}(\mathcal{L})$ *as* $\{A \in Ar \mid \mathcal{L}(A) = \texttt{undec}\}$.

In a *reinstatement labelling*, an argument is "in" iff all its defeaters are "out" and an argument is "out" if it has a defeater that is "in", as is stated in the following definition.

Definition 5. *Let $\mathcal{L}$ be an AF-labelling. We say that $\mathcal{L}$ is a* reinstatement labelling *iff it satisfies the following:*

- $\forall A \in Ar : (\mathcal{L}(A) = \texttt{out} \equiv \exists B \in Ar : (B def A \wedge \mathcal{L}(B) = \texttt{in}))$ *and*
- $\forall A \in Ar : (\mathcal{L}(A) = \texttt{in} \equiv \forall B \in Ar : (B def A \supset \mathcal{L}(B) = \texttt{out}))$.

The above definitions can be illustrated using the argumentation frameworks in Figure 1. Here, an argumentation framework is depicted as a directed graph, in which the vertices represent the arguments and the edges represent the defeat relation. In the leftmost argumentation framework, there exists just one reinstatement labelling ($\mathcal{L}_1$) with $\mathcal{L}_1(A) = \texttt{in},\ \mathcal{L}_1(B) = \texttt{out},\ \mathcal{L}_1(C) = \texttt{in}$. In the middle argumentation framework, there exist three reinstatement labellings ($\mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$) with $\mathcal{L}_2(D) = \texttt{in}$, $\mathcal{L}_2(E) = \texttt{out}$, $\mathcal{L}_3(D) = \texttt{out}$, $\mathcal{L}_3(E) = \texttt{in}$, $\mathcal{L}_4(D) = \texttt{undec}$ and $\mathcal{L}_4(E) = \texttt{undec}$. In the rightmost argumentation framework, there exists just one reinstatement labelling ($\mathcal{L}_5$) with $\mathcal{L}_5(F) = \texttt{undec}$.

Notice that Definition 5 can actually be seen as a *postulate*, as it specifies a restriction on an AF-labelling. It turns out that different kinds of reinstatement labellings correspond with different kinds of Dung-style semantics. This is explored in the remainder of this paper.

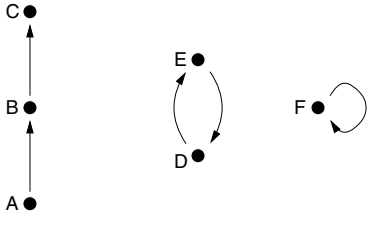


Fig. 1. Three argumentation frameworks

4 Labellings Versus Semantics

We now define two functions that, given an argumentation framework, allow a set of arguments to be converted to a labelling and vice versa. The function $\texttt{Ext2Lab}_{(Ar,def)}$ takes a conflict-free set of arguments (sometimes an extension) and converts it to a labelling. The function $\texttt{Lab2Ext}_{(Ar,def)}$ takes an AF-labelling and converts it to a set of arguments (sometimes an extension). Notice that as an AF-labelling is defined as a function (Definition 4), which in its turn is essentially a relation, it is possible to represent the labelling as a set of pairs.

In the following definition, the resulting AF-labelling does not yet need to satisfy the properties of a reinstatement labelling as stated in Definition 5.

Definition 6. *Let* (Ar, def) *be an argumentation framework,* $\mathcal{A}rgs \subseteq Ar$ *such that* $\mathcal{A}rgs$ *is conflict-free, and* $\mathcal{L} : Ar \longrightarrow \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$ *an AF-labelling. We define* $\mathtt{Ext2Lab}_{(Ar,def)}(\mathcal{A}rgs)$ *as* $\{(A, \mathtt{in}) \mid A \in \mathcal{A}rgs\} \cup \{(A, \mathtt{out}) \mid \exists A' \in \mathcal{A}rgs : A' def\, A\} \cup \{(A, \mathtt{undec}) \mid A \notin \mathcal{A}rgs \wedge \neg\exists A' \in \mathcal{A}rgs : A' def\, A\}$*. We define* $LabToExt_{(Ar,def)}(\mathcal{L})$ *as* $\{A \mid (A, \mathtt{in}) \in \mathcal{L}\}$*.*

The fact that $\mathcal{A}rgs$ is conflict-free in the above definition makes that $\mathtt{Ext2Lab}_{(Ar,def)}(\mathcal{A}rgs)$ is indeed an AF-labelling. When the associated argumentation framework is clear, we sometimes simply write `Ext2Lab` and `Lab2Ext` instead of $\mathtt{Ext2Lab}_{(Ar,def)}$ and $\mathtt{Lab2Ext}_{(Ar,def)}$.

4.1 Reinstatement Labellings Without Restrictions

It is interesting to notice that a reinstatement labelling coincides with Dung's notion of complete semantics.

Theorem 1. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling then* $\mathtt{Lab2Ext}(\mathcal{L})$ *is a complete extension. If* $\mathcal{A}rgs$ *is a complete extension then* $\mathtt{Ext2Lab}(\mathcal{A}rgs)$ *is a reinstatement labelling.*

It is interesting to observe that, when the domain and range of `Lab2Ext` is restricted to reinstatement labellings and complete extensions, and the domain and range of `Ext2Lab` is restricted to complete extensions and reinstatement labellings, then the resulting functions (call them $\mathtt{Lab2Ext}^r$ and $\mathtt{Ext2Lab}^r$) are bijective (that is, they are both injective and surjective) and each other's inverse.

Theorem 2.
Let $\mathtt{Lab2Ext}^r_{(Ar,def)} : \{\mathcal{L} \mid \mathcal{L}$ *is a reinstatement labelling of* $(Ar, def)\} \longrightarrow \{\mathcal{A}rgs \mid \mathcal{A}rgs$ *is a complete extension of* $(Ar, def)\}$ *be a function defined by* $\mathtt{Lab2Ext}^r_{(Ar,def)}(\mathcal{L}) = \mathtt{Lab2Ext}_{(Ar,def)}(\mathcal{L})$*.*
Let $\mathtt{Ext2Lab}^r_{(Ar,def)} : \{\mathcal{A}rgs \mid \mathcal{A}rgs$ *is a complete extension of* $(Ar, def)\} \longrightarrow \{\mathcal{L} \mid \mathcal{L}$ *is a reinstatement labelling of* $(Ar, def)\}$ *be a function defined by* $\mathtt{Ext2Lab}^r_{(Ar,def)}(\mathcal{A}rgs) = \mathtt{Ext2Lab}_{(Ar,def)}(\mathcal{A}rgs)$*.*
The functions $\mathtt{Lab2Ext}^r$ *and* $\mathtt{Ext2Lab}^r$ *are bijective and are each other's inverse.*

As $\mathtt{Lab2Ext}^r$ and $\mathtt{Ext2Lab}^r$ are each other's inverse, there exists a strong similarity between complete extensions and reinstatement labellings.

4.2 Reinstatement Labellings with Empty `undec`

Reinstatement labellings where `undec` is empty coincide with stable semantics.

Theorem 3. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling with* $\mathtt{undec}(\mathcal{L}) = \emptyset$ *then* $\mathtt{Lab2Ext}(\mathcal{L})$ *is a stable extension. If* $\mathcal{A}rgs$ *be a stable extension then* $\mathtt{Ext2Lab}(\mathcal{A}rgs)$ *is a labelling with* $\mathtt{undec}(\mathcal{L}) = \emptyset$*.*

4.3 Reinstatement Labellings with Maximal `in`, Maximal `out` and Maximal `undec`

Reinstatement labellings where `in` is maximal coincide with preferred semantics.

Theorem 4. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{in}(\mathcal{L})$ *is maximal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is a preferred extension. If* $\mathcal{A}rgs$ *is a preferred extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a labelling where* $\texttt{in}(\mathcal{L})$ *is maximal.*

It is interesting to notice that, contrary to what one might expect, reinstatement labellings in which `out` is maximized coincide with preferred semantics, just like (as was proved earlier) labellings in which `in` is maximized. This has to do with the fact that when `in` increases, `out` also increases, and conversely. This is stated by the following lemma.

Lemma 1. *Let* $\mathcal{L}$ *and* $\mathcal{L}'$ *be two reinstatement labellings. If* $\texttt{in}(\mathcal{L}) \subsetneq \texttt{in}(\mathcal{L}')$ *then* $\texttt{out}(\mathcal{L}) \subsetneq \texttt{out}(\mathcal{L}')$. *If* $\texttt{out}(\mathcal{L}) \subsetneq \texttt{out}(\mathcal{L}')$ *then* $\texttt{in}(\mathcal{L}) \subsetneq \texttt{in}(\mathcal{L}')$.

Theorem 5. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{out}(\mathcal{L})$ *is maximal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is a preferred extension. If* $\mathcal{A}rgs$ *is a preferred extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a labelling such that* $\texttt{out}(\mathcal{L})$ *is maximal.*

A reinstatement labelling with maximal `undec` coincides with grounded semantics.

Theorem 6. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{undec}(\mathcal{L})$ *is maximal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is the grounded extension. If* $\mathcal{A}rgs$ *is the grounded extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a reinstatement labelling where* $\texttt{undec}(\mathcal{L})$ *is maximal.*

4.4 Reinstatement Labellings with Minimal `in`, Minimal `out` and Minimal `undec`

A reinstatement labelling with minimal `in` coincides with grounded semantics.

Theorem 7. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{in}(\mathcal{L})$ *is minimal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is the grounded extension. If* $\mathcal{A}rgs$ *is the grounded extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a reinstatement labelling where* $\texttt{in}(\mathcal{L})$ *is minimal.*

A reinstatement labelling with minimal `out` coincides with grounded semantics.

Theorem 8. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{out}(\mathcal{L})$ *is minimal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is the grounded extension. If* $\mathcal{A}rgs$ *is the grounded extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a reinstatement labelling where* $\texttt{out}(\mathcal{L})$ *is minimal.*

The last remaining case to be examined is the one of reinstatement labellings where `undec` is minimized. We show that this does not coincide with any of Dung's standard semantics.

There is a one-way relation between reinstatement labellings with minimal undec and preferred extensions, as is stated in the following theorem.

Theorem 9. *Let* (Ar, def) *be an argumentation framework and* $\mathcal{L}$ *be a reinstatement labelling such that* $\texttt{undec}(\mathcal{L})$ *is minimal. Then* $\texttt{Lab2Ext}(\mathcal{L})$ *is a preferred extension.*

Unfortunately, it does not work the other way around. If $\mathcal{A}rgs$ is a preferred extension, then it is not necessarily the case that $\texttt{Ext2Lab}(\mathcal{A}rgs)$ is a reinstatement labelling where $\texttt{undec}(\mathcal{L})$ is minimal. This is shown in the following example.

Example 1. Let $Ar = \{A, B, C, D, E\}$ and let A defeat B, B defeat A, B defeat C, C defeat D, D defeat E, and E defeat C (see also Figure 2). Here, there exists two preferred extensions: $\mathcal{E}_1 = \{B, D\}$ and $\mathcal{E}_2 = \{A\}$. As $\mathcal{E}_1$ is also a stable extension, it holds that $\texttt{Ext2Lab}(\mathcal{E}_1)$ yields a labelling (say $\mathcal{L}$) with $\texttt{undec}(\mathcal{L}) = \emptyset$. However, $\texttt{Ext2Lab}(\mathcal{E}_2)$ yields a labelling (say $\mathcal{L}'$) with $\texttt{undec}(\mathcal{L}') = \{C, D, E\}$. So, even though $\mathcal{E}_2$ is a preferred extension, $\texttt{Ext2Lab}(\mathcal{E}_2)$ is not a reinstatement labelling in which undec is minimal.

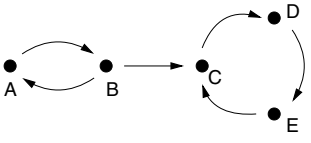


Fig. 2. A preferred extension does not necessarily imply minimal undec

Labellings in which undec is minimized can be seen as produced by an agent that is eager to take a position (in or out) on as many arguments as possible. It is not too difficult to specify what these would look like as a Dung-style semantics.

Definition 7. *Let* (Ar, def) *be an argumentation framework and* $\mathcal{A}rgs \subseteq Ar$. $\mathcal{A}rgs$ *is called a* semi-stable extension *iff* $\mathcal{A}rgs$ *is a complete extension where* $\mathcal{A}rgs \cup \mathcal{A}rgs^+$ *is maximal.*

The following theorem states that semi-stable semantics indeed coincides with reinstatement labellings in which undec is minimal.

Theorem 10. *Let* (Ar, def) *be an argumentation framework. If* $\mathcal{L}$ *is a reinstatement labelling where* $\texttt{undec}(\mathcal{L})$ *is minimal then* $\texttt{Lab2Ext}(\mathcal{L})$ *is a semi-stable extension. If* $\mathcal{A}rgs$ *is a semi-stable extension then* $\texttt{Ext2Lab}(\mathcal{A}rgs)$ *is a reinstatement labelling where* $\texttt{undec}(\mathcal{L})$ *is minimal.*

An interesting property is that when there exists at least one stable extension, the semi-stable extensions coincide with the stable extensions. This is because a stable extension lables every argument in or out, without labelling any argument undec. As for this stable extension, the set of undec labelled arguments is empty, every labelling in which undec is minimized must then also have the set of undec labelled arguments empty, and is therefore also a stable extension.

Theorem 11. *Let* (Ar, def) *be an argumentation framework. If there exists a stable extension, then the semi-stable extensions coincide with the stable extensions.*

It should be mentioned that Theorem 11 does not hold when semi-stable semantics is replaced by preferred semantics. That is, it is *not* the case that if there exists a stable extension, the preferred extensions coincide with the stable extensions (see Figure 2 for a counterexample). Semi-stable semantics is thus very close to stable semantics (closer than, for instance, preferred semantics) without the traditional disadvantage of stable semantics (the potential absence of extensions).

The idea of semi-stable semantics is not entirely new. It is quite similar to Verheij's concept of an *admissible stage extension*, which fits within Verheij's approach of using *stages* to deal with the reinstatement of arguments [12].

Definition 8 ([12], condensed). *An* admissible stage extension *is a pair* $(\mathcal{A}rgs, \mathcal{A}rgs^+)$ *where* $\mathcal{A}rgs$ *is an admissible set of arguments and* $\mathcal{A}rgs \cup \mathcal{A}rgs^+$ *is maximal.*

Theorem 12. *Let* (Ar, def) *be an argumentation framework and* $\mathcal{A}rgs \subseteq Ar$. $(\mathcal{A}rgs, \mathcal{A}rgs^+)$ *is an admissible stage extension iff* $\mathcal{A}rgs$ *is a semi-stable extension.*

4.5 Overview

From the previous discussion, it is clear that there exists a connection between the various forms of reinstatement labellings on one hand and the various Dung-style semantics on the other hand. This connection is summarized in Table 1.

Table 1. Reinst. labellings versus Dung-style semantics

restriction reinst. labellings	Dung-style semantics	linked by Theorem
no restrictions	complete semantics	1
empty `undec`	stable semantics	3
maximal `in`	preferred semantics	4
maximal `out`	preferred semantics	5
maximal `undec`	grounded semantics	6
minimal `in`	grounded semantics	7
minimal `out`	grounded semantics	8
minimal `undec`	semi-stable semantics	10

There also exists a partial ordering between the various Dung-style semantics. Every stable extension is a semi-stable extension, every semi-stable extension is a preferred extension, every preferred extension is a complete extension, and every grounded extension is a complete extension. This is graphically depicted in Figure 3.

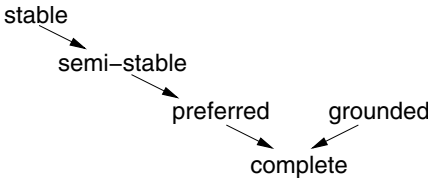


Fig. 3. An overview of the different semantics

5 Semantics Revisited

In essence, a reinstatement labelling can be seen as a subjective but reasonable point of view that an agent can take with respect to which arguments are `in`, `out` or `undec`. Each such position is internally coherent in the sense that, if questioned, the agent can use its own position to defend itself. It is possible for the position to be disagreed with, but at least one cannot point out an internal inconsistency. The set of all reinstatement labellings thus stands for all possible and reasonable positions an agent can take. This can be seen as a good reason for applying complete semantics, as reinstatement labellings coincide with complete extensions (as was explained in section 4.1). In the remainder of this section, we compare the approach of applying complete semantics with alternative approaches (in particular with preferred semantics).

When determining the overall justified arguments, two approaches are possible: the sceptical and the credulous one. Under the credulous approach, an argument is justified iff there is at least one reasonable position (= reinstatement labelling) where it is labelled `in`. Under the sceptical approach, an argument is justified iff it is `in` in every reasonable position; that is, a reasonable agent cannot deny that the argument is `in`.

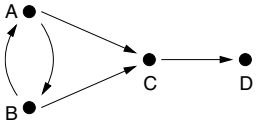


Fig. 4. A floating argument

As an example, consider the argumentation framework of Figure 4. Here there are three reinstatement labellings, as stated in Figure 5. When all reinstatement labellings are taken into account (such is the case in complete semantics) then A, B and D are credulously justified, whereas no arguments are sceptically justified.

It is interesting to compare this approach with preferred semantics, which has been the subject of much recent research [2, 13, 14]. As was explained earlier, a preferred extension coincides with a reinstatement labelling in which the set of

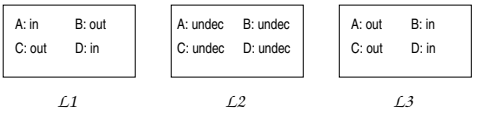


Fig. 5. Three reinstatement labellings

arguments labelled `in` is maximal. In case of Figure 4, for instance, the relevant labellings are only $\mathcal{L}_1$ and $\mathcal{L}_3$; thus, $\mathcal{L}_2$ is ruled out (see Figure 6).

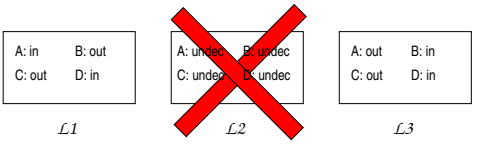


Fig. 6. Preferred semantics rules out particular labellings

What preferred semantics essentially does is to rule out zero or more reinstatement labellings before determining which arguments are credulously or sceptically justified. Under the sceptical approach, this can lead to more conclusions becoming justified. In the case of Figure 4, for instance, argument D is sceptically justified under preferred semantics but not under complete semantics.

The fact that under preferred semantics, reinstatement labelling $\mathcal{L}_2$ is ruled out can be seen as odd. $\mathcal{L}_2$, after all, is a perfectly valid reinstatement labelling. The fact that it is ruled out under preferred semantics means that those who defend preferred semantics must have some reason to justify this. This reason should state why $\mathcal{L}_2$ is "wrong" or "irrelevant", thus making it possible to ignore $\mathcal{L}_2$. One such reason could be (Theorem 4) "$\mathcal{L}_2$ should be ignored because the set of `in`-labelled arguments is not maximal." This reason does not appear to be a very strong one.

A more pragmatic reason in favor of preferred semantics is the issue of floating conclusions and floating arguments. Suppose the following information is available [15]: (1) Lars's mother is Norwegian, (2) Lars's father is Dutch, (3) Norwegians like ice-skating and (4) Dutch like ice-skating. We can now construct two arguments that defeat each other (assuming that double nationality is not possible): (A) Lars likes ice-skating because he's Norwegian and (B) Lars likes ice-skating because he's Dutch. Under sceptical complete semantics, the proposition that Lars likes ice-skating is not justified, despite the fact that, intuitively, it should be. Under sceptical preferred semantics, on the other hand, the proposition that Lars likes ice-skating *is* justified. At a first sight, this seems to illustrate a clear advantage of preferred semantics to complete semantics.

If we take a closer look, however, the situation becomes more complex. This is because the issue of whether or not Lars likes ice-skating depends on whether

or not the principle of excluded middle is regarded as valid. In monotonic logic, the validity of a statement $p \vee \neg p$ depends, among other things, on the number of truth-values. Whereas in a two-valued logic (where each proposition is either `true` or `false` in a given model) the proposition $p \vee \neg p$ is usually regarded as valid, it is *not* regarded as valid in, for instance, three-valued logics [16, 17]. Similarly, for one of the two conflicting arguments A and B to be regarded as valid (or *justified*), one should require that an argument is either `in` or `out`, resulting in a two-valued reinstatement labelling (without `undec`). In section 4.2, it was shown that this essentially boils down to stable semantics. Stable semantics, however, suffers from the problem that for some argumentation frameworks, no stable extensions exist. Consequently, it is not always possible to have a reinstatement labelling with only `in` and `out`. A third possibility (`undec`) is needed. Therefore, the principle of excluded middle, as an absolute criterion, should be rejected.[1] For those who nevertheless feel that the principle of the excluded middle should perhaps not hold at all times, but at least as much as possible (thus not completely ruling out `undec` but merely *minimizing* it), semi-stable semantics would seem a more appropriate choice than preferred semantics.

Given the observation that the principle of complete semantics can be given a decent philosophical justification, it is interesting to examine how complete semantics could be implemented. Fortunately, it turns out that both sceptical and credulous complete semantics have relatively easy and well-documented proof procedures.

As for sceptical semantics, an argument is in each complete extension iff it is in the grounded extension.

Theorem 13 ([1]). *Let $\{CE_1, \ldots, CE_n\}$ be the set of complete extensions and GE be the grounded extension. Let A be an argument. It holds that $A \in GE$ iff $A \in CE_1 \cap \ldots \cap CE_n$.*

As for credulous semantics, an argument is in some complete extension iff it is in some admissible set.

Theorem 14. *Let $CE_1, \ldots, CE_n$ be the set of complete extensions and $AS_1, \ldots, AS_m$ be the set of admissible sets. Let A be an argument. It holds that $\exists CE_i \in \{CE_1, \ldots, CE_n\} : A \in CE_i$ iff $\exists AS_j \in \{AS_1, \ldots, AS_m\} : A \in AS_j$.*

The fact that sceptical complete semantics coincides with grounded semantics, and credulous complete semantics coincides with credulous preferred semantics is advantageous, as these have relatively straightforward and well-studied proof procedures. Proof procedures for grounded semantics are given in [4, 18], and proof procedures for credulous preferred semantics are given in [2, 3].

[1] Another issue where the principle of excluded middle does not hold in most formalisms for defeasible reasoning is in handling disjunctive information. If $\{p \vee q\} \subseteq \mathcal{P}$ and $\{p \Rightarrow r;\ \ q \Rightarrow r\} \subseteq \mathcal{D}$ then in most formalisms for defeasible reasoning, r is not justified, although intuitively it should be, if one accepts the principle of excluded middle.

6 Summary and Discussion

In this paper, we showed it is possible to describe Dung's standard semantics in terms of reinstatement labellings, which provide an intuitive and relatively simple way of dealing with the issue of reinstatement. We also showed how reinstatement labellings can be used to pinpoint the exact differences between Dung's standard semantics. Using a systematic analysis of reinstatement labellings, we were also able to specify an additional form of semantics (semi-stable semantics) and showed how this semantics fits into the overall picture (Figure 3). We then reexamined the various semantical approaches and made a case for grounded semantics for sceptical entailment and credulous preferred semantics for credulous entailment.[2]

One of the researchers who has done some work on the relation between reinstatement labellings ("status assignments") and Dung's various semantics is Prakken [15]. In particular, Prakken proves (in his own terms and particular formalization) that reinstatement labellings without `undec` correspond to stable extensions, and that reinstatement labellings with maximal `in` correspond to preferred extensions [15]. It was the work of Prakken that served as an inspiration for the more thorough analysis in this paper.

Other recent work on reinstatement labellings has been done by Jakobovits and Vermeir [19]. Their definition of a labelling, however, is different than ours. First of all, they allow for an argument to be labelled `in`, `out`, both `in` and `out`, or neither `in` or `out`. Furthermore, their main reinstatement postulate is different.

Definition 9 ([19], syntax and formulation adjusted). *$\mathcal{L}$ is a labelling iff:*

- $\forall A \in Ar : (\mathcal{L}(A) = \texttt{out} \equiv \exists B \in Ar : (B def A \wedge \mathcal{L}(B) = \texttt{in}))$ *and*
- $\forall A \in Ar : (\mathcal{L}(A) = \texttt{in} \supset \forall B \in Ar : (B def A \supset \mathcal{L}(B) = \texttt{out}))$.

The difference between Definition 9 and the earlier presented Definition 5 is that the former does not require an argument of which all defeaters are `out` to be labelled `in`. This is quite strange, since it also means that an argument that has no defeaters at all is not required to be labelled `in`. To some extent, this problem is repaired for *complete labellings*, in which each argument is labelled either `in`, `out` or both.

The overall aim of Jakobovits and Vermeir is to come up with a semantics that is different from Dung's. Jakobovits and Vermeir justify their approach by discussing a number of small examples. However, the general approach of using examples in order to justify a particular formalism has some important downsides. To illustrate our main point, consider the following example provided in [19].

[2] This also implies that we do not support the approach of sceptical preferred semantics, as is for instance examined by [13]. We reject sceptical preferred semantics for reasons discussed in the previous section. We do, however, support the approach of credulous preferred semantics, as this coincides with credulous complete semantics.

Example 2.
A: As the bacteria in the patient's blood is not of type X, it must be of type Y.
B: As the bacteria in the patient's blood is not of type Y, it must be of type X.
C: As the patient does not have bacterial infection, giving antibiotics to the patient is superfluous.
D: As it is not superfluous to give the patient antibiotics, the antibiotics should be prescribed.

Example 2 is represented in the argumentation framework of Figure 4. Jakobovits and Vermeir argue that the correct outcome should be that argument D is justified. However, it is quite easy to provide another example, with essentially the same structure, where the desired outcome is totally different.

Example 3.
A: The suspect killed the victim by stabbing him with a knife, as witness #1 says so.
B: The suspect killed the victim by shooting him with a gun, as witness #2 says so.
C: The suspect is innocent.
D: The suspect should go to jail.

This essentially gives the same argumentation framework as Figure 4. However, an analysis of this case yields a different outcome. As essentially none of the witness statements is without doubt, none of them can serve as a good reason to refute the innocence of the suspect, and the conclusion that suspect should go to jail is not an intuitive or desired one, at least not from a legal point of view.

The main point here is that some researchers try to justify a particular design decision by giving an abstract example (like Figure 4) an informal meaning (like Example 2 or Example 3) and then arguing that the outcome of the abstract example should be in line with the "intuitive" outcome of the informal example. Although this approach has been applied by various researchers in the past, it has also been criticized [20, 18] for its inherent ad-hoc nature.

It is the author's opinion that a better justification for the design of a particular logical formalism can be found in postulates, as these have a more general nature than separate examples. And for reasons explained earlier, we feel that Definition 5 can serve as a more intuitive and acceptable postulate for reinstatement than Definition 9. It is the author's firm opinion that Dung's traditional semantics have a solid basis and that one should have very good reasons for adjusting them.

References

1. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artificial Intelligence **77** (1995) 321–357
2. Vreeswijk, G.A.W., Prakken, H.: Credulous and sceptical argument games for preferred semantics. In: Proceedings of the 7th European Workshop on Logic for Artificial Intelligence (JELIA-00). Number 1919 in Springer Lecture Notes in AI, Berlin, Springer Verlag (2000) 239–253

3. Cayrol, C., Doutre, S., Mengin, J.: Dialectical Proof Theories for the Credulous Preferred Semantics of Argumentation Frameworks. In: Proceedings of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU-2001). Volume 2143 of LNAI., Springer-Verlag (2001) 668–679
4. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Non-Classical Logics **7** (1997) 25–75
5. Governatori, G., Maher, M., Antoniou, G., Billington, D.: Argumentation semantics for defeasible logic. Journal of Logic and Computation **14** (2004) 675–702
6. ASPIC-consortium: Deliverable D2.5: Draft formal semantics for ASPIC system (2005)
7. Horty, J.: Argument construction and reinstatement in logics for defeasible reasoning. Artificial Intelligence and Law **9** (2001) 1–28
8. Prakken, H.: Intuitions and the modelling of defeasible reasoning: some case studies. In: Proceedings of the Ninth International Workshop on Nonmonotonic Reasoning (NMR-2002), Toulouse, France (2002) 91–99
9. Caminada, M., Amgoud, L.: An axiomatic account of formal argumentation. In: Proceedings of the AAAI-2005. (2005) 608–613
10. Caminada, M.: On the issue of reinstatement in argumentation. Technical Report UU-CS-2006-023, Institute of Information and Computing Sciences, Utrecht University (2006)
11. Caminada, M.: Contamination in formal argumentation systems. In: Proceedings of the 17th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC). (2005) 59–65
12. Verheij, B.: Two approaches to dialectical argumentation: admissible sets and argumentation stages. In Meyer, J.J., van der Gaag, L., eds.: Proceedings of the Eighth Dutch Conference on Artificial Intelligence (NAIC'96), Utrecht, Utrecht University (1996) 357–368
13. Doutre, S., Mengin, J.: On sceptical versus credulous acceptance for abstract argument systems. In: Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA-2004). (2004) 462–473
14. Dimopoulos, Y., Nebel, B., Toni, F.: Finding Admissible and Preferred Arguments Can be Very Hard. In: Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-2000). (2000) 53–61
15. Prakken, H.: Commonsense reasoning. Technical report, Institute of Information and Computing Sciences, Utrecht University (2004) Reader.
16. Urquhart, A.: Basic many-valued logic. In Gabbay, D., Günthner, F., eds.: Handbook of Philosophical Logic. Volume 2. Second edn. Kluwer Academic Publishers, Dordrecht/Boston/London (2001) 249–295
17. Hähnle, R.: Advanced many-valued logic. In Gabbay, D., Günthner, F., eds.: Handbook of Philosophical Logic. Volume 2. Second edn. Kluwer Academic Publishers, Dordrecht/Boston/London (2001) 297–395
18. Caminada, M.: For the sake of the Argument. Explorations into argument-based reasoning. Doctoral dissertation Free University Amsterdam (2004)
19. Jakobovits, H., Vermeir, D.: Robust semantics for argumentation frameworks. Journal of logic and computation **9(2)** (1999) 215–261
20. Vreeswijk, G.A.W.: Studies in defeasible argumentation. PhD thesis at Free University of Amsterdam (1993)

Comparing Action Descriptions Based on Semantic Preferences*

Thomas Eiter, Esra Erdem, Michael Fink, and Ján Senko

Institute of Information Systems, Vienna University of Technology, Austria
{eiter, esra, michael, jan}@kr.tuwien.ac.at

Abstract. We consider action domain descriptions whose meaning can be represented by transition diagrams. We introduce several semantic measures to compare such action descriptions, based on preferences over possible states of the world and preferences over some given conditions (observations, assertions, etc.) about the domain, as well as the probabilities of possible transitions. This preference information is used to assemble a weight which is assigned to an action description. As an application of this approach, we study the problem of updating action descriptions with respect to some given conditions. With a semantic approach based on preferences, not only, for some problems, we get more plausible solutions, but also, for some problems without any solutions due to too strong conditions, we can identify which conditions to relax to obtain a solution. We conclude with computational issues, and characterize the complexity of computing the semantic measures.

1 Introduction

This paper discusses how to compare action descriptions, whose meaning can be represented by transition diagrams—a directed graph whose nodes correspond to states and edges correspond to transitions caused by action occurrences and non-occurrences, with respect to some given conditions. Comparison of action descriptions is important for applications, when an agent has to prefer one description more than the others. One such application is the action description update problem [1]: when an agent tries to update an action description with respect to some given information, she usually ends up with several possibilities and has to choose one of these action descriptions. Another application is related to representing an action domain in an elaboration tolerant way (for a definition of elaboration tolerance see, e.g., [2, 3]): among several action descriptions representing the same action domain, which one is the most elaboration tolerant one, with respect to some given conditions describing possible elaborations?

The preference of an agent over action descriptions may be based on a syntactic measure, such as the number of formulas: the less the number of formulas contained in an action description, the more preferred it is. A syntactic measure can be defined also in terms of set containment with respect to a given action description D: an action description is more preferred if it is a maximal set among others that is contained in D. For instance, according to the syntactic measure used in [1] for updating an action

* Work supported by the Austrian Science Fund (FWF) under grant P16536-N04.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 124–137, 2006.

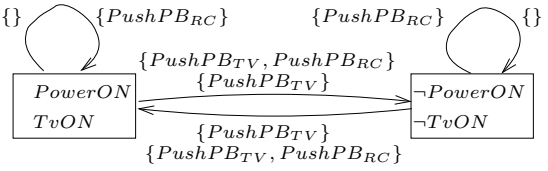


Fig. 1. A transition diagram

description D with some new knowledge Q, an action description D' is more preferred if D' is a maximal set among others containing D and contained in $D \cup Q$ is maximum.

In this paper, we describe the preference of an agent over action descriptions, with respect to some semantic measure. The idea is to describe a semantic measure by assigning weights (i.e., real numbers) to action descriptions, with respect to their transition diagrams and some given conditions; then, once the weights of action descriptions are computed, to compare two descriptions by comparing their weights.

We consider action descriptions, in a fragment of the action language $\mathcal{C}$ [4], which consists of "causal laws." For instance, the causal law

$$\textbf{caused } PowerON \textbf{ after } PushPB_{TV} \wedge \neg PowerON, \tag{1}$$

expresses that the action $PushPB_{TV}$ causes the value of the fluent $PowerON$ to change from f to t; such causal laws describe direct effects of actions. The causal law

$$\textbf{caused } TvON \textbf{ if } PowerON, \tag{2}$$

expresses that if the fluent $PowerON$ is caused to be true, then the fluent $TvON$ is caused to be true as well; such causal laws describe state constraints. The meaning of an action description D can be represented by a transition diagram, like in Fig. 1. In this transition diagram, the nodes of the graph (shown by boxes) denote the states of the world: (s) one where both the power and the TV is on, and (s') the other where both the power and the TV is off. The edges denote action occurrences. For instance, the edge from s to s' labeled by the action of pushing the power button on the TV describes that executing this action at s leads to s'. The edges labeled by the empty set are due to the law of inertia.

Suppose that we are given another action description D' describing the domain above; and that the transition diagram of D' is almost the same as that of D, except that there is no outgoing edge from the state $\{PowerON, TvON\}$ with the label $\{PushPB_{RC}\}$. Which action description should be preferred? In answering this question, we also take given *conditions* (observations, assertions, etc.) on the action domain into account. We describe conditions in an action query language, like in [5], by "queries." For instance,

$$\textbf{ALWAYS } \bigvee_{A \in 2^{\mathbf{A}}} \textbf{executable } A, \tag{3}$$

where $2^{\mathbf{A}}$ denotes the set of all actions, expresses that, at every state, there is some action executable. The query

$$\textbf{SOMETIMES evolves } PowerON; \{PushPB_{RC}\}; PowerON \tag{4}$$

expresses that, at some state when the power is on, pushing the power button on the remote control does not turn the power off.
The question we consider in this paper is then the following:

> *Given a set $\mathcal{D}$ of action descriptions and a set C of queries, which action description in $\mathcal{D}$ is a most preferred one with respect to C?*

Our main contributions are briefly summarized as follows.

- We provide an answer to the above question with respect to mainly four *semantically-oriented* approaches, by assigning weights to action descriptions in $\mathcal{D}$, based on their transition diagrams. The weights express preferences of the agent over possible states of the world and preferences over conditions, as well as the probabilities of possible transitions.

  A simple weight measure is to count the number of queries in C which an action description D entails. In the example above, D entails according to its transition diagram (3) and (4), so D has weight 2; D' entails according to its transition diagram only (3), so D' has weight 1. Hence, D is preferred over D'.
- We apply these approaches to the problem of updating an action description, and observe two benefits. First, if a problem has many solutions with the syntactic approach of [1], a semantic approach can be used to pick one. Second, if a problem does not have any solution with any of the approaches due to too strong conditions, a semantic approach can be used to identify which conditions to relax to find a solution.
- We characterize the computational cost of computing the weight assignments, which lays the foundations for efficient computation.

For space reasons, we omit the definitions of transition diagrams and action descriptions. *They are as in [1] and given in an extended version [6],*[1] *which contains further explanation of the examples, additional examples, another application, and a detailed discussion of the complexity results and algorithms.*

2 Action Queries

To talk about observations of the world, or assertions about the effects of the execution of actions, we use an action query language consisting of queries described as follows. We start with *basic queries*: (a) *static queries* of the form

$$\textbf{holds } F, \tag{5}$$

where F is a fluent formula; (b) *dynamic queries* of the form

$$\textbf{necessarily } Q \textbf{ after } A_1; \ldots; A_n, \tag{6}$$

where Q is a basic query and each A_i is an action; and (c) every propositional combination of basic queries. An *existential query* is an expression of the form

$$\textbf{SOMETIMES } Q, \tag{7}$$

[1] Available at `http://www.kr.tuwien.ac.at/research/ad-cmp.pdf`.

where Q is a basic query; a *universal query* is of the form

$$\textbf{ALWAYS}\ Q, \tag{8}$$

where Q is a basic query. A *query* q is a propositional combination of existential queries and universal queries.

As for the semantics, let $T = \langle S, V, R\rangle$ be a transition diagram, with a set S of states, a value function V mapping, at each state s, every fluent P to a truth value, and a set R of transitions. A *history* of T of length n is a sequence

$$s_0, A_1, s_1, \ldots, s_{n-1}, A_n, s_n \tag{9}$$

where each $\langle s_i, A_{i+1}, s_{i+i}\rangle$ $(0 \leq i < n)$ is in R. We say that a state $s \in S$ *satisfies* a basic query Q' of form (5) (resp. (6)) relative to T (denoted $T, s \models Q'$), if the interpretation $P \mapsto V(P, s)$ satisfies F (resp. if, for every history $s = s_0, A_1, s_1, \ldots, s_{n-1}, A_n, s_n$ of T of length n, basic query Q is satisfied at state s_n). For other forms of basic queries Q, *satisfaction* is defined by the truth tables of propositional logic. If T is described by an action description D, then the satisfaction relation between s and a basic query Q can be denoted by $D, s \models Q$ as well.

Note that, for every state s and for every fluent formula F, $D, s \models$ **holds** F iff $D, s \models \neg$**holds** $\neg F$. For every state s, every fluent formula F, and every action sequence $A_1, \ldots, A_n$ $(n \geq 1)$, if $D, s \models$ **necessarily** (**holds** F) **after** $A_1; \ldots; A_n$ then $D, s \models \neg$**necessarily** ($\neg$**holds** F) **after** $A_1; \ldots; A_n$.

We say that D *entails* a query q (denoted $D \models q$) if one of the following holds:

- q is an existential query (7) and $D, s \models Q$ for some state $s \in S$;
- q is a universal query (8) and $D, s \models Q$ for every state $s \in S$,
- $q = \neg q'$ and $D \not\models q'$;
- $q = q_1 \wedge q_2$ and $D \models q_1$ and $D \models q_2$; or
- $q = q_1 \vee q_2$ and $D \models q_1$ or $D \models q_2$.

For every basic query Q, $D \models$ **SOMETIMES** Q iff $D \models \neg$**ALWAYS** $\neg Q$. For a set C of queries, we say that D *entails* C (denoted $D \models C$) if D *entails* every query in C. Consider, e.g., the action description consisting of (1), (2), and

$$\begin{array}{l}\textbf{caused}\ \neg PowerON\ \textbf{after}\ PushPB_{TV} \wedge PowerON\\ \textbf{caused}\ \neg TvON\ \textbf{if}\ \neg PowerON\\ \textbf{inertial}\ PowerON, \neg PowerON, TvON, \neg TvON\end{array} \tag{10}$$

encoding how a TV system operates; here **inertial** $L_1, \ldots, L_k$ stands for the causal laws **caused** L_i **if** L_i **after** L_i $(1 \leq i \leq k)$. It does not entail any set of queries containing

$$\textbf{ALWAYS necessarily}\ (\textbf{holds}\ \neg TvON)\ \textbf{after}\ \{PushPB_{RC}\}$$

because this query is not satisfied at the state $\{TvON, PowerON\}$; but, it entails the queries:

$$\begin{array}{l}\textbf{ALWAYS holds}\ PowerON \equiv TvON,\\ \textbf{ALWAYS holds}\ PowerON \wedge TvON \supset\\ \qquad \neg\textbf{necessarily}\ (\textbf{holds}\ TvON)\ \textbf{after}\ \{PushPB_{TV}\}.\end{array} \tag{11}$$

In the rest of the paper, an expression of the form

$$\textbf{possibly } Q \textbf{ after } A_1; \ldots; A_n,$$

where Q is a basic query and each A_i is an action, stands for the dynamic query $\neg$**necessarily** $\neg Q$ **after** $A_1; \ldots; A_n$; an expression of the form

$$\textbf{evolves } F_0; A_1; F_1; \ldots; F_{n-1}; A_n; F_n, \tag{12}$$

where each F_i is a fluent formula, and each A_i is an action, stands for **holds** $F_0 \ \wedge$ **possibly** (**holds** $F_1 \wedge$ **possibly** (**holds** $F_2 \wedge$ ...) **after** A_2) **after** A_1; and

$$\textbf{executable } A_1; \ldots; A_n,$$

where each A_i is an action, stands for **possibly** $True$ **after** $A_1; \ldots; A_n$. We sometimes drop **holds** from static queries appearing in dynamic queries.

Queries allow us to express various pieces of knowledge about the domain. For instance, we can express the existence of states where a formula F holds by means of the query **SOMETIMES holds** F. Similarly, we can express the existence of a transition from some state where a formula F holds to another state where a formula F' holds, by the execution of an action A:

$$\textbf{SOMETIMES holds } F \wedge \textbf{possibly } F' \textbf{ after } A\ .$$

In general, the existence of a history (9) such that, for each s_i of the history, the interpretation $P \mapsto V(P, s_i)$ satisfies some formula F_i is expressed by the query:

$$\textbf{SOMETIMES evolves } F_0; A_1; F_1; \ldots; F_{n-1}; A_n; F_n. \tag{13}$$

For instance, the query

$$\begin{aligned} \textbf{SOMETIMES evolves } & PowerON; \{PushPB_{TV}\}; \\ & \neg PowerON; \{PushPB_{TV}\}; PowerON. \end{aligned} \tag{14}$$

describes the presence of the following history in Fig. 1:

$$\begin{aligned} & \{PowerON, TvON\}, \{PushPB_{TV}\}, \\ & \{\neg PowerON, \neg TvON\}, \{PushPB_{TV}\}, \{PowerON, TvON\}. \end{aligned} \tag{15}$$

That at some state where formula F holds no action is possible is expressed by

$$\textbf{SOMETIMES holds } F \wedge \bigwedge\nolimits_{A \in 2^{\mathbf{A}}} \textbf{necessarily } False \textbf{ after } A.$$

Like in [1], executability of an action sequence $A_1, \ldots, A_n$ ($n \geq 1$) at every state can be described by **ALWAYS executable** $A_1; \ldots; A_n$; mandatory effects of a sequence $A_1, \ldots, A_n$ ($n \geq 1$) of actions in a given context by **ALWAYS holds** $G \supset$ **necessarily** F **after** $A_1; \ldots; A_n$; and possible effects of a sequence of actions in a context by **ALWAYS holds** $G \supset$ **possibly** F **after** $A_1; \ldots; A_n$. In the last two queries, F describes the effects and G the context.

3 Weight Assignments for Action Descriptions

To compare action descriptions with respect to their semantics, we can assign weights to them, based on their transition diagrams and a given set of conditions. We present below

four weight assignments, each with a different motivation expressing some appeal of the action description, however, without an a priori epistemic meaning. They are by no means exhaustive, i.e., many more are conceivable, but allow to specify preferences over the main semantic constituents—states, transitions, queries, and a combination thereof. Corresponding orders are total and, unlike more general preferences (partial orders), beneficial wrt. discrimination of choices or component-wise comparability.

3.1 Weighted States

We can specify our preference over states of a transition diagram $\langle S, V, R\rangle$ by assigning a weight to each state in S, by a function g. Such a function assigning real numbers to states of the world can be considered as a *utility function*, as in decision theory. If one state of the world is preferred to another state of the world then it has higher utility for the agent; here "utility" is understood as "the quality of being useful" as in [7]. Alternatively, the function g can be viewed as a *reward function*: being at a state s will give a reward of $g(s)$ to the agent.

Given a utility function for a set S of states, the highly preferred states relative to a given number l are states with a weight greater than l. Then, one way to define the weight of an action description D relative to g and l is as follows:

$$weight_s(D) = |\{s : s \in S, g(s) > l\}|.$$

With respect to this definition, the more the number of states that are highly preferred by the agent, the more preferred the action description is.

For instance, consider the transition diagram in Fig. 1 described by D. Take, for each $s \in S$,

$$g(s) = \begin{cases} 2 \text{ if } PowerON \in s \\ 1 \text{ otherwise.} \end{cases} \tag{16}$$

Take $l = 1$. Then $weight_s(D) = 1$.

3.2 Weighted Queries

We can assign weights to queries to specify preferences over conditions they express:

Let C be a set of queries, along with a weight function f mapping each query in C to a real number. Then one way to define the weight of D (relative to C and f) is by

$$weight_q(D) = \sum\nolimits_{c \in C, D \models c} f(c).$$

Intuitively, the weight of an action description defined relative to the weights of queries shows how much the set C of given preferable queries are satisfied. (Note that f can easily express a threshold function as well.) *With this definition, the more the highly preferred queries are satisfied, the more preferred the action description is.*

For instance, suppose that C consists of (14) and

$$\textbf{ALWAYS executable } \{PushPB_{RC}\}, \tag{17}$$

with weights 1 and 2 respectively. For the description D with the transition diagram in Fig. 1, $weight_q(D) = 3$.

3.3 Weighted Histories

In a transition diagram $T = \langle S, V, R\rangle$, we will say that a history (9) of length n is *desired* with respect to a given query (13), if, for each i, the interpretation $P \mapsto V(P, s_i)$ satisfies F_i.

Let D be an action description, and $T = \langle S, V, R\rangle$ be the transition diagram described by D. Let C be a set of queries, along with a weight function f mapping each condition in C to a number. Let H_C be the set of pairs (w, c) such that w is a desired history in T with respect to the query c of form (13) in C. Let us denote by $st(w)$ the starting state s_0 of a history w of form (9). We define a function h mapping each desired history w appearing in H_C to a real number, in terms of the utility $u(w)$ of state $st(w)$ with respect to w:

$$h(w) = u(w) \times \textstyle\sum_{(w,c)\in H_C} f(c).$$

The function u mapping a history w of form (9) to a real number can be defined in terms of a sequence of functions u_i. Given a utility function (or a reward function) g mapping each state in S to a real number, and a *transition model* m mapping each transition $\langle s, A, s'\rangle$ in R to a probability (i.e., the probability of reaching s' from s after execution of A):

$$\begin{aligned} u_n(w) &= g(s_n) \\ u_i(w) &= g(s_i) + m(\langle s_i, A_{i+1}, s_{i+1}\rangle) \times u_{i+1}(w)\ (0 \le i < n) \\ u(w) &= u_0(w). \end{aligned}$$

These equations are essentially obtained from the equations used for value determination in the policy-iteration algorithm described in [7, Chapter 17]: take $\{s_0, \ldots, s_n\}$ as the set of states, $\langle s_i, A_{i+1}, s_{i+1}\rangle$ as the possible transitions, the mapping $s_i \mapsto A_{i+1}$ as the fixed policy, U as u, U_i as u_i, R as g, and M as m. Then we can define the weight of D in terms of the weights of desired histories $w_1, \ldots, w_z$ appearing in H_C as follows:

$$weight_h(D) = \sum\nolimits_{i=1}^{z} h(w_i).$$

The more the utilities of desired histories (or trajectories) satisfied by the action description, the more preferred the action description is.

For instance, suppose that C consists of query (14), with weight 3. Consider the transition diagram $T = \langle S, V, R\rangle$ in Fig. 1. Let us denote history (15) by w, and query (14) by c. Then H_C contains (w, c). Take $g(s)$ as in (16). Take $l = 1$. Suppose that, for each transition $\langle s, A, s'\rangle$ in R,

$$m(\langle s, A, s'\rangle) = \begin{cases} 0.5 \text{ if } s = \{PowerON, TvON\} \wedge\ |A| = 1 \\ 1 \quad \text{otherwise.} \end{cases} \tag{18}$$

Then $u(w)$ is computed as 3.5. and $h(w) = u(w) \times \sum_{(w,c)\in H_C} f(c) = 3.5 \times 3 = 10.5$. Hence $weight_h(D) = 10.5$.

3.4 Weighted Queries Relative to Weighted States

The three approaches above can be united by also considering to what extent each universal query in C is entailed by the action description. The idea is while computing the

weight of a description relative to weighted queries, to take into account the states at which these queries are satisfied.

Let D be an action description. Let $T = \langle S, V, R\rangle$ be the transition diagram described by D, along with a weight function g mapping each state in T to a real number. Let C be a set of queries such that every query q in C is an existential query, a universal query, or a disjunction of both.

First, for each state s in S, we compute its new weight $g'(s)$, taking into account utilities of the desired histories starting with s. Let H_C be the set of pairs (w, c) such that w is a desired history in T with respect to the query c of form (13) in C. Let W be the set of histories that appear in H_C. Let u be a function mapping a history w to a real number, describing the utility of state s with respect to w. Then the new weight function g' is defined as follows:

$$g'(s) = \begin{cases} g(s) & \text{if } \nexists w(w \in W \wedge st(w) = s) \\ \sum_{w \in W, st(w)=s} u(w) & \text{otherwise.} \end{cases}$$

Next, for each query c in C, we compute its new weight $f'(c)$. Let f be a function mapping each condition in C to a real number. We will denote by $S_D(B)$ the set of states s such that $D, s \models B$. Then we define f' as follows:

$$f'(q) = \begin{cases} f'(q') + f'(q'') & \text{if } q = q' \vee q'' \\ \beta & \text{if } q = \textbf{ALWAYS } B \\ \gamma & \text{if } q = \textbf{SOMETIMES } B \wedge |S_D(B)| > 0 \\ 0 & \text{if } q = \textbf{SOMETIMES } B \wedge |S_D(B)| = 0, \end{cases}$$

where $\beta = f(q) \times \sum_{s \in S_D(B)} g'(s)$ and $\gamma = f(q) \times [(\sum_{s \in S_D(B)} g'(s))/|S_D(B)|]$. Intuitively, f' describes to what extent each preferable query q is satisfied.

Then the weight of D (relative to C and f') is the sum:

$$weight_{qs}(D) = \sum\nolimits_{q \in C} f'(q).$$

Intuitively, $weight_{qs}(D)$ describes how much and to what extent the given preferable queries are satisfied by D. For instance, suppose C consists of three queries:

$$\textbf{ALWAYS executable } \{PushPB_{TV}\}, \tag{19}$$

$$\textbf{SOMETIMES } \neg\textbf{executable } \{PushPB_{RC}, PushPB_{TV}\}, \tag{20}$$

and query (14), denoted by c_1, c_2 and c_3 respectively. Consider an action description D, with the transition diagram in Fig. 1. Let us denote history (15) by w; then $H_C = \{(w, c_3)\}$. Take the utility function g as in (16), and the transition model m as in (18). Take $f(c_1) = 1$, $f(c_2) = 2$, $f(c_3) = 3$. Then $g'(\{PowerON, TvON\}) = 3.5$, $g'(\{\neg PowerON, \neg TvON\}) = 1$, and $f'(c_1) = 4$, $f'(c_2) = 4$, $f'(c_3) = 10.5$. Therefore, $weight_{qs}(D) = 18.5$.

Further discussion and additional examples considering the weight functions in different action domains are given in the extended version [6].

4 Application: Updating an Action Description

Suppose that an action description D consists of two parts: D_u (unmodifiable causal laws) and D_m (modifiable causal laws); and a set C of conditions is partitioned into two: C_m (must) and C_p (preferable). We define an *Action Description Update (ADU)* problem by an action description $D = (D_u, D_m)$, a set Q of causal laws, a set $C = (C_m, C_p)$ of queries, all with the same signature, and a weight function $weight$ mapping an action description to a number. The weight function can be defined relative to a set of queries, a utility function, or a transition model, as seen in the previous section. We say that a consistent action description D' is a *solution* to the ADU problem $(D, Q, C, weight)$ if

(i) $Q \cup D_u \subseteq D' \subseteq D \cup Q$,
(ii) $D' \models C_m$,
(iii) there is no other consistent action description D'' such that $Q \cup D_u \subseteq D'' \subseteq D \cup Q$, $D'' \models C_m$, and $weight(D'') > weight(D')$.

The definition of an ADU problem in [1] is different from the one above mainly in two ways. First, $C_p = \emptyset$. Second, instead of (iii) above, the following syntactic condition is considered: there is no consistent action description D'' such that $D' \subset D'' \subseteq D \cup Q$, and $D'' \models C$.

The semantic approach above has mainly two benefits, compared to the syntactic approach of [1]. First, there may be more than one solution to some ADU problems with the syntactic approach. In such cases, a semantic approach may be applied to pick one of those solutions. Example 1 illustrates this benefit. Second, for an ADU problem, if no consistent action description D' satisfying (i) satisfies the must queries (C_m), there is no solution to this problem with either syntactic or semantic approach. In such a case, we can use the semantic approach with weighted queries, to relax some must queries in C_m (e.g., move them to C_p). The idea is first to solve the ADU problem $((D_u, D_m), Q, (\emptyset, C'_m), weight)$, where C'_m is obtained from C_m by complementing each query, and where the weights of queries in C'_m are equal to some very small negative integer; and then to identify the queries of C'_m satisfied in a solution and add them C_p, with weights multiplied by -1. This process of relaxing some conditions of C_m to find a solution is illustrated in Example 2.

Example 1. Consider, for instance, an action description $D = (D_m, D_u)$, where $D_m = \{(1), (2)\}$ and D_u is (10), that describes a TV system with a remote control. Suppose that, later the following information, Q, is obtained:

caused $TvON$ **after** $PushPB_{RC} \wedge PowerON \wedge \neg TvON$
caused $\neg TvON$ **after** $PushPB_{RC} \wedge TvON$.

Suppose that we are given the set $C = (C_m, C_p)$ of queries where C_m consists of the queries (3) and

$$\textbf{SOMETIMES evolves } \neg TvON; \{PushPB_{TV}\}; \neg TvON, \tag{21}$$

and C_p consists of the queries (14), (20), (19), (17), (4), denoted by $c_1, \ldots, c_5$ respectively. When Q is added to D, the meaning of $D \cup Q$ can be represented by a transition

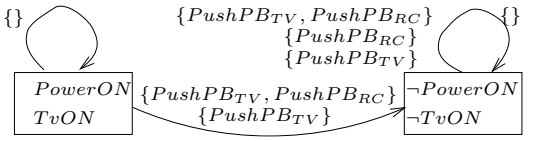


Fig. 2. Transition diagram of $D^{(2)} = D_u \cup Q \cup \{(2)\}$

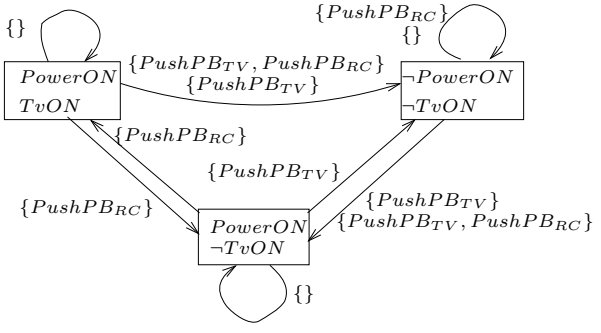


Fig. 3. Transition diagram of $D^{(3)} = D_u \cup Q \cup \{(1)\}$

diagram almost the same as in that of D (Fig. 1), except that there is no outgoing edge from the state $\{PowerON, TvON\}$ with the label $\{PushPB_{RC}\}$; thus only (3), (21), and (14) in C are entailed by $D \cup Q$. The question is how to update D by Q so that the must conditions, C_m, are satisfied, and the preferable conditions, C_p, are satisfied as much as possible.

The consistent action descriptions for which (i) holds are $D^{(1)} = D \cup Q$, $D^{(2)} = D_u \cup Q \cup \{(2)\}$, $D^{(3)} = D_u \cup Q \cup \{(1)\}$, $D^{(4)} = D_u \cup Q$. With the syntactic approach of [1], we have to choose between $D^{(2)}$ and $D^{(3)}$, since they have more causal laws. Consider the semantic approach based on weighted histories (i.e., $weight = weight_h$), with (16) as the utility function g, (18) as the transition model m, and $f(c_1) = 3, f(c_2) = 1, f(c_3) = 4, f(c_4) = 3, f(c_5) = 2$. Let us consider the states $s_0 = \{PowerON, TvON\}$, $s_1 = \{PowerON, \neg TvON\}$, $s_2 = \{\neg PowerON, \neg TvON\}$; and the histories

$$w_0 = s_0, \{PushPB_{RC}\}, s_1, \qquad w_2 = s_0, \{PushPB_{TV}\}, s_2, \{PushPB_{TV}\}, s_1,$$
$$w_1 = s_1, \{PushPB_{RC}\}, s_0, \qquad w_3 = s_1, \{PushPB_{TV}\}, s_2, \{PushPB_{TV}\}, s_1$$

with utilities $u(w_0) = 3, u(w_1) = 4, u(w_2) = 3.5, u(w_3) = 5$.

For $D^{(2)}$ (Fig. 2), since $H_{C_p} = \emptyset$, $weight_h(D^{(2)}) = 0$. For $D^{(3)}$ (Fig. 3), since H_{C_p} contains (w_0, c_5), (w_1, c_5), (w_2, c_3), and (w_3, c_3), $weight_h(D^{(3)}) = 48$. Thus $D^{(3)}$ is the solution.

Example 2. Let D, Q, C_p, and $D^{(1)}$–$D^{(4)}$ as in Example 1 and C_m consist of

$$\textbf{SOMETIMES} \neg \bigvee_{A \in 2^{\mathbf{A}}} \textbf{executable}\ A, \tag{22}$$

$$\textbf{ALWAYS}\ \neg\textbf{evolves}\ \neg TvON; \{PushPB_{TV}\}; \neg TvON, \tag{23}$$

denoted by c_1' and c_2' respectively. None of the descriptions $D^{(1)}$ – $D^{(4)}$ entails C_m. Therefore, there is no solution to the ADU problem above with either the syntactic approach of [1] or any of the semantic approaches above. To identify which queries in C_m we shall move to C_p, first we obtain C_m' from C_m by negating each query in C_m, and assigning a very small negative integer, say –100, as their weights. So C_m' consists of the queries (3) and (21), denoted by c_1'' and c_2'', with weights -100. With the semantic approach based on weighted queries (i.e., $weight = weight_q$),

$$\begin{aligned} &weight_q(D^{(1)}) = f(c_1'') = -100, \\ &weight_q(D^{(2)}) = weight_q(D^{(3)}) = f(c_1'') + f(c_2'') = -200, \\ &weight_q(D^{(4)}) = f(c_1'') + f(c_2'') = -200 \end{aligned}$$

the description $D^{(1)}$ is the solution to the ADU problem given by $((D_u, D_m), Q, (\emptyset, C_m'), weight_q)$. This suggests relaxing the must query (22) (i.e., adding the query (22) to C_p with the weight 100) and solving the new ADU problem, $((D_u, D_m), Q, \{(23)\}, C_p \cup \{(22)\}, weight_q)$, for which the description $D_u \cup Q$ is the solution.

Other semantic approaches to action description updates. Given a consistent action description E, condition (iii) of an ADU problem $(D, Q, C, weight)$ can be replaced by

(iii)′ there is no other consistent D'' such that $Q \cup D_u \subseteq D'' \subseteq D \cup Q$, $D'' \models C_m$, and $|weight(D'') - weight(E)| < |weight(D') - weight(E)|$

to express that, among the consistent action descriptions D' for which (i) and (ii) hold, an action description that is "closest" to (or most "similar" to) E is picked. Here, for instance, E may be $D \cup Q$, to incorporate as much of the new information as possible, although $D \cup Q$ may not entail C. What is meant by closeness or similarity is based on the particular definition of the weight function. For instance, based on the weights of the states only, with $g(s) = 1$ if s is a state of E, and 0 otherwise, the closeness of an action description to E is defined in terms of the common world states.

A further application of weight-based comparison of action descriptions to assess the elaboration tolerance of different representations of an action domain is considered in [6].

5 Computational Aspects

We confine here to discuss the complexity, in order to shed light on the cost of computing the weight measures. We assume that the basic functions $g(s)$, $f(q)$, as well as $m(\langle s, A, s' \rangle)$ are computable in polynomial time. For a background on complexity, we refer to the literature (see e.g. [8]).[2]

Apparently, none of the different weights above is polynomially computable from an input action description D and a set C of queries in general. Indeed, deciding whether S has any states is NP-complete, thus intractable. Furthermore, evaluating arbitrary queries q on D ($D \models q$) is a PSPACE-complete problem. Indeed, q can be evaluated by a simple recursive procedure in polynomial space. On the other hand, evaluating Quantified Boolean Formulas, which is PSPACE-complete, can be reduced to deciding $D \models q$.

[2] See also `http://qwiki.caltech.edu/wiki/Complexity_Zoo`

Table 1. Complexity of computing weights (completeness)

Input / Weight	$weight_s$	$weight_q$	$weight_h$	$weight_{qs}$
D, C	#P	FPSPACE	GapP *	FPSPACE
D, C, S	polynomial			
D_{pol}**, C	in $\mathrm{FP}_{\|}^{\mathrm{NP}}$			

* #P for non-negative $g(s)$,$f(q)$; ** $|S|$ is polynomially bounded

Computation given D and C. As it turns out, all four weights are computable in polynomial space. This is because each weight is a sum of (in some cases exponentially many) terms, each of which can be easily computed in polynomial space, using exhaustive enumeration. In some cases, the computation is also PSPACE-hard, but in others supposedly easier:

Theorem 1. *Given an action description D, a set C of queries, and polynomial-time computable basic functions $g(s)$, $f(q)$, and $m(\langle s, A, s'\rangle)$,*

(i) Computing $weight_s(D)$ relative to g is, #P-complete;
(ii) Computing $weight_q(D)$ relative to C and f is FPSPACE-complete;
(iii) Computing $weight_h(D)$ relative to C, f, g and m is #P-complete (modulo a normalization, which casts the problem to one on integers), if the range of f and g are nonnegative numbers, and GapP-complete for arbitrary f and g;
(iv) Computing $weight_{qs}(D)$ relative to C, f, g and m is FPSPACE-complete.

These results are also shown in the first row of Table 1. Here #P [8] is the class of the problems where the output is an integer that can be obtained as the number of the runs of an NP Turing machine accepting the input, and GapP [9, 10] is the closure of #P under subtraction (equivalently, the functions expressible as the number of accepting computations minus the number of rejecting computations of an NP Turing machine). These problems are trivially solvable in polynomial time with an oracle #P, and no such problem is believed to be PSPACE-hard.

Computation given D, C, and states S of D. Informally, a source of complexity is that D may specify an exponentially large transition diagram T. If T is given, then all four weights are polynomially computable. In fact, not all of T is needed, but only a *relevant part*, denoted $T_C(D)$, which comprises all states and all transitions that involve actions appearing in C.

Now if the state set S is known (e.g., after computation with CCALC [11]) or computable in polynomial time, then $T_C(D)$ is constructible in polynomial time. Indeed, for each states $s, s' \in S$ and each action A occurring in some query, we can test in polynomial time whether $\langle s, A, s'\rangle$ is a legal transition with respect to D; the total number of such triples is polynomial in $|S|$. Then the following result (the second row of Table 1) holds.

Theorem 2. *Given an action description D, the set S of states described by D, a set C of queries, and polynomial-time computable basic functions $g(s)$, $f(q)$, and $m(\langle s, A, s'\rangle)$. Then $weight_s(D)$ (relative to g), $weight_q(D)$ (relative to C and f),*

$weight_h(D)$ (relative to C, f, g and m), and $weight_{qs}(D)$ (relative to C, f, g and m), are all computable in polynomial time.

Intuitively, for $weight_q(D)$ this holds since we can decide whether a query q from C holds with respect to $T_C(D)$ in polynomial time using standard labeling methods from model checking [12]. We can compute $weight_h(D)$ with similar labeling techniques, reshuffling the weight and utility functions $h(w)$ and $u(w)$, respectively, such that considering exponentially many paths in $T_C(D)$ explicitly is avoided.

Computation given D and C for polynomial state set S. Finally, if the state space S is not large, i.e., $|S|$ is polynomially bounded, S is computable with the help of an NP-oracle in polynomial time; in fact, this is possible with parallel NP oracles queries, and thus computing S is in the respective class $\mathrm{FP}_{\parallel}^{\mathrm{NP}}$. From Theorem 2, we thus obtain the following results (the third row of Table 1):

Theorem 3. *Given an action description D such that $|S|$ is polynomially bounded, a set C of queries, and polynomial-time computable basic functions $g(s)$, $f(q)$, and $m(\langle s, A, s'\rangle)$, Then computing each of the weight functions, $weight_s(D)$ (relative to g), $weight_q(D)$ (relative to C and f), $weight_h(D)$ (relative to C, f, g and m), and $weight_{qs}(D)$ (relative to C, f, g and m), is in $\mathrm{FP}_{\parallel}^{\mathrm{NP}}$.*

On the other hand, tractability of any of the weight functions in the case where $|S|$ is polynomially bounded is unlikely, since solving SAT under the assertion that the given formula F has at most one model (which is still considered to be intractable) is reducible to computing $weight_p(D)$ for each $p \in \{s, q, h, qs\}$.

6 Conclusion

We have presented four ways of assigning weights to action descriptions, based on the preferences over states, preferences over conditions, and probabilities of transitions, so that one can compare the action descriptions by means of their weights. To the best of our knowledge, this paper is the first attempt in this direction. Moreover, we have characterized the computational cost of the weight assignments, providing a basis for efficient algorithms.

We have illustrated the usefulness of such a semantically-oriented approach of comparing action descriptions, on the problem of updating an action description, in comparison with the syntactic approach of [1]. Further examples and applications are considered in the extended version of this paper [6].

Further work will aim at implementations of the weight measures, based on the complexity characterizations and algorithms obtained (cf. [6]) and to investigate restricted problem classes. Another issue is to explore further measures.

References

1. Eiter, T., Erdem, E., Fink, M., Senko, J.: Updating action domain descriptions. In: Proc. IJCAI. (2005) 418–423
2. McCarthy, J.: Elaboration tolerance. In: Proc. CommonSense. (1998)

3. Amir, E.: Towards a formalization of elaboration tolerance: Adding and deleting axioms. In: Frontiers of Belief Revision. Kluwer (2000)
4. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. AAAI. (1998) 623–630
5. Gelfond, M., Lifschitz, V.: Action languages. ETAI **3** (1998) 195–210
6. Eiter, T., Erdem, E., Fink, M., Senko, J.: Comparing action descriptions based on semantic preferences. Extended manuscript. Available at `http://www.kr.tuwien.ac.at/research/ad-cmp.pdf` (2006)
7. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall (1995)
8. Papadimitriou, C.: Computational Complexity. Addison-Wesley (1994)
9. Fenner, S.A., Fortnow, L., Kurtz, S.A.: Gap-definable counting classes. Journal of Computer and System Sciences **48** (1994) 116–148
10. Gupta, S.: Closure properties and witness reduction. Journal of Computer and System Sciences **50** (1995) 412–432
11. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. AI **153** (2004) 49–104
12. Clark, E., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)

Modal Logics of Negotiation and Preference

Ulle Endriss and Eric Pacuit

Institute for Logic, Language and Computation
University of Amsterdam

Abstract. We develop a dynamic modal logic that can be used to model scenarios where agents negotiate over the allocation of a finite number of indivisible resources. The logic includes operators to speak about both preferences of individual agents and deals regarding the reallocation of certain resources. We reconstruct a known result regarding the convergence of sequences of mutually beneficial deals to a Pareto optimal allocation of resources, and discuss the relationship between reasoning tasks in our logic and problems in negotiation. For instance, checking whether a given restricted class of deals is sufficient to guarantee convergence to a Pareto optimal allocation for a specific negotiation scenario amounts to a model checking problem; and the problem of identifying conditions on preference relations that would guarantee convergence for a restricted class of deals under all circumstances can be cast as a question in modal logic correspondence theory.

1 Introduction

Negotiation between autonomous agents over the allocation of resources has become a central topic in AI. In this paper, we present some first steps towards using (modal) logic to model negotiation scenarios. We explore to what extent known results about negotiation can be reconstructed in such a logic and whether it is possible to derive new insights about a negotiation framework by studying its formalisation in logic. The particular negotiation framework we are interested in here, which has recently been studied by several authors [1, 2, 3], involves a number of autonomous agents negotiating over the reallocation of a number of indivisible goods amongst themselves. Agents have preferences over the resources they hold, and they will only agree to take part in a deal if that deal would leave them with a preferred bundle of goods. That is, negotiation is driven by the rational interests of the participating agents. At the same time, we can observe different phenomena at the global level. For instance, it may or may not be the case that the sequence of deals implemented by the agents converges to a socially optimal allocation of resources (say, a Pareto optimal allocation).

Our aim in this paper is to show how such a negotiation setting can be formalised using modal logic. More specifically, we are developing a logic in the style of propositional dynamic logic (PDL) that allows us to speak both about the preferences of individual agents and the aggregated preferences of the society as a whole (to model Pareto improvements), as well as deals between agents involving the reassignment of specific resources to other agents. We show that

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 138–150, 2006.

properties such as guaranteed convergence to a Pareto optimal allocation can be expressed in this logic, and we discuss how to apply logical reasoning techniques, such as model checking, to decision problems arising in the context of negotiation.

This work also fits in with the larger project of "social software" first discussed by Parikh [4]. The main idea of social software is that tools and techniques from computer science (in particular logic of programs) can be used to reason about *social procedures* (see [5] for a survey of the relevant literature). Much of the work on social software is concerned with developing logics intended to verify the "correctness" of social procedures [6]. There are often two key features of these logics. First, they should be expressive enough to capture the relevant concepts in order to state correctness conditions. Second, the logics should have well-behaved computational properties (for example, a decidable satisfiability problem and polynomial time model checking). The present paper will pay close attention to both of these issues.

The paper is organised as follows. In Section 2 we introduce a PDL-style logic for reasoning about negotiation settings, prove its decidability, and discuss some illustrative examples. Then we show in Section 3 how the language of this logic can express a property known as guaranteed convergence to a Pareto optimal allocation. Our discussion shows that this can be reduced to a statement about Pareto improvements alone; and we consequently introduce a second, more basic logic to reason about Pareto efficiency in Section 4. Section 5 concludes with an extensive discussion of further possibilities of linking reasoning tasks in our logic of negotiation spaces and questions arising in the context of negotiation. The appendix summarises relevant results about PDL and its extensions.

2 The Logic of Negotiation Spaces

In this section, we are going to develop a logic to describe negotiation scenarios of the following sort. There are a (finite) number of *agents* and a (finite) number of *resources*, which are indivisible and cannot be shared amongst more than one agent at a time. An *allocation* is a partitioning of the resources amongst the agents (each resource has to be assigned to exactly one agent). Agents have *preferences* over the bundles of resources they receive (but they are indifferent to what resources are being received by other agents; that is, we do not want to model allocative externalities). To improve their situation, agents can agree on *deals* to exchange some of the resources currently in their possession. In the most general case, we allow for any kind of multilateral deal. That is, a single deal may involve the reassignment of any number of resources amongst any number of agents. Agents are assumed to be *rational* in the sense of never accepting a deal that would leave them with a bundle that they like less than the bundle they did hold prior to that deal.

As outside observers, we are not actually interested in the preferences of individual agents, but we do care about the quality of allocations from a social point of view. In particular, we are going to be interested in allocations of resources

that are *Pareto optimal* as well as in sequences of deals that lead to such Pareto optimal allocations. To describe such scenarios, we develop the logic $\mathcal{L}_{\langle \mathcal{A},\mathcal{R}\rangle}$, which is parametrised by a finite set of *agents* $\mathcal{A}$ and a finite set of *resources* $\mathcal{R}$.

2.1 Preliminaries

An *allocation* is a total function $A : \mathcal{R} \to \mathcal{A}$ specifying for each resource item which agent currently holds that item. As we shall see, the set $\mathcal{A}^{\mathcal{R}}$ of all allocations will be the "set of worlds" in the (intended) models of our logic. An *atomic deal* is of the form $(a \leftarrow r)$, for $a \in \mathcal{A}$ and $r \in \mathcal{R}$. It specifies that resource r is being reassigned to agent a (which agent held r before the deal is left unspecified). Each of these atomic deals induces a binary relation $R_{a\leftarrow r}$ over the set of allocations $\mathcal{A}^{\mathcal{R}}$: given two allocations x and y, we have $xR_{a\leftarrow r}y$ iff x and y are identical except possibly for the assignment of resource r which must be assigned to agent a in allocation y.

Each agent $i \in \mathcal{A}$ is equipped with a preference relation R_i over alternative *bundles* of resources: $R_i \subseteq 2^{\mathcal{R}} \times 2^{\mathcal{R}}$. We require preference relations to be *reflexive* and *transitive* (but not necessarily monotonic, for instance). Each R_i extends to a preference relation over alternative *allocations* of resources: for allocations $A, A' \in \mathcal{A}^{\mathcal{R}}$, we have $(A, A') \in R_i$ iff $(\{r \in \mathcal{R} \mid A(r) = i\}, \{r \in \mathcal{R} \mid A'(r) = i\}) \in R_i$. That is, agent i prefers allocation A' over allocation A iff they prefer the bundle they receive in A' over the bundle they receive in A. While the R_i are defined in terms of bundles, we are mostly going to use them in this derived form, as relations over allocations. Union ($\cup$), intersection ($\cap$), complement ($\overline{R}$), converse (R^{-1}), and iteration (R^*) of relations are defined in the usual manner.

2.2 Syntax

Atomic propositions. Let At be a finite or countable set of atomic propositions, including the special symbols H_{ij} for all $i \in \mathcal{A}$ and all $j \in \mathcal{R}$. The intended meaning of H_{ij} is that agent i holds resource j.

Relations and formulas. We first define the range of terms that can be used to index a modal operator, and then the set of formulas itself. We assume there is a set of atomic relation terms, one for each atomic deal relation and one for each preference relation. We will use the same symbol to represent both a relation term and the relation. We trust this abuse of notation will not cause any confusion. A relation term has the following syntactic form:

$$R \ ::= \ r \mid R \cup R' \mid R \cap R' \mid R^{-1} \mid \overline{R} \mid R^*,$$

where r is an atomic relation of the form $R_{a\leftarrow r}$ or R_i. Formulas have the following syntactic form:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \psi \mid \langle R\rangle\varphi,$$

where $p \in \mathsf{At}$ and R is a relation term. Further logical operators, such as conjunction, can be defined in terms of the above in the usual manner. The box-operator, in particular, is defined as the dual of the diamond: $[R]\varphi = \neg\langle R\rangle\neg\varphi$.

2.3 Semantics

Frames. A *frame* $\mathcal{F} = (\mathcal{A}, \mathcal{R}, \{R_i\}_{i\in\mathcal{A}})$ is a triple consisting of a set of agents $\mathcal{A}$, a set of resources $\mathcal{R}$, and a set of preference relations R_i over allocations, one for each agent. This would corresponds to the frame $(\mathcal{A}^{\mathcal{R}}, \{R_i\}_{i\in\mathcal{A}})$ in the standard Kripke semantics for a multi-modal logic; that is, the "worlds" in a frame are allocations of resources. Note that the deal relations $R_{a\leftarrow r}$ are fully specified by $\mathcal{A}$ and $\mathcal{R}$ already, so these need not be specified as relations of the frame.

Models. A model $\mathcal{M} = (\mathcal{F}, V)$ is a pair consisting of a frame $\mathcal{F} = (\mathcal{A}, \mathcal{R}, \{R_i\}_{i\in\mathcal{A}})$ and a valuation function V mapping atomic propositions to subsets of $\mathcal{A}^{\mathcal{R}}$. Intuitively, $V(p)$ will be the set of allocations at which the proposition p is true. V has to respect the condition $V(H_{ij}) = \{A \in \mathcal{A}^{\mathcal{R}} \mid A(j) = i\}$. That is, H_{ij} is true in exactly those allocations where agent i holds resource j.

Truth in a model. Truth of a formula φ at a world w (an allocation) in a given model $\mathcal{M}$ is defined as follows:

(1) $\mathcal{M}, w \models p$ iff $w \in V(p)$ for atomic propositions p;
(2) $\mathcal{M}, w \models \neg\varphi$ iff not $\mathcal{M}, w \models \varphi$;
(3) $\mathcal{M}, w \models \varphi \vee \psi$ iff $\mathcal{M}, w \models \varphi$ or $\mathcal{M}, w \models \psi$;
(4) $\mathcal{M}, w \models \langle R\rangle\varphi$ iff there is a $v \in \mathcal{A}^{\mathcal{R}}$ such that wRv and $\mathcal{M}, v \models \varphi$.

For instance, $\langle R_i\rangle\varphi$ means that φ is true in some allocation that agent i prefers over the current allocation. Notions such as validity and satisfiability are defined in the usual manner [7, 8]. The formula $[R_{a\leftarrow r_1} \cup R_{a\leftarrow r_2}]\varphi$, for instance, expresses that in every allocation that we can reach by giving either item r_1 or item r_2 to agent a satisfies φ.

2.4 Decidability

Next we are going to show that the logic $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$ is decidable. This may seem surprising at first, given the close connection of our logic to PDL extended with the complement operator, which is known to be undecidable (see appendix). In short, the reason why $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$ is decidable is that, for this logic, fixing the language of formulas involves fixing the set $\mathcal{A}$ of agents and the set $\mathcal{R}$ of resources. This is turn amounts to fixing the set of possible worlds of our models.

Proposition 1 (Decidability). *The logic $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$ is decidable.*

Proof. A formula φ in the language of $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$ is valid iff it is true at every world in every model of $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$. The number of frames of $\mathcal{L}_{\langle\mathcal{A},\mathcal{R}\rangle}$ is finite: $\mathcal{A}$ and $\mathcal{R}$ are fixed and the number of choices for each preference relation R_i is bound above by the square of the number of bundles of resources from $\mathcal{R}$. The definition of the valuation function over atomic propositions not appearing in φ is not relevant, so we only need to consider a finite number of valuation functions, and hence a finite number of models. Each of these models is itself finite, and checking whether φ is true at a given world in a given model is a decidable problem. Hence, checking validity amounts to deciding a finite number of decidable problems, so it must be a decidable problem itself. □

2.5 Examples

We are now going to give a couple of examples that demonstrate what can be expressed in our logic $\mathcal{L}_{\langle \mathcal{A},\mathcal{R}\rangle}$ of negotiation spaces.

Describing bundles and allocations. Formulas of the following form completely specify the bundle held by agent i (there is one such formula for each $X \subseteq \mathcal{R}$):

$$\textsc{bun}_i^X = \bigwedge_{j \in X} H_{ij} \ \wedge \bigwedge_{j \in \mathcal{R} \setminus X} \neg H_{ij} \tag{1}$$

Conjunctions of such BUN-formulas (with one conjunct for each $i \in \mathcal{A}$) completely specify an allocation. Let $\langle X_1, \ldots, X_n \rangle$ be a partitioning of the set of resources $\mathcal{R}$. The following formula identifies the corresponding allocation:

$$\textsc{alloc}_{\langle X_1,\ldots,X_n\rangle} = \bigwedge_{i=1}^{n} \textsc{bun}_i^{X_i} \tag{2}$$

Given our semantics, any such ALLOC-formula will be true in exactly one world (by definition); that is, these formulas have a similar role as *nominals*, familiar from hybrid logic [7]. In fact, an alternative approach would have been to introduce a nominal for each allocation, and to define the propositions H_{ij} in terms of these nominals, rather than giving the H_{ij} a special status.

No externalities. In our definition of the preference relations R_i we have stipulated that they should be free of externalities by defining them as being induced by preferences over bundles. Next we are going to see that this could in fact also be defined syntactically; that is, we may define the R_i as preference relations over allocations and additionally impose axioms that exclude the option of externalities, when this is desired. We first define a modality that allows us to move to any world in the model from any given starting point. This is possible, because all worlds (allocations) can be reached by a sequence of atomic deals (as long as no conditions on the acceptability of a deal are being imposed).

$$[*]\varphi = [(\bigcup_{a \in \mathcal{A}, r \in \mathcal{R}} R_{a \leftarrow r})^*]\varphi \tag{3}$$

Since any two states of our model are connected via a finite sequence of deals, $[*]$ is a universal modality. That is, $[*]\varphi$ is true at a state provided φ is true at *every* state in the model.

Intuitively, the preferences depend only on the bundles if, whenever there is a situation in which agent i prefers bundle Y over bundle X, then whenever the agent has bundle X, then the agent prefers a situation in which it has bundle Y. With the help of the universal modality we can express this as follows:

$$(\textsc{bun}_i^X \wedge \langle R_i \rangle \textsc{bun}_i^Y) \rightarrow [*](\textsc{bun}_i^X \rightarrow \langle R_i \rangle \textsc{bun}_i^Y) \tag{4}$$

The conjunction of the above type of implication for all bundles $X, Y \in 2^{\mathcal{R}}$ would then describe the fact that preferences only depend on bundles (no externalities).

3 Convergence to a Pareto Optimal Allocation

A central question in negotiation concerns *convergence* [1, 2, 3]: under what circumstances can we be sure that any sequence of deals negotiated by the agents will eventually lead to an allocation with certain desirable properties? Such "desirable properties" are usually expressed in terms of an aggregation of the preferences of the individual agents. A fundamental criterion for economic efficiency is the concept of Pareto optimality: an allocation of resources is *Pareto optimal* iff there is no other alternative that would be strictly better for one agent without being worse for any of the others [9]. In this paper, we are going to be interested under what circumstances a sequence of deals can be guaranteed to converge to a Pareto optimal allocation of resources. More specifically, in this section, we are going to reconstruct a result of [2], which may be paraphrased as stating that any sequence of deals that are beneficial for all the agents involved and that are not subject to any structural restrictions (say, on the number of agents involved in a single deal), will eventually result in a Pareto optimal allocation.

We are now going to formalise this result as a formula of $\mathcal{L}_{\langle \mathcal{A}, \mathcal{R} \rangle}$. This formula will have the following general structure: $[\Phi^*]\langle \Phi^* \rangle \textsc{opt}$. Here Φ stands for the union of all deals that are possible and OPT is a formula describing that the allocation in question is "optimal". So the formula says that for any initial allocation, if we implement any sequence of Φ-deals, we can always reach an optimal allocation by implementing a further such sequence (or we are already at the optimal allocation).

To instantiate this template to a concrete formula, we first need to say what it means for a deal (a move to another allocation) to be "beneficial" (or *rational*) for everyone involved. For this we use the notion of Pareto improvement. We first need to define an agent's strict preference. Given any preference R_i, we can define its strict version, R_i^s as follows. For allocations w and v, say that $wR_i^s v$ if $wR_i v$ and it is not the case that $vR_i w$. Thus,

$$R_i^s = R_i \cap \overline{R_i^{-1}} \tag{5}$$

Thus the intended interpretation of $\langle R_i^s \rangle \varphi$ is that φ is true at an alternative which agent i *strictly* prefers to the current state.

We can now define a relation, denoted PAR, with intended interpretation of $\langle \textsc{par} \rangle \varphi$ being that φ is true at an alternative which is a Pareto improvement to the current alternative. Formally, we define PAR as follows:

$$\textsc{par} = \bigcap_{i \in \mathcal{A}} R_i \cap \bigcup_{i \in \mathcal{A}} R_i^s \tag{6}$$

Now if $\mathcal{M}, w \models [\textsc{par}]\bot$, then w is an "end-state" with respect to the PAR relation. Thus, there is no state which is a Pareto improvement over w. In other words, w is *Pareto efficient*.

Requiring deals to be rational is one way of restricting the range of possible deals. Another form of restriction are *structural* constraints. For instance, a particular negotiation protocol may only permit agents to negotiate bilateral

deals (deals involving only two agents each), or there may be an upper limit on the number of resources that can be reassigned in a single deal. Let D be the set of deals licensed by our negotiation protocol. For instance, D could be the set of all *atomic* deals:

$$D = \bigcup_{a \in \mathcal{A}, r \in \mathcal{R}} R_{a \leftarrow r} \tag{7}$$

Another option would be to define D as the set of *all* deals (observe that every deal can be implemented as a sequence of atomic deals):

$$D = (\bigcup_{a \in \mathcal{A}, r \in \mathcal{R}} R_{a \leftarrow r})^* \tag{8}$$

We should note that, of course, not every restriction of interest can be expressed using our language for describing deals. This is due to the fact that we define atomic deals in terms of a single resource and the agent receiving that resource, but we do not specify from which other agent that resource is being taken.

The set of deals that are both rational and subject to the structural constraints defining D are given by the intersection $D \cap \textsc{par}$. Sequences of such deals belong to $(D \cap \textsc{par})^*$. We can now state the convergence property:

$$[(D \cap \textsc{par})^*]\langle (D \cap \textsc{par})^* \rangle [\textsc{par}]\bot \tag{9}$$

This formula expresses that any sequence of deals that are rational and belong to D will either lead to a Pareto optimal allocation, or to an allocation from which a Pareto optimal allocation is still reachable by means of such a sequence. In case we also know that any such sequence is bound to *terminate*, then this reduces to every sequence of rational D-deals eventually resulting in a Pareto optimal allocation of resources. For D being the full set of deals (without any structural restrictions), this has been proved to hold in [2]. Hence, formula (9) with D being the full set of deals must be valid in our logic $\mathcal{L}_{\langle \mathcal{A}, \mathcal{R} \rangle}$.

We can see this also as follows. If D is the full set of deals, *i.e.* D is defined by equation (8), then D is a universal relation, linking any two allocations in $\mathcal{A}^{\mathcal{R}}$. Hence the intersection $D \cap \textsc{par}$ is actually just the relation $\textsc{par}$. It is not difficult to see (and we are going to explain precisely why in the following section), that $\textsc{par}$ must be a transitive relation. Hence, $\textsc{par}^*$ is just the reflexive closure of $\textsc{par}$. Thus formula (9) reduces to the formula $[\textsc{par}^*]\langle \textsc{par}^* \rangle [\textsc{par}]\bot$. Observe that this formula is valid on a given frame iff the following is:

$$[\textsc{par}]\bot \vee \langle \textsc{par} \rangle [\textsc{par}]\bot \tag{10}$$

That is, either we are already at a Pareto efficient state or there is a $\textsc{par}$-path that leads to a Pareto efficient state. Thus our convergence theorem reduces to a statement purely about Pareto improvements, which can be expressed in a fragment of our logic in which the modalities contain only preference relation symbols. Since this logic may be of independent interest, we treat it in detail in the next section.

4 The Logic of Pareto Efficiency

The goal of this section is to develop a logic of Pareto efficiency. We start with an arbitrary set of alternatives W and assume each agent has a (reflexive and transitive) preference over W. This is the setting of a recent paper by van Benthem *et al.* [10]. In fact, studying preferences from a logical perspective has been studied by a number of different authors (cf. Hansson [11]). Of course, since each R_i is assumed to be reflexive and transitive, the class of all preference models is axiomatized by multi-agent **S4**. Van Benthem *et al.* [10] show that taking the above language as a starting point, a number of different game-theoretic notions, such as the Nash equilibrium and the backward induction solution, can be expressed and studied from a modal preference logic point of view. To that end, standard tools from extended modal logic, such as nominals, dynamic epistemic operators, and the universal modality, are used. The logic presented in this section continues this line of thinking.

Let At be a finite or countable set of atomic propositions. The language of the logic $\mathcal{L}_{\mathsf{Pareto}}$ of Pareto efficiency is defined as follows (with $p \in \mathsf{At}$):

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle R_i\rangle\varphi \mid \langle R_i^s\rangle\varphi \mid \langle \textsc{par}\rangle\varphi$$

The standard boolean connectives and the operators $[R_i]$, $[R_i^s]$ and $[\textsc{par}]$ are defined as usual. Truth in a model is defined as usual. Here we are working in a multi-modal language interpreted over standard Kripke structures in which the accessibility relation for each $\langle R_i^s\rangle$ and the $\langle \textsc{par}\rangle$ modal operator are defined in terms of the R_i relations. This is analogous to working in a multi-agent epistemic logic with a common knowledge operator (in this case, the accessibility for the common knowledge operator is defined to be the reflexive transitive closure of the union of the individual accessibility relations). Recall the definitions of R_i^s and PAR from the previous section. Putting everything together, a *preference model* is a tuple $(W, \{R_i\}_{i\in\mathcal{A}}, V)$ where each R_i is reflexive and transitive, and the R_i^s and PAR relations are defined as above.

For issues of decidability and axiomatization it will be convenient to interpret the above language as a fragment of PDL with converse, intersection and complement operators. In this case, each R_i is an atomic program, and the modalities $\langle R_i^s\rangle$ and $\langle \textsc{par}\rangle$ can be defined by the appropriate operations on the R_i. See the appendix for a discussion of the relevant issues. We end this section with two simple observations.

Observation 1. *If each R_i is transitive, then* PAR *is transitive.*

Proof. Suppose that $w\textsc{par}v$ and $v\textsc{par}z$. By transitivity of the R_i, it is easy to see that $(w, z) \in \bigcap_i R_i$. Since $v\textsc{par}z$, there is some agent i such that vR_iz but not zR_iv. Our claim is that not zR_iw. Suppose that zR_iw. Then by transitivity of R_i, since zR_iw and wR_iv, zR_iv which contradicts our assumption. □

Consider again formula (10): $[\textsc{par}]\bot \vee \langle \textsc{par}\rangle[\textsc{par}]\bot$. Intuitively, this formula will be true at an alternative w provided either w is Pareto efficient or there is

a Pareto improvement v that is. That is, $\mathcal{M}, w \models [\text{PAR}]\bot \vee \langle \text{PAR} \rangle [\text{PAR}]\bot$ just in case either there is no v such that $w\text{PAR}v$ or $w\text{PAR}v$ and v is an "end state". Our last observation is that assuming W is finite, this formula is valid.

Observation 2. *Suppose that W is finite and $\mathcal{M} = (W, \{R_i\}_{i \in \mathcal{A}}, V)$ is a preference model. Then for each $w \in W$, we have $\mathcal{M}, w \models [\text{PAR}]\bot \vee \langle \text{PAR} \rangle [\text{PAR}]\bot$.*

Proof. The proof follows easily from the fact that PAR is irreflexive and W is assumed to be finite. Under these assumptions it is easy to see that for each state $w \in W$, if w is not an PAR end state, then it is PAR accessible to an PAR end state. That is, for each $w \in W$, either there is no state v such that $w\text{PAR}v$ or there is a state $v \in W$ such that $w\text{PAR}v$ and for each $v' \in W$, it is not the case that $v\text{PAR}v'$. This is precisely what it means to say that $\mathcal{M}, w \models [\text{PAR}]\bot \vee \langle \text{PAR} \rangle [\text{PAR}]\bot$. □

From a modal logic perspective, these observations are easy exercises. However, from the perspective of this paper, they demonstrate that modal logic, and in particular variants of PDL, can provide an interesting perspective on negotiation.

5 Discussion

In this section we are going to explore further connections between different types of reasoning tasks in our logic $\mathcal{L}_{\langle \mathcal{A}, \mathcal{R} \rangle}$ and questions arising in the context of negotiation.

5.1 Necessity of Complex Deals and Satisfiability

Besides convergence, another important property of negotiation systems that has been studied in the literature concerns the *necessity* of specific deals [1, 2]. A given deal or class of deals, characterised by structural constraints (rather than rationality conditions), is said to be necessary in view of reaching an allocation with a certain desired property (such as being Pareto optimal) by means of rational deals iff there are an initial allocation and individual preference relations such that any path leading to such a desirable allocation would have to involve that particular deal. A known result [2] states that if you do not allow all structural types of deals, but do require rationality, then you cannot guarantee Pareto optimal outcomes in all cases. In this section, we are going to discuss what this result corresponds to in our logic $\mathcal{L}_{\langle \mathcal{A}, \mathcal{R} \rangle}$.

Consider again our convergence formula (9). The claim is that, if the set of deals D excludes even a single deal, then formula (9) will cease to be valid. In other words, its negation will become satisfiable:

$$\neg[(D \cap \text{PAR})^*]\langle (D \cap \text{PAR})^* \rangle [\text{PAR}]\bot \tag{11}$$

The proof of the necessity theorem given in [2] amounts to giving a general algorithm for constructing individual preference relations and an initial allocation such that the one deal not included in D will be the only deal taking us from

the initial allocation to the (only) allocation that Pareto-dominates the initial allocation. This constructive element of the proof would correspond to giving a general method for proving satisfiability of formula (11). Vice versa, the known necessity theorem shows that formula (11) must be satisfiable for any given set of deals D that is not the full set of complex deals.

The discussion of necessity theorems highlights the fact that the exact form of presentation chosen for specifying deals can lead to somewhat different results. In [2] deals are represented as pairs of allocations, which amounts to a more fine-grained representation than we have opted for in this paper. For example, the deal $R_{a \leftarrow r}$ does in fact represent n different deals: for any of the n agents (including a itself), that agent could have owned r before the deal. If the more fine-grained representation is chosen, then certain deals need to be excluded from the statement of the theorem: a deal that is *independently decomposable* (meaning there are two groups of agents involved in the deal, but not a single resource is changing group) is not necessary for convergence, but can always be decomposed into two smaller deals. If deals are specified in terms of reassignments, as in this paper, however, each such deal does in fact correspond to a class of deals involving both independently decomposable deals and deals that are not independently decomposable. Hence, excluding that whole class from the negotiation protocol will always cause a problem, and therefore any such deal must be necessary.

5.2 Reachability Properties and Model Checking

Recall the formulation of the convergence property as given by formula (9). It states that any sequence of rational D-deals will eventually result in a Pareto optimal allocation (or in an allocation from which a Pareto optimal allocation is still accessible by means of such a sequence). We have seen that the formula is valid if D is the full set of deals, and that it is not valid if D is any subset of the full set of deals (that is, every single deal is necessary).

Dunne and colleagues [3, 12] have studied the complexity of deciding whether a given negotiation scenario allows for convergence to an optimal allocation by means of a structurally restricted class of (rational) deals. To be precise, these authors have concentrated on a framework where agent preferences are represented using utility functions (rather than ordinal preference relations) and where an allocation is considered optimal if it maximises the sum of individual utilities (so-called utilitarian social welfare [9]), a notion that is stronger than Pareto optimality. Nevertheless, conceptually there are interesting parallels to be explored.

This problem of deciding whether a given negotiation scenario admits convergence for a given restricted class of deals amounts to a *model checking* problem in our logic. This is interesting for at least two reasons. Firstly, model checking as a well-developed algorithmic technique may turn out to be a useful tool for deciding such questions in practice. Secondly, it may be of interest to compare and relate complexity results for negotiation frameworks and PDL model checking. A discussion of the latter may be found in the appendix. As shown by

Lange [13], model checking is PTIME-complete for all conceivable extensions of PDL (e.g. with intersection). It is important to note, however, that such complexity results must be understood with respect to the number of worlds in a model. In our case (as in many other applications), this will be an exponential number. Dunne and Chevaleyre [12] have recently shown that deciding whether a given negotiation scenario admits convergence by means of rational atomic deals is PSPACE-complete for the "numerical" version of the problem (with utility functions). A deeper understanding of the exact relationship between the two problems may allow us to obtain complexity results for model checking in our logic expressed in terms of the numbers of agents and resources (rather than the exponential number of allocations).

5.3 Guaranteed Convergence and Correspondence Theory

While Dunne *et al.* [3] have concentrated on establishing complexity results for deciding when convergence is possible, another line of work has attempted to establish general conditions (on the preferences of individual agents) that would guarantee that convergence by means of structurally simple deals is always possible [2, 14]. These results mostly relate to the numerical negotiation framework (with utility functions, monetary side payments, and maximal utilitarian social welfare as the chosen notion of optimality). Also, these results are either very simple (for instance, if all agents use modular utility functions, then convergence to an optimal allocation can be guaranteed by rational atomic deals alone) or require an overly complex specification of conditions. Here the logic-based representation of the problem promises to offer some real help in identifying further interesting cases of guaranteed convergence.

This kind of question can be cast as a question in modal logic *correspondence theory* [7]. Suppose we want to identify suitable conditions on agent preferences that would allow us to guarantee convergence by means of rational deals all belonging to a class of deals D. Then we have to identify a class of frames on which formula (9) would be valid. Again, this is an issue we put forward for detailed investigation in the future.

References

1. Sandholm, T.W.: Contract types for satisficing task allocation: I Theoretical results. In: Proc. AAAI Spring Symposium: Satisficing Models. (1998)
2. Endriss, U., Maudet, N., Sadri, F., Toni, F.: Negotiating socially optimal allocations of resources. Journal of Artificial Intelligence Research **25** (2006) 315–348
3. Dunne, P.E., Wooldridge, M., Laurence, M.: The complexity of contract negotiation. Artificial Intelligence **164**(1–2) (2005) 23–46
4. Parikh, R.: Social software. Synthese **132** (2002) 187–211
5. Pacuit, E., Parikh, R.: Social interaction, knowledge, and social software. In: Interactive Computation: The New Paradigm. Springer-Verlag (forthcoming)
6. Pauly, M., Wooldridge, M.: Logic for mechanism design: A manifesto. In: Proc. 5th Workshop on Game-theoretic and Decision-theoretic Agents. (2003)

7. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2002)
8. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Boston (2000)
9. Moulin, H.: Axioms of Cooperative Decision Making. Cambridge University Press, Cambridge (1988)
10. van Benthem, J., van Otterloo, S., Roy, O.: Preference logic, conditionals and solution concepts in games. In: Modality Matters: Twenty-Five Essays in Honour of Krister Segerberg. University of Uppsala (2006)
11. Hansson, S.O.: Preference logic. In: Handbook of Philosophical Logic. 2nd edn. Kluwer Academic Publishers (2001)
12. Dunne, P.E., Chevaleyre, Y.: Negotiation can be as hard as planning: Deciding reachability properties of distributed negotiation schemes. Technical Report ULCS-05-009, Department of Computer Science, University of Liverpool (2005)
13. Lange, M.: Model checking propositional dynamic logic with all extras. Journal of Applied Logic **4**(1) (2005) 39–49
14. Chevaleyre, Y., Endriss, U., Lang, J., Maudet, N.: Negotiating over small bundles of resources. In: Proc. 4th International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press (2005)
15. Passy, S., Tinchev, T.: An essay in combinatory dynamic logic. Information and Computation **93**(2) (1991) 263–332
16. Balbiani, P., Vakarelov, D.: Iteration-free PDL with intersection: A complete axiomatization. Fundamenta Informaticae **45** (2001) 1–22
17. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**(2) (1979) 194–211
18. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: Proc. 17th Annual Symposium on Foundations of Computer Science, IEEE (1976) 109–121
19. Lutz, C., Walther, D.: PDL with negation of atomic programs. Journal of Applied Non-Classical Logics **15**(2) (2005) 189–214
20. Danecki, R.: Non-deterministic propositional dynamic logic with intersection is decidable. In: Proc. 5th Workshop on Computation Theory. Springer-Verlag (1985)
21. Lange, M., Lutz, C.: 2-EXPTIME lower bounds for propositional dynamic logics with intersection. Journal of Symbolic Logic **70**(4) (2005) 1072–1086

A PDL and Its Extensions

In this short appendix we list the relevant results surrounding propositional dynamic logic and its extensions. Much of this information can be found in the textbook *Dynamic Logic* by Harel, Kozen and Tiuryn [8]. The reader is also referred to Passy and Tinchev [15] for more information.

Let At be a set of atomic propositions and Pr a set of atomic programs. Formulas and programs have the following syntactic form ($p \in \mathsf{At}$ and $r \in \mathsf{Pr}$):

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi \vee \psi \mid \langle\alpha\rangle\varphi$$

$$\alpha \ ::= \ r \mid \alpha \cup \beta \mid \alpha \cap \beta \mid \alpha;\beta \mid \alpha^* \mid \overline{\alpha} \mid \alpha^{-1}$$

Other connectives and operators are defined as usual. For example, $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$ and $[\alpha]\varphi = \neg\langle\alpha\rangle\neg\varphi$. Note that for simplicity we do not include the test-operator. Let $\mathcal{L}_{\mathsf{PDL}}$ be the set of all such well-formed formulas. Given an

arbitrary program α, we define relations R_α as usual [8]. Formulas are interpreted in Kripke structures $\mathcal{M} = (W, \{R_r\}_{r \in \mathsf{Pr}}, V)$ where each $R_r \subseteq W \times W$ and $V : \mathsf{At} \to 2^W$. Truth in a model is defined as usual (see Section 2.3 and [8]). A model $\mathcal{M}$ is called a PDL model provided $\mathcal{M}$ and the relations R_α for any program α are defined as above. By PDL we mean the set of all formulas which are valid in any PDL model. We now survey the main results relevant for our discussion in this paper.

Harel [8] showed that assuming that all atomic programs are deterministic, PDL with intersection is highly undecidable. However, the result is more positive if we allow for arbitrary (non-deterministic) atomic programs. Balbiani and Vakarelov [16] showed that PDL with intersection is axiomatizable with the use of an infinitary proof rule. Passy and Tinchev [15] prove a similar result using nominals. Early on it was shown by Fischer and Ladner [17] that the satisfiability problem for $\mathcal{L}_{\mathsf{PDL}}$ with respect to the class of all PDL models is decidable. Pratt [18] went on to show that it is EXPTIME-complete. It was observed by Harel [8] that the validity problem with complementation is undecidable. However, recently it was shown that allowing complementation of *atomic* programs only allows us to retain decidability.

Theorem 1 (Lutz & Walther [19]). *The satisfiability problem for $\mathcal{L}_{\mathsf{PDL}}$ with complement applied only to atomic programs is decidable.*

The satisfiability problem for $\mathcal{L}_{\mathsf{PDL}}$ (with or without complement) interpreted over PDL models in which the atomic programs are deterministic is Σ^1_1-complete. If the restriction to deterministic atomic programs is dropped then the situation becomes much more manageable.

Theorem 2 (Danecki [20]; Lange & Lutz [21]). *The satisfiability problem for $\mathcal{L}_{\mathsf{PDL}}$ with intersection (but without complement) is 2-EXPSPACE-complete.*

Finally, in a recent paper Lange [13] points out that model checking $\mathcal{L}_{\mathsf{PDL}}$ formulas remains in PTIME,

Theorem 3 (Lange [13]). *The model checking problem for $\mathcal{L}_{\mathsf{PDL}}$ with respect to PDL models is in PTIME.*

Returning to the logics presented in this paper, it is not hard to see that the language $\mathcal{L}_{\mathsf{Pareto}}$ is a fragment of $\mathcal{L}_{\mathsf{PDL}}$. The idea is to interpret each preference relation R_i as an atomic program. Then the operators $\langle \textsc{par} \rangle$ and $\langle R_i^s \rangle$ become definable in $\mathcal{L}_{\mathsf{PDL}}$. Of course, this interpretation uses the converse, complement and intersection operators. Thus as remarked above, in the presence of the complement operator, the validity problem for $\mathcal{L}_{\mathsf{PDL}}$ is undecidable. However, we are working in a fragment in which the complement operator is only applied to atomic and the converse of atomic programs. The logic $\mathcal{L}_{\langle \mathcal{A}, \mathcal{R} \rangle}$ is decidable due to the chosen semantics which fixes the set of possible worlds (cf. Proposition 1).

Representing Action Domains with Numeric-Valued Fluents*

Esra Erdem[1] and Alfredo Gabaldon[2,3]

[1] Institute of Information Systems, Vienna University of Technology, Vienna, Austria
[2] National ICT Australia
[3] School of Comp. Sci. and Eng., UNSW, Sydney, Australia

Abstract. We present a general method to formalize action domains with numeric-valued fluents whose values are incremented or decremented by executions of actions, and show how it can be applied to the action description language $\mathcal{C}+$ and to the concurrent situation calculus. This method can handle nonserializable concurrent actions, as well as ramifications on numeric-valued fluents, which are described in terms of some new causal structures, called contribution rules.

1 Introduction

Numeric-valued fluents are used for describing measurable quantities, such as weight, money, memory. In many cases, the values of such fluents are incremented/decremented by the execution of actions, such as adding/removing some weight, depositing/withdrawing some money, or allocating/deallocating memory. How to compute the value of a numeric-valued fluent after a concurrent execution of several such actions, possibly with indirect effects, is the question we study in this paper. We consider true concurrency: actions occur at the same time and may not be serializable (i.e., their effect may not be equivalent to the effect of executing the same actions consecutively in any order). For instance, consider two boats towing a barge upriver by applying forces via cables tied to the barge, where the force applied by either boat is not enough to move the barge against the current of the river; here the concurrent action of two boats applying forces can not be serialized. True concurrency makes the problem more challenging, because actions that are individually executable may not be executable concurrently, e.g., due to conflicting effects, and actions that are individually nonexecutable may be executable concurrently, e.g., due to synergistic effects, like in the example above.

This question is important for real-world applications that involve reasoning tasks, like planning or prediction, related to resource allocation. For instance, allocation of memory storage for use by computer programs is one such application. It is also important for applications that involve modeling the behavior of physical systems. For instance, how water pressure changes at a piston when some water is pumped from above and some force is applied from the bottom is important for modeling the behavior of a hydraulic elevator.

* We thank Selim T. Erdoğan, Joohyung Lee, and Vladimir Lifschitz for helpful comments on an earlier version of the paper. Esra Erdem was supported in part by the Austrian Science Fund (FWF) under project P16536-N04.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 151–163, 2006.

There are several planning systems designed to work in concurrent domains with resources, like [1, 2, 3]. However, they consider a simpler concurrency: they either require the serializability of actions, or that no concurrent action contain two actions, one producing and the other consuming the same resource.

Lee and Lifschitz [4] show, in the action language $\mathcal{C}+$ [5], how to formalize action domains involving additive fluents—numeric-valued fluents on which the effect of a concurrent action is computed by adding the effects of its primitive actions. However, since additive fluents range over finite sets, a concurrent action is executable only if its effect on each additive fluent is in that fluent's range, and it is not easy to handle indirect effects of actions (ramifications) on additive fluents (e.g., an indirect effect of adding too much water into a small container is an increase in the amount of water in a large container, into which the excess water overflows from the small container). Similarly, [6] defines the cumulative direct effects of concurrent actions on additive fluents, in an extension of the action language $\mathcal{A}$ [7]; however, it is not easy to handle ramifications (not only the ones on numeric-valued fluents) in this formalism.

In [8], the authors show, in the concurrent situation calculus [9], how to formalize action domains containing numeric-valued fluents, that do not require serializability of actions, and that take into account ramifications caused by too much increase/decrease of a numeric-valued fluent. However, with this formalization, it is not easy to capture other forms of ramifications (e.g., whenever the amount of water increases in the large container, the force towards the bottom of the container increases).

In this paper, we present a general method to formalize action domains with numeric-valued fluents whose values are incremented/decremented by executions of actions. This method is applicable to both the concurrent situation calculus and the action language $\mathcal{C}+$; and thus can be used with the reasoning systems CCALC and GOLOG. The idea is to compute the total effect of a concurrent action on a numeric-valued fluent, in terms of the direct and indirect effects of its primitive actions on that fluent, while also taking into account the range restrictions (e.g., the capacity of the small container).

To describe direct effects, like in [4, 8], we introduce new constructs and functions in the original formalisms. To describe ramifications, like in [10, 11, 12], we introduce an explicit notion of causality, specific for numeric-valued fluents. We characterize this notion by *contribution rules*, motivated by the equation-like causal structures of [13, 14, 8]. With contribution rules, both forms of ramifications above can be handled. The idea of introducing these new constructs is to be able to represent effects of actions on numeric-valued fluents concisely. Semantically these constructs are treated as "macros" on top of the original formalisms; like the constructs introduced in [4] and in [8], they are compiled into causal laws or formulas in the original formalisms.

The paper consists of three parts. The first two parts describe how action domains with numeric-valued fluents can be formalized in the action language $\mathcal{C}+$ and in the concurrent situation calculus, using the new constructs; the semantics of these constructs is defined by showing how to treat them as abbreviations in the original formalims. The third part includes a comparison of these two formalizations, and a discussion of related work. We refer the reader to [5] and [9] for descriptions of the action language $\mathcal{C}+$ and the concurrent situation calculus. For the proofs, and the CCALC and GOLOG files describing our running example, see `http://www.kr.tuwien.ac.at/staff/esra/papers/cr.pdf`.

2 Describing Additive Fluents in the Action Language $\mathcal{C}$+

To formalize action domains with additive fluents, we extend the action description language $\mathcal{C}$+, similar to [4].

Additive fluents. According to this extension, some numeric-valued fluent constants can be designated as *additive*. Each additive fluent constant has a finite set of numbers as its domain. As in [4], we understand numbers as symbols for elements of any set with an associative and commutative operation $+$ that has a neutral element 0; in particular, we consider the additive group of integers (since this case can be implemented for CCALC). We suppose that the domain of each additive fluent constant f is specified as a range $[L_f, U_f]$, so that, at any state, $L_f \leq f \leq U_f$. We suppose that heads of causal laws do not contain any additive fluent constants.

Direct effects of actions. Direct effects of a boolean action constant a on an additive fluent f are expressed by *increment laws* of [4], expressions of the form

$$a \textbf{ increments } f \textbf{ by } n \textbf{ if } \psi \tag{1}$$

where n is an integer and ψ is a fluent formula. We drop the '**if** ψ' part if $\psi \equiv \top$; we call f the *head* of the causal law. Intuitively, an increment law of form (1) expresses that, if ψ holds, the direct contribution of the action a to the value of the additive fluent f is n. The idea is then, to compute the cumulative direct contribution of concurrently executed primitive actions to the value of an additive fluent f, denoted *DContr*(f), by adding the direct contributions of those primitive actions to f. Translation of these laws into causal laws is different from that of [4] (see the definition of *DContr* in the next section).

Preconditions of actions. We describe preconditions of actions with the **nonexecutable** construct of [5]. For instance, the expression

$$\textbf{nonexecutable } \mathit{Move}(A, B) \textbf{ if } \neg \mathit{Clear}(B)$$

describes that moving Block A onto Block B is not possible if B is not clear.

Ramifications on additive fluents. Ramifications on an additive fluent f are described by *contribution rules*, expressions of the form:

$$f \xleftarrow{\oplus} \mathcal{E}(h) \tag{2}$$

where h is one of the additive fluents that f depends on, $\mathcal{E}$ is a numeric-valued function, and $\oplus$ is an element of $\{+, -, ++, +-, -+, --\}$; we call f the *head* of the rule. These rules allow us to describe both kinds of ramifications mentioned in the introduction. The first kind of ramifications is expressed with $\oplus = +$ or $\oplus = -$.

The meaning of a rule of form (2) with $\oplus = +$ (respectively, with $\oplus = -$) can be described as follows: whenever the sum of the direct and indirect contributions of a concurrent action to h, when added to h, exceeds the upper bound U_h (respectively, goes beyond its lower bound L_h), that action indirectly contributes to f by the amount $\mathcal{E}(\mathit{DContr}(h)+\mathit{IContr}(h)-\mathit{TContr}(h))$, where *IContr*$(h)$ denotes the indirect contribution of a concurrent action to h, and *TContr*(h) denotes the total contribution of a concurrent action to h respecting the range restriction $[L_h, U_h]$. Intuitively, *DContr*(h) + *IContr*(h) − *TContr*(h) describes the excess amount being contributed to h.

The other form of ramifications is expressed with $\oplus \in \{++, +-, -+, --\}$. A rule of form (2) with $\oplus = ++$ (respectively, with $\oplus = +-$) expresses that whenever there is an increase (respectively, decrease) n in the value of h, i.e., $TContr(h) = n$, the value of f increases (respectively, decreases) by $\mathcal{E}(n)$; the rules with $\oplus \in \{-+, --\}$ are similar, but they specify a decrease in the value of f. This form of ramification, unlike the one above, is not due to the range restrictions imposed on the values of fluents, although these restrictions must be satisfied at all times.

The indirect contribution of an action to an additive fluent f is the sum of the increases/decreases described by the contribution rules with the head f.

Once the direct and indirect contributions of a concurrent action to an additive fluent f are computed, we can compute the total contribution of that action to f as follows. If f appears on the right hand side of a contribution rule of form (2) with $\oplus = +, -$, then we add $DContr(f)$ and $IContr(f)$, considering the range restriction $[L_f, U_f]$:

$$TContr(f) = \begin{cases} U_f - f & \text{if } DContr(f) + IContr(f) > U_f - f \\ L_f - f & \text{if } DContr(f) + IContr(f) < L_f - f \\ DContr(f) + IContr(f) & \text{otherwise.} \end{cases}$$

Otherwise, we do not need to consider the range restriction, and $TContr(f)$ is defined as $DContr(f) + IContr(f)$.

We consider action domains only where the causal influence among fluents is acyclic. Here is an example.

Example 1. Consider three containers, small, medium, and large, for storing water. The small container is suspended over the medium, and the medium container is suspended over the large so that, when the small (respectively, medium) container is full of water, the water poured into the small (respectively, medium) container overflows into the medium (respectively, large) container. Suppose that there are three taps: one directly above the small container, by which some water can be added to the containers from an external source, one on the small container, by which some water can be released into the medium container, and a third tap on the large container to release water to the exterior. Suppose also that one unit increase (respectively, decrease) of water in the large container increases (respectively, decreases) the amount of force applied downwards to the bottom of the large container by two units. Also assume that some force is exerted upwards at the bottom of the large container, e.g., by a piston, to lift it up.

A formalization of this action domain in the extended $\mathcal{C}$+ is presented in Figure 1. Here the additive fluent constants *Small*, *Medium*, and *Large* describe the amount of water in each container; *Force* describes the force exerted upwards at the bottom of the large container. The boolean action constant $AddS(n)$ describes the action of adding n units of water to the small container by opening the tap over it; $ReleaseS(n)$ and $ReleaseL(n)$ describe the action of releasing n units of water from the small, respectively large, container by opening its tap; and $Exert(n)$ represents the action of exerting n amount of force upwards.

Suppose that the range restrictions are specified as follows: $L_{Small} = L_{Medium} = L_{Large} = 0$, $L_{Force} = -8$, $U_{Small} = 2$, $U_{Medium} = 3$, $U_{Large} = 4$, $U_{Force} = 8$. If initially $Small = Medium = Large = 1$, $Force = -2$, then, after executing the concurrent action $c = \{AddS(8), ReleaseS(1), ReleaseL(2), Exert(8)\}$, the values of fluents are computed by CCALC as follows: $Small = 2, Medium = 3, Large = 4, Force = 0$.

Notation: n ranges over $\{Min, .., Max\}$ and a ranges over action constants.

Action constants:	Domains:
$AddS(n)$, $ReleaseS(n)$, $ReleaseL(n)$, $Exert(n)$	Boolean

Additive fluent constants:	Domains:
Small	$\{L_{Small}, .., U_{Small}\}$
Medium	$\{L_{Medium}, .., U_{Medium}\}$
Large	$\{L_{Large}, .., U_{Large}\}$
Force	$\{L_{Force}, .., U_{Force}\}$

Causal laws:

$AddS(n)$ **increments** *Small* **by** n
$ReleaseS(n)$ **increments** *Small* **by** $-n$
$ReleaseS(n)$ **increments** *Medium* **by** n
$ReleaseL(n)$ **increments** *Large* **by** $-n$
$Exert(n)$ **increments** *Force* **by** n

nonexecutable $AddS(n)$ **if** $AddS(n')$ $(n \neq n')$
nonexecutable $ReleaseS(n)$ **if** $ReleaseS(n')$ $(n \neq n')$
nonexecutable $ReleaseL(n)$ **if** $ReleaseL(n')$ $(n \neq n')$
nonexecutable $Exert(n)$ **if** $Exert(n')$ $(n \neq n')$

exogenous a

Contribution rules:

$Medium \xleftarrow{+} Small$ $\quad$ $Large \xleftarrow{+} Medium$
$Force \xleftarrow{+-} 2 \times Large$ $\quad$ $Force \xleftarrow{-+} 2 \times Large$

Fig. 1. Containers domain described in the extended C+

Indeed, the direct effect of c on *Small* is the sum of the direct contributions of its primitive actions (described by the increment laws with the head *Small*, in Figure 1): $DContr(Small) = 8 - 1 = 7$. Since there is no contribution rule with the head *Small*, in Figure 1, there is no ramification on it: $IContr(Small) = 0$. Since $Small + DContr(Small) + IContr(Small) = 7$ exceeds the capacity of the small container, the total contribution of c to *Small* is just the amount that fills the small container: $TContr(Small) = U_{Small} - Small = 2 - 1 = 1$. Then the value of *Small* after the execution of c is 2.

On the other hand, since the function $\mathcal{E}$ in $Medium \xleftarrow{+} Small$ is the identity function, the indirect contribution of c to *Medium* is the amount of the excess water overflown into the medium container: $DContr(Small) + IContr(Small) - TContr(Small) = 7 + 0 - 1 = 6$. Since the direct contribution of c to *Medium* is 1, the total contribution of c to *Medium* is just the amount that fills the medium container: $TContr(Medium) = 2$. Then, after the execution of c, $Medium = 3$.

Similarly, the direct and indirect contributions of c to *Large* can be computed as follows: $DContr(Large) = -2, IContr(Large) = 5$. Since *Large* does not appear on the right hand side of a contribution rule of form (2) with $\oplus = +, -$, the total contribution of c to *Large* is simply the addition of these two: $TContr(Large) = 3$. Then the value of *Large* after the execution of c is 4.

Since the total contribution of c to *Large* is 3, and since the function $\mathcal{E}$ in $Force \xleftarrow{-+} 2 \times Large$ is $(\lambda x.2 \times x)$, the indirect contribution of c to *Force* is $-(2 \times 3) = -6$.

Since the direct contribution of c to *Force* is +8, the total contribution of c to *Force* is 2. Therefore, the value of *Force* after the execution of c is 0.

3 Obtaining an Action Description

To obtain an action description in $\mathcal{C}$+ from a formalization of an action domain like in Figure 1, we translate increment laws, and contribution rules into causal laws as follows.

1. To describe the direct effects of primitive actions, first we introduce new action constants, $\mathit{Contr}(a, f)$, of sort integer, where a is an action constant and f is an additive fluent constant; an atom of the form $\mathit{Contr}(a, f) = v$ expresses that the action a contributes to f by the amount v. We define $\mathit{Contr}(a, f)$ to be 0 by default:

   $$\textbf{default } \mathit{Contr}(a, f) = 0.$$

   Then we replace every increment law (1) with

   $$\textbf{caused } \mathit{Contr}(a, f) = n \textbf{ if } a \wedge \psi.$$

2. To describe the cumulative effects of concurrent actions, we introduce new action constants, $\mathit{DContr}(f)$, $\mathit{IContr}(f)$, $\mathit{TContr}(f)$, of sort integer, where f is an additive fluent constant. Intuitively, an atom of the form $\mathit{DContr}(f) = v$ (respectively, $\mathit{IContr}(f) = v$) expresses that the direct (respectively, indirect) contribution of a concurrent action to f is v. An atom of the form $\mathit{TContr}(f) = v$ expresses that the total contribution of a concurrent action to f is v.

   We define $\mathit{DContr}(f)$ as follows:

   $$\textbf{caused } \mathit{DContr}(f) = \textstyle\sum_a v_a \textbf{ if } \bigwedge_a \mathit{Contr}(a, f) = v_a$$

   where $\mathit{Min} \leq \sum_a v_a \leq \mathit{Max}$.

   Let us denote by C the set of all contribution rules. We define $\mathit{IContr}(f)$ to be 0 by default:

   $$\textbf{default } \mathit{IContr}(f) = 0.$$

   Then we translate contribution rules in C into the causal laws:

   $$\begin{array}{l}\textbf{caused } \mathit{IContr}(f) = v \textbf{ if } v = \\ \quad \sum_{f \xleftarrow{+} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{IContr}(h) + \mathit{DContr}(h) - \mathit{TContr}(h)) \\ -\sum_{f \xleftarrow{-} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{IContr}(h) + \mathit{DContr}(h) - \mathit{TContr}(h)) \\ +\sum_{f \xleftarrow{++} \mathcal{E}(h) \in C, \mathit{TContr}(h) > 0} \mathcal{E}(\mathit{TContr}(h)) \\ +\sum_{f \xleftarrow{+-} \mathcal{E}(h) \in C, \mathit{TContr}(h) < 0} \mathcal{E}(\mathit{TContr}(h)) \\ -\sum_{f \xleftarrow{-+} \mathcal{E}(h) \in C, \mathit{TContr}(h) > 0} \mathcal{E}(\mathit{TContr}(h)) \\ -\sum_{f \xleftarrow{--} \mathcal{E}(h) \in C, \mathit{TContr}(h) < 0} \mathcal{E}(\mathit{TContr}(h)) \quad (\mathit{Min} \leq v \leq \mathit{Max}).\end{array}$$

   For instance, with the contribution rules in Figure 1, for *Medium*, we add

   $$\begin{array}{l}\textbf{caused } \mathit{IContr}(\mathit{Medium}) = v \textbf{ if} \\ \mathit{IContr}(\mathit{Small}) + \mathit{DContr}(\mathit{Small}) - \mathit{TContr}(\mathit{Small}) = v \ \ (\mathit{Min} \leq v \leq \mathit{Max}).\end{array}$$

If f appears on the right hand side of a contribution rule of form (2), then we define $\mathit{TContr}(f)$ by adding the direct and indirect contributions of actions, respecting the range restriction $[L_f, U_f]$:

caused $\mathit{TContr}(f)=v+v'$ **if** $\mathit{DContr}(f)=v \wedge \mathit{IContr}(f)=v'$ $(L_f \leq v+v'+f \leq U_f)$
caused $\mathit{TContr}(f)=U_f-f$ **if** $\mathit{DContr}(f)=v \wedge \mathit{IContr}(f)=v'$ $(v+v'+f > U_f)$
caused $\mathit{TContr}(f)=L_f-f$ **if** $\mathit{DContr}(f)=v \wedge \mathit{IContr}(f)=v'$ $(v+v'+f < L_f)$

such that the values assigned to $\mathit{TContr}(f)$ are in the range $[\mathit{Min}, \mathit{Max}]$. Otherwise, we define $\mathit{TContr}(f)$ simply by adding the direct and indirect contributions of actions, i.e., by the first set of causal laws above.

3. To determine the value of an additive fluent constant f after an execution of a concurrent action, we add

caused $f = v + v'$ **if** $\top$ **after** $f = v \wedge \mathit{TContr}(f) = v'$ $(\mathit{Min} \leq v+v' \leq \mathit{Max})$.

With the translation above, the meaning of an action description D in the extended $\mathcal{C}+$ can be represented by the transition diagram described by the action description D' obtained from D as described above (see [7] for a definition of a transition diagram). Then a query Q (in a query language, like $\mathcal{R}$ [7]), which describes a planning problem, a prediction problem, etc., is entailed by D if Q is entailed by D'. This allowed us to compute the values of additive fluents in Example 1 using CCALC.

4 Describing Additive Fluents in the Concurrent Situation Calculus

To formalize action domains with additive fluents, we extend the concurrent situation calculus, as in [8].

Additive fluents. According to this extension, some functional fluents that range over numbers (not necessarily integers) can be designated as *additive*. For each additive fluent f, we understand a given range $[L_f, U_f]$ as follows: in every situation s, $L_f \leq f(s) \leq U_f$.

Direct effects of actions. For describing direct effects of actions on additive fluents, we introduce a function $\mathit{contr}_f(\boldsymbol{x}, a, s)$ for each additive fluent f. Intuitively, $\mathit{contr}_f(\boldsymbol{x}, a, s)$ is the amount that the action a contributes to f when executed in situation s. In the following, free variables are implicitly universally quantified. We describe the direct effects of primitive actions on additive fluents by axioms of the form:

$$\kappa_f(\boldsymbol{x}, v, a, s) \supset \mathit{contr}_f(\boldsymbol{x}, a, s) = v \tag{3}$$

where $\kappa_f(\boldsymbol{x}, v, a, s)$ is a first-order formula whose only free variables are $\boldsymbol{x}, v, a, s$, doesn't mention function contr_g for any g, and s is its only term of sort situation. If there is no axiom (3) describing the effect of an action a on an additive fluent f, we assume that the direct contribution of a to f is zero. This assumption allows us to derive, for each function contr_f, a definitional axiom:

$$\mathit{contr}_f(\boldsymbol{x}, a, s) = v \equiv \kappa_f(\boldsymbol{x}, v, a, s) \ \vee v = 0 \wedge \neg(\exists v')\kappa_f(\boldsymbol{x}, v', a, s).$$

Notation: n, n', v are object (number) variables, s is a situation variable, a, a' are action variables, and c is a concurrent variable.

Action functions: $addS(n)$, $releaseS(n)$, $releaseL(n)$, $exert(n)$.

Additive fluent functions:	Ranges:
small	$[L_{small}, U_{small}]$
medium	$[L_{medium}, U_{medium}]$
large	$[L_{large}, U_{large}]$
force	$[L_{force}, U_{force}]$

Direct effect axioms:

$$(\exists n)[a = addS(n) \wedge v = n] \supset contr_{small}(a, s) = v$$
$$(\exists n)[a = releaseS(n) \wedge v = -n] \supset contr_{small}(a, s) = v$$
$$(\exists n)[a = releaseS(n) \wedge v = n] \supset contr_{medium}(a, s) = v$$
$$(\exists n)[a = releaseL(n) \wedge v = -n] \supset contr_{large}(a, s) = v$$
$$(\exists n)[a = exert(n) \wedge v = n] \supset contr_{force}(a, s) = v$$

Preconditions of actions:

$$Poss(a, s)$$
$$conflict(c, s) = (\exists n, n').[addS(n) \in c \wedge addS(n') \in c \wedge n \neq n'] \vee$$
$$[releaseS(n) \in c \wedge releaseS(n') \in c \wedge n \neq n'] \vee$$
$$[releaseL(n) \in c \wedge releaseL(n') \in c \wedge n \neq n'] \vee$$
$$[exert(n) \in c \wedge exert(n') \in c \wedge n \neq n']$$

Contribution rules:

$$medium \xleftarrow{+} small \qquad large \xleftarrow{+} medium$$
$$force \xleftarrow{+-} 2 \times large \qquad force \xleftarrow{-+} 2 \times large$$

Fig. 2. Containers domain described in the extended concurrent situation calculus

Preconditions of actions. We describe preconditions of primitive actions as in [9]. For preconditions of a concurrent action c, we describe by a formula $conflict(c, s)$ the conditions under which the primitive actions in c conflict with each other. This is required to handle cases where a set of primitive actions each of which is individually possible may be impossible when executed concurrently.

Ramifications on additive fluents. As in the language $\mathcal{C}$+, we consider two kinds of ramifications on numeric-valued fluents, and we express them by acyclic contribution rules (2), where f and h do not contain a situation term.

For instance, Figure 2 shows a formalization of the containers example in this extended version of the concurrent situation calculus. With such a formalization, we can compute the values of fluents, as in Example 1, using GOLOG.

5 Obtaining a Basic Action Theory

From a formalization of an action domain, like in Figure 2, we can obtain a basic action theory in the concurrent situation calculus as follows. In the following, as in [9], instead of axiomatizing sets, numbers, and arithmetic operations, we use them assuming their standard interpretation.

1. We consider the foundational axioms of [9].
2. From the preconditions of primitive actions, conflicts between actions, and range restrictions on additive fluents, we can formalize preconditions of a concurrent action c as in [8], by an axiom of the form

$$\begin{aligned}&\mathit{Poss}(c, s) \equiv\\ &\quad (\exists a)(a \in c) \wedge (\forall a \in c)\mathit{Poss}(a, s) \wedge \neg \mathit{conflict}(c, s) \wedge \mathcal{R}^1[RC(\mathit{do}(c, s))].\end{aligned}$$

Denoted by $\mathcal{R}^1[W]$ is a formula equivalent to the result of applying one step of Reiter's regression procedure [9] on W. We use $RC(s)$ to denote the conjunction of the range constraints on each additive fluent f (i.e., $\bigwedge_f L_f \leq f(s) \leq U_f$) conjoined with additional qualification constraints if given. By this way, a concurrent action is possible if it results in a situation that satisfies the range constraints on additive fluents. For Example 1,

$$\begin{aligned}&RC(s) = L_{\mathit{small}} \leq \mathit{small}(s) \leq U_{\mathit{small}} \wedge L_{\mathit{medium}} \leq \mathit{medium}(s) \leq U_{\mathit{medium}} \wedge\\ &\quad L_{\mathit{large}} \leq \mathit{large}(s) \leq U_{\mathit{large}} \wedge L_{\mathit{force}} \leq \mathit{force}(s) \leq U_{\mathit{force}}.\end{aligned}$$

3. From the direct effect axioms and contribution rules in such a formalization, we can derive successor state axioms for additive fluents by the same kind of transformation in [9], which is based on an explanation closure assumption.

   First, we express the cumulative effects of actions on f, by adding the direct and indirect contributions of actions on f, respecting the given range $[L_f, U_f]$. For each additive fluent f, we introduce three new functions: dContr_f, iContr_f, and tContr_f. Intuitively, $\mathit{dContr}_f(\boldsymbol{x}, c, s)$ describes the cumulative direct contributions of primitive actions in c at a situation s:

$$\mathit{dContr}_f(\boldsymbol{x}, c, s) = \sum_{a \in c} \mathit{contr}_f(\boldsymbol{x}, a, s).$$

The indirect contribution of a concurrent action c on f at a situation s is described by $\mathit{iContr}_f(\boldsymbol{x}, c, s)$, relative to a set C of contribution rules:

$$\begin{aligned}&\mathit{iContr}_f(\boldsymbol{x}, c, s) =\\ &\quad \sum_{f \xleftarrow{+} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{iContr}_h(\boldsymbol{y}, c, s) + \mathit{dContr}_h(\boldsymbol{y}, c, s) - \mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \sum_{f \xleftarrow{-} \mathcal{E}(h) \in C} \mathcal{E}(\mathit{iContr}_h(\boldsymbol{y}, c, s) + \mathit{dContr}_h(\boldsymbol{y}, c, s) - \mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &+ \sum_{f \xleftarrow{++} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) > 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &+ \sum_{f \xleftarrow{+-} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) < 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \sum_{f \xleftarrow{-+} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) > 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s))\\ &- \sum_{f \xleftarrow{--} \mathcal{E}(h) \in C, \mathit{tContr}_h(\boldsymbol{y}, c, s) < 0} \mathcal{E}(\mathit{tContr}_h(\boldsymbol{y}, c, s)).\end{aligned}$$

For instance, relative to the contribution rules in Figure 2:

$$\mathit{iContr}_{\mathit{medium}}(c, s) = \mathit{iContr}_{\mathit{small}}(c, s) + \mathit{dContr}_{\mathit{small}}(c, s) - \mathit{tContr}_{\mathit{small}}(c, s).$$

After defining direct and indirect contributions of actions on an additive fluent f, we can define the total contribution of actions as follows. If f appears on the right hand side of a contribution rule of form (2), then we add the direct and indirect contributions of actions respecting the range restriction $[L_f, U_f]$:

$$\mathit{tContr}_f(\boldsymbol{x}, c, s) = \begin{cases} U_f - f(\boldsymbol{x}, s) & \text{if } \mathit{sum}_f > U_f - f(\boldsymbol{x}, s) \\ L_f - f(\boldsymbol{x}, s) & \text{if } \mathit{sum}_f < L_f - f(\boldsymbol{x}, s) \\ \mathit{sum}_f & \text{otherwise} \end{cases}$$

where sum_f stands for $\mathit{dContr}_f(\boldsymbol{x}, c, s) + \mathit{iContr}_f(\boldsymbol{x}, c, s)$. Otherwise, the total contribution of actions is simply the sum of the direct and indirect contributions of actions, i.e., sum_f.

Finally, we define the successor state axiom for an additive fluent f:

$$f(\boldsymbol{x}, \mathit{do}(c, s)) = f(\boldsymbol{x}, s) + \mathit{tContr}_f(\boldsymbol{x}, c, s).$$

4. From the given action functions, we can obtain unique names axioms, like $\mathit{addS}(n) \neq \mathit{releaseS}(n')$, etc.
5. We suppose that a description of the initial world is given.

6 Comparing the Two Formalizations

We have described how to formalize an action domain with additive fluents, in two formalisms: the action language $\mathcal{C}$+ and the concurrent situation calculus. We can see in Figures 1 and 2 that two such formalizations look similar. In fact, under some conditions, a formalization D of an action domain in the extended version of $\mathcal{C}$+ and a description I of the initial world can be translated into an action theory $\mathit{sit}(D, I)$ in the extended version of the concurrent situation calculus, such that, for every additive fluent f and for every concurrent action c, the value of f after execution of c is the same according to each formalization.

Suppose that D consists of the following:

- additive fluent constants $F_1, \ldots, F_m$, each F_i with the domain $\{L_{F_i}, ..., U_{F_i}\}$ ($\mathit{Min} \leq L_{F_i}, U_{F_i} \leq \mathit{Max}$); and boolean action constants $A_1, \ldots, A_{m'}$;
- increment laws of form (1) where a is a boolean action constant, f is an additive fluent constant, n is an integer, and ψ is true;
- preconditions of actions of the form

$$\textbf{nonexecutable } a \textbf{ if } \psi \tag{4}$$

  where ψ is a conjunction of atoms that does not contain the action constant a.
- acyclic contribution rules of form (2).

Suppose that I consists of the following:

$$0 : F_i = N_i \quad (0 \leq i \leq m)$$

where N_i is an integer in the given range $\{L_{F_i}, \ldots, U_{F_i}\}$, expressing that, at time stamp 0, the value of F_i is N_i.

Then we can obtain $\mathit{sit}(D, I)$ from D and I as follows:

1. For each additive fluent constant $F_i \in D$, declare a corresponding unary additive fluent function $\mathrm{f_i}(s)$ with the range $[L_{\mathrm{f}_i}, U_{\mathrm{f}_i}]$. such that $L_{F_i} = L_{\mathrm{f}_i}$ and $U_{F_i} = U_{\mathrm{f}_i}$. For each boolean action constant $A_i \in D$, declare a corresponding nullary

action function A_i. For instance, for the fluent constant *Small* with the domain $\{L_{Small}, \ldots, U_{Small}\}$ in Figure 1, we declare in Figure 2 the fluent function *small* with the range $[L_{small}, U_{small}]$.

Schemas are frequently used in $\mathcal{C}$+ to represent a large number of constants or statements. For example, *AddS*(n) in the declarations part denotes the action constants *AddS*(*Min*), ..., *AddS*(*Max*). In a situation calculus representation, for such a set of action constants, we can introduce a single action function (e.g., *addS*(n)).

2. For each increment law A_i **increments** F_j **by** N in D, add the formula

$$[a = \mathrm{A}_i \wedge v = N] \supset \mathit{contr}_{\mathrm{f}_j}(a, s) = v. \tag{5}$$

With a function $\mathrm{A}_i(n)$, we can use a single formula to represent all of the formulas (5) for A_i, as seen in Figure 2.

3. Let NEX_F be the set of all causal laws (4) in D such that ψ is a fluent formula. Let $\psi(s)$ be the formula obtained from a fluent formula ψ by replacing every additive fluent atom $F_i = N$ by $\mathrm{f}_i(s) = N$. For each action constant A_i in D, add the formula

$$\mathit{Poss}(\mathrm{A}_i, s) \equiv \bigwedge_{(\textbf{nonexecutable } A_i \textbf{ if } \psi) \in NEX_F} \neg\psi(s).$$

If for every action constant A_i, the right hand side of the equivalence above is $\top$ then we can simply replace all of the equivalences above by the single formula *Poss*(a, s) as in Figure 2 (recall a is implicitly universally quantified.)

4. Let NEX_A be the set of all causal laws (4) in D such that ψ is a formula that contains an action constant. Let $\psi(c, s)$ be the formula obtained from a concurrent action c and a formula ψ by replacing every fluent atom $F_i = N$ with $\mathrm{f}_i(s) = N$, and every action atom A_j (respectively, $\neg A_k$) with $\mathrm{A}_j \in c$ (respectively, $\mathrm{A}_k \notin c$). Then add the following definition:

$$\mathit{conflict}(c, s) \equiv \bigvee_{(\textbf{nonexecutable } A_i \textbf{ if } \psi) \in NEX_A} [\mathrm{A}_i \in c \wedge \psi(c, s)].$$

5. For each contribution rule $F \overset{\oplus}{\leftarrow} \mathcal{E}(H)$ in D, add the contribution rule $\mathrm{f} \overset{\oplus}{\leftarrow} \mathcal{E}(\mathrm{h})$.
6. For each expression $0 : F_i = N_i$ in I, add the fact $\mathrm{f}_i(S_0) = N_i$.

Suppose that the range $[\mathit{Min}, \mathit{Max}]$ is wide enough that, when compiling D into an action description as described in Section 3, the auxiliary actions *DContr*$_f$, *IContr*$_f$, and *TContr*$_f$ are never undefined due to range violation.

Proposition 1. *Let C be a set of action constants in D and c be the set of corresponding action functions in sit(D, I). Then the following hold:*

(i) C is executable at time stamp 0 with respect to D and I iff Poss(c, S_0) with respect to sit(D, I);

(ii) for every fluent constant F_i, if C is executable at time stamp 0 and $1 : F_i = N_i'$ after the execution of C at time stamp 0, with respect to D and I, then $\mathrm{f}_i(do(c, S_0)) = N_i'$ with respect to sit(D, I).

(iii) for every fluent constant F_i, if Poss(c, S_0) and $\mathrm{f}_i(do(c, S_0)) = N_i'$ with respect to sit(D, I), then $1 : F_i = N_i'$ after the execution of C at time stamp 0, with respect to D and I.

The assumption above is required for the 'if' part of (i), and for (iii).

Although we have incorporated contribution rules into two formalisms in a similar way, and we have shown that, under some conditions, a formalization of an action domain in $\mathcal{C}+$ can be transformed into a formalization in the concurrent situation calculus, these two formalisms are different in general: $\mathcal{C}+$ action descriptions are nonmonotonic and propositional, while the situation calculus action theories are monotonic and first-order. This work can be viewed in part as an attempt to bridge the gap between these two formalisms, in the spirit of [15].

7 Related Work

There are mainly two lines of work related to ours. The first one, [13] and [14], introduces methods to obtain a causal ordering of variables (denoting numeric-valued fluents) from a set of equation-like causal structures, confluence equations and structural equations, each describing a mechanism in a device. Such a causal ordering describes which fluents are directly causally dependent on which other fluents. The goal is, by this way, to understand the causal behavior of a device.

The other line of work, [16] and [8], explicitly represents causal relations among variables by equation-like causal structures, structural equations and contribution equations; so the goal is not to obtain a causal ordering on numeric-valued variables. They use these equations for various problems of reasoning about actions and change. For instance, [16] represents each mechanism with a structural equation, and uses them for modeling counterfactuals. On the other hand, [8] represents each mechanism with a contribution equation, compiles them into an action theory, allowing one to solve problems of reasoning about effects of actions, like planning and prediction.

All [14, 16, 8] suppose that the causal influence among fluents is acyclic. The method of [13] can not in general determine the effects of disturbances by propagation when the causal influences are cyclic. [14, 16] require each variable to be classified as either exogenous or endogenous; the others and we do not.

In our approach, each mechanism is described by a set of contribution rules with the same head. These rules explicitly represent the flow of causal influences among variables; in this sense it can be considered along the second line of work above. Contribution rules are assumed to be acyclic. As in [8], by compiling contribution rules into an action theory, we can solve problems of reasoning about effects of actions. On the other hand, unlike with contribution equations, there is no obvious correspondence between contribution rules and algebraic equations. For instance, in the containers example, with the contribution equations $inner(s) = medium(s) + small(s)$ and $total(s) = inner(s) + large(s)$, one can verify that $total(s) = small(s) + medium(s) + large(s)$. In our approach, we can verify this equation by introducing an auxiliary fluent $total(s)$ and contribution rules for it, but there is no direct correspondence between the equation and the contribution rules. Another difference between contribution equations and contribution rules, is that auxiliary fluents such as $total$ and $inner$ are necessary to write contribution equations, while they are not required in writing contribution rules. This is due to the ability of contribution rules to express more directly the causal influence relationships among fluents. Finally, although contribution equations can handle the first kind of ramifications mentioned in the introduction, we cannot directly express the sec-

ond kind of ramifications by them; there is no direct way to describe these ramifications by the other causal structures mentioned above.

8 Conclusion

We have described how to formalize an action domain with additive fluents, in two formalisms: the action language $\mathcal{C}+$ and the concurrent situation calculus. In both cases, first we have extended the formalisms, e.g., by introducing some new constructs or functions and by modifying some axioms. Since some ramifications are not easy to describe in the original formalisms, or using the existing causal structures, we have introduced contribution rules, which express causal influences between additive fluents. After that we have formalized an action domain in the extended versions in four parts: specification of additive fluents with their domains/ranges and actions affecting them, direct effects of actions on additive fluents, preconditions of actions, and ramifications on additive fluents. The formalizations obtained this way can handle not only nonserializable actions, but also ramifications on additive fluents. Investigating the application of our method to other formalisms, such as TAL [17], is a possible future research direction.

References

1. Koehler, J.: Planning under resource constraints. In: Proc. ECAI. (1998) 489–493
2. Kvarnström, J., Doherty, P., Haslum, P.: Extending TALplanner with concurrency and resources. In: Proc. ECAI. (2000) 501–505
3. Bacchus, F., Ady, M.: Planning with resources and concurrency: A forward chaining approach. In: Proc. IJCAI. (2001) 417–424
4. Lee, J., Lifschitz, V.: Describing additive fluents in action language $\mathcal{C}+$. In: Proc. IJCAI. (2003)
5. Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H.: Nonmonotonic causal theories. AIJ **153** (2004) 49–104
6. Baral, C., Son, T.C., Tuan, L.: A transition function based characterization of actions with delayed and continuous effects. In: Proc. KR. (2002)
7. Gelfond, M., Lifschitz, V.: Action languages. ETAI **3** (1998) 195–210
8. Erdem, E., Gabaldon, A.: Cumulative effects of concurrent actions on numeric-valued fluents. In: Proc. AAAI. (2005) 627–632
9. Reiter, R.: Knowledge in action: Logical Foundations for specifying and implementing dynamical systems. MIT Press (2001)
10. Lin, F.: Embracing causality in specifying the indirect effects of actions. In: Proc. IJCAI. (1995) 1985–1991
11. McCain, N., Turner, H.: A causal theory of ramifications and qualifications. In: Proc. IJCAI. (1995) 1978–1984
12. Thielscher, M.: Ramification and causality. AIJ **89** (1997) 317–364
13. de Kleer, J., Brown, J.S.: A qualitative physics based on confluences. AIJ **24** (1984) 7–83
14. Iwasaki, Y., Simon, H.: Causality in device behavior. AIJ **29** (1986) 3–32
15. Giunchiglia, E., Lifschitz, V.: Action languages, temporal action logics and the situation calculus. In: Proc. NRAC. (1999)
16. Halpern, J., Pearl, J.: Causes and explanations: a structural-model approach–Part I: Causes. In: Proc. UAI. (2001) 194–202
17. Doherty, P., Gustafsson, J., Karlsson, L., Kvarnstrom, J.: (TAL) Temporal Action Logics: Language specification and tutorial. ETAI **2** (1998) 273–306

Model Representation over Finite and Infinite Signatures

Christian G. Fermüller and Reinhard Pichler

Technische Universität Wien, A-1040 Vienna, Austria
{chrisf, reini}@logic.at

Abstract. Computationally adequate representation of models is a topic arising in various forms in logic and AI. Two fundamental decision problems in this area are: (1) to check whether a given clause is true in a represented model, and (2) to decide whether two representations of the same type represent the same model. ARMs, contexts and DIGs are three important examples of model representation formalisms. The complexity of the mentioned decision problems has been studied for ARMs only for finite signatures, and for contexts and DIGs only for infinite signatures, so far. We settle the remaining cases. Moreover we show that, similarly to the case for infinite signatures, contexts and DIGs allow one to represent the same classes of models also over finite signatures; however DIGs may be exponentially more succinct than all equivalent contexts.

1 Introduction

Computing with term models — aka. Herbrand models — is an important topic in a number of subfields of AI. Apart from general interest in Model Computation[1], the subject can be motivated by applications in Automated Model Building, Logic Programming, Automated Deduction, Machine Learning, and Non-Monotonic Reasoning (see, e.g., [9, 6, 10, 15, 7, 8]). For instance, in Automated Model Building, the target of the model building process is usually a term model. Likewise, the semantics of logic programs is defined by means of term models. Thus, proving the correctness of program simplifications comes down to checking that the term model corresponding to a given logic program remains unchanged, etc.

A term model can be identified with a (generally infinite) set of ground atoms. The simplest and most natural finite representation of a term model consists in a set $\{A_1, \ldots, A_n\}$ of general atoms, where the ground instances of the A_i constitute the represented model. Observe that such an *atomic representation of a term model* or *ARM* is unique only with respect to a given *signature*, i.e., a fixed set of predicate, constant, and function symbols. Two basic decision problems arise: Testing Equivalence, where, given two ARMs, one asks whether they represent the same term model; and Clause-Evaluation, where one wants to know

[1] See www.uni-koblenz.de/~peter/CADE17-WS-MODELS/ and www.uni-koblenz.de/~peter/models03/ for proceedings of relevant workshops.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 164–176, 2006.

whether a given clause is true in the model represented by a given ARM. The complexity of these problems has been studied in detail in [12]. However only finite signatures have been considered there. This is at variance with investigations of related model representation mechanisms (see below). Moreover, infinite signatures are used in potential areas of applications: e.g., in [7, 8] infinite signatures facilitate the study of equivalence of program clauses with respect to arbitrary contexts of datalog programs and answer set programs, respectively.

At a first glance one might be tempted to suppose that it does not make much difference whether the underlying signature is finite or infinite. However, there is an essential difference between these two scenarios. This can be highlighted by considering the closely related problem of *atomic H-subsumption* [10], where one asks whether each instance of a given atom is an instance of some element in a given set of atoms. Note that this amounts to the evaluation of unit-clauses over ARMs. This decision problem has been shown to be coNP-complete for any non-trivial finite signature Σ (i.e., Σ contains at least two constant or function symbols), see [14, 13]. In contrast, for any infinite signature Σ, this problem is in PTIME, see [15].

Of course, the class of term models that can be represented by an ARM is rather limited. Moreover, it is not hard to see that this class is not closed under complementation. Two generalizations of ARMs that enable an explicit representation (also) of false ground atoms are of particular interest: *disjunctions of implicit generalisations* (*DIGs*) and so-called *contexts* as introduced in [1] in connection with the 'model evolution' calculus. Motivated mainly by the latter work, we have investigated in [11] the expressive power and the complexity of Clause-Evaluation and of deciding Equivalence for DIGs and contexts, respectively. Since the intended application of model evolution relies on repeated introductions of new constants we followed [1] in assuming an infinite underlying signature. However, DIGs and contexts are of interest also for finite signatures (see, e.g., [3]). In connection with the above mentioned state of knowledge about ARMs (and atomic H-subsumption), the following open questions naturally arise:

- How is the expressiveness of DIGs and contexts related over finite signatures?
- Does the complexity of Equivalence and Clause-Evaluation increase when we generalize from ARMs to DIGs and contexts over finite signatures?
- Does the complexity of these decision problems decrease for ARMs when we move from finite to infinite signatures?

Results. The following table summarizes results of this paper answering the second and the third question and puts these in the context of directly related previous results. (The new results are written in bold face.)

	finite signature		infinite signature	
	Equivalence	Clause-Evaluation	Equivalence	Clause-Evaluation
ARMs	coNP-complete see [12]	coNP-complete see [12]	in **PTIME**	**coNP-complete**
Contexts and DIGs	**coNP-complete**	**coNP-complete**	coNP-complete see [11]	coNP-complete see [11]

Concerning the first question, we will see in Section 3 that, like for infinite signatures, term models over finite signatures are DIG representable iff they are context representable, but that DIGs may be exponentially more succinct.

Discussion. One possible way to summarize the complexity results in the above table is the following. In moving from ARMs to the more expressive DIGs and contexts there is only one case, where the complexity of the decision problem moves up in the polynomial hierarchy: Testing Equivalence over *infinite* signatures is tractable for ARMs. but coNP-complete for DIGs and contexts. All other cases of Clause-Evaluation and Equivalence are coNP-complete for all model representations considered here.

2 Basic Concepts of Model Representation

A *model representation* is a syntactic structure D associated with a unique model $\mathcal{M}$ over a given signature Σ. Hereafter, we will denote by $\mathcal{M}_\Sigma(D)$ a model $\mathcal{M}$ over the signature Σ represented by D. For the intended applications, any model representation D should satisfy the following properties (cf. [10, 6]):

(1) It can be checked efficiently whether a given ground atom is true in $\mathcal{M}_\Sigma(D)$.
(2) Given a clause C, it is decidable whether C is true in $\mathcal{M}_\Sigma(D)$.
(3) Given another structure D' (of the same type as D), it is decidable whether D and D' are equivalent, i.e., whether $\mathcal{M}_\Sigma(D') = \mathcal{M}_\Sigma(D)$.

For all representation formalisms that are considered in this paper it will be obvious that (1) is fulfilled via (a properly bounded number of) instance checks. The decision problems defined in (2) and (3) are called Clause-Evaluation and Equivalence, respectively, as already mentioned in the introduction.

For a paradigmatic example of model representation, one may think of the explicit specification of *finite* models by tables ('diagrams'). However, we restrict our attention to *term models* over some signature Σ. A term model or, more precisely, Σ-*model*, is identified with the set of ground atoms, i.e., the variable free atoms over Σ, that are true in it. To prevent the corresponding set of ground terms from being empty we assume that Σ contains at least one constant. Σ is called *infinite* if it contains infinitely many constants or function symbols; otherwise it is called *finite*. The cardinality of the (always non-empty) set of predicate symbols plays no role in our investigations. In some proofs we will assume that predicate symbols of any finite arity are available; however, we mention in passing that standard coding techniques allow to strengthen all results to the case where Σ contains a single monadic predicate symbol, as long as at least one non-monadic function symbol is also contained in Σ.

Notation. We use a, b, and c to denote constants; f, g, h will denote function symbols. Concerning variables, we remind the reader that in introducing *contexts*, Baumgartner and Tinelli [1] distinguish between '*universal variables*' and '*parameters*'. Whereas universal variables can be seen as placeholders for arbitrary ground terms, parameters indicate that instantiation is potentially restricted. (This distinction will get clearer, below.) Each atom contains either

only universal variables or only parameters, but not both. We will speak of *universal atoms (literals)* and *parameter atoms (literals)*, respectively. We use x, y, z to denote universal variables, and u, v, w to denote parameters. *Terms* (s, t, ...) and *atoms* (A, B, ...) are built up from constants, variables, and parameters, using function and predicate symbols. On some occasions, we will speak of *positions* in terms and atoms and their respective *depth*, all defined as usual. *Literals* (denoted by K, L, M) are either atoms, called *positive literals*, or negated atoms, called *negative literals*. We write $\overline{L}$ for the literal that is dual to L; i.e., $\overline{A} = \neg A$ and $\overline{\neg A} = A$. The special parameter v is used to denote the *pseudo (parameter) literal* $\neg v$, intended to represent all negative literals whose dual is not explicitly contained in a context. (What exactly this means will get clear from Definitions 2 and 3, below.) A *clause* $C = L_1 \vee \cdots \vee L_k$ is a disjunction of literals.

Substitutions are mappings from variables and parameters to terms that have fixpoints almost everywhere. A substitution is called a *renaming* if it is a permutation that maps variables to variables and parameters to parameters. If the restriction of a substitution σ to parameters is a renaming, then we call σ *p-preserving*. We write $s \precsim t$ if s is an *instance* of t; i.e., if there is a substitution σ such that $s = t\sigma$. If $s = t\sigma$ is a ground term (over Σ) then s is called a *(Σ-)ground instance* of t and σ is called a *(Σ-)ground substitution*. In case σ is a renaming, then we call s a *variant* of t; otherwise s is a *proper instance* of t. A literal L is called *most specific* among literals $L_1, \ldots, L_n$ if for no $i \in \{1, \ldots, n\}$ L_i is a proper instance of L. We assume the reader to be familiar with unification. We use $mgu(E_1, \ldots, E_n)$ to denote the most general unifier of the terms or atoms E_i ($1 \leq i \leq n$). For our proofs below, it is important to remember that when terms and atoms are represented by directed acyclic graphs (rather than by strings), unification can be carried out in polynomial time and space [16].

Given this notational background, we are ready to define the representation mechanisms investigated in this paper.

Definition 1. *An* atomic representation of a term model*, or* ARM*, $\mathcal{A}$ over a signature Σ is a finite set of atoms over Σ. The corresponding* represented Σ-model $\mathcal{M}_\Sigma(\mathcal{A})$ *is the set of Σ-ground instances of atoms in $\mathcal{A}$.*

For ARMs the distinction between parameters and variables is immaterial. For simplicity, we assume that the elements of an ARM are universal atoms. This distinction, however, is essential for contexts:

Definition 2. *A* context Λ *is a finite set of literals including the pseudo literal $\neg v$. Λ is* contradictory *iff $L\sigma = \overline{K}\sigma$ for some variants L, K of elements in Λ and a p-preserving substitution σ.*

Example 1. The context $\Lambda_1 = \{\neg v, P(x, f(y)), \neg P(a, x)\}$ is contradictory, since $P(x, f(y))$ and $P(a, x')$ are unifiable, and the corresponding unifier is p-preserving (since no parameters occur in the atoms). However the context $\Lambda_2 = \{\neg v, P(x, f(y)), \neg P(a, u)\}$ is non-contradictory since, for all substitutions σ such that $P(x, f(y))\sigma = P(a, u)\sigma$ holds, the parameter u has to be instantiated. Likewise the special parameter v has to be instantiated when unified with $P(x, f(y))$.

From now on, we only consider *non-contradictory* contexts and thus drop the adjective, unless we want to emphasize this property.

The central notion for defining the model represented by a context Λ is that of certain ground literals being *produced* in Λ. Like in [11], we provide a slightly simplified definition, that is equivalent to the original one given in [1].

Definition 3. *A ground literal K is produced by a context Λ iff one of the following conditions holds:*

1. *K is an instance of some universal literal L in Λ, or*
2. *K is an instance of a parameter literal $L \in \Lambda$, but $\overline{K}$ is not an instance of a literal $M \in \Lambda$, where M is either universal or a proper instance of L.*

If L is the most specific literal among all literals in Λ with properties 1 and 2, then we say that L produces *K.*

The Σ-model $\mathcal{M}_\Sigma(\Lambda)$ induced by Λ is the set of ground atoms over Σ that are produced by Λ. We call a model $\mathcal{N}$ context representable *if $\mathcal{N} = \mathcal{M}_\Sigma(\Lambda)$ for some context Λ.*

Example 2. Consider the context $\Lambda = \{\neg v,\ P(a,x,y),\ P(u,b,v),\ P(u,v,c),\ \neg P(u,u,c)\}$ and a signature Σ containing only the constants a, b, and c and no function symbols. The universal atom $P(a,x,y)$ and therefore Λ produces all atoms of the form $P(a,s,t)$ for $s,t \in \{a,b,c\}$. $P(u,b,v)$ produces, for instance, the atom $P(b,b,b)$. On the other hand, $\neg P(u,u,c)$ prevents $P(u,v,c)$ from producing $P(c,c,c)$. $P(c,c,c)$ is not produced by Λ at all. Note that $\neg P(u,u,c)$ also prevents $P(u,v,c)$ from producing $P(a,a,c)$ and $P(b,b,c)$. Nevertheless, both $P(a,a,c)$ and $P(b,b,c)$ are produced by Λ, namely by $P(a,x,y)$ and $P(u,b,v)$, respectively. We obtain $\mathcal{M}_\Sigma(\Lambda) = \{P(r,s,t) \mid r = a \vee s = b \vee (r \neq s \wedge t = c)\}$.

A (general) atom is sometimes called an 'explicit generalization' of its ground instances. Consequently, an ARM can be viewed as a disjunction of explicit generalizations. Following ideas in [15], these notions have been generalized in [11] as follows.

Definition 4. *An* implicit generalization *Γ is an expression of the form $A/\mathcal{B}$, where A is an atom and $\mathcal{B}$ is a finite set of atoms. We simply write A for $A/\{\}$. Every ground atom that is an instance of A, but not an instance of any $B \in \mathcal{B}$ is said to be* contained *in $A/\mathcal{B}$.*

A disjunction of implicit generalizations *Δ (shortly:* DIG*) is defined as an expression of the form $A_1/\mathcal{B}_1 \sqcup \ldots \sqcup A_n/\mathcal{B}_n$, also written as $\bigsqcup_{1 \leq i \leq n} A_i/\mathcal{B}_i$. A ground atom is said to be* contained *in Δ if it is contained in $A_i/\mathcal{B}_i$ for some $i \in \{1,\ldots,n\}$.*

The Σ-model $\mathcal{M}_\Sigma(\Delta)$ represented by a DIG Δ is the set of all Σ-ground atoms that are contained in Δ. We call a model $\mathcal{N}$ DIG representable *if $\mathcal{N} = \mathcal{M}_\Sigma(\Delta)$ for some DIG Δ.*

Natural and useful as it is, the concept of DIGs seemingly has been (re)discovered several times in slightly different syntactic disguises. For more references, examples and comments on notation, we refer to [11].

Like for ARMs, the difference between universal variables and parameters is, in principle, irrelevant for DIGs. Staying with the notation of [11], we will assume that DIGs only contain parameters. Note that for any implicit generalization $A/\mathcal{B}$, every $B \in \mathcal{B}$ can be replaced by $B\sigma$, where σ is the most general unifier of (parameter disjoint copies of) A and B, without affecting the set of contained ground atoms. If $B\sigma$ is a variant of A (i.e., if A is an instance of B) then $\mathcal{M}_\Sigma(A/\mathcal{B})$ is empty. In other words, for any implicit generalization $A/\mathcal{B}$, one may assume without loss of generality that the atoms in $\mathcal{B}$ are *proper instances* of A. We call a DIG *normalized* if, for all implicit generalizations $A/\mathcal{B}$ in it, $\mathcal{B}$ consists only of proper instances of A and, moreover, all atoms occurring in the DIG are pairwise parameter disjoint.

Note that a single normalized implicit generalization can be considered as a special form of contexts. Indeed, let $\Gamma = A/\mathcal{B}$ be normalized. Then the set $\Lambda_\Gamma = \{\neg v\} \cup \{A\} \cup \{\neg B \mid B \in \mathcal{B}\}$ is a context with $\mathcal{M}_\Sigma(\Lambda_\Gamma) = \mathcal{M}_\Sigma(\Gamma)$.

3 Expressive Power of DIGs and Contexts

Analogously to the case of an infinite signature Σ, investigated in [11], the following relationship between DIGs and contexts holds:

Theorem 1. *Let Σ be a finite signature.*

1. *Contexts and DIGs have the same expressive power, i.e., a Σ-model $\mathcal{N}$ is context representable iff $\mathcal{N}$ is DIG representable.*
2. *Given a context Λ, a normalized DIG Δ with $\mathcal{M}_\Sigma(\Lambda) = \mathcal{M}_\Sigma(\Delta)$ can be computed in polynomial time.*
3. *If Σ contains at least one function symbol of arity ≥ 2, then there exists a sequence Δ_n $(n > 1)$ of DIGs, where the size of Δ_n is polynomial (in n), but where all contexts representing the same Σ-model as Δ_n are of exponential size (in n).*

Proof. The proof of claims 1 and 2 of the theorem can be taken over literally from the case of an infinite signature Σ: the corresponding constructions presented in [11] do not depend on the cardinality of the signature.

The case of the last claim is different. We need a new sequence of DIGs. For $n \geq 1$, let $\Delta_n = \bigsqcup_{1 \leq i \leq n} P(u_1, \ldots, u_n)/\{P(u_1, \ldots, u_{i-1}, f(u_i, u_i), u_{i+1}, \ldots, u_n), P(u_1, \ldots, u_{i-1}, g(u_i, u_i), u_{i+1}, \ldots, u_n)\}$, where the u_i $(1 \leq i \leq n)$ are pairwise distinct parameters and f and g are two distinct function symbols. Observe that $\mathcal{M}_\Sigma(\Delta_n)$ contains all ground atoms over Σ except those of the form $P(\varphi_1(t_1, t_1), \ldots, \varphi_n(t_n, t_n))$, where $\varphi_i \in \{f, g\}$ and t_i is an arbitrary ground term for $1 \leq i \leq n$. Let Λ_n be a context representing $\mathcal{M}_\Sigma(\Delta_n)$. We show that Λ_n must contain instances of literals of the form $\neg P(\varphi_1(u_1, u_1'), \ldots, \varphi_n(u_n, u_n'))$ for all choices of $\varphi_i \in \{f, g\}$. Since the 2^n different literals of the exhibited form are pairwise non-unifiable it follows that Λ_n contains exponentially many literals.

Consider a literal $L = P(\varphi_1(t, t), \ldots, \varphi_n(t, t))$, where $\varphi_i \in \{f, g\}$ and t is some ground term in which a constant c occurs at a position π that is deeper than

any position in any literal in Λ_n. Moreover let $L' = P(\varphi_1(t', t), \ldots, \varphi_n(t, t))$, where t' is obtained from t by replacing c at π with another constant d, or with $f(c, c)$ if Σ contains only one constant. In other words, L' is exactly as L except for a tiny difference at a position that is so deep that any literal in Λ_n that has L' as an instance is bound to have also L as an instance. Note that L is false in $\mathcal{M}_\Sigma(\Delta_n)$, whereas L' is true in $\mathcal{M}_\Sigma(\Delta_n)$. To prevent L from being produced in Δ_n, there must be some $K \in \Delta_n$ which produces $\neg L$. We claim that $K \precsim \neg P(\varphi_1(u_1, u_1'), \ldots, \varphi_n(u_n, u_n'))$, where the u_i, u_i' for $1 \leq i \leq n$, are different parameters. Suppose, to the contrary, that K is not an instance of $\neg P(\varphi_1(u_1, u_1'), \ldots, \varphi_n(u_n, u_n'))$. Then K is either (1) the pseudo literal $\neg v$ or (2) of the form $\neg P(s_1, \ldots, s_n)$, where $\varphi_i(u_i, u_i')$ is not an instance of s_i for at least one $i \in \{1, \ldots, n\}$. $\neg L \precsim K$ implies $\varphi_i(t, t) \precsim s_i$; consequently s is a parameter. In both cases we have not only $\neg L \precsim K$ but also $\neg L' \precsim K$. On the other hand, since L' is true in $\mathcal{M}_\Sigma(\Delta_n) = \mathcal{M}_\Sigma(\Lambda_n)$, L' has to be produced by some positive literal $K' \in \Lambda_n$. But we have defined L and L' in such a way that if K' produces L', then it also produces L, which contradicts the fact that L is false in $\mathcal{M}_\Sigma(\Delta_n) = \mathcal{M}_\Sigma(\Lambda_n)$.

We have made use of two binary function symbols f and g. To see that a single function symbol h of arity ≥ 2 is sufficient for the claim to hold, one simply has to replace terms of the form $f(s, t)$ everywhere by, e.g., $h(s, t, \ldots, t)$ and terms of the form $g(s, t)$ by, e.g., $h(h(s, \ldots, s), t, \ldots, t)$. □

The class of DIGs that can be transformed polynomially into equivalent contexts is strictly increasing when shifting from infinite to finite signatures. To provide a better understanding of the effect of restricting to finite signatures we re-visit the sequence of DIGs $\Delta_n' = \bigsqcup_{1 \leq i \leq n} P(u_1, \ldots, u_n)/\{P(u_1, \ldots, u_{i-1}, a, u_{i+1}, \ldots, u_n), P(u_1, \ldots, u_{i-1}, b, u_{i+1}, \ldots, u_n)\}$, that are shown in [11] to have no equivalent contexts of polynomial size for any *infinite* signature Σ. $\mathcal{M}_\Sigma(\Delta_n')$ consists in all ground instances of $P(u_1, \ldots, u_n)$ except those where all the (pairwise different) parameters u_i are replaced by either the constant a or the constant b. Over any *finite* signature Σ (that contains at least a and b) we have $\mathcal{M}_\Sigma(\Delta_n') = \mathcal{M}_\Sigma(\Lambda_n)$ for $\Lambda_n = \{P(t_1, \ldots, t_n) \mid t_i \in \mathcal{F}_\Sigma, t_j \neq a \vee t_j \neq b$ for all i and some $j \in \{1, \ldots n\}\}$. Here, $\mathcal{F}_\Sigma$ is the set of terms that contains all constants in Σ and for each function symbol $f \in \Sigma$ a term of the form $f(x_1, \ldots, x_m)$, where the x_i are pairwise different variables. Note that Λ_n not only is a context if augmented by $\neg v$, but even is an ARM. Obviously, the size of Λ_n is polynomial in n for any fixed Σ.

4 ARMs: Clause Evaluation and Testing Equivalence over Infinite Signature

In [12], Equivalence and Clause-Evaluation for ARMs were shown to be coNP-complete for any non-trivial *finite* signature. In this section, we consider the case of an *infinite* signature Σ. The following property (see [15], Proposition 4.1) plays an important role in this analysis :

Proposition 1. *Let $A, B_1, \ldots, B_n$ be atoms over an infinite signature Σ, where $B_i \precsim A$ for all $i \in \{1, \ldots, n\}$. Then $\{A\}$ and $\{B_1, \ldots B_n\}$ are equivalent, i.e., $\mathcal{M}_\Sigma(\{A\}) = \mathcal{M}_\Sigma(\{B_1, \ldots B_n\})$ iff A is a variant of B_j for some $j \in \{1, \ldots, n\}$.*

From this, the tractability of Equivalence for ARMs is an easy consequence:

Theorem 2. *Over any infinite signature Σ,* Equivalence *for ARMs is in PTIME.*

Proof. Let $\mathcal{A} = \{A_1, \ldots, A_m\}$ and $\mathcal{B} = \{B_1, \ldots, B_n\}$ be two ARMs. Obviously, $\mathcal{A}$ and $\mathcal{B}$ are equivalent over Σ iff (1) for every $i \in \{1, \ldots, m\}$, all Σ-ground instances of A_i are in $\mathcal{M}_\Sigma(\mathcal{B})$ and (2) for every $j \in \{1, \ldots, n\}$, all Σ-ground instances of B_j are in $\mathcal{M}_\Sigma(\mathcal{A})$.

Each of these $m+n$ checks can be reduced in polynomial time (via unification) to linearly many checks of the form in Proposition 1. More exactly, for any $i \in \{1, \ldots, m\}$ let $\mathcal{B}'_i = \{B\vartheta \mid \vartheta = mgu(A_i, B), B \in \mathcal{B}\}$. Then the Σ-ground instances of A_i are in $\mathcal{M}_\Sigma(\mathcal{B})$ iff $\{A_i\}$ and $\mathcal{B}'_i$ are equivalent. (Analogously for the B_j and $\mathcal{A}$.) □

For *finite* signatures Σ, the coNP-hardness of Clause-Evaluation for ARMs is shown in [12] by reducing the Equivalence problem to it. By the above tractability result for Equivalence for *infinite* signatures, the question naturally arises whether Clause-Evaluation also becomes tractable when we consider ARMs over an infinite signature. We provide a negative answer to this question:

Theorem 3. *Over any infinite signature Σ,* Clause-Evaluation *with respect to ARMs is coNP-complete.*

Proof. As already mentioned, ARMs can be considered as a special case of contexts (with no negative literals apart from the pseudo literal $\neg v$). The membership part of the claim therefore follows immediately from the coNP-membership of Clause-Evaluation for contexts, shown in [11]. The coNP-hardness is shown by the following reduction from the 3SAT problem.

Recall that an instance of the 3SAT problem is given through a set $X = \{x_1, \ldots, x_k\}$ of propositional variables and a Boolean formula $E = (l_{11} \vee l_{12} \vee l_{13}) \wedge \cdots \wedge (l_{n1} \vee l_{n2} \vee l_{n3})$, where the l_{ij} are literals over X, i.e., every l_{ij} is either a propositional variable x_γ or a negated propositional variable $\bar{x}_\gamma$ for some $\gamma \in \{1, \ldots, k\}$.

Let 0 and 1 denote two distinct ground terms over Σ. We define the clause C_E and the ARM $\mathcal{A}_E$ as follows. By slight abuse of notation, we use the symbols x_γ and $\bar{x}_\gamma$ ($\gamma \in \{1, \ldots, k\}$) to denote also first-order variables:

$$C_E = \neg P(l_{11}, l_{12}, l_{13}) \vee \cdots \vee \neg P(l_{n1}, l_{n2}, l_{n3}) \vee \neg Q(x_1, \bar{x}_1) \vee \cdots \vee \neg Q(x_k, \bar{x}_k)$$

$$\mathcal{A}_E = \{P(0,0,1), P(0,1,0), P(0,1,1), P(1,0,0), P(1,0,1), P(1,1,0), P(1,1,1), Q(0,1), Q(1,0)\}.$$

This problem reduction is clearly feasible in polynomial time. The underlying idea is as follows. C_E evaluates to false in $\mathcal{M}_\Sigma(\mathcal{A}_E)$ iff there exists a substitution σ such that all literals of $C_E\sigma$ are false. In other words, all dual atoms

$P(l_{i1}, l_{i2}, l_{i3})\sigma$ and $Q(x_j, \bar{x}_j)\sigma$ have to be true, i.e., they have to be equal to one of the (ground) atoms in $\mathcal{A}_E$. It is easy to check that such a substitution (which assigns the terms 0 and 1 to the first-order variables x_γ and $\bar{x}_\gamma$) exists iff the Boolean formula E has a satisfying truth assignment (which accordingly assigns values 'false' and 'true' to the propositional literals x_γ and $\bar{x}_\gamma$). □

In order to better understand the source of complexity in Theorem 3 we consider the special cases of positive and negative clauses, respectively. (A clause is called positive if it consists only of positive literals and negative if consists only of negative literals.)

Proposition 2. *Over any infinite signature Σ,* Clause-Evaluation *over ARMs restricted to positive clauses is in PTIME.* Clause-Evaluation *when restricted to negative clauses is coNP-complete.*

Proof. The coNP-completeness in the case of negative clauses is already settled by the proof of Theorem 3. The target of the problem reduction given there is in fact a clause with negative literals only. The tractability of Clause-Evaluation for ARMs in case of positive clauses and infinite signature Σ follows from the following fact, which is related to Proposition 1:

$C = A_1 \vee \cdots \vee A_k$ is true in $\mathcal{M}_\Sigma(\mathcal{A})$ iff for some $i \in \{1, \ldots, k\}$ the atom A_i is a Σ-instance of some atom in the ARM $\mathcal{A}$.

The 'if'-direction is obvious. For the 'only if'-direction, suppose that C is true in $\mathcal{M}_\Sigma(\mathcal{A})$. Let σ be a ground substitution that assigns a unique new constant to every variable in C, i.e., the terms $\sigma(x)$ are pairwise distinct constants that do not occur in $\mathcal{A}$. By assumption, $C\sigma$ evaluates to true in $\mathcal{M}_\Sigma(\mathcal{A})$. Hence, there exists an $i \in \{1, \ldots, k\}$ such that $A_i\sigma$ is an instance of some $B \in \mathcal{A}$. But, by the special form of σ, also A_i is an instance of B. □

5 DIGs and Contexts: Clause Evaluation and Testing Equivalence over Finite Signatures

In the following we will assume that the underlying signature Σ is finite, but non-trivial (i.e., Σ contains at least two constant or function symbols). The coNP-hardness of the four decision problems considered here — Clause-Evaluation and Equivalence for DIGs and contexts, respectively — follows directly from the coNP-hardness of Equivalence and Clause-Evaluation for ARMs, which was shown in [12]. In this section, we show that all of the four problems are in fact coNP-*complete*. By the polynomial-time transformation of contexts into DIGs (see Theorem 1), it suffices to establish the coNP-membership of Equivalence and Clause-Evaluation for DIGs.

For this purpose, we first recall from [12] how the 'complement' of an atom A (i.e., the ground atoms over Σ which are not instances of A) can be represented by means of 'constrained atoms'. Constrained atoms are constructs of the form $[B : \mathcal{X}]$ consisting of an atom B and an equational formula $\mathcal{X}$ such that $[B : \mathcal{X}]$

contains precisely those ground instances $B\sigma$ of B for which σ is a solution of $\mathcal{X}$ (see [5]). Any atom B can be considered as a constrained atom by adding the trivially true formula $\top$ as a constraint, i.e., B and $[B : \top]$ are equivalent.

Then the complement of an atom A can be constructed in the following way. Consider the tree representation of A, 'deviate' from this representation at some node and close all other branches of A as early as possible with new, pairwise distinct variables. Depending on the label of a node, this deviation can be done in two different ways: If a node is labelled by a (constant, function, or predicate) symbol from Σ, then this node has to be labelled by a different symbol from Σ. If a node is labelled by a variable which also occurs at some other position, then the two occurrences of this variable have to be replaced with two fresh variables z_1, z_2 and the constraint $z_1 \neq z_2$ has to be added. However, if a node is labelled by a variable which occurs nowhere else, then no deviation at all is possible at this node. This idea is illustrated by the following example:

Example 3. Let $\Sigma = \{P, Q, f, g, a\}$ and let $A = P(f(x, y), g(x))$ be an atom over Σ. The complement of A can be represented by the set $\mathcal{C} = \{Q(z), P(a, z), P(g(z_1), z_2), [P(f(z_1, y), g(z_2)) : z_1 \neq z_2], P(z, a), P(z_1, f(z_1, z_2))\}$. In other words, a ground atom A' over Σ is not an instance of A iff A' is an instance of one of the (constrained) atoms in $\mathcal{C}$.

We only need the following properties of this construction via 'deviations' (for details of this construction and for a proof of these properties, see [12]):

Theorem 4. *Let A be an atom over a finite signature Σ. There exists a set of constrained atoms $\mathcal{C} = \{[B_1 : \mathcal{X}_1], \ldots, [B_n : \mathcal{X}_n]\}$ with the following properties:*

1. *$\mathcal{C}$ represents the complement of A, i.e., a ground atom A' over Σ is not an instance of A iff A' is an instance of one of the constrained atoms in $\mathcal{C}$.*
2. *For every $i \in \{1, \ldots, n\}$, $\mathcal{X}_i$ is either the trivially true formula $\top$ or a quantifier-free disequation.*
3. *The size of every constrained atom in $\mathcal{C}$ is linearly bounded by the size of A (assuming compact representations of terms as directed acyclic graphs).*

Let $comp_\Sigma(A)$ denote the complement of an atom A with respect to signature Σ. This notion is readily generalized to implicit generalizations I and to DIGs Δ. We write $comp_\Sigma(I)$ and $comp_\Sigma(\Delta)$ for the respective complements. Note that the complement $comp_\Sigma(I)$ of a single implicit generalization $I = B/\mathcal{B}$ coincides with $comp_\Sigma(B) \cup \mathcal{M}_\Sigma(\mathcal{B})$. Moreover, $comp_\Sigma(\Delta)$ is obtained as the intersection of the complements of the implicit generalizations in Δ. The distinction between universal variables and parameters is irrelevant here. Thus we will simply speak of variables also for DIGs.

To obtain a coNP-algorithm for Clause-Evaluation for DIGs we need an efficient method for testing whether some constrained atoms have at least one ground instance in common (see [12]):

Theorem 5. *Let $\{[B_1 : \mathcal{X}_1], \ldots, [B_m : \mathcal{X}_m]\}$ denote a set of constrained atoms, where the constraints are either $\top$ or quantifier-free disequations. Then it can*

be tested in polynomial time whether there exists a ground atom A' that is an instance of $[B_i : \mathcal{X}_i]$ for every $i \in \{1, \ldots, m\}$.

Proof. Without loss of generality, assume that the B_i's are pairwise variable disjoint. If there exists at least one common ground instance of the constrained atoms, then all these common ground instances can be represented by the constrained atom $[B_1\mu : \mathcal{Z}]$, where $\mu = mgu(B_1, \ldots, B_m)$ and $\mathcal{Z}$ is defined as $\mathcal{Z} \equiv \mathcal{X}_1 \wedge \cdots \wedge \mathcal{X}_m$. In order to test whether at least one common ground instance exists, we just have to check whether μ exists and whether $\mathcal{Z}\mu$ has at least one solution. Since $\mathcal{Z}\mu$ is a conjunction of disequations, the latter condition holds iff $\mathcal{Z}\mu$ contains no trivial disequation of the form $t \neq t$. (For a proof of this latter fact, see [4], Lemma 2). □

Lemma 1. *Over any non-trivial finite signature Σ,* Clause-Evaluation *for DIGs is in coNP.*

Proof. Let $C = A_1 \vee \cdots \vee A_k \vee \neg A'_1 \vee \cdots \vee \neg A'_\ell$ and $\Delta = \bigsqcup_{1 \leq i \leq n} B_i/\mathcal{B}_i$. In order to simplify the notation, we assume that all sets $\mathcal{B}_i$ have the same cardinality m. Of course, this can be easily achieved by adding an appropriate number of copies of some $B \in \mathcal{B}_i$ to $\mathcal{B}_i$. Thus, $\mathcal{B}_i$ is of the form $\mathcal{B}_i = \{\bar{B}_{i1}, \ldots, \bar{B}_{im}\}$. The clause C is false in $\mathcal{M}_\Sigma(\Delta)$ iff for some ground instance $C\sigma$ all literals of $C\sigma$ evaluate to false. In other words, C evaluates to false in $\mathcal{M}_\Sigma(\Delta)$ iff there exists a ground substitution σ such that

(a) for every $\alpha \in \{1, \ldots, k\}$, $A_\alpha\sigma$ is contained in $comp_\Sigma(\Delta)$ and
(b) for every $\delta \in \{1, \ldots, \ell\}$, $A'_\delta\sigma$ is contained in $\mathcal{M}_\Sigma(\Delta)$.

It remains to show that these conditions can be tested by an NP-algorithm. Our algorithm first carries out the following non-deterministic **guesses**:

1. Guess $k \cdot n$ constrained atoms $[E_{\alpha\beta} : \mathcal{X}_{\alpha\beta}]$ with $\alpha \in \{1, \ldots, k\}$ and $\beta \in \{1, \ldots, n\}$ such that $[E_{\alpha\beta} : \mathcal{X}_{\alpha\beta}]$ is a constrained atom from the complement representation of the implicit generalization $B_\beta/\{\bar{B}_{\beta 1}, \ldots, \bar{B}_{\beta m}\}$.
2. Guess ℓ indices $\gamma(1), \ldots, \gamma(\ell)$ with $\gamma(\delta) \in \{1, \ldots, n\}$ for every $\delta \in \{1, \ldots, \ell\}$.
3. Guess $\ell \cdot m$ constrained atoms $[F_{\delta\epsilon} : \mathcal{Y}_{\delta\epsilon}]$ with $\delta \in \{1, \ldots, \ell\}$ and $\epsilon \in \{1, \ldots, m\}$ such that $[F_{\delta\epsilon} : \mathcal{Y}_{\delta\epsilon}]$ is a constrained atom from the complement representation of the atom $\bar{B}_{\gamma(\delta)\epsilon} \in \mathcal{B}_{\gamma(\delta)}$.

Again, we may assume that the clause C and all atoms $B_{\gamma(1)}, \ldots, B_{\gamma(\ell)}$ (i.e., the left-hand sides of the implicit generalizations whose indices $\gamma(1), \ldots, \gamma(\ell)$ are guessed in step 2 above) as well as all constrained atoms $[E_{\alpha\beta} : \mathcal{X}_{\alpha\beta}]$ and $[F_{\delta\epsilon} : \mathcal{Y}_{\delta\epsilon}]$ are pairwise variable disjoint. Then we carry out the following **checks**:

4. Check that the most general unifier $\mu = mgu(\mathcal{U})$ of the simultaneous unification problem $\mathcal{U}$ exists, where $\mathcal{U}$ is defined as follows.
$$\mathcal{U} = \{A_1 = E_{11} = \cdots = E_{1n},\ \ldots,\ A_k = E_{k1} = \cdots = E_{kn},\\ A'_1 = B_{\gamma(1)} = F_{11} = \cdots = F_{1m},\ \ldots,\ A'_\ell = B_{\gamma(\ell)} = F_{\ell 1} = \cdots = F_{\ell m}\}$$
5. Check that the equational problem $\mathcal{Z}\mu$ contains no trivial disequation of the form $t \neq t$ where $\mathcal{Z} \equiv \big(\bigwedge_{\alpha=1}^{k} \bigwedge_{\beta=1}^{n} \mathcal{X}_{\alpha\beta}\big) \wedge \big(\bigwedge_{\delta=1}^{\ell} \bigwedge_{\epsilon=1}^{m} \mathcal{Y}_{\delta\epsilon}\big)$.

Obviously, this algorithm works in non-deterministic polynomial time, provided that an efficient unification algorithm is used (see [16]). The correctness of this algorithm can be seen as follows. The checks in steps 4 and 5 above correspond to a generalization of Theorem 5 in that we check whether some ground instance $C\sigma$ of C exists such that the resulting ground atoms $A_1\sigma, \ldots, A_k\sigma$, $A'_1\sigma, \ldots, A'_\ell\sigma$ are contained in the following intersections of constrained atoms:

(a) For every $\alpha \in \{1, \ldots, k\}$, $A_\alpha\sigma$ has to be in $comp_\Sigma(\Delta)$. For this purpose, we guess in step 1 the constrained atoms $[E_{\alpha\beta} : \mathcal{X}_{\alpha\beta}]$ from the complement representation of every implicit generalization $B_\beta/\{\bar{B}_{\beta 1}, \ldots, \bar{B}_{\beta m}\}$ with $\beta \in \{1, \ldots, n\}$. Obviously, $A_\alpha\sigma$ is in $comp_\Sigma(\Delta)$, iff it is a common instance of these constrained atoms.

(b) For every $\delta \in \{1, \ldots, \ell\}$, $A'_\delta\sigma$ has to be contained in Δ. In other words, for every δ, there exists an index $\gamma(\delta)$ – which is guessed in step 2 – such that $A'_\delta\sigma$ is contained in $B_{\gamma(\delta)}/\{\bar{B}_{\gamma(\delta)1}, \ldots, \bar{B}_{\gamma(\delta)m}\}$. Therefore, $A'_\delta\sigma$ is an instance of $B_{\gamma(\delta)}$ (for this purpose, the conditions $A'_\delta = B_{\gamma(\delta)}$ are part of the unification problem $\mathcal{U}$) and $A'_\delta\sigma$ is in the complement of every $\bar{B}_{\gamma(\delta)\epsilon}$. The constrained atoms $[F_{\delta\epsilon} : \mathcal{Y}_{\delta\epsilon}]$ guessed in step 3 take care of the latter condition. □

In [11], the coNP-membership of Equivalence for DIGs over infinite signatures was shown by reducing the problem to linearly many instances of Clause-Evaluation for DIGs. This reduction is completely independent of the underlying signature. In other words: the same problem reduction works also if we consider a finite signature. Together with the coNP-membership shown above, we immediately obtain:

Theorem 6. *Over any non-trivial finite signature Σ, the* Clause-Evaluation *and the* Equivalence *problem for contexts and DIGs, respectively, are coNP-complete.*

6 Conclusion

We have been motivated by the fact that basic decision problems relating to ARMs had been investigated previously only for *finite* signatures, whereas the same problems for two important generalizations of ARMs — namely, DIGs and contexts — had been studied for an underlying *infinite* signature only. In this paper, we have completed the picture by studying ARMs over an infinite signature and DIGs and contexts over a finite signature. It has turned out that — apart from one case (namely Equivalence over an infinite signature) — the complexity does not increase when we move from ARMs to the much more expressive contexts and DIGs. Note however that this does not mean that ARMs are "useless". In fact, the usefulness of a model representation depends to a large extent on the existence of a calculus which constructs models via this representation. And this is clearly the case for ARMs (see [10]).

One may now ask for a more fine grained analysis of the problems considered here. In particular, a more detailed picture of the borderline between tractable and intractable cases would be interesting. For instance, what happens to the complexity if we restrict ourselves to DIGs with only one atom on the right-hand

side of each implicit generalization or to linear atoms (in the ARMs, DIGs and contexts)? Proposition 2 already contains an observation along this line.

Another natural topic for future investigations is the effect of integrating equality literals into model representation formalisms. In fact, Baumgartner and Tinelli have recently [2] generalized model evolution, including contexts, to clause logic with equality. It is clear at once that the expressive power, but also the complexity of corresponding decision problems, increases dramatically in presence of equality literals. We delegate more detailed assertions to future work.

References

1. P. Baumgartner and C. Tinelli. The model evolution calculus. In *Proceedings of CADE-19*, LNCS 2741, pages 350–364, Springer, 2003.
2. P. Baumgartner and C. Tinelli. The model evolution calculus with Equality. In *Proceedings of CADE 2005*, LNCS 3632, pages 392–408, Springer, 2005.
3. P. Baumgartner, A. Fuchs, and C. Tinelli. Lemma Learning in the Model Evolution Calculus. Submitted.
4. H. Comon and C. Delor. Equational formulae with membership constraints. *Information and Computation*, 112(2):167–216, 1994.
5. R. Caferra and N. Zabel. Extending resolution for model construction. In *Proceedings of JELIA'90*, LNAI 478, pages 153–169, Springer, 1991.
6. R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building*, volume 31 of *Applied Logic Series*, Kluwer Academic Publishers, 2004.
7. T. Eiter, W. Faber, and P. Traxler. Testing strong equivalence of datalog programs - implementation and examples. In *Proceedings of LPNMR 2005*, LNCS 3662, pages 437–441, Springer, 2005.
8. T. Eiter, M. Fink, H. Tompits, P. Traxler, and S. Woltran. Replacements in non-ground answer set programming. In *Proc. of WLP 2006*, pages 145–153, 2006.
9. C.G. Fermüller and A. Leitsch. Model building by resolution. In *Proceedings of CSL'92*, LNCS 702, pages 134–148, Springer, 1993.
10. C.G. Fermüller and A. Leitsch. Hyperresolution and automated model building. *Journal of Logic and Computation*, 6(2):173–203, 1996.
11. C.G. Fermüller and R. Pichler. Model representation via contexts and implicit generalizations. In *Proc. of CADE-20*, LNCS 3632, pages 409–423, Springer, 2005.
12. G. Gottlob and R. Pichler. Working with ARMs: Complexity results on atomic representations of Herbrand models. *Information and Computation*, 165:183–207, 2001.
13. D. Kapur, P. Narendran, D. Rosenkrantz, and H. Zhang. Sufficient-completeness, ground-reducibility and their complexity. *Acta Informatica*, 28(4):311–350, 1991.
14. K. Kunen. Answer sets and negation as failure. In *Proceedings of ICLP'87*, pages 219–228, MIT Press, 1987.
15. J.-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3(3):301–317, 1987.
16. A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.

Deciding Extensions of the Theory of Arrays by Integrating Decision Procedures and Instantiation Strategies

Silvio Ghilardi[1], Enrica Nicolini[2], Silvio Ranise[1,3], and Daniele Zucchelli[1,3]

[1] Dipartimento di Informatica, Università degli Studi di Milano (Italia)
[2] Dipartimento di Matematica, Università degli Studi di Milano (Italia)
[3] LORIA & INRIA-Lorraine, Nancy (France)

Abstract. The theory of arrays, introduced by McCarthy in his seminal paper "Toward a mathematical science of computation", is central to Computer Science. Unfortunately, the theory alone is not sufficient for many important verification applications such as program analysis. Motivated by this observation, we study extensions of the theory of arrays whose satisfiability problem (i.e. checking the satisfiability of conjunctions of ground literals) is decidable. In particular, we consider extensions where the indexes of arrays has the algebraic structure of Presburger Arithmetic and the theory of arrays is augmented with axioms characterizing additional symbols such as dimension, sortedness, or the domain of definition of arrays.

We provide methods for integrating available decision procedures for the theory of arrays and Presburger Arithmetic with automatic instantiation strategies which allow us to reduce the satisfiability problem for the extension of the theory of arrays to that of the theories decided by the available procedures. Our approach aims to reuse as much as possible existing techniques so to ease the implementation of the proposed methods. To this end, we show how to use both model-theoretic and rewriting-based theorem proving (i.e., superposition) techniques to implement the instantiation strategies of the various extensions.

1 Introduction

Since its introduction by McCarthy in [13], the theory of arrays ($\mathcal{A}$) has played a very important role in Computer Science. Hence, it is not surprising that many papers [4, 17, 20, 10, 12, 19, 2, 3] have been devoted to its study in the context of verification and many reasoning techniques, both automatic - e.g., [2] - and manual [17], have been developed to reason in such a theory.

Unfortunately, as many previous works [20, 10, 12, 3] have already observed, $\mathcal{A}$ alone or even extended with extensional equality between arrays (as in [19, 2]) is not sufficient for many applications of verification. For example, the works in [20, 10, 12] tried to extend the theory to reason about sorted arrays. More recently, Bradley et al. [3] have shown the decidability of the satisfiability problem for a restricted class of (possibly quantified) first-order formulae that allows one to express many important properties about arrays.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 177–189, 2006.

In this paper, we consider the theory of arrays with extensionality [19, 2] whose indexes have the algebraic structure of Presburger Arithmetic ($\mathcal{P}$), and extend it with additional (function or predicate) symbols expressing important features of arrays (e.g., the dimension of an array or an array being sorted). The *main contribution* of the paper is a method to integrate two decision procedures, one for the theory of arrays without extensionality ($\mathcal{A}$) and one for $\mathcal{P}$, with instantiation strategies that allow us to reduce the satisfiability problem of the extension of $\mathcal{A} \cup \mathcal{P}$ to the satisfiability problems decided by the two available procedures.

Our approach to integrating decision procedures and instantiation strategies is inspired by model-theoretic considerations and by the rewriting-approach [2, 1, 11]. For the rewriting-based method, we follow the lines of [11], where, facing the satisfiability problem, it is suggested that the (ground) formulae derived by the superposition calculus [16] between ground literals and the axioms of a theory T (extending the theory of equality Eq) can be passed to a decision procedure for Eq. In this paper, we use superposition to generate enough (ground) instances of an extension of $\mathcal{A}$ so to enable the decision procedures for $\mathcal{P}$ and $\mathcal{A}$ to decide its satisfiability problem. An immediate by-product of our approach is the fact that the various extensions of $\mathcal{A}$ can be combined together to decide the satisfiability of the union of the various extensions.

Related work. The work most closely related to ours is [3]. The main difference is that we have a semantic approach to extending $\mathcal{A}$ since we consider only the satisfiability of ground formulae and we introduce additional functions and predicates while in [3], a syntactic characterization of a class of full first-order formulae, which turns out to be expressive enough to specify many properties of interest about arrays, is considered. Our approach allows us to get a more refined characterization of some properties of arrays, yielding the decidability of the extension of $\mathcal{A}$ with injective arrays (see Section 5.1), which is left as an open problem in [3].

Our instantiation strategy based on superposition (see Section 5.2) has a similar spirit of the work in [7], where equational reasoning is integrated in instantiation-based theorem proving. The main difference with [7] is that we solve the state-explosion problem, due to the recombination of formulae caused by the use of standard superposition rules, by deriving a new termination result for an extension of $\mathcal{A}$ as recommended by the rewriting approach to satisfiability procedures of [2].

Plan of the paper. Section 2 introduces some formal notions necessary to develop the results in this paper. Section 3 gives some motivation for the first extension of $\mathcal{A}$ by a dimension function together with its formal definition while Section 4 describe an extensible decision procedure. Section 5 considers two extensions of the theory defined in Section 3. For lack of space, further extensions of $\mathcal{A}$ and the proofs of the results in this paper are included in a Technical Report [9].

2 Formal Preliminaries

We work in *many-sorted first-order logic with equality* and we assume the basic syntactic and semantic concepts as in, e.g., [6]. A *signature* Σ is a non-empty

set of sort symbols together with a set of function symbols and a set of predicate symbols (both equipped with suitable lists of sort symbols as arity). The set of predicate symbols contains a symbol $=_S$ for equality for every sort S (we usually omit its subscript). If Σ is a signature, a *simple* expansion of Σ is a signature Σ' obtained from Σ by adding a set $\underline{a} := \{a_1, ..., a_n\}$ of "fresh" constants (each of them again equipped with a sort), i.e. $\Sigma' := \Sigma \cup \underline{a}$, where $\underline{a}$ is such that Σ and $\underline{a}$ are disjoint. Below, we write $\Sigma^{\underline{a}}$ as the simple expansion of Σ with a set $\underline{a}$ of fresh constant symbols. First-order terms and formulae over a signature Σ are defined in the usual way, i.e., they must respect the arities of function and predicate symbols and the variables occurring in them must also be equipped with sorts (well-sortedness). A *Σ-atom* is a predicate symbol applied to (well-sorted) terms. A *Σ-literal* is a Σ-atom or its negation. A *ground* literal is a literal not containing variables. A *constraint* is a finite conjunction $\ell_1 \wedge \cdots \wedge \ell_n$ of literals, which can also be seen as a finite set $\{\ell_1, \ldots, \ell_n\}$. A *$\Sigma$-sentence* is a first-order formula over Σ without free variables.

A *Σ-structure* $\mathcal{M}$ consists of non-empty and pairwise disjoint domains $S^{\mathcal{M}}$ for every sort S, and interprets each function symbol f and predicate symbol P as functions $f^{\mathcal{M}}$ and relations $P^{\mathcal{M}}$, respectively, according to their arities. If t is a ground term, we also use $t^{\mathcal{M}}$ for the element denoted by t in the structure $\mathcal{M}$. Validity of a formula ϕ in a Σ-structure $\mathcal{M}$ (in symbols, $\mathcal{M} \models \phi$), satisfiability, and logical consequence are defined in the usual way. The Σ-structure $\mathcal{M}$ is a *model* of the Σ-theory T iff all axioms of T are valid in $\mathcal{M}$. A *Σ-theory* T is a (possibly infinite) set of Σ-sentences. Let T be a theory. We refer to the signature of T as Σ_T. If there exists a set $Ax(T)$ of sentences in T such that every formula ϕ of T is a logical consequence of $Ax(T)$, then we say that $Ax(T)$ is a set of axioms of T. A theory T is *complete* iff, given a sentence ϕ, we have that ϕ is either true or false in all the models of T.

In this paper, we are concerned with the *(constraint) satisfiability problem* for a theory T, also called the T-satisfiability problem, which is the problem of deciding whether a Σ_T-constraint is satisfiable in a model of T. Notice that a constraint may contain variables: since these variables may be equivalently replaced by free constants, we can reformulate the constraint satisfiability problem as the problem of deciding whether a finite conjunction of ground literals in a simply expanded signature $\Sigma_T^{\underline{a}}$ is true in a $\Sigma_T^{\underline{a}}$-structure whose Σ_T-reduct is a model of T. We say that a Σ_T-constraint is *T-satisfiable* iff there exists a model of T satisfying it. Two Σ_T-constraints ϕ and ψ are T-equisatisfiable iff there exists a structure $\mathcal{M}_1$ such that $\mathcal{M}_1 \models T \wedge \phi$ iff the following condition holds: there exists a structure $\mathcal{M}_2$ such that $\mathcal{M}_2 \models T \wedge \psi$.

Without loss of generality, when considering a set L of ground literals to be checked for satisfiability, we may assume that each literal ℓ in L is *flat*, i.e. ℓ is required to be either of the form $a = f(a_1, \ldots, a_n)$, $P(a_1, \ldots, a_n)$, or $\neg P(a_1, \ldots, a_n)$, where $a, a_1, \ldots, a_n$ are (sort-conforming) free constants, f is a function symbol, and P is a predicate symbol (possibly also equality).

3 Finite Arrays with Dimension as a Combined Theory

Given a set A, by $Arr(A)$ we denote the set of finite arrays with natural numbers as indexes and whose elements are from A. We model such an array a as a sequence $a : \mathbb{N} \longrightarrow A \cup \{\bot\}$ which is eventually equal to $\bot$ (here $\bot$ is an element not in A denoting an "undefined" or "default" value). In this way, for every array $a \in Arr(A)$ there is a smallest index $n \geq 0$, called the *dimension* of a, such that the value of a at index j is equal to $\bot$ for $j \geq n$. Contrary to finite sequences, we do not require that any value of a at $k < n$ be distinct from $\bot$: this is also the reason to use the word 'dimension' rather than 'length', as for sequences. There is just one array whose dimension is zero which we indicate by ε and call it the *empty* array. Since many applications of verification require arithmetic expressions on indexes of arrays, we introduce Presburger arithmetic $\mathcal{P}$ over indexes: any other decidable fragment of Arithmetic would be a good alternative. Thus the relevant operations on our arrays include *addition over indexes, read, write, and dimension*. The resulting theory (to be formally introduced later on) $\mathcal{ADP}$ can be seen as a combination of well-known theories such as $\mathcal{P}$ and the theory $\mathcal{A}_e$ of arrays with extensionality (see, e.g., [2]), extended with a function for dimension which takes an array and returns a natural number. Because of the function for dimension, the combination is *non-disjoint* and cannot be handled by classical combination schemas such as Nelson-Oppen [15]. Nevertheless, following [8], it is convenient to see $\mathcal{ADP}$ as a combination of $\mathcal{P}$ with a theory of array with dimension $\mathcal{A}_{dim}$: $\mathcal{A}_{dim}$ extends $\mathcal{A}_e$ (both in the signature and in the axioms), but is contained in $\mathcal{ADP}$, because indexes are only endowed with a discrete linear poset structure (the next subsection fixes the details). In this way, we have that $\mathcal{ADP} = \mathcal{A}_{dim} \cup \mathcal{P}$ and the theories $\mathcal{A}_{dim}$ and $\mathcal{P}$ share the well-known complete theory $\mathcal{T}_0$ of natural numbers endowed with zero and successor (see e.g., [5]): this theory admits quantifier elimination, so that the $\mathcal{T}_0$-compatibility hypothesis of [8] needed for the non-disjoint Nelson-Oppen combination is satisfied. Unfortunately, the combination result in [8] cannot be applied to $\mathcal{ADP}$ for mainly two reasons. First, $\mathcal{T}_0$ is not locally finite (see, e.g., [8] for details). Secondly, $\mathcal{A}_{dim}$ is a proper extension of the theory $\mathcal{A}_e$, hence the decision procedures for the $\mathcal{A}_e$-satisfiability problem (such as, e.g., the one in [2]) must be extended. In the rest of the paper, we will show that it is sufficient to use decision procedures for the $\mathcal{P}$- and $\mathcal{A}_e$-satisfiability problem to solve the $\mathcal{ADP}$-satisfiability problem provided that a suitable pre-processing of the input set of literals is performed.

Here, we formally introduce the basic theories of interests for this paper.

$\boxed{\mathcal{T}_0}$ has just one sort symbol INDEX, the following function and predicate symbols: 0 : INDEX, s : INDEX $\rightarrow$ INDEX, and $<$: INDEX $\times$ INDEX. It is axiomatized by the the following formulae:[1]

[1] Here and in the following, we omit the outermost universal quantification for the sake of readability.

$$y \neq 0 \rightarrow \exists z(y = \mathrm{s}(z)) \tag{1}$$
$$x < \mathrm{s}(y) \leftrightarrow (x < y \vee x = y) \tag{2}$$
$$\neg(x < 0) \tag{3}$$
$$x < y \vee x = y \vee y < x \tag{4}$$
$$x < y \rightarrow \neg(y < x) \tag{5}$$
$$x < y \rightarrow (y < z \rightarrow x < z) \tag{6}$$

where x, y and z are variables of sort INDEX. This theory admits elimination of quantifiers and it is complete, see [5] for details.

$\boxed{\mathcal{P}}$ is the well-known Presburger arithmetic, see, e.g., [5], over indexes. The signature is that of $\mathcal{T}_0$ extended with the function symbol for addition $+$: INDEX $\times$ INDEX $\rightarrow$ INDEX, written infix. Since $\mathcal{P}$ is not finitely axiomatizable (see, again [5]), we assume as axioms all valid sentences in the theory. Notice that $\mathcal{T}_0 \subset \mathcal{P}$.

$\boxed{\mathcal{A}}$ is the theory of arrays (see, e.g., [2]) which has the following signature:

- sort symbols: INDEX, ELEM, ARRAY and
- function symbols: select : ARRAY $\times$ INDEX $\rightarrow$ ELEM and store : ARRAY $\times$ INDEX $\times$ ELEM $\rightarrow$ ARRAY

and it is axiomatized by the following formulae:

$$\mathrm{select}(\mathrm{store}(a, i, e), i) = e \tag{7}$$
$$i \neq j \rightarrow \mathrm{select}(\mathrm{store}(a, i, e), j) = \mathrm{select}(a, j) \tag{8}$$

$\boxed{\mathcal{A}_e}$ is the theory of arrays with extensionality (see, e.g., [2]) which has the same signature of $\mathcal{A}$ and it is axiomatized by (7), (8), and the axiom of extensionality:

$$\forall i(\mathrm{select}(a, i) = \mathrm{select}(b, i)) \rightarrow a = b \tag{9}$$

Notice that $\mathcal{A} \subset \mathcal{A}_e$.

$\boxed{\mathcal{A}_{dim}}$ is the simple theory of arrays with dimension whose signature is the union of the signatures of $\mathcal{T}_0$ and $\mathcal{A}_e$ extended with the following three symbols: $\perp$: ELEM, ε : ARRAY, and dim : ARRAY $\rightarrow$ INDEX. It is axiomatized by the axioms in $\mathcal{T}_0$, those in $\mathcal{A}_e$, and the following formulae:

$$\dim(a) \leq i \rightarrow \mathrm{select}(a, i) = \perp \tag{10}$$
$$\dim(a) = \mathrm{s}(i) \rightarrow \mathrm{select}(a, i) \neq \perp \tag{11}$$
$$\dim(\varepsilon) = 0 \tag{12}$$

Notice that $\mathcal{T}_0 \subset \mathcal{A}_{dim}$ and $\mathcal{A}_e \subset \mathcal{A}_{dim}$.

$\boxed{\mathcal{ADP}}$ is the theory of arrays with dimension whose signature is the union of the signatures of $\mathcal{A}_{dim}$ and $\mathcal{P}$ and is axiomatized by the axioms in $\mathcal{A}_{dim}$ and all valid sentences in $\mathcal{P}$.

The theories $\mathcal{T}_0$, $\mathcal{P}$, $\mathcal{A}$, and $\mathcal{A}_e$ are decidable (see [5] for the first two and [2] for the last two). This is an important observation for the results of this paper,

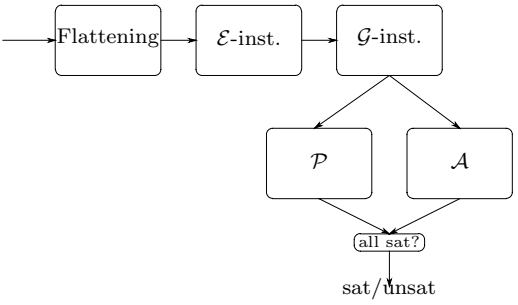


Fig. 1. The architecture of the decision procedure for $\mathcal{ADP}$

since the decision procedure for $\mathcal{ADP}$-satisfiability will assume the availability of two decision procedures for the constraint satisfiability problems of $\mathcal{P}$ and $\mathcal{A}$. The theories $\mathcal{A}_e$, $\mathcal{A}_{dim}$, and $\mathcal{ADP}$ admit a particular subclass of models, which we call the *standard* ones and are exactly those introduced above in order to motivate the definition of $\mathcal{ADP}$. Such models are characterized by the fact that the sort INDEX is always interpreted as the set $\mathbb{N}$ of natural numbers, and the sort ARRAY is interpreted as the set of all the sequences of elements from ELEM that are eventually equal to $\perp$; the dimension of each array is the successor of the index of the last element different from $\perp$. Of course, when investigating constraint satisfiability we are mainly interested in satisfiability of constraints in standard models and we shall in fact prove that a constraint is satisfiable in a model of $\mathcal{ADP}$ iff it is satisfiable in a standard model (see Lemma 4.3, below).

4 A Decision Procedure for Arrays with Dimension

We assume the availability of two decision procedures solving the $\mathcal{A}_e$- and $\mathcal{P}$-satisfiability problems. The overall schema of the procedure for $\mathcal{ADP}$-satisfiability problems is depicted in Figure 1. The idea is to reduce the $\mathcal{ADP}$-satisfiability problem to the constraint satisfiability problems for $\mathcal{A}_e$ and $\mathcal{P}$. The module Flatten pre-processes the literals in the input constraint so to make them flat and easily recognizable as belonging to one theory among those used to define $\mathcal{ADP}$, i.e. $\mathcal{T}_0$, $\mathcal{P}$, $\mathcal{A}$, or $\mathcal{A}_e$. The module $\mathcal{E}$-instantiation produces suitable instances of the extensionality axiom of arrays, i.e. (9), so that a simple satisfiability procedure for $\mathcal{A}$ is assumed available (rather than one for $\mathcal{A}_e$). The module $\mathcal{G}$-instantiation is non-deterministic and guesses sufficiently many instances of the axioms about dim, i.e. (10) and (11), as well as some facts entailed by the constraints in $\mathcal{P}$. The modules $\mathcal{P}$ and $\mathcal{A}$ implement the decision procedures for Presburger arithmetic and the theory of arrays without extensionality. The module 'all sat?' returns 'sat' if both decision procedures for $\mathcal{P}$ and $\mathcal{A}$ returned 'sat', and, otherwise, returns 'unsat'. We are now ready to describe the internal workings of each module.

4.1 Flattening

It is well-known (see, e.g., [2]) that it is possible to transform a constraint ϕ into an equisatisfiable constraint ϕ' containing only flat literals in linear time by introducing sufficiently many fresh constant symbols to name sub-terms. In our case, we assume that the module Flatten in Figure 1 transforms (in linear time) a set of arbitrary literals over the signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$, into an equisatisfiable set of flat literals on the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$, for some set $\underline{c} \supseteq \underline{a}$ of constants (the constants in $\underline{c} \setminus \underline{a}$ are said to be fresh). Notice that a flattened set of literals L can be represented as a set-theoretic union $L = L_{\mathcal{A}} \cup L_{\mathcal{P}}$, where $L_{\mathcal{A}}$ collects all the literals from L whose signature is the signature of $\mathcal{A}$ and $L_{\mathcal{P}}$ collects all the literals from L whose signature is the signature of $\mathcal{P}$ (thus $L_{\mathcal{A}} \cap L_{\mathcal{P}}$ contains precisely the literals from L whose signature is the signature of $\mathcal{T}_0$).

4.2 $\mathcal{E}$-Instantiation Closure

The $\mathcal{E}$-instantiation module in Figure 1 is based on the Skolemization of axiom (9).

Definition 4.1 ($\mathcal{E}$-instantiation closed set of literals). *A set L of ground flat literals is $\mathcal{E}$-instantiation closed iff for every negative literal of the kind $a \neq b$ that belongs to L (with $a, b :$ ARRAY), we have that $\{select(a, i) = e_1, select(b, i) = e_2, e_1 \neq e_2\} \subseteq L$, for some constants $i :$ INDEX, $e_1, e_2 :$ ELEM;*

The correctness of the module is stated below.

Lemma 4.1. *There exists a linear time algorithm which takes a set L of flat literals over the signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ and returns a $\mathcal{E}$-instantiation closed set $L^{\mathcal{E}}$ of flat literals over the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$ such that (i) $L \subseteq L^{\mathcal{E}}$, (ii) L and $L^{\mathcal{E}}$ are $\mathcal{ADP}$-equisatisfiable, and (iii) $\underline{a} \subseteq \underline{c}$.*

4.3 $\mathcal{G}$-Instantiation Closure

The module $\mathcal{G}$-instantiation is non-deterministic and it is responsible to produce suitable instances of the axioms (10) and (11) as well as to guess (hence the name of $\mathcal{G}$-instantiation) enough facts of $\mathcal{P}$ entailed by the input constraint.

Definition 4.2 ($\mathcal{G}$-instantiation closed set of literals). *A set L of ground flat literals is $\mathcal{G}$-instantiation closed iff the following conditions are satisfied:*

1. *if ε occurs in L, then $dim(\varepsilon) = 0 \in L$.*
2. *if $dim(a) = i \in L$, with $a :$ ARRAY and $i :$ INDEX, then either $\{i = 0\} \subseteq L$ or $\{e \neq \bot, select(a, j) = e, s(j) = i\} \subseteq L$ for some constant $j :$ INDEX;*
3. *if i, j occur in L, with $i, j :$ INDEX, then either $i = j \in L$ or $j \neq i \in L$;*
4. *if i, j occur in L, with $i, j :$ INDEX and $i \neq j \in L$, then either $i < j \in L$, or $j < i \in L$;*
5. *if $\{dim(a) = i, i \leq j\} \subseteq L$, with $a :$ ARRAY and $i, j :$ INDEX, then $\{select(a, j) = \bot\} \subseteq L$ (here $i \leq j$ stands for $i < j$ or $i = j$).*

```
T ⟵ {A, P}
function DP_ADP (L: set of flat literals)
    L^E ⟵ E-instantiation(L)
    for each L^G ⟵ G-instantiation(L^E) do begin
        for each T ∈ T do ρ_T ⟵ DP_T(L^G_T)
        if ⋀_{T∈T}(ρ_T = sat) then return sat
    end
    return unsat
end
```

Fig. 2. The (extensible) decision procedure for $\mathcal{ADP}$

It is not difficult to see that, given a set of literals, it is always possible to compute its $\mathcal{G}$-instantiation in (non-deterministic) polynomial time.

Lemma 4.2. *There exists a non-deterministic polynomial time algorithm which takes as input a set L of ground flat literals over a signature $\Sigma^{\underline{a}}_{\mathcal{ADP}}$ and returns a $\mathcal{G}$-instantiation closed set $L^{\mathcal{G}}$ of flat literals over the signature $\Sigma^{\underline{c}}_{\mathcal{ADP}}$ such that (i) $L \subseteq L^{\mathcal{G}}$, (ii) L and $L^{\mathcal{G}}$ are $\mathcal{ADP}$-equisatisfiable, and (iii) $\underline{a} \subseteq \underline{c}$.*

For the correctness of our decision procedure, we need sets of literals that are both $\mathcal{E}$- and $\mathcal{G}$-instantiation closed. To this aim, one can check that the $\mathcal{E}$-instantiation module has to be invoked first, followed by the $\mathcal{G}$-instantiation module.

4.4 The Decision Procedure for $\mathcal{ADP}$

Figure 2 gives an algorithmic and non-deterministic description of the decision procedure to solve the $\mathcal{ADP}$-satisfiability problem. Without loss of generality (see Section 4.1), we assume that L contains only flat literals. For a theory T with decidable satisfiability problem, we write DP_T for the decision procedure solving the T-satisfiability problem: DP_T takes a set L of Σ_T-literals and returns *sat* when L is T-satisfiable; *unsat*, otherwise. If L is a set of flat literals, then

$$L_T := \{\ell \mid \ell \in L \text{ is a } \Sigma_T\text{-literal}\},$$

where $T \in \{\mathcal{A}, \mathcal{P}\}$. So, for example, $L^{\mathcal{G}}_{\mathcal{P}}$ is the subset of the $\Sigma_{\mathcal{P}}$-literals in $L^{\mathcal{G}}$. The set T in Figure 2 contains the names of the theories for which a decision procedure is assumed available. It will be used for modularly extending the procedure in Section 5.

Let L be a set of flat $\Sigma_{\mathcal{ADP}}$-literals to be checked for $\mathcal{ADP}$-satisfiability. The decision procedure $DP_{\mathcal{ADP}}$ first computes the $\mathcal{E}$-instantiation $L^{\mathcal{E}}$ of L (recall from Lemma 4.1 that this can be done in linear time). Then, it enumerates all possible $\mathcal{G}$-instantiations (cf. the **for each** loop in Figure 2). If it is capable of finding a $\mathcal{G}$-instantiation $L^{\mathcal{G}}$ such that its $\Sigma_{\mathcal{P}}$-literals are $\mathcal{P}$-satisfiable and its $\Sigma_{\mathcal{A}}$-literals are $\mathcal{A}$-satisfiable, then $DP_{\mathcal{ADP}}$ returns the $\mathcal{ADP}$-satisfiability of the input set L of literals. Otherwise, if all possible $\mathcal{G}$-instantiations are enumerated and the test of the conditional in the body of the loop always fails, then $DP_{\mathcal{ADP}}$ returns the $\mathcal{ADP}$-unsatisfiability of the input set L of literals.

4.5 Correctness of the Decision Procedure for $\mathcal{ADP}$

The termination of $DP_{\mathcal{ADP}}$ is immediate, whereas its soundness and completeness (Theorem 4.1 below) are consequences of the following Combination Lemma.

Lemma 4.3 (Combination). *Let L be a $\mathcal{E}$- and $\mathcal{G}$-instantiation closed finite set of flat literals. Then, the following conditions are equivalent:*

(i) *L is satisfiable in a standard model of $\mathcal{ADP}$;*
(ii) *L is $\mathcal{ADP}$-satisfiable;*
(iii) *$L_{\mathcal{A}}$ is $\mathcal{A}$-satisfiable and $L_{\mathcal{P}}$ is $\mathcal{P}$-satisfiable.*

The soundness and correctness of $DP_{\mathcal{ADP}}$ is stated in the following

Theorem 4.1. *$DP_{\mathcal{ADP}}$ is a decision procedure for the $\mathcal{ADP}$-satisfiability problem, i.e. for any set L of flat literals, L is $\mathcal{ADP}$-satisfiable iff $DP_{\mathcal{ADP}}(L)$ returns sat. Furthermore, $DP_{\mathcal{ADP}}$ decides the satisfiability problem in the standard models of $\mathcal{ADP}$.*

5 Extensions of the Theory of Arrays with Dimension

We show the decidability of two interesting extensions of $\mathcal{ADP}$ (more extensions can be found in the Technical Report [9]).

5.1 Injective Arrays

The first extension of $\mathcal{ADP}$ is obtained by adding an axiom recognizing injective arrays which, according to [14], may characterize memory configurations where pointers satisfy the no-aliasing property. We extend the (empty) set of predicate symbols $\mathcal{ADP}$ by the unary predicate symbol $\mathsf{Inj} : \text{ARRAY}$ which holds for arrays containing no repeated elements, with the exception of the undefined element $\perp$ (the decidability of a similar problem is left open in [3]). To formalize the intended meaning of Inj, we consider the theory $\mathcal{ADP}_{\mathsf{inj}}$ obtained by extending $\mathcal{ADP}$ with the following defining axiom:

$$\mathsf{Inj}(a) \leftrightarrow \forall i, j(\text{select}(a, i) = \text{select}(a, j) \rightarrow i = j \vee \text{select}(a, i) = \perp) \quad (13)$$

where a is a variable of sort ARRAY. In order to obtain a decision procedure for $\mathcal{ADP}_{\mathsf{inj}}$, it is necessary to find suitable extensions of Definitions 4.1 and 4.2 so that enough instances of (13) are considered, and the results of the available decision procedures for $\mathcal{A}$ and $\mathcal{P}$ are conclusive about the satisfiability of the original constraint in the extended theory. We formalize the meaning of "enough instances" for $\mathcal{ADP}_{\mathsf{inj}}$ in the following two definitions.

Definition 5.1 ($\mathcal{E}_{\mathsf{inj}}$-instantiation closed set of literals). *A set L of ground flat literals is $\mathcal{E}_{\mathsf{inj}}$-instantiation closed iff (i) L is $\mathcal{E}$-instantiation closed (cf. Definition 4.1) and moreover for every negative literal $\neg\mathsf{Inj}(a) \in L$, there are constants $e : \text{ELEM}, i, j : \text{ARRAY}$ such that $\{select(a, i) = e, select(a, j) = e, i < j, e \neq \perp\} \subseteq L$.*

Definition 5.2 ($\mathcal{G}_{\mathsf{inj}}$-instantiation closed set of literals). *A set L of ground flat literals is $\mathcal{G}_{\mathsf{inj}}$-instantiation closed iff L is $\mathcal{G}$-instantiation closed and the following conditions are satisfied:*

1. *if $\mathsf{Inj}(a) \in L$ then, for each constant i of sort* INDEX *occurring in L, either $select(a, i) = \bot \in L$ or $\{select(a, i) = e, e \neq \bot\} \subseteq L$ for some constant e :* ELEM*;*
2. *if $\{\mathsf{Inj}(a), i < j, select(a, i) = e_1, select(a, j) = e_2, e_1 \neq \bot, e_2 \neq \bot\} \subseteq L$, then $e_1 \neq e_2 \in L$.*

Lemmas 4.1 and 4.2 can easily be adapted to the theory $\mathcal{ADP}_{\mathsf{inj}}$. Since the combination Lemma 4.3 continues to hold with Definitions 5.1 and 5.2, we can show the correctness of the decision procedure $DP_{\mathcal{ADP}_{\mathsf{inj}}}$ for $\mathcal{ADP}_{\mathsf{inj}}$, which is obtained from $DP_{\mathcal{ADP}}$ by replacing the modules for $\mathcal{E}$- and $\mathcal{G}$-instantiation in Figure 1 with those taking into account Definitions 5.1 and 5.2.

Theorem 5.1. *$DP_{\mathcal{ADP}_{\mathsf{inj}}}$ is a decision procedure for the $\mathcal{ADP}_{\mathsf{inj}}$-satisfiability problem. Furthermore, $DP_{\mathcal{ADP}_{\mathsf{inj}}}$ decides the satisfiability problem in the standard models of $\mathcal{ADP}_{\mathsf{inj}}$.*

5.2 Arrays with Domain

The *second extension* of $\mathcal{ADP}$ we consider is again motivated by applications in program verification. As already observed in [17], it is quite helpful to regard arrays as functions equipped with an operator to compute their domains. This is used, for example, to define the semantics of separating connectives (supporting local reasoning) of Separation Logic [18]. So, we extend $\mathcal{ADP}$ with a set of axioms characterizing a function which, given an array a, returns the domain $dom(a)$ of a, i.e. $dom(a)$ is the set of indexes i such that $\mathrm{select}(a, i) \neq \bot$.

To formalize this extension of $\mathcal{A}_{dim}$, we need to introduce a very simple theory of sets of indexes, which is a straightforward extension of that used in [2]. Let $\mathcal{S}^{\emptyset}$ be the theory whose sort symbols are BOOL and SET, whose function symbols are $\mathrm{true}, \mathrm{false} :$ BOOL, $\emptyset :$ SET, $\mathrm{mem} :$ INDEX $\times$ SET $\rightarrow$ BOOL, $\mathrm{ins} :$ INDEX $\times$ SET $\rightarrow$ SET, and whose axioms are the following:

$$\mathrm{mem}(i, \emptyset) = \mathrm{false} \tag{14}$$

$$\mathrm{mem}(i, \mathrm{ins}(i, s)) = \mathrm{true} \tag{15}$$

$$i_1 \neq i_2 \rightarrow \mathrm{mem}(i_1, \mathrm{ins}(i_2, s)) = \mathrm{mem}(i_1, s) \tag{16}$$

$$\mathrm{true} \neq \mathrm{false} \wedge (\forall x : \mathrm{BOOL}\ x = \mathrm{true} \vee x = \mathrm{false}) \tag{17}$$

where i, i_1, i_2 (s) are variables of sort INDEX (SET, respectively). Intuitively, $\emptyset$ denotes the empty set, mem is the test for membership of an index to a set, ins adds an index to a set if it is not already in the set. It is possible to adapt the decidability result of [2] to $\mathcal{S}^{\emptyset}$ (see [9] for details). Since we want to be able to compare sets by using the membership predicate mem, we need to consider the theory $\mathcal{S}^{\emptyset}_{e}$ obtained from $\mathcal{S}^{\emptyset}$ by adding the following axiom of extensionality for sets (here s_1, s_2 are variables of sort SET):

$$\forall i(\mathrm{mem}(i, s_1) = \mathrm{mem}(i, s_2)) \rightarrow s_1 = s_2. \tag{18}$$

Let $\mathcal{ADP}_{\mathsf{dom}}$ be the theory obtained by extending the (disjoint) union of $\mathcal{ADP}$ with $\mathcal{S}_e^{\emptyset}$ by the function symbol dom : ARRAY $\rightarrow$ SET together with the following axiom:

$$\mathrm{select}(a, i) = \bot \leftrightarrow \mathrm{mem}(i, \mathrm{dom}(a)) = \mathrm{false} \tag{19}$$

where i and a are variables of sort INDEX and ARRAY, respectively.

In order to obtain a decision procedure for $\mathcal{ADP}_{\mathsf{dom}}$, it is necessary to find suitable extensions of Definitions 4.1 and 4.2 so that enough instances of axioms (18) and (19) are considered and the results of the available decision procedures for $\mathcal{A}$, $\mathcal{P}$, and $\mathcal{S}^{\emptyset}$ are conclusive about the satisfiability of the original constraint in the extended theory. We formalize the meaning of "enough instances" for axiom (18) in the following definition.

Definition 5.3 ($\mathcal{E}_{\mathsf{set}}$-instantiation closed set of literals). *A set L of ground flat literals is $\mathcal{E}_{\mathsf{set}}$-instantiation closed iff L is $\mathcal{E}$-instantiation closed (cf. Definition 4.1) and for every literal of the kind $s_1 \neq s_2 \in L$ (with s_1, s_2 constants of sort* SET*), there are constants b_1, b_2 : BOOL, i : INDEX such that $\{mem(i, s_1) = b_1, mem(i, s_2) = b_2, b_1 \neq b_2\} \subseteq L$.*

Instead of using guessing as for $\mathcal{ADP}_{\mathsf{inj}}$ in Section 5.2, we adopt the rewriting-approach to satisfiability procedures of [2]. We use the superposition calculus (from now on denoted by $\mathcal{SP}$) to build a rewriting-based decision procedure for the satisfiability problem in the union of the theories $\mathcal{A}_e$ and $\mathcal{S}_e^{\emptyset}$ extended with axiom (19). Such a procedure is then combined with a decision procedure for the satisfiability problem in $\mathcal{P}$ to build a decision procedure for $\mathcal{ADP}_{\mathsf{dom}}$.

In [2], it is shown how to use $\mathcal{SP}$ to build decision procedures for theories axiomatized by a finite set of first-order clauses. The key observation is that, in order to show that $\mathcal{SP}$ is a decision procedure, it is sufficient to prove that $\mathcal{SP}$ terminates on the set of clauses obtained by the union of the axioms of the theory and an arbitrary set of ground and flat literals. According to [2], $\mathcal{SP}$ terminates also for some of the theories considered in this paper, e.g., $\mathcal{A}$ and $\mathcal{S}_-^{\emptyset} := \mathcal{S}^{\emptyset} \setminus \{(17)\}$ (when considered in isolation). Modularity results in [1] allow us to conclude that $\mathcal{SP}$ also terminates for the union $\mathcal{A} \cup \mathcal{S}_-^{\emptyset}$. Unfortunately, this is not enough here since our goal is to build a decision procedure $\mathcal{ADP}_{\mathsf{dom}}$ whose set of axioms also contains (17) and (19).

Below, we develop the termination result for $\mathcal{SP}$ necessary to replace guessing as for $\mathcal{ADP}_{\mathsf{inj}}$ (cf. Section 5.1) with $\mathcal{SP}$. Notice that $\mathcal{SP}$ is used in two ways: to check for unsatisfiability in the theory of equality and to find enough instances of the axioms of $\mathcal{A} \cup \mathcal{S}_-^{\emptyset}$ together with (17) and (19). A similar approach has also been investigated in [11] (for the theories already considered in [2]) to enable the efficient combination of rewriting-based satisfiability procedures with a decision procedure for $\mathcal{P}$.

Let L be a set of ground and flat $\Sigma_{\mathcal{A}\cup\mathcal{S}^\emptyset}$-literals; we define $\mathcal{I}_L$ to be the following set of (partial) instances of axioms (17) and (19):

$$\begin{aligned} \mathrm{select}(a,x) \neq \bot \ \ \vee \ \ \mathrm{mem}(x,\mathrm{dom}(a)) \neq \mathrm{true}, \\ \mathrm{select}(a,x) = \bot \ \ \vee \ \ \mathrm{mem}(x,\mathrm{dom}(a)) = \mathrm{true}, \\ \mathrm{true} \neq \mathrm{false}, \text{ and } b = \mathrm{true} \vee b = \mathrm{false} \end{aligned}$$

for each $\mathrm{dom}(a) = s$ in L and for each constant $b : \mathrm{BOOL}$ occurring in L.

Lemma 5.1. *$\mathcal{SP}$ terminates on $\mathcal{A}\cup\mathcal{S}^\emptyset_-\cup\mathcal{I}_L\cup L$ for every set L of $\Sigma_{\mathcal{A}\cup\mathcal{S}^\emptyset}$-literals.*

In the following, we denote with $DP_{\mathcal{SP}}$ the function taking a set L of $\mathcal{E}_{\mathsf{set}}$-instantiated $\Sigma_{\mathcal{A}\cup\mathcal{S}^\emptyset}$-literals, computing $\mathcal{I}_L$, and then invoking $\mathcal{SP}$ on the clauses $\mathcal{A} \cup \mathcal{S}^\emptyset_- \cup \mathcal{I}_L \cup L$. If the empty clause is derived by $\mathcal{SP}$, then $DP_{\mathcal{SP}}$ returns *unsat*; *sat*, otherwise. The decision procedure $DP_{\mathcal{ADP}_{\mathsf{dom}}}$ for the theory $\mathcal{ADP}_{\mathsf{dom}}$ is obtained from $DP_{\mathcal{ADP}}$ by replacing the module for $\mathcal{E}$-instantiation in Figure 1 with a module for $\mathcal{E}_{\mathsf{set}}$-instantiation (cf. Definition 5.3) and by calling $DP_{\mathcal{SP}}$ instead of $DP_{\mathcal{A}}$ in the loop of Figure 2.

Theorem 5.2. *$DP_{\mathcal{ADP}_{\mathsf{dom}}}$ is a decision procedure for the $\mathcal{ADP}_{\mathsf{dom}}$-satisfiability problem.*

6 Conclusion

We have considered extensions of the theory of arrays which are relevant for many important applications such as program verification. These extensions are such that the indexes of arrays has the algebraic structure of Presburger Arithmetic and the theory of arrays is augmented with axioms characterizing additional symbols such as dimension, injectivity, or the domain of definition of arrays. We have obtained the decidability of all the considered extensions by a combination of decision procedures for the theories of arrays and Presburger Arithmetic with various instantiation strategies based both on model-theoretic and rewriting-based methods.

References

1. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal. In *Proc. of 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *LNCS*, 2005.
2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.
3. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Proc. of 7th Int. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, volume 3855 of *LNCS*, 2006.
4. P. J. Downey and R. Sethi. Assignment commands with array references. *Journal of the ACM*, 25(4):652–666, 1978.

5. H. B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York-London, 1972.
6. J. H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.
7. H. Ganzinger and K. Korovin. Integrating equational reasoning in instantiation-based theorem proving. In *Proc. of Computer Science in Logic (CSL'04)*, volume 3210 of *LNCS*, 2004.
8. S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4):221–249, 2004.
9. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Deciding extension of the theory of arrays by integrating decision procedures and instantiation strategies. Rapporto Interno DSI 309-06, Università degli Studi di Milano, Milano (Italy), 2006. Available at `http://homes.dsi.unimi.it/~zucchell/publications/techreport/GhiNiRaZu-RI309-06.pdf`.
10. J. Jaffar. Presburger arithmetic with array segments. *Information Processing Letters*, 12(2):79–82, 1981.
11. H. Kirchner, S. Ranise, C. Ringeissen, and D.-K. Tran. On superposition-based satisfiability procedures and their combination. In *Proc. of the 2nd Int. Conf. on Theoretical Aspects of Computing (ICTAC'05)*, volume 3722 of *LNCS*, 2005.
12. P. Mateti. A decision procedure for the correctness of a class of programs. *Journal of the ACM*, 28(2):215–232, 1981.
13. J. McCarthy. Towards a mathematical theory of computation. In *Proceedings of IFIP Congress*, 1962.
14. S. McPeak and G. Necula. Data structures specification via local equality axioms. In *Proc. of 17th Int. Conf. on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, 2005.
15. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Transaction on Programming Languages and Systems*, 1(2):245–257, 1979.
16. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*. 2001.
17. J. C. Reynolds. Reasoning about arrays. *Communications of the ACM*, 22(5):290–299, 1979.
18. J. C. Reynolds. Separation logic: a logic for shared mutable data structures, 2002.
19. A. Stump, C. W. Barrett, D. L. Dill, and J. Levitt. A decision procedure for an extensional theory of arrays. In *Proc. of the 16th IEEE Symposium on Logic in Computer Science (LICS'01)*. IEEE Computer Society, 2001.
20. N. Suzuki and D. R. Jefferson. Verification decidability of presburger array programs. *Journal of the ACM*, 27(1):191–205, 1980.

Analytic Tableau Calculi for KLM Rational Logic R

Laura Giordano[1], Valentina Gliozzi[2], Nicola Olivetti[3], and Gian Luca Pozzato[2]

[1] Dipartimento di Informatica - Università del Piemonte Orientale A. Avogadro - via Bellini 25/G - 15100 Alessandria, Italy
laura@mfn.unipmn.it
[2] Dipartimento di Informatica - Università degli Studi di Torino, corso Svizzera 185 - 10149 Turin - Italy
{gliozzi, pozzato}@di.unito.it
[3] LSIS - UMR CNRS 6168 Université Paul Cézanne (Aix-Marseille 3) Avenue Escadrille Normandie-Niemen 13397 Marseille Cedex 20 - France
nicola.olivetti@univ.u-3mrs.fr, nicola.olivetti@lsis.org

Abstract. In this paper we present a tableau calculus for the rational logic **R** of default reasoning, introduced by Kraus, Lehmann and Magidor. Our calculus is obtained by introducing suitable modalities to interpret conditional assertions, and makes use of labels to represent possible worlds. We also provide a decision procedure for **R**, and study its complexity.

1 Introduction

In [1] Kraus, Lehmann and Magidor (KLM) proposed a formalization of nonmonotonic reasoning that led to a classification of nonmonotonic consequence relations, determining a hierarchy of stronger and stronger systems. The so called *KLM properties* have been widely accepted as the "conservative core" of default reasoning. The role of KLM logics is similar to the role of AGM postulates in Belief Revision [2]: they give a set of postulates for default reasoning that any concrete reasoning mechanism should satisfy.

In the recent literature it is shown that many different approaches to default reasoning are characterized by these properties. In particular, a recent work by Halpern and Friedman [3] has shown that two of these systems, namely preferential logic **P** and rational logic **R**, are natural and general systems: surprisingly enough, the axiom systems of these logics are complete with respect to a wide spectrum of semantics (including κ-rankings, parametrized probabilistic structures, ϵ-semantics and possibilistic structures). The reason is that all these structures are examples of *plausibility structures* and the truth in them is captured by the axioms of preferential or rational logic.

The results presented in [3], and their extensions to the first order setting [4], are the source of a renewed interest in the KLM framework. A considerable amount of research in the area has then concentrated in developing concrete mechanisms for plausible reasoning in accordance with KLM systems (**P** and **R** mostly). These mechanisms are defined by exploiting a variety of models of reasoning under uncertainty (ranked models, belief functions, possibilistic logic, etc. [5, 6, 7, 8, 9, 10]) that provide, as we remarked, alternative semantics to KLM systems. The mechanisms can be seen as restricting the consideration to preferred classes of models of KLM logics; this is also the case of Lehmann's notion of rational closure (not to be confused with the logic **R**).

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 190–202, 2006.

More recent research has also explored the integration of KLM framework with paraconsistent logics [11]. Finally, there has been some recent investigation on the relation between KLM systems and decision-theory [12, 13].

In KLM logics, defeasible knowledge is assumed to be represented by a set of nonmonotonic conditionals or assertions of the form $A \mathrel{|\!\sim} B$, whose reading is *normally (or typically) the A's are B's*. The operator "$\mathrel{|\!\sim}$" is nonmonotonic, in the sense that $A \mathrel{|\!\sim} B$ does not imply $A \wedge C \mathrel{|\!\sim} B$.

In this paper we focus on logic **R**, whose axiom system includes the rule of *rational monotonicity*: if $A \mathrel{|\!\sim} B$ and $\neg(A \mathrel{|\!\sim} \neg C)$ hold, then one can infer $A \wedge C \mathrel{|\!\sim} B$. This rule allows a conditional to be inferred from a set of conditionals in absence of other information. More precisely, "it says that an agent should not have to retract any previous defeasible conclusion when learning about a new fact the negation of which was not previously derivable" [14].

Consider, for instance, a knowledge base K containing the following set of conditional assertions: $adult \mathrel{|\!\sim} worker$, $adult \mathrel{|\!\sim} taxpayer$, $student \mathrel{|\!\sim} adult$, $student \mathrel{|\!\sim} \neg worker$, $student \mathrel{|\!\sim} \neg taxpayer$, whose meaning is that adults typically work, adults typically pay taxes, students are typically adults, but they typically do not work, nor do they pay taxes. In rational logic **R** one can infer the following conditional assertions from the knowledge base K: $adult \mathrel{|\!\sim} \neg student$ (i.e. typical adults are not students), $adult \wedge student \mathrel{|\!\sim} \neg worker$ (giving preference to more specific information). Moreover, if one further knows that $\neg(adult \mathrel{|\!\sim} \neg married)$ (i.e. it is not the case the adults are typically unmarried), one can also infer that $adult \wedge married \mathrel{|\!\sim} worker$. Observe that one cannot infer $student \mathrel{|\!\sim} worker$.

From a semantic point of view, the models of rational logic are possible-world structures equipped with a preference relation among worlds. The preference relation (an irreflexive and transitive relation) on worlds is further assumed to be *modular*. The meaning of a conditional assertion $A \mathrel{|\!\sim} B$ is that B holds in the *most preferred* worlds where A holds.

In this work we extend the investigation of tableau procedures for propositional KLM logics developed in [15] by considering the case of **R**. Our approach is based on a novel interpretation of **R** into modal logics. As a difference with previous approaches (e.g. Crocco et. al [16] and Boutillier [17]), that take S4.3 as the modal counterpart of **R**, we consider here an extension of modal logic G, including modularity of the preference relation. The idea is simply to interpret the preference relation as an accessibility relation: a conditional $A \mathrel{|\!\sim} B$ holds in a model if B is true in all minimal A-worlds, i.e. worlds in which A holds, that are minimal w.r.t. $<$. A world is A-minimal if all smaller worlds are not A-worlds. The relation with modal logic G is motivated by the fact that we assume, following KLM, the so-called *smoothness condition*, which is related to the well-known *limit assumption*. This condition ensures that minimal A-worlds exist, by preventing infinitely descending chains of worlds. This condition is therefore ensured by the finite-chain condition on the accessibility relation (as in modal logic G). As it has been done in [15] for preferential logic and loop-cumulative logic, our tableau method provides a sort of run-time translation of **R** into the extension of modal logic G. As a difference with [15], we develop here a *labelled* tableau system, which seems to be the most natural approach.

The paper presents a tableau calculus for **R** which is sound, complete and terminating. Moreover, it defines a systematic procedure which allows the satisfiability problem for **R** to be decided in nondeterministic polynomial time, in accordance with the known complexity results for this logic.

2 KLM Rational Logic R

In this section we briefly recall the axiomatization and semantics of the rational logic **R**. For a complete description of KLM systems, see [1] and [14]. The language of KLM logics consists just of conditional assertions $A \mathrel{|\!\sim} B$. We consider a richer language allowing boolean combinations of conditional and propositional formulas[1]. Our language $\mathcal{L}$ is defined from a set of propositional variables ATM, the boolean connectives and the conditional operator $\mathrel{|\!\sim}$. We use $A, B, C, ...$ to denote propositional formulas, whereas $F, G, ...$ are used to denote all formulas (even conditionals); $\Gamma, \Delta, ...$ represent sets of formulas. The formulas of $\mathcal{L}$ are defined as follows: if A is a propositional formula, $A \in \mathcal{L}$; if A and B are propositional formulas, $A \mathrel{|\!\sim} B \in \mathcal{L}$; if F is a boolean combination of formulas of $\mathcal{L}$, $F \in \mathcal{L}$. $\mathcal{L}$ corresponds to the fragment of the language of conditional logics without nested conditionals $\mathrel{|\!\sim}$.

The axiomatization of **R** consists of all axioms and rules of propositional calculus together with the following axioms and rules (notice that $\vdash$ denotes provability in the propositional calculus):

- REF. $A \mathrel{|\!\sim} A$ (reflexivity)
- LLE. If $\vdash A \leftrightarrow B$, then $\vdash (A \mathrel{|\!\sim} C) \rightarrow (B \mathrel{|\!\sim} C)$ (left logical equivalence)
- RW. If $\vdash A \rightarrow B$, then $\vdash (C \mathrel{|\!\sim} A) \rightarrow (C \mathrel{|\!\sim} B)$ (right weakening)
- AND. $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \mathrel{|\!\sim} B \wedge C)$
- OR. $((A \mathrel{|\!\sim} C) \wedge (B \mathrel{|\!\sim} C)) \rightarrow (A \vee B \mathrel{|\!\sim} C)$
- CM. $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \wedge B \mathrel{|\!\sim} C)$ (cautious monotonicity)
- RM. $((A \mathrel{|\!\sim} B) \wedge \neg(A \mathrel{|\!\sim} \neg C)) \rightarrow (A \wedge C \mathrel{|\!\sim} B)$ (rational monotonicity)

REF states that A is always a default conclusion of A. LLE states that the syntactic form of the antecedent of a conditional formula is irrelevant. RW describes a similar property of the consequent. This allows to combine default and logical reasoning [3]. AND states that it is possible to combine two default conclusions. OR states that it is allowed to reason by cases: if C is the default conclusion of two premises A and B, then it is also the default conclusion of their disjunction. CM states that if B and C are two default conclusions of A, then adding one of the two conclusions to A will not cause the retraction of the other conclusion. As explained in the Introduction, RM captures a natural form of monotonicity.

The semantics of **R** is defined by considering possible world structures with a preference relation (a strict partial order) $w < w'$ whose meaning is that w is preferred to w'. Moreover, the preference relation is supposed to be *modular*, i.e. for all w, w_1 and w_2, if $w_1 < w_2$ then either $w_1 < w$ or $w < w_2$. We have that $A \mathrel{|\!\sim} B$ holds in

[1] In [14] it is shown that in a language lacking boolean combinations of conditionals, system **R** collapses into system **P**. As shown in [3], the two systems are distinct given a richer language (as ours) allowing boolean combinations of conditionals.

a model $\mathcal{M}$ if B holds in all *minimal A-worlds* (w.r.t. $<$). This definition makes sense provided minimal A-worlds exist (whenever there are A-worlds). This is ensured by the *smoothness condition* in the next definition.

Definition 1 (Semantics of R, Definition 14 in [14]). *A rational model is a triple $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ where: $\mathcal{W}$ is a non-empty set of items called worlds; $<$ is an irreflexive, transitive and modular relation on $\mathcal{W}$; V is a function $V : \mathcal{W} \longmapsto pow(ATM)$, which assigns to every world w the set of atoms holding in that world. We define the truth conditions for a formula F as follows:*

- *If F is a boolean combination of formulas, $\mathcal{M}, w \models F$ is defined as for propositional logic;*
- *Let A be a propositional formula; we define $Min_<(A) = \{w \in \mathcal{W} \mid \mathcal{M}, w \models A$ and $\forall w'$, $w' < w$ implies $\mathcal{M}, w' \not\models A\}$;*
- *$\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if for all w', if $w' \in Min_<(A)$ then $\mathcal{M}, w' \models B$.*

The relation $<$ satisfies the smoothness condition*: if $\mathcal{M}, w \models A$ then $w \in Min_<(A)$ or $\exists w' \in Min_<(A)$ such that $w' < w$.*
We say that a formula F is valid in a model $\mathcal{M}$ *($\mathcal{M} \models F$), if $\mathcal{M}, w \models F$ for every $w \in \mathcal{W}$. A formula is* valid *if it is valid in every model $\mathcal{M}$.*

Notice that the truth conditions for conditional formulas are given with respect to single possible worlds for uniformity sake. Since the truth value of a conditional only depends on global properties of $\mathcal{M}$, we have that: $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ iff $\mathcal{M} \models A \mathrel{|\!\sim} B$.

By transitivity of the relation $<$, the smoothness condition is equivalent to the following *strong smoothness condition*: given a formula A and a world w, if there is $w' < w$ such that $\mathcal{M}, w' \models A$, then either $w' \in Min_<(A)$ or there exists $w'' < w$ such that $w'' \in Min_<(A)$. Observe also that by the modularity of $<$ it follows that possible worlds of $\mathcal{W}$ are *clustered* into equivalence classes, each class consisting of worlds that are incomparable to one another; the classes are totally ordered[2]. In other words the property of modularity determines a *ranking* of worlds so that the semantics of **R** can be specified equivalently in terms of *ranked* models [14]. By means of the modularity condition on the preference relation, we can also prove the following theorem:

Theorem 1 (Small Model Theorem). *For any $\Gamma \subseteq \mathcal{L}$, if Γ is satisfiable in a rational model, then it is satisfiable in a rational model containing at most n worlds, where n is the size of Γ, i.e. the length of the string representing Γ.*

3 The Tableau Calculus for R

In this section we present $\mathcal{T}\mathbf{R}$, a labelled tableau calculus for rational logic **R**. The calculus makes use of labels to represent possible worlds. We consider a language $\mathcal{L}_R$ and a denumerable alphabet of labels $\mathcal{A}$, whose elements are denoted by *x, y, z, ...* $\mathcal{L}_R$ extends $\mathcal{L}$ by formulas of the form $\Box A$, where A is propositional, whose intuitive meaning is as follows: $\Box A$ holds in a world w if A holds in all the worlds w' such that $w' < w$, that is to say:

[2] Notice that the worlds themselves may be incomparable since the relation $<$ is not assumed to be (weakly) connected.

Definition 2 (Truth condition of $\Box$). *$\mathcal{M}, w \models \Box A$ if for every $w' \in \mathcal{W}$ if $w' < w$ then $\mathcal{M}, w' \models A$.*

It is easy to see that $\Box$ has (among others) the properties of the modal system G, whose characterizing axiom is $\Box(\Box A \rightarrow A) \rightarrow \Box A$ (see for instance [18]). This axiom guarantees that the accessibility relation (defined as xRy if $y < x$) is transitive and does not have infinite ascending chains. From definition of $Min_<(A)$ in Definition 1 above, it follows that for any formula A, $w \in Min_<(A)$ iff $\mathcal{M}, w \models A \wedge \Box\neg A$. As we will see, $\mathcal{TR}$ will only make use of boxed formulas with a negated argument, i.e. with the form $x : \Box\neg A$.

Our tableau calculus comprises two kinds of labelled formulas: (i) *world formulas* $x : F$, whose meaning is that F holds in the possible world represented by x; (ii) *relation formulas* of the form $x < y$, where $x, y \in \mathcal{A}$, used to represent the relation $<$. We denote by $\alpha, \beta \ldots$ a world or a relation formula.

We define $\Gamma^M_{x \rightarrow y} = \{y : \neg A, y : \Box\neg A \mid x : \Box\neg A \in \Gamma\}$. The calculus $\mathcal{TR}$ is presented in Figure 1. We call *dynamic* the rules $(\mid\!\sim^-)$ and $(\Box^-)$ that introduce new labels in their conclusion; all the other rules are called *static*.

$$(\mathbf{AX})\ \Gamma, x : P, x : \neg P \quad \text{with } P \in ATM \qquad (\neg)\ \frac{\Gamma, x : \neg\neg F}{\Gamma, x : F}$$

$$(\wedge^+)\ \frac{\Gamma, x : F \wedge G}{\Gamma, x : F, x : G} \qquad (\wedge^-)\ \frac{\Gamma, x : \neg(F \wedge G)}{\Gamma, x : \neg F \qquad \Gamma, x : \neg G}$$

$$(\mid\!\sim^+)\ \frac{\Gamma, u : A \mid\!\sim B}{\Gamma, x : \neg A, u : A \mid\!\sim B \qquad \Gamma, x : \neg\Box\neg A, u : A \mid\!\sim B \qquad \Gamma, x : B, u : A \mid\!\sim B}$$

$$(\mid\!\sim^-)\ \frac{\Gamma, u : \neg(A \mid\!\sim B)}{\Gamma, x : A, x : \Box\neg A, x : \neg B}\ x \text{ new label} \qquad (\Box^-)\ \frac{\Gamma, x : \neg\Box\neg A}{\Gamma, y < x, \Gamma^M_{x \rightarrow y}, y : A, y : \Box\neg A}\ y \text{ new label}$$

$$(<)\ \frac{\Gamma, x < y}{\Gamma, x < y, z < y, \Gamma^M_{y \rightarrow z} \qquad \Gamma, x < y, x < z, \Gamma^M_{z \rightarrow x}}\ z \text{ occurs in } \Gamma \text{ and } \{x < z, z < y\} \cap \Gamma = \emptyset$$

Fig. 1. The calculus $\mathcal{TR}$. To save space, rules for $\rightarrow$ and $\vee$ are omitted.

Definition 3 (Truth conditions of formulas of $\mathcal{TR}$). *Given a model $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ and a labelled alphabet $\mathcal{A}$, we consider a mapping $I : \mathcal{A} \mapsto \mathcal{W}$. Given a formula α of the calculus $\mathcal{TR}$, we define $\mathcal{M} \models_I \alpha$ as follows: $\mathcal{M} \models_I x : F$ iff $\mathcal{M}, I(x) \models F$; $\mathcal{M} \models_I x < y$ iff $I(x) < I(y)$.*

We say that a set Γ of formulas of $\mathcal{TR}$ is satisfiable if, for all formulas $\alpha \in \Gamma$, we have that $\mathcal{M} \models_I \alpha$, for some model $\mathcal{M}$ and some mapping I.

A tableau is a tree whose nodes are sets of formulas Γ. Therefore, a branch is a sequence of sets of formulas $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ Each node Γ_i is obtained by its immediate predecessor Γ_{i-1} by applying a rule of $\mathcal{T}\mathbf{R}$, having Γ_{i-1} as the premise and Γ_i as one of its conclusions. A branch is closed if one of its nodes is an instance of $(\mathbf{AX})$, otherwise it is open. We say that a tableau is closed if all its branches are closed.

In order to verify that a set of formulas Γ is unsatisfiable, we label all the formulas in Γ with a new label x, and verify that the resulting set of labelled formulas has a closed tableau. For instance, in order to verify that the set $\{adult \mathrel{|\!\sim} worker,\ \neg(adult \mathrel{|\!\sim} \neg married),\ \neg(adult \wedge married \mathrel{|\!\sim} worker)\}$ is unsatisfiable (thus $adult \wedge married \mathrel{|\!\sim} worker$ is entailed by $\{adult \mathrel{|\!\sim} worker,\ \neg(adult \mathrel{|\!\sim} \neg married)\}$), we can build the closed tableau in Figure 2.

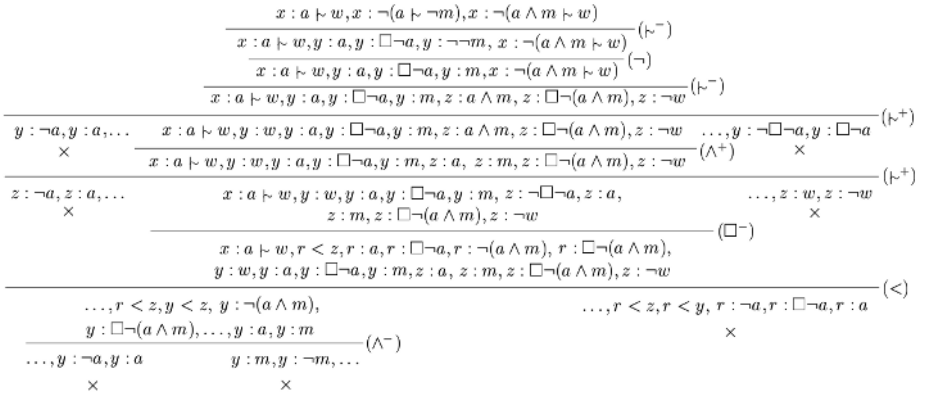


Fig. 2. A derivation in $\mathcal{T}\mathbf{R}$ of $\{adult \mathrel{|\!\sim} worker,\ \neg(adult \mathrel{|\!\sim} \neg married),\ \neg(adult \wedge married \mathrel{|\!\sim} worker)\}$. To save space, we use a for $adult$, m for $married$, and w for $worker$.

Lemma 1. *For any set of formulas Γ and any world formula $x : F$, there is a closed tableau for $\Gamma, x : F, x : \neg F$.*

The calculus $\mathcal{T}\mathbf{R}$ is sound and complete w.r.t. the semantics.

Theorem 2 (Soundness). *If there is a closed tableau for a set of formulas Γ, then Γ is unsatisfiable.*

Proof. By induction on the height of the closed tableau for Γ. If Γ is an axiom, then $x : P \in \Gamma$ and $x : \neg P \in \Gamma$, therefore there is no $w \in \mathcal{W}$ such that $\mathcal{M}, w \models P$ and $\mathcal{M}, w \not\models P$, and Γ is unsatisfiable. For the inductive step, we prove the contrapositive, i.e. we prove for each rule that, if the premise is satisfiable, so is (at least) one of the conclusions. To save space, we only present the most interesting case of $(\Box^-)$. Since the premise is satisfiable, then there is a model $\mathcal{M}$ and a mapping I such that $\mathcal{M} \models_I \Gamma, x : \neg\Box\neg A$. Let $w \in \mathcal{W}$ such that $I(x) = w$; this means that $\mathcal{M}, w \not\models \Box\neg A$, hence there exists a world $w' < w$ such that $\mathcal{M}, w' \models A$. By the

strong smoothness condition, we have that there exists a *minimal* such world, so we can assume that $w' \in Min_<(A)$, thus $\mathcal{M}, w' \models \Box\neg A$. In order to prove that the conclusion of the rule is satisfiable, we construct a mapping I' as follows: let y be a new label, not occurring in the current branch; we define (1) $I'(u) = I(u)$ for all $u \neq y$ and (2) $I'(y) = w'$. Since y does not occur in Γ, it follows that $\mathcal{M} \models_{I'} \Gamma$. By Definition 3, we have that $\mathcal{M} \models_{I'} y < x$ since $w' < w$. Moreover, since $I'(y) = w'$, we have that $\mathcal{M} \models_{I'} y : A$ and $\mathcal{M} \models_{I'} y : \Box\neg A$. Finally, $\mathcal{M} \models_{I'} \Gamma^M_{x\to y}$ follows from the fact that $I'(y) < I'(x)$ and from the transitivity of $<$. The only conclusion of the rule is then satisfiable in $\mathcal{M}$ via I'. ■

In order to prove the completeness of the calculus, we introduce the notion of saturated branch and we show that $\mathcal{T}\mathbf{R}$ introduces a finite number of labels in a tableau.

Definition 4 (Saturated branch). *We say that a branch* $\mathbf{B} = \Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ *of a tableau is* saturated *if the following conditions hold:* (1) *For the boolean connectives, the condition of saturation is defined in the usual way. For instance, if* $x : A \wedge B \in \Gamma_i$ *in* $\mathbf{B}$*, then there exists* Γ_j *in* $\mathbf{B}$ *such that* $x : A \in \Gamma_j$ *and* $x : B \in \Gamma_j$*.* (2) *If* $x : A \mathrel{|\!\sim} B \in \Gamma_i$*, then for any label* y *in* $\mathbf{B}$*, there exists* Γ_j *in* $\mathbf{B}$ *such that either* $y : \neg A \in \Gamma_j$ *or* $y : \neg\Box\neg A \in \Gamma_j$ *or* $y : B \in \Gamma_j$*.* (3) *If* $x : \neg(A \mathrel{|\!\sim} B) \in \Gamma_i$*, then there is a* Γ_j *in* $\mathbf{B}$ *such that, for some* y*,* $y : A \in \Gamma_j$*,* $y : \Box\neg A \in \Gamma_j$*, and* $y : \neg B \in \Gamma_j$*.* (4) *If* $x : \neg\Box\neg A \in \Gamma_i$*, then there exists* Γ_j *in* $\mathbf{B}$ *such that, for some* y*,* $y < x \in \Gamma_j$*,* $y : A \in \Gamma_j$ *and* $y : \Box\neg A \in \Gamma_j$*.* (5) *If* $x < y \in \Gamma_i$*, then for all labels* z *in* $\mathbf{B}$*, there exists* Γ_j *in* $\mathbf{B}$ *such that either* $z < y \in \Gamma_j$ *or* $x < z \in \Gamma_j$*.*

We can easily show the following Lemma:

Lemma 2. *Given a tableau starting with* $x_0 : F$*, for any saturated branch* $\mathbf{B} = \Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$*, we have that:*

- *if* $z < y \in \Gamma_i$ *in* $\mathbf{B}$ *and* $y < x \in \Gamma_j$ *in* $\mathbf{B}$*, then there exists* Γ_k *in* $\mathbf{B}$ *such that* $z < x \in \Gamma_k$*;*
- *if* $x : \Box\neg A \in \Gamma_i$ *in* $\mathbf{B}$ *and* $y < x \in \Gamma_j$ *in* $\mathbf{B}$*, then there exists* Γ_k *in* $\mathbf{B}$ *such that* $y : \neg A \in \Gamma_k$ *and* $y : \Box\neg A \in \Gamma_k$*;*
- *for no* Γ_i *in* $\mathbf{B}$*,* $x < x \in \Gamma_i$*.*

Notice that in $\mathcal{T}\mathbf{R}$ the order of application of the rules is not relevant, since all the rules are invertible. Hence, no backtracking is required in the calculus, and we can assume without loss of generality that a given set of formulas Γ has a unique tableau.

In Theorem 3 below we prove that the tableau for a given set of formulas Γ_0 contains a finite number of labels. Indeed, the only rules that can introduce new labels in the tableau are $(\mathrel{|\!\sim}^-)$ and $(\Box^-)$. We prove that in the tableau there can be only finitely many applications of these rules. Intuitively, the rule $(\mathrel{|\!\sim}^-)$ can be applied only once for each negated conditional Γ (hence it introduces only a finite number of labels). Furthermore, the generation of infinite branches due to the interplay between rules $(\mathrel{|\!\sim}^+)$ and $(\Box^-)$ cannot occur. Indeed, each application of $(\Box^-)$ to a formula $x : \neg\Box\neg A$ (introduced by $(\mathrel{|\!\sim}^+)$) adds the formula $y : \Box\neg A$ to the conclusion, so that $(\mathrel{|\!\sim}^+)$ can no longer

consistently introduce $y : \neg\Box\neg A$. This is due to the properties of $\Box$, that are similar to the corresponding modality of modal system G.

In order to prove this result in a rigorous manner, we proceed as follows: first, we introduce the measure of Definition 6, and the auxiliary Definition 5; then, we prove that each application of $(\mid\!\sim^-)$ and $(\Box^-)$ reduces this measure, until the two rules are no longer applicable. We write $A \mid\!\sim B \in_+ \Gamma$ (resp. $A \mid\!\sim B \in_- \Gamma$) if $A \mid\!\sim B$ occurs positively (resp. negatively) in Γ, where positive and negative occurrences are defined in the standard way.

Definition 5. *Given an initial set of formulas Γ_0, we define: (i) the set $\mathcal{L}^{\Gamma_0}_{\Box+}$ of boxed formulas $\Box\neg A$ that can be generated in a tableau for Γ_0, i.e. $\mathcal{L}^{\Gamma_0}_{\Box+} = \{\Box\neg A \mid A \mid\!\sim B \in_+ \Gamma_0\} \cup \{\Box\neg A \mid A \mid\!\sim B \in_- \Gamma_0\}$. We let $n_0 =\mid \mathcal{L}^{\Gamma_0}_{\Box+} \mid$; (ii) the* multiset $\mathcal{L}^{\Gamma_0}_{\Box-}$ *of negated boxed formulas that can be generated in a tableau for Γ_0, i.e. $\mathcal{L}^{\Gamma_0}_{\Box-} = [\neg\Box\neg A \mid A \mid\!\sim B \in_+ \Gamma_0]$. We let $k_0 =\mid \mathcal{L}^{\Gamma_0}_{\Box-} \mid$.*
Given a label x and a set of formulas Γ in the tableau for the initial set Γ_0, we define: (i) *the number n_x of positive boxed formulas $\Box\neg A$ not labelled by x, i.e. $n_x = n_0 - \mid \{\Box\neg A \in \mathcal{L}^{\Gamma_0}_{\Box+} \mid x : \Box\neg A \in \Gamma\} \mid$;* (ii) *the number k_x of negated boxed formulas $\neg\Box\neg A$ not yet expanded in a world x, i.e. $k_x = k_0 - \mid [\neg\Box\neg A \in \mathcal{L}^{\Gamma_0}_{\Box-} \mid x : \neg\Box\neg A$ has been expanded$] \mid$.*

Definition 6. *We define $p(\Gamma) = \langle c_1, c_2\rangle$ where:*

- $c_1 =\mid \{u : A \mid\!\sim B \in_- \Gamma\} \mid$
- *c_2 is the multiset given by $[c_2^{x_1}, c_2^{x_2}, \ldots, c_2^{x_n}]$, where $x_1, x_2, \ldots, x_n$ are the labels occurring in Γ and, given a label x, c_2^x is a pair (n_x, k_x) in a lexicographic order (n_x and k_x are defined as in Definition 5). We consider the integer multiset ordering given by c_2.*

We consider the lexicographic order given by $p(\Gamma)$.

Roughly speaking, c_1 is the number of negated conditionals that can still be expanded in the tableau. c_2 keeps track of positive conditionals *which can still create a new world*. The application of $(\mid\!\sim^-)$ reduces c_1. The application of $(\Box^-)$ reduces c_2. Indeed, if $(\mid\!\sim^+)$ is applied to $u : A \mid\!\sim B$, this application introduces a branch containing $x : \neg\Box\neg A$; when a new world y is generated by an application of $(\Box^-)$ on $x : \neg\Box\neg A$, $y : \Box\neg A$ is added to the current set of formulas. If $(\mid\!\sim^+)$ is applied to $u : A \mid\!\sim B$ by using the new world y, then the conclusion where $y : \neg\Box\neg A$ is introduced is closed, by the presence of $y : \Box\neg A$.

Theorem 3. *Given a set of formulas Γ, the tableau generated by $\mathcal{T}\mathbf{R}$ for Γ only contains a finite number of labels.*

Proof sketch. First, we can easily prove that each application of $(\mid\!\sim^-)$ and $(\Box^-)$ reduces $p(\Gamma)$. This means that a finite number of applications of these rules leads either to a node containing $x : F, x : \neg F$ (see Lemma 1) or to a node to which the two rules are no further applicable. In particular, when $c_1 = 0$, $(\mid\!\sim^-)$ is no longer applicable. When

$c_2 = [(0,0),(0,0),\ldots,(0,0)]$, we can reason as follows: suppose there is $x : \neg\Box\neg A \in \Gamma$; since $c_2^x = (0,0)$, it follows that $x : \Box\neg A \in \Gamma$, and we conclude by Lemma 1. ■

Theorem 4 (Completeness). *If a set of formulas Γ is unsatisfiable, then it has a closed tableau.*

Proof sketch. We show the contrapositive, i.e. if there is no closed tableau for Γ, then Γ is satisfiable. Consider the tableau starting with the set of formulas $\{x : F$ such that $F \in \Gamma\}$ and any open, saturated branch $\mathbf{B} = \Gamma_1, \Gamma_2, \ldots, \Gamma_n$ in it. Starting from $\mathbf{B}$, we build a canonical model $\mathcal{M} = \langle \mathcal{W}_B, <, V\rangle$ satisfying Γ, where: $\mathcal{W}_B$ is the set of labels that appear in the branch $\mathbf{B}$; for each $x, y \in \mathcal{W}_B$, $x < y$ iff there exists Γ_i in $\mathbf{B}$ such that $x < y \in \Gamma_i$; for each $x \in \mathcal{W}_B$, $V(x) = \{P \in ATM \mid$ there is Γ_i in $\mathbf{B}$ such that $x : P \in \Gamma_i\}$. We can easily prove that:

(i) by Theorem 3, we have that $\mathcal{W}_B$ is finite;
(ii) $<$ is an irreflexive, transitive and modular relation on $\mathcal{W}_B$ satisfying the smoothness condition. Irreflexivity, transitivity and modularity are obvious, given Definition 4 and Lemma 2 above. Since $<$ is irreflexive and transitive, it can be easily shown that it is also acyclic. This property together with the finiteness of $\mathcal{W}_B$ entails that $<$ cannot have infinite descending chains. In turn this last property together with the transitivity of $<$ entails the smoothness condition.
(iii) We show that, for all formulas F and for all Γ_i in $\mathbf{B}$, (i) if $x : F \in \Gamma_i$ then $\mathcal{M}, x \models F$ and (ii) if $x : \neg F \in \Gamma_i$ then $\mathcal{M}, x \not\models F$. The proof is by induction on the complexity of the formulas. If $F \in ATM$ this immediately follows from definition of V. For the inductive step, due to space limitations, we only present the case of $F = A \mathrel{|\!\sim} B$. The other cases are similar and then left to the reader. Let $x : A \mathrel{|\!\sim} B \in \Gamma_i$. By Definition 4, we have that, for all y, there is Γ_j in $\mathbf{B}$ such that either $y : \neg A \in \Gamma_j$ or $y : B \in \Gamma_j$ or $y : \neg\Box\neg A \in \Gamma_j$. We show that for all $y \in Min_<(A)$, $\mathcal{M}, y \models B$. Let $y \in Min_<(A)$. This entails that $\mathcal{M}, y \models A$, hence $y : \neg A \notin \Gamma_j$. Similarly, we can show that $y : \neg\Box\neg A \notin \Gamma_j$. It follows that $y : B \in \Gamma_j$, and by inductive hypothesis $\mathcal{M}, y \models B$. (ii) If $x : \neg(A \mathrel{|\!\sim} B) \in \Gamma_i$, since $\mathbf{B}$ is saturated, there is a label y in some Γ_j such that $y : A \in \Gamma_j$, $y : \Box\neg A \in \Gamma_j$, and $y : \neg B \in \Gamma_j$. By inductive hypothesis we can easily show that $\mathcal{M}, y \models A$, $\mathcal{M}, y \models \Box\neg A$, hence $y \in Min_<(A)$, and $\mathcal{M}, y \not\models B$, hence $\mathcal{M}, x \not\models A \mathrel{|\!\sim} B$. ■

4 Termination of $\mathcal{T}\mathbf{R}$ and Optimal Proof Search

In this section, we refine $\mathcal{T}\mathbf{R}$ in order to ensure termination. Moreover, we describe an optimal decision procedure for $\mathbf{R}$ that allows to decide the satisfiability in $\mathbf{R}$ in nondeterministic polynomial time.

In general, non-termination in tableau calculi can be caused by two different reasons: 1. some rules copy their principal formula in the conclusion, so that they can be reapplied over the same formula without any control; 2. dynamic rules can generate infinitely-many worlds, creating infinite branches.

As far as $\mathcal{T}\mathbf{R}$ is concerned, Theorem 3 excludes the second source of non termination (point 2). Concerning point 1, the above calculus $\mathcal{T}\mathbf{R}$ does not ensure a terminating

proof search due to $(\mid\!\sim^+)$, which can be applied without any control. We ensure the termination by putting some constraints on $(\mid\!\sim^+)$ in $\mathcal{T}\mathbf{R}$. It is easy to observe that it is useless to apply the rule on the same conditional formula more than once by using the same label x. Indeed, all formulas in the premise of $(\mid\!\sim^+)$ are kept in the conclusions, then we can assume, without loss of generality, that two applications of $(\mid\!\sim^+)$ on x are consecutive. We observe that the second application is useless, since each of the conclusions has already been obtained after the first application, and can be removed. We prevent redundant applications of $(\mid\!\sim^+)$ by keeping track of labels (worlds) in which a conditional $u : A \mid\!\sim B$ has already been applied in the current branch. To this purpose, we add to each positive conditional a list of *used* labels; we restrict the application of $(\mid\!\sim^+)$ only to labels not occurring in the corresponding list. Notice that also the rule $(<)$ copies its principal formula $x < y$ in the conclusion; however, this rule will be applied only a finite number of times. This is a consequence of the side condition of the rule application and the fact that the number of labels in a tableau is finite (Theorem 3).

The terminating calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ is obtained by replacing the $(\mid\!\sim^+)$ rule in Figure 1 with the one presented in Figure 3.

$$\frac{\Gamma, u : A \mid\!\sim B^{L}}{\Gamma, x : \neg A, u : A \mid\!\sim B^{L,x} \qquad \Gamma, x : \neg\Box\neg A, u : A \mid\!\sim B^{L,x} \qquad \Gamma, x : B, u : A \mid\!\sim B^{L,x}}\ (\mid\!\sim^+)$$

with $x \notin L$

Fig. 3. The rule $(\mid\!\sim^+)$ in the tableau system $\mathcal{T}\mathbf{R}^{\mathbf{T}}$

Theorem 5 (Soundness and completeness of $\mathcal{T}\mathbf{R}^{\mathbf{T}}$). *The calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ is sound and complete w.r.t. the semantics.*

Theorem 6 (Termination of $\mathcal{T}\mathbf{R}^{\mathbf{T}}$). *Let Γ be a finite set of formulas, then any tableau generated by $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ is finite.*

Let n be the size of the starting set Γ of which we want to verify the satisfiability. The number of applications of the rules is proportional to the number of labels introduced in the tableau. In turn, this is $O(2^n)$ due to the interplay between the rules $(\mid\!\sim^+)$ and $(\Box^-)$. Hence, the complexity of the calculus $\mathcal{T}\mathbf{R}^{\mathbf{T}}$ is exponential in n.
In order to obtain a better complexity bound for validity in **R** we provide the following procedure. Intuitively, we do not apply $(\Box^-)$ to all negated boxed formulas, but only to formulas $y : \neg\Box\neg A$ not already expanded, i.e. such that $z : A, z : \Box\neg A$ do not belong to the current branch. As a result, we build a *small* model for the initial set of formulas in accordance with Theorem 1. This is made possible by the modularity of $<$ in **R**.

Let us define a nondeterministic procedure `CHECK`(Γ) to decide whether a given set of formulas Γ is satisfiable. Let `EXPAND`(Γ) be a procedure that returns one saturated expansion of Γ w.r.t. all static rules. In case of a branching rule, `EXPAND` nondeterministically selects (guesses) and applies one conclusion of the rule.

CHECK(Γ)
1. $\Gamma \longleftarrow$ EXPAND(Γ);
2. **if** Γ contains an axiom **then return** UNSAT;
3. $\Gamma \longleftarrow$ result of applying $(\mid\!\sim^-)$ to each negated conditional in Γ;
4. $\Gamma \longleftarrow$ EXPAND(Γ);
5. **if** Γ contains an axiom **then return** UNSAT;
while Γ contains a $y : \neg\Box\neg A$ not marked as CONSIDERED **do**
   6. select $y : \neg\Box\neg A \in \Gamma$ not already marked as CONSIDERED;
      6a. **if** there is z in Γ such that $z : A \in \Gamma$ and $z : \Box\neg A \in \Gamma$
         then 6a'. add $z < y, \Gamma^M_{y\to z}$ to Γ;
         else 6a". $\Gamma \longleftarrow$ result of applying $(\Box^-)$ to $y : \neg\Box\neg A$;
      6b. mark $y : \neg\Box\neg A$ as CONSIDERED;
   7. $\Gamma \longleftarrow$ EXPAND(Γ);
   8. **if** Γ contains an axiom **then return** UNSAT;
endWhile
9. **return** SAT;

Observe that the addition of the set of formulas $z < y, \Gamma^M_{y\to z}$ in step 6a' could be omitted and it has been added mostly to enhance the understanding of the procedure. Indeed, the rule $(<)$, which is applied at each iteration to assure modularity, already takes care of adding such formulas. The procedure CHECK nondeterministically builds an open branch for Γ.

Theorem 7 (Soundness and completeness of the procedure). *The above procedure is sound and complete w.r.t. the semantics.*

Proof sketch. (*Soundness*). We prove that if the initial set of formulas Γ is satisfiable, then the above procedure returns SAT. More precisely, we prove that each step of the procedure preserves the satisfiability of Γ. As far as EXPAND is concerned, notice that it only applies the static rules of $\mathcal{TR}^{\mathbf{T}}$ and the soundness follows from the fact that these rules preserve satisfiability (see Theorems 2 and 5). Consider now step 6. Let $y : \neg\Box\neg A$ the formula selected in this step. If $(\Box^-)$ is applied to $y : \neg\Box\neg A$ (step 6a") we are done, since $(\Box^-)$ preserves satisfiability (see Theorems 2 and 5). If Γ already contains $z : A, z : \Box\neg A$, then step 6a' is executed, and the relation $z < y$ is added. In this case we reason as follows. Since Γ is satisfiable, we have that there is a model $\mathcal{M}$ and a mapping I such that (1) $\mathcal{M}, I(y) \models \neg\Box\neg A$ and (2) $\mathcal{M}, I(z) \models A$ and $\mathcal{M}, I(z) \models \Box\neg A$. We can observe that $I(z) < I(y)$ in $\mathcal{M}$. Indeed, by the truth condition of $\neg\Box\neg A$ (see Definitions 2 and 3) and by the strong smoothness condition, we have that there exists w such that $w < I(y)$ and $\mathcal{M}, w \models A, \Box\neg A$. By modularity of $<$, either 1. $w < I(z)$ or 2. $I(z) < I(y)$. 1 is impossible, since otherwise we would have $\mathcal{M}, w \models \neg A$, which contradicts $\mathcal{M}, w \models A$. Hence, 2 holds. Therefore, we can conclude that step 6a' preserves satisfiability.
(*Completeness*). It can be easily shown that in case the procedure above returns SAT, then the branch built is saturated (see Definition 4). Therefore, we can build a canonical model for the initial Γ, as done in the proof of Theorem 4. ■

Theorem 8 (Complexity of the CHECK procedure). *By means of the procedure CHECK the satisfiability of a set of formulas of logic* **R** *can be decided in nondeterministic polynomial time.*

Proof. Observe that the procedure generates at most *O(n)* labels by applying the rule $(\mathrel{|\!\sim}^{-})$ (step 3) and that the while loop generates at most one new label for each $\neg\Box\neg A$ formula. Indeed, the rule $(\Box^{-})$ is applied to a labelled formula $y : \neg\Box\neg A$ to generate a new world only if there is not a label z such that $z : A \in \Gamma$ and $z : \Box\neg A \in \Gamma$ are already on the branch. In essence, the procedure does not add a new minimal A-world on the branch if there is already one. As the number of different $\neg\Box\neg A$ formulas is at most $O(n)$, then the while loop can add at most $O(n)$ new labels on the branch. Moreover, for each different label x, the expansion step can add at most $O(n)$ formulas $x : \neg\Box\neg A$ on the branch, one for each positive conditional $A \mathrel{|\!\sim} B$ occurring in the set Γ. We can therefore conclude that the while loop can be executed at most $O(n^2)$ times.

As the number of generated labels is at most $O(n)$, by the subformula property, the number of labelled formulas on the branch is at most $O(n^2)$. Hence, the execution of step 6a has complexity $O(n^2)$. The execution of the nondeterministic procedure `EXPAND` has complexity $O(n^2)$, including a guess of size $O(n^2)$, whereas to verify if Γ contains an axiom has complexity $O(n^4)$ (since it requires to check whether, for each labelled formula $x : P \in \Gamma$, the formula $x : \neg P$ is also in Γ, and Γ contains at most $O(n^2)$ labelled formulas). We can therefore conclude that the execution of the `CHECK` procedure requires at most $O(n^6)$ steps. ■

By Theorem 8, the validity problem for **R** is in **coNP**. **coNP**-hardness is immediate, since **R** includes classical propositional logic. Thus, we can conclude that:

Theorem 9 (Complexity of R). *The problem of deciding the validity for rational logic* **R** *is* **coNP***-complete.*

5 Conclusions and Future Work

In this paper we have developed an analytic tableau calculus $\mathcal{T}\mathbf{R}$ for the rational logic **R**. To the best of our knowledge, this is the first calculus for **R** directly based on preferential semantics. We have proved the termination of the calculus and provided a systematic procedure for deciding the satisfiability of a set of formulas in nondeterministic polynomial time. The paper is complementary to the work [15], where the other KLM systems are considered.

We briefly remark on some related work for deductive approaches to KLM logics.

Proof methods for the other KLM logics and for conditional logics related to them have been presented in [19, 20, 21]. Decidability of **P** and **R** has also been obtained by interpreting them into standard modal logics, as it is done by Boutilier [17]. However, Boutilier rejects the smoothness condition, which is essential in KLM framework. Furthermore, Boutilier gives a less natural and more complicated mapping into modal logic S4 and S4.3 for **P** and **R** respectively. Our logic and S4.3 are incomparable: finite-chain condition corresponding to the axiom G does not hold in S4.3, reflexivity and weak connectedness (holding in S4.3) do not hold in our logic. In [15] analytic tableaux calculi for **P** and **CL** are presented. These calculi are based on the same idea of using suitable modalities to interpret conditional assertions. In [15] authors show that the problem of deciding validity is **co-NP** complete for both logics **P** and **CL**.

We plan to extend our calculi to the first order case. The starting point will be the analysis of first order rational logic by Friedman, Halpern and Koller in [4]. In sub-

sequent research we also intend to investigate how to find models in the alternative semantics of **P** and **R** [3] of a set of conditional assertions by using our tableau methods. This could be a step in order to use our tableau procedures to uniformly implement a variety of default reasoning mechanisms built upon KLM logics **P** and **R** [5, 6, 7, 8].

References

1. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence **44**(1-2) (1990) 167–207
2. Gardenförs, P.: Knowledge in Flux. MIT Press (1988)
3. Friedman, N., Halpern, J.Y.: Plausibility measures and default reasoning. Journal of the ACM **48**(4) (2001) 648–685
4. Friedman, N., Halpern, J.Y., Koller, D.: First-order conditional logic for default reasoning revisited. ACM TOCL **1**(2) (2000) 175–207
5. Benferhat, S., Dubois, D., Prade, H.: Nonmonotonic reasoning, conditional objects and possibility theory. Artificial Intelligence **92**(1-2) (1997) 259–276
6. Benferhat, S., Saffiotti, A., Smets, P.: Belief functions and default reasoning. Artificial Intelligence **122**(1-2) (2000) 1–69
7. Weydert, E.: System jlz - rational default reasoning by minimal ranking constructions. Journal of Applied Logic **1**(3-4) (2003) 273–308
8. Pearl, J.: System z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In: Proc. of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge, Morgan Kaufmann Publishers Inc. (1990) 121–135
9. Makinson, D.: Bridges from Classical to Nonmonotonic logic. London: King's College Publications. Series: Texts in Computing, vol 5 (2005)
10. Makinson, D.: Bridges between classical and nonmonotonic logic. Logic Journal of the IGPL **11**(1) (2003) 69–96
11. Arieli, O., Avron, A.: General patterns for nonmonotonic reasoning: From basic entailments to plausible relations. Logic Journal of the IGPL **8**(2) (2000) 119–148
12. Dubois, D., Fargier, H., Perny, P., Prade, H.: Qualitative decision theory: from savages axioms to nonmonotonic reasoning. Journal of the ACM **49**(4) (2002) 455–495
13. Dubois, D., Fargier, H., Perny, P.: Qualitative decision theory with preference relations and comparative uncertainty: An axiomatic approach. Art. Int. **148**(1-2) (2003) 219–260
14. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence **55**(1) (1992) 1–60
15. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux for KLM Preferential and Cumulative Logics. In: Proc. of LPAR 2005, LNAI 3835, Springer (2005) 666–681
16. Crocco, G., Lamarre, P.: On the connection between non-monotonic inference systems and conditional logics. In: Proc. of KR 92. (1992) 565–571
17. Boutilier, C.: Conditional logics of normality: a modal approach. Art. Int. **68**(1) (1994) 87–154
18. Hughes, G., Cresswell, M.: A Companion to Modal Logic. Methuen (1984)
19. Artosi, A., Governatori, G., Rotolo, A.: Labelled tableaux for non-monotonic reasoning: Cumulative consequence relations. J. of Logic and Computation **12**(6) (2002) 1027–1060
20. Giordano, L., Gliozzi, V., Olivetti, N., Schwind, C.: Tableau calculi for preference-based conditional logics. In: Proc. of TABLEAUX 2003, LNAI 2796, Springer (2003) 81–101
21. Giordano, L., Gliozzi, V., Olivetti, N., Schwind, C.: Extensions of tableau calculi for preference-based conditional logics. In: Proc. of M4M-4, Informatik-Bericht 194 (2005) 220–234

On the Semantics of Logic Programs with Preferences

Sergio Greco, Irina Trubitsyna, and Ester Zumpano

DEIS, University of Calabria,
87030 Rende, Italy
{greco, irina, zumpano}@deis.unical.it

Abstract. This work is a contribution to realizing prioritized reasoning in logic programming in the presence of preference relations involving atoms. In more details, the case of dynamic preferences is investigated and a semantics interpreting each preference rule as a tool for representing a choice over alternative options is proposed. The technique, providing a new interpretation for prioritized logic programs, is inspired by the one proposed by Sakama and Inoue in [19] and enriched with the use of structural information of preference rules as proposed by Brewka et al. in [6]. Specifically, the analysis of the logic program is carried out together with the analysis of preferences in order to determine the choice order and the sets of comparable models. The proposed approach is compared with those in [6, 19]. Complexity analysis is also performed showing that the use of additional information does not increase the complexity of computing preferred stable models.

1 Introduction

The increased interest in preferences is reflected by an extensive number of proposals and systems for preference handling [17, 22, 24, 25]. The literature distinguish *static* and *dynamic* preferences. Static preferences are fixed at the time a theory is specified, i.e. they are "external" to the logic program [19, 26], whereas dynamic preferences appear within the logic program and are determined "on the fly" [6, 8, 10, 12, 24]. The most common form of preference consists in specifying preference conditions among rules [2, 3, 4, 5, 10, 11, 12, 16, 20, 21, 23, 27, 28], whereas, some recent proposals admit the expression of preference relations among atoms [6, 7, 19, 24]. More sophisticated forms of preferences also allow specifying priorities between conjunctive (disjunctive) knowledge with preconditions [6, 10, 19] and numerical penalties for suboptimal options [7].

This work is a contribution to realizing prioritized reasoning in logic programming in the presence of preference conditions involving atoms. In more details, the case of dynamic preferences is investigated and a semantics interpreting each preference rule as tool for representing a choice over alternative options is proposed. In particular, priorities are applied by following the natural ordering defined by dependencies, as proposed in [6], and the comparison strategy, proposed in [19], is extended by introducing the concept of *comparable models*. Next example describes the intuition at the basis of the proposed approach.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 203–215, 2006.

Example 1. The following prioritized program $\langle \mathcal{P}_1, \Phi_1 \rangle$, inspired by a program presented in [6], describes different menus and the preferences among drinks and desserts:

$$\begin{array}{ll}
\texttt{fish} \oplus \texttt{beef} \leftarrow & \varrho_1 : \texttt{white} > \texttt{red} \leftarrow \texttt{fish} \\
\texttt{red} \oplus \texttt{white} \leftarrow & \varrho_2 : \texttt{red} > \texttt{white} \leftarrow \texttt{beef} \\
\texttt{pie} \oplus \texttt{ice-cream} \leftarrow & \varrho_3 : \texttt{pie} > \texttt{ice-cream} \leftarrow \texttt{red} \\
\leftarrow \texttt{fish}, \texttt{white} & \\
\leftarrow \texttt{beef}, \texttt{pie} & \\
\leftarrow \texttt{fish}, \texttt{ice-cream} &
\end{array}$$

The symbol $\oplus$ denotes exclusive disjunction, i.e. if the body of the rule is true only one atom in the head is true, whereas a rule with empty head defines a constraint, i.e. a rule which is satisfied only if the body is false. The first three rules of $\mathcal{P}_1$ select the main dish, the drink and the dessert; the last three rules are constraints and state that a feasible solution cannot contain (i) `fish` and `white` or (ii) `beef` and `pie` or (iii) `fish` and `ice-cream`. Prioritized rules in Φ_1 introduce preferences among drinks (ϱ_1, ϱ_2) and desserts (ϱ_3).

The program $\mathcal{P}_1$ has three stable models: $M_1 = \{\texttt{fish}, \texttt{red}, \texttt{pie}\}$, $M_2 = \{\texttt{beef}, \texttt{white}, \texttt{ice-cream}\}$ and $M_3 = \{\texttt{beef}, \texttt{red}, \texttt{ice-cream}\}$. The $\mathcal{PLP}$ (Prioritized Logic Program) technique [19] returns M_1 as unique preferred model; whereas the $\mathcal{ASO}$ (Answer Set Optimization) technique [6], following the natural ordering of preference rules, derives that M_3 is the unique solution. Thus, the two approaches provide different results. The structure of preference rules suggests that (i) the choice of drink precedes the choice of dessert and depends on the selected main dish; (ii) `fish` and `beef` are alternative options for main dish. The latter conclusion is based on the observation that ϱ_1 and ϱ_2 provide opposite valuation to the drink choice. This is possible if their bodies define two different classes of models (menus), which should be considered separately. In other words, the model M_1 (associated to the menu containing `fish`) should not be compared with the models M_2 and M_3 (associated with the menus containing `beef`). Consequently, both M_1 and M_3 should be preferred. □

Contribution. The paper provides a new semantics for prioritized logic programs enriching the one proposed in [19] with additional information gained from the structure of preference rules as proposed in [6]. In particular, the new semantics introduces a natural ordering among preferences that fixes the order of choices, looking at the *stratification* of the preference program. Each decision is determined by the set of choices belonging to the corresponding level and provides the subset of models given in input as solution. Once a decision is made, this output subset becomes the input set of the following decision and so on. The proposed semantics drives the decision process by catching additional information regarding non comparable sets of models; the concept of incomparability has not been taken into account by previous approaches.

The paper also analyzes the complexity of computing preferred answer sets and shows that, w.r.t. others previous proposals such as the one proposed in [19], the use of additional information does not increase the complexity of computing preferred stable models.

2 Preliminaries

We assume familiarity with relational database theory, disjunctive logic programs, disjunctive deductive databases, (disjunctive) stable model semantics and computational complexity [13, 14, 15, 18].

2.1 Background

A *(disjunctive) logic program* is a finite set of rules of the form $A_1 \vee \cdots \vee A_k \leftarrow B_1, ..., B_m, not\, C_1, ..., not\, C_n, k{+}m{+}n{>}0$, where $A_1, ..., A_k, B_1, ..., B_m, C_1, ..., C_n$ are atoms. The disjunction $A_1 \vee \cdots \vee A_k$, denoted by $head(r)$, is called the *head* of r; while the conjunction $B_1, ..., B_m, not\, C_1, ..., not\, C_n$, denoted by $body(r)$, is called *body* of r. The intuitive meaning of previous disjunctive rule is that if $body(r)$ is true, i.e. $B_1, ..., B_m$ are true and $C_1, ..., C_n$ are false, then $head(r)$ is true, i.e at least one of $A_1, ..., A_k$ is true (otherwise r is not satisfied). In this paper exclusive disjunction, denoted by $\oplus$, is used in the head; the statement $head(r) = A_1 \oplus ... \oplus A_k$ is true, if exactly one of $A_1, ..., A_k$ is true. Rules with empty head, called *denials* or *constraints*, will be used to define constraints and are satisfied only if the body is false. The solution of a logic program $\mathcal{P}$ is given in term of stable model (answer set) semantics [14, 15].

An interpretation M for $\mathcal{P}$ is a model of $\mathcal{P}$ if M satisfies all rules in $ground(\mathcal{P})$. The minimal model semantics, defined for positive $\mathcal{P}$, assigns to $\mathcal{P}$ the set of its *minimal models* $\mathcal{MM}(\mathcal{P})$, where a model M for $\mathcal{P}$ is minimal, if no proper subset of M is a model for $\mathcal{P}$. The more general *disjunctive stable model semantics* also applies to programs with (unstratified) negation [15]. Disjunctive stable model semantics generalizes stable model semantics, previously defined for normal programs [14]. For any interpretation M, denote with $\mathcal{P}^M$ the ground positive program derived from $ground(\mathcal{P})$ by (i) removing all rules that contain a negative literal $not\ a$ in the body and $a \in M$, and (ii) removing all negative literals from the remaining rules. An interpretation M is a (disjunctive) stable model of $\mathcal{P}$ if and only if $M \in \mathcal{MM}(\mathcal{P}^M)$. For general $\mathcal{P}$, the stable model semantics assigns to $\mathcal{P}$ the set $\mathcal{SM}(\mathcal{P})$ of its *stable models*. It is well known that stable models are minimal models (i.e. $\mathcal{SM}(\mathcal{P}) \subseteq \mathcal{MM}(\mathcal{P})$) and that for negation free programs, minimal and stable model semantics coincide (i.e. $\mathcal{SM}(\mathcal{P}) = \mathcal{MM}(\mathcal{P})$).

The rest of this section will briefly review the two main approaches for prioritizing reasoning we refer to, i.e. Prioritized Logic Programs and Answer Set Optimization, proposed respectively in [19] and [6].

2.2 Prioritized Logic Programs

A *(partial) preference relation* $\succeq$ among atoms is defined as follows: given two atoms e_1 and e_2, the statement $e_1 \succeq e_2$ (called *priority*) means that e_1 has higher priority than e_2. Moreover, if $e_1 \succeq e_2$ and $e_2 \succeq e_3$, then $e_1 \succeq e_3$. A priority statement $e_1 \succeq e_2$ states that for each a_1 instance of e_1 and for each a_2 instance of e_2 it is $a_1 \succeq a_2$. The statement $e_1 \succ e_2$ stands for $e_1 \succeq e_2$ and $e_2 \not\succeq e_1$. Clearly, if $e_1 \succ e_2$, the sets of ground instantiations of e_1 and e_2 have an empty intersection.

A *prioritized logic program* (PLP) is a pair $\langle \mathcal{P}, \Phi \rangle$ where $\mathcal{P}$ is a disjunctive program and Φ is a set of priorities. Φ^* denotes the set of priorities which can be reflexively or transitively derived from Φ.

Definition 1. Given a prioritized logic program $\langle \mathcal{P}, \Phi \rangle$, the relation $\sqsupseteq$ is defined over the stable models of $\mathcal{P}$ as follows. For any stable models M_1, M_2 and M_3 of $\mathcal{P}$:

1. $M_1 \sqsupseteq M_1$,
2. $M_1 \sqsupseteq M_2$ if $\exists e_1 \in M_1 - M_2$, $\exists e_2 \in M_2 - M_1$ such that $(e_1 \succeq e_2) \in \Phi^*$ and $\nexists e_3 \in M_2 - M_1$ such that $(e_3 \succ e_1) \in \Phi^*$,
3. if $M_1 \sqsupseteq M_2$ and $M_2 \sqsupseteq M_3$, then $M_1 \sqsupseteq M_3$.

If $M_1 \sqsupseteq M_2$ then M_1 is *preferable* to M_2. Moreover, if $M_1 \sqsupseteq M_2$ and $M_1 \not\sqsupseteq M_2$ then $M_1 \sqsupset M_2$. □

An interpretation M is a *preferred* stable model of $\langle \mathcal{P}, \Phi \rangle$ if M is a stable model of $\mathcal{P}$ and $N \sqsupseteq M$ implies $M \sqsupseteq N$ for any stable model N. The set of preferred stable models of $\langle \mathcal{P}, \Phi \rangle$ will be denoted by $\mathcal{PSM}(\langle \mathcal{P}, \Phi \rangle)$. Note that the relation $\Phi_1 \subseteq \Phi_2$ between two PLPs $\langle \mathcal{P}, \Phi_1 \rangle$ and $\langle \mathcal{P}, \Phi_2 \rangle$ does not imply $\mathcal{PSM}(\langle \mathcal{P}, \Phi_1 \rangle) \subseteq \mathcal{PSM}(\langle \mathcal{P}, \Phi_2 \rangle)$.

In a prioritized logic program $\langle \mathcal{P}, \Phi \rangle$ the basic priority relations are defined over atoms by means of static preference rules. The priorities over more general forms of knowledge (conjunctive, disjunctive knowledge, rules, preconditions) can be then express by a simple rewriting of the preference program. For instance, a dynamic preference rule of the form $(e_1 \succeq e_2) \leftarrow B$ is equivalent to $e_1' \succeq e_2'$, where $e_1' \leftarrow e_1, B$ and $e_2' \leftarrow e_2, B$.

The semantics of prioritized programs proposed by Sakama and Inoue will be denoted by $\mathcal{PLP}$ semantics. More details can be found in [19] (see also [9, 28] for related material).

The complexity of answering queries over PLP programs is one level above the complexity of answering queries over standard programs (without preferences). In particular, let $\langle \mathcal{P}, \Phi \rangle$ be a prioritized logic program, then (i) deciding the existence of a preferred stable model is $\Sigma_P^2 - complete$; (ii) deciding whether a literal is true in some (resp. all) preferable stable model of $\langle \mathcal{P}, \Phi \rangle$ is Σ_P^3-complete (resp. Π_P^3-complete) [19].

Sakama et al. in [24] propose a sound and complete procedure that allows preferred answer sets for a prioritized logic program to be computed using a *generate and test algorithm*. This algorithm translates a PLP program $\langle \mathcal{P}, \Phi \rangle$ and any answer set S of the program $\mathcal{P}$ into a single logic program $T[\mathcal{P}, \Phi, S]$, such that its answer sets are answer sets of $\mathcal{P}$ preferable to S. Dynamic preferences are expressed by a stratified logic program whose rules have the standard form: $head(r) \leftarrow body(r)$, where $head(r)$ can be either a standard atom or a prioritized fact, and $body(r)$ is a conjunction of ground literals.

2.3 Answer Set Optimization

An answer set optimization program, denoted as ASO program, is a pair $\langle \mathcal{P}, \Phi \rangle$, where $\mathcal{P}$ is called *Generation Program* and Φ is called *Preference Program* and consists of a finite set of rules of the form: $C_1 > \cdots > C_k \leftarrow a_1, ..., a_n, not\ b_1, ..., not\ b_m$ where a_is and b_js are literals and C_is are boolean combinations[1] of literals; here a literal is either an atom A or its negation $\neg A$. Φ determines a preference ordering on the answer sets described by the generation program $\mathcal{P}$.

[1] A boolean combination is a formula built of atoms by means of disjunctions, conjunctions, strong and default negation with the restriction that strong negation is allowed to appear only in front of atoms and default negation only in front of literals.

Definition 2. Let $\Phi = \{r_1, ..., r_n\}$ be a preference program and S be an answer set, then S induces a satisfaction vector $V_s = (v_s(r_1), ..., v_s(r_n))$ where:

- $v_s(r_j) = I$, if r_j is *Irrelevant* to S, i.e. (i) the body of r_j is not satisfied in S or (ii) the body of r_j is satisfied, but none of the C_is is satisfied in S.
- $v_s(r_j) = min\{i : S \models C_i\}$, otherwise. □

In the comparison of models it is assumed that I is equal to 1 (i.e., $v_{S_j}(r_i) = I$ is equivalent to $v_{S_j}(r_i) = 1$).

Definition 3. Let S_1 and S_2 be two answer sets, then (i) $V_{S_1} \leq V_{S_2}$ if $v_{S_1}(r_i) \leq v_{S_2}(r_i)$ for every $i \in [1..n]$; (ii) $V_{S_1} < V_{S_2}$ if $V_{S_1} \leq V_{S_2}$ and for some $i \in [1..n]$ $v_{S_1}(r_i) < v_{S_2}(r_i)$. In these cases $S_1 \geq S_2$ and $S_1 > S_2$, respectively.

A set of literals S is an *optimal model* of an ASO program $\langle \mathcal{P}, \Phi \rangle$ if S is an answer set of $\mathcal{P}$ and there is no answer set S' of $\mathcal{P}$ such then $S' > S$. □

The complexity of ASO programs depends on the class of generating programs. For disjunctive programs we have the same complexity of prioritized programs, while for disjunction-free programs the complexity is one level lower.

The strategy is further extended by introducing *meta-preferences* among preference rules: a ranked ASO program is a sequence $\langle \mathcal{P}, \Phi_1, ..., \Phi_n \rangle$ consisting of a generation program $\mathcal{P}$ and a sequence of pairwise disjoint preference programs Φ_i. The rank of a rule $r \in \Phi_1 \cup \cdots \cup \Phi_n$, denoted $rank(r)$, is the unique integer i for which $r \in \Phi_i$. $S_1 \geq_{rank} S_2$ if for every preference rule r' such that $v_{S_1}(r') \leq v_{S_2}(r')$ does not hold, there is a rule r'' such that $rank(r'') < rank(r')$ and $v_{S_1}(r'') < v_{S_2}(r'')$.

Moreover, a procedure deriving the *natural ordering* of the preference rules is introduced. Firstly, given a preference program Φ, its dependency graph $G(\Phi)$ is defined. The atoms appearing in Φ form the vertex set of $G(\Phi)$. There is a directed edge from a vertex b to a vertex a in $G(\Phi)$ if there is a rule r in Φ such that a appears in the head of r and b appears in the body of r. If the graph $G(\Phi)$ is acyclic, there is a natural ranking of its atoms which can be defined recursively as follows: $rank(a) = 0$ for every atom a that has no predecessors in $G(\Phi)$; otherwise $rank(a)$ is the maximum of the ranks of all predecessors of a in $G(\Phi)$ incremented by 1. The rank of a preference rule r is then defined as the maximum rank of atoms in its head.

The standard semantics of ASO programs, where priorities are examined all together, will be denoted as $\mathcal{ASO}$ semantics. The alternative semantics, where priorities are divided into layers following the natural order, will be denoted by $\mathcal{RASO}$ (ranked $\mathcal{ASO}$) semantics.

3 Well Formed Prioritized Logic Programs

In this paper a syntax similar to the one proposed in [6] is used. Given two atoms A_1 and A_2, the statement $A_2 > A_1$ means that A_2 has higher priority than A_1. A *(partial) preference relation* $>$ among atoms is defined as follows.

Definition 4. A *prioritized program* is a pair $\langle \mathcal{P}, \Phi \rangle$ where $\mathcal{P}$ is a disjunctive program and Φ is set of *preference rules* of the form:

$$A_1 > A_2 > \cdots > A_k \leftarrow B_1, ..., B_m, not\, C_1, ..., not\, C_n \quad (1)$$

where $k > 1$ and $A_1, ..., A_k, B_1, ..., B_m, C_1, ..., C_n$ are atoms.

A *ground* prioritized program, denoted by $ground(\langle \mathcal{P}, \Phi \rangle) = \langle ground(\mathcal{P}), ground(\Phi) \rangle$ is a prioritized program, where each rule $r \in (\mathcal{P} \cup \Phi)$ with variables is replaced with the set of its ground instances, i.e the set of rules obtained by replacing variables with constants. □

Intuitively, a preference rule ϱ of the form (1) describes the *choice* between $A_1, ..., A_k$ (*choice options*) under the condition specified by the body of ϱ. The head of ϱ introduces the preference order between choice options: A_i is preferred to A_j, $i < j$ and $i, j \in [1..k]$. As ϱ can be applied only if $body(\varrho)$ is true, the body of ϱ specifies the decisions which have to precede this choice. For instance, $a > c \leftarrow b$ states that if b is $true$, then a is preferred to c. In the following the short cut of the form $a \vee d > c \leftarrow body$ will be used, stating for the two preference rules $a > c \leftarrow body$ and $d > c \leftarrow body$, whose meaning is that a and d are preferred to c if $body$ is $true$.

A preference rule with exactly two atoms in the head will be called *binary* preference rule, whereas preference rules with empty bodies will be called *preference facts*. A prioritized program is said to be in *binary form* if all its preference rules are binary. Preference rules can be rewritten into binary preferences. Thus, a preference rule of the form $A_1 > A_2 > \cdots > A_k \leftarrow body$ is equivalent to $k - 1$ binary rules of the form $A_i > A_{i+1} \leftarrow body$.

The following example, presenting a classical program borrowed from [6], will be used as running example.

Example 2. Consider the prioritized program $\langle \mathcal{P}_2, \Phi_2 \rangle$ whose stable models define menus of a restaurant:

$\mathtt{fish} \oplus \mathtt{beef} \leftarrow$
$\mathtt{red} \oplus \mathtt{white} \oplus \mathtt{beer} \leftarrow$
$\mathtt{pie} \oplus \mathtt{ice\text{-}cream} \leftarrow$
$\leftarrow \mathtt{beef}, \mathtt{pie}$
$\leftarrow \mathtt{fish}, \mathtt{ice\text{-}cream}$

$\varrho_1 : \mathtt{white} > \mathtt{red} > \mathtt{beer} \leftarrow \mathtt{fish}$
$\varrho_2 : \mathtt{red} \vee \mathtt{beer} > \mathtt{white} \leftarrow \mathtt{beef}$
$\varrho_3 : \mathtt{pie} > \mathtt{ice\text{-}cream} \leftarrow \mathtt{beer}$

The first three rules of $\mathcal{P}_2$ select the main dish, the drink and the dessert; the rules in Φ_2 introduce preferences among drinks and desserts. The program $\mathcal{P}_2$ has six stable models:

$\mathtt{M}_1 = \{\mathtt{fish}, \mathtt{white}, \mathtt{pie}\}$
$\mathtt{M}_2 = \{\mathtt{fish}, \mathtt{red}, \mathtt{pie}\}$
$\mathtt{M}_3 = \{\mathtt{fish}, \mathtt{beer}, \mathtt{pie}\}$
$\mathtt{M}_4 = \{\mathtt{beef}, \mathtt{white}, \mathtt{ice\text{-}cream}\}$
$\mathtt{M}_5 = \{\mathtt{beef}, \mathtt{red}, \mathtt{ice\text{-}cream}\}$
$\mathtt{M}_6 = \{\mathtt{beef}, \mathtt{beer}, \mathtt{ice\text{-}cream}\}$

Both techniques proposed in [19] and [6] select the stable models M_1 and M_5 as preferred ones. □

Before presenting the formal semantics of programs, some preliminary definitions are needed. Given a prioritized program $\langle \mathcal{P}, \Phi \rangle$, the (ground) transitive closure of Φ is $\Phi^* = \Phi' \cup \{a > c \leftarrow body_1, body_2 \mid a > b \leftarrow body_1 \in \Phi^* \wedge b > c \leftarrow body_2 \in \Phi^* \wedge a \neq c\}$, where Φ' is the binary form of $ground(\Phi)$.

Φ^* is defined as the set of rules, explicitly representing the preference relations between choice options. In order to ensure that these relations regard alternative choice options, the following property is introduced:

Definition 5. *Well-formed programs.* A prioritized program $\langle \mathcal{P}, \Phi \rangle$ is said to be well-formed if there is no model $M \in \mathcal{SM}(\mathcal{P})$ and preference rule $a > b \leftarrow body$ in Φ^*, such that $body$ is $true$ in M and $a, b \in M$. □

Observe that in order to guarantee that programs are well formed it is sufficient to add rules which guarantee that for each preference rule $a > b \leftarrow body$ in Φ^* there is a constraint $\leftarrow a, b, body$ in $ground(\mathcal{P})$. All programs considered so far are well-formed (the constraints which guarantee that programs are well formed are not necessary as exclusive disjunction has been used).

In the following we consider only well-formed programs.

Definition 6. *Contradictory preferences.* Two ground (binary) preferences of the form $a > b \leftarrow body_1$ and $b > a \leftarrow body_2$ are said to be *contradictory*. A set of preferences Φ is said to be contradictory if Φ^* contains two contradictory preference rules. □

For instance, the preferences ϱ_1 : `white` $>$ `red` $\leftarrow$ `fish` and ϱ_2 : `red` $>$ `white` $\leftarrow$ `beef` of Example 1 are contradictory, whereas the preferences ϱ_1 and ϱ_2' : `red` $\vee$ `white` $>$ `water` $\leftarrow$ `beef` are not.

Definition 7. *Relevant models.* Given a prioritized program $\langle \mathcal{P}, \Phi \rangle$ and a preference $\varrho \in \Phi$, the set of stable models *relevant* for ϱ is $\mathcal{SM}(\mathcal{P}, \varrho) = \{M \mid M \in \mathcal{SM}(\mathcal{P}) \wedge M \models body(\varrho)\}$. □

Definition 8. *Conflicting preferences.* Let $\langle \mathcal{P}, \Phi \rangle$ be a prioritized program, a pair of contradictory preferences ϱ_1 and ϱ_2 is *conflicting* if $\mathcal{SM}(P, \varrho_1) \cap \mathcal{SM}(P, \varrho_2) \neq \emptyset$. □

Example 3. Considering the preference rules of Example 1, we have that $\mathcal{SM}(\mathcal{P}_1, \varrho_1) = \{M_1\}$ and $\mathcal{SM}(\mathcal{P}_1, \varrho_2) = \{M_2, M_3\}$; the contradictory preferences ϱ_1 and ϱ_2 are not conflicting as $\mathcal{SM}(\mathcal{P}_1, \varrho_1) \cap \mathcal{SM}(\mathcal{P}_1, \varrho_2) = \emptyset$. For the preferences of Example 2 we have that $\mathcal{SM}(\mathcal{P}_2, \varrho_1) = \{M_1, M_2, M_3\}$ and $\mathcal{SM}(\mathcal{P}_2, \varrho_2) = \{M_4, M_5, M_6\}$; also in this case the two contradictory preference rules are not conflicting. □

Thus, two contradictory preferences ϱ_1 and ϱ_2 are conflicting if there is a stable model satisfying the bodies of both ϱ_1 and ϱ_2.

Definition 9. *Stratification.* A (ground) preference program Φ is stratified if it is possible to determine the stratification into $\langle \Phi[0], \Phi[1], ..., \Phi[n] \rangle$ such that:

- Every atom A is associated with the least possible level i (denoted $A[i]$) in such a way that for each preference rule $\varrho \in \Phi$ the level of head atoms is greater than the level of each body atom; the level of body atoms that do not appear in any head is assumed to be equal to 0;
- Every preference rule $\varrho \in \Phi$ is associated with a level i (denoted by $\varrho[i]$) consisting of the maximum level of the atoms in $body(\varrho)$;
- $\Phi[i]$ consists of all preference rules associated with the level i. □

The above definition of stratification of preference rules defines the order in which preferences are applied. Observe, that the preference program may have only one stratification, or may be not stratified. In the latter case we consider all preference rules together by introducing the default stratification $\langle \Phi[0] \rangle$, where $\Phi[0] = \Phi^*$.

It should be noticed that the assignment of the level to each atom can be performed following the first part of the declarative procedure establishing the natural ordering of preference rules, defined in [6]; whereas the assignment of the level to each rule, performed on the second step, differs from the one proposed in [6] as it considers body's instead of head's atoms. A more detailed comparison of the two approaches will be presented in the next section.

Example 4. Consider the prioritized program $\langle \mathcal{P}_2, \Phi_2 \rangle$ of Example 2, where $\Phi_2 = \{\varrho_1, \varrho_2, \varrho_3\}$. The stratification of Φ_2 consists of: $\Phi_2[0] = \{\varrho_1, \varrho_2\}$ and $\Phi_2[1] = \{\varrho_3\}$. □

The intuition at the basis of our approach is clarified in this example. Suppose there are two contradictory preferences $\varrho_1 : a > b \leftarrow c$ and $\varrho_2 : b > a \leftarrow d$. Intuitively, the two contradictory preferences ϱ_1 and ϱ_2 are meaningful if they are applied to different sets of models, i.e. models defined by alternative decisions associated respectively, with c and d (or with atoms on which c and d depend). Thus when defining two contradictory preferences ϱ_1 and ϱ_2 the user assumes that their bodies define alternative decisions. Moreover, once the alternative decisions have been made, the associated solutions are no longer comparable. In order to capture the previously mentioned intuition, use is made of the following concept.

Definition 10. *Comparable models.* Let $\langle \mathcal{P}, \Phi \rangle$ be a prioritized program, M_1 and M_2 two stable models for $\mathcal{P}$ and $\Phi^*[0], ..., \Phi^*[n]$ be a stratification of Φ^*, then

1. M_1 and M_2 are comparable on $\Phi^*[0]$.
2. M_1 and M_2 are comparable on $\Phi^*[i+1]$, if
   (a) they are comparable on $\Phi^*[i]$, and
   (b) there do not exist two contradictory preferences $\varrho_1, \varrho_2 \in \Phi^*[i]$ such that M_1 is relevant for ϱ_1 and M_2 is relevant for ϱ_2, i.e. $M_1 \models body(\varrho_1)$ and $M_2 \models body(\varrho_2)$. □

Observe that, the second condition in the previous definition of comparable models states that given two models M_1 and M_2 associated with two alternative decisions, if an alternative decision has been performed in the previous level (i.e. if two contradictory preferences exist in the previous level), then no further comparison can be made, i.e. M_1 and M_2 are not comparable in the current and next levels. In other words, if M_1 and M_2 are relevant for two contradictory preferences in the previous level, they have to be considered separately, i.e. they are not comparable.

Example 5. Let's consider $M_3 = \{\texttt{fish}, \texttt{beer}, \texttt{pie}\}$ and $M_6 = \{\texttt{beef}, \texttt{beer}, \texttt{ice-cream}\}$ with respect to the preferences Φ_2 of Example 2. M_3 and M_6 are comparable on $\Phi_2^*[0]$ by definition, while they are not comparable on $\Phi_2^*[1]$, because M_3 is relevant for ϱ_1, M_6 is relevant for ϱ_2, and these contradictory preferences belong to $\Phi_2^*[0]$. □

Fact 1. *Let $\langle P, \Phi \rangle$ be a prioritized program without contradictory preferences and $\langle \Phi^*[0], \Phi^*[1], ..., \Phi^*[n] \rangle$ the stratification of Φ^*. Then, each pair of models M_1, M_2 is comparable on $\Phi^*[i]$, $i \in [0..n]$.* □

The proof of the fact above follows directly from Definition 10.

On the basis of Definition 10 the declarative semantics of prioritized logic programs can be now provided. This new semantics, denoted with $\mathcal{PAS}$ (Preferred Answer Sets), is given by preferred answer sets as follows:

Definition 11. *Preference between Answer Sets.* Given a prioritized program $\langle \mathcal{P}, \Phi \rangle$, the relation $\sqsupseteq$ is defined over the stable models of $\mathcal{P}$ as follows. For any stable models M_1, M_2 and M_3 of $\mathcal{P}$, let $\Phi^*[0], \Phi^*[1], ..., \Phi^*[n]$ be a stratification of Φ^*, then

- $M_1 \sqsupseteq M_1$,
- $M_1 \sqsupseteq M_2$ if $\exists i$, such that M_1 and M_2 are comparable on $\Phi^*[i]$ and
  - $\exists e_1 \in M_1 - M_2$, $\exists e_2 \in M_2 - M_1$ such that $\varrho : (e_1 > e_2) \leftarrow body_1 \in \Phi^*[i]$, and both models M_1, M_2 are relevant for ϱ.
  - $\nexists e_3 \in M_2 - M_1$ such that $\varrho : (e_3 > e_1) \leftarrow body_3 \in \Phi^*[j]$, $j \leq i$, and both models M_1, M_2 are relevant for ϱ.
- if $M_1 \sqsupseteq M_2$ and $M_2 \sqsupseteq M_3$, then $M_1 \sqsupseteq M_3$. □

If $M_1 \sqsupseteq M_2$ we say that M_1 is *preferable* to M_2. Moreover, we write $M_1 \sqsupset M_2$ if $M_1 \sqsupseteq M_2$ and $M_2 \not\sqsupseteq M_1$.

Definition 12. *Preferred Answer Sets.* An interpretation M is a *preferred* stable model for a prioritized program $\langle \mathcal{P}, \Phi \rangle$ if M is a stable model of $\mathcal{P}$ and $N \sqsupseteq M$ implies $M \sqsupseteq N$ for any stable model N. The set of preferred stable models for $\langle \mathcal{P}, \Phi \rangle$ will be denoted by $\mathcal{PAS}(\langle \mathcal{P}, \Phi \rangle)$. □

Note that Definition 11 extends the $\mathcal{PLP}$ semantics. In particular, $\mathcal{PAS}$ semantics defines priorities between pairs of models, and can be seen as a $\mathcal{PLP}$ semantics enriched with additional information gained from the structure of preference rules.

Example 6. Consider the prioritized program $\langle P_2, \Phi_2 \rangle$ of Example 2. We have that

- all models are comparable on $\Phi^*[0]$ by definition and
  - due to ϱ_1, $M_1 \sqsupseteq M_2 \sqsupseteq M_3$;
  - due to ϱ_2, $M_5 \sqsupseteq M_4$ and $M_6 \sqsupseteq M_4$;
- as ρ_1 and ρ_2 are contradictory, models satisfying the body of ρ_1 (M_1, M_2 and M_3) cannot be compared in $\Phi^*[1]$ with models satisfying the body of ρ_2 (M_4, M_5, M_6).

Therefore, M_3 and M_6 are not comparable on $\Phi_2^*[1]$, as discussed in Example 5. The preferred models are: M_1, M_5 and M_6. M_6 is considered as good as M_5 because both them present `beef` as main dish, the best choice of drink and the same (unique possible) dessert.

Observe that both $\mathcal{ASO}$ [6] and $\mathcal{PLP}$ [19] semantics discard M_6. The $\mathcal{ASO}$ semantics deduces that M_1 and M_5 are preferable to M_6 owing to ϱ_3, while the $\mathcal{PLP}$ semantics states that M_1 is preferable to M_3 and M_3 is preferable to M_6, owing to ϱ_1, ϱ_3.

Consider the program $\langle \mathcal{P}_2, \widehat{\Phi}_2 \rangle$ where $\widehat{\Phi}_2$ is derived from Φ_2 by replacing ϱ_3 with ϱ_3' : `pie` $>$ `ice-cream` $\leftarrow$. The new preference program has the unique level $\widehat{\Phi}_2[0] = \{\varrho_1, \varrho_2, \varrho_3'\}$; thus, due to ϱ_3' the following relations hold $M_3 \sqsupseteq M_5$ and $M_3 \sqsupseteq M_6$. Therefore M_1 is the unique preferred model. The same result is obtained by both $\mathcal{PLP}$ and $\mathcal{ASO}$ semantics. □

Complexity Result

Theorem 2. *Let $\langle \mathcal{P}, \Phi \rangle$ be a prioritized program. Then*

1. *Deciding the existence of a preferred stable model is Σ_P^2-complete.*
2. *Deciding whether a literal is true in some (all) preferred stable models of $\langle \mathcal{P}, \Phi \rangle$ is Σ_P^3-complete (Π_P^3-complete).*

Proof sketch: The lower bound derives from analogous results presented in [19], in which static preferences are considered (in our framework static preferences belong to the same stratum, the first). Concerning the upper bound, the computational complexity does not increase with respect to the semantics proposed in [19]. In fact, the difference states in the introduction of the stratification of Φ, which can be done in polynomial time. The comparison of models is carried out by considering the preferences one stratum at time, instead of considering the preferences all together. The test of comparability can be also done in a polynomial time. □

Corollary 1. *Let $\langle \mathcal{P}, \Phi \rangle$ be a disjunction-free, prioritized program. Then deciding whether a literal is true in some (all) preferred stable models of $\langle \mathcal{P}, \Phi \rangle$ is Σ_P^2-complete (Π_P^2-complete).* □

Previous results states that the use of additional information does not increase the computational complexity of the proposed approach with respect to the $\mathcal{PLP}$ semantics [19].

4 Analysis and comparison

This section compares the proposed semantics with the $\mathcal{PLP}$ and $\mathcal{ASO}$ semantics proposed in [6, 19, 24]. We also briefly consider other semantics recently proposed.

The $\mathcal{PLP}$ technique is very elegant and simple and compares pairs of models on the basis of their common preferences and not on the basis of their degree of satisfaction. It does not consider the natural ordering between preference rules and, in some cases, as in Example 1 and 2, compares (and consequently discards) models which in the $\mathcal{PAS}$ approach are not comparable.

The $\mathcal{ASO}$ technique is a very powerful tool as it determines the preferred models by evaluating the degree of satisfaction of all preference rules (thus it compares two models also in the absence of common preferences). In more detail, it considers the structure of preference rules by associating a degree of satisfaction to choice options and introduces a natural ordering among preferences. As in the case of $\mathcal{PLP}$ semantics, also the $\mathcal{RASO}$ semantics compares and, consequently, discards models which are not comparable using the $\mathcal{PAS}$ technique. For instance, for the program $\langle P_1, \Phi_1 \rangle$, presented in the Introduction, it discards M_1, having the second best option of drink, even if this is the unique possible choice in the presence of `fish`.

The $\mathcal{PAS}$ semantics extends the semantics proposed in [19] by introducing the concept of comparable models and by considering a refinement of the natural order among preferences so defining the order of choices. This latter aim is modelled by refining the stratification of preference rules of [6]: levels to rules are assigned on the basis of the body atoms instead of the head atoms. Moreover, $\mathcal{PAS}$ semantics only considers well-formed programs, i.e. programs in which preferences are defined over alternative

choice options. To better understand the introduction of stratification consider the following example:

Example 7. The problem defined by means of the below prioritized program $\langle P_7, \Phi_7 \rangle$ consists in selecting the colors of the trouser and the shirt having only `black` or `blue` trousers ($\mathtt{r}_1$) and `white`, `yellow` or `red` shirts ($\mathtt{r}_2$) available. The fashion consultant suggests that `blue` trousers are better than `black` ones (ϱ_1); a `white` shirt is better than a `yellow` shirt (ϱ_2); and in the case of `black` trousers a `white` shirt is preferred to a `red` one (ϱ_3). Moreover, `blue` trousers do not go with a `white` shirt ($\mathtt{c}_1$) and a `red` shirt does not go with `blue` trousers ($\mathtt{c}_2$).

$\mathtt{r}_1 :$ `black` $\oplus$ `blue` $\leftarrow$
$\mathtt{r}_2 :$ `white` $\oplus$ `yellow` $\oplus$ `red` $\leftarrow$
$\mathtt{c}_1 : \leftarrow$ `blue`, `white`
$\mathtt{c}_2 : \leftarrow$ `red`, `blue`

$\varrho_1 :$ `blue` $>$ `black` $\leftarrow$
$\varrho_2 :$ `white` $>$ `yellow` $\leftarrow$
$\varrho_3 :$ `white` $>$ `red` $\leftarrow$ `black`

The program P_7 has four stable models: $M_1 = \{\mathtt{black}, \mathtt{white}\}$, $M_2 = \{\mathtt{black}, \mathtt{yellow}\}$, $M_3 = \{\mathtt{blue}, \mathtt{yellow}\}$ and $M_4 = \{\mathtt{black}, \mathtt{red}\}$. In order to define the stratification of preference rules, both $\mathcal{RASO}$ and $\mathcal{PAS}$ semantics firstly assign the level to atoms: first level to `blue`, `black` and `yellow` and second level to `white` and `red`. On the second step $\mathcal{RASO}$ approach, by considering the maximum level of head atoms, assigns ϱ_1 to the first level and ϱ_2 and ϱ_3 to the second level, whereas $\mathcal{PAS}$ defines the level of preferences on the basis of body atoms and assigns ϱ_1 and ϱ_2 to the first level and ϱ_3 to the second level. Note that in this case the order of ϱ_2 is relevant for determining the preferred models. In fact, $\mathcal{RASO}$ gives only M_3, while $\mathcal{PAS}$ returns M_1 and M_3 as preferred models. □

We point out that the stratification here proposed always assigns static preferences to the first level because the level of a rule is fixed by looking at the level of body atoms. Our technique introduces the concept of comparable models in order to avoid to compare models which (in our opinion) should not be compared because are associated to alternative decisions. Moreover, the presented approach does not increase the computational complexity with respect to the above mentioned techniques. Basically, in all approaches the introduction of priorities increases the complexity and expressivity of the languages by one level in the polynomial hierarchy. However, it should be pointed out that an advantage of the $\mathcal{PAS}$ technique lies in a significant reduction in the number of models to be examined. In fact, the stratification of the preference program permits the search space to be cut, because on each level i only the "best" models from the level i-1 are considered. Moreover, the introduction of *non comparable* sets of models reduces the number of preferences which have to be applied.

Other Approaches. Following the approach in [6], in [7, 8] it is proposed an extension of the $\mathcal{ASO}$ semantics. In more details, in [7] a preference description language is provided, allowing to express complex preferences by combining qualitative and quantitative, penalty based preferences. In [8] a framework to specify problem solutions (outcomes) and preferences among them is provided. The proposal combines ideas from answer-set programming, answer-set optimization and CP-nets [1]. The semantics that

we have proposed in this paper is different from both those proposed in [7, 8] as in some case, returns different results (see Examples 2 and 4).

Besides the approaches managing preferences among atoms, some other works proposed in the literature specify preferences among rules.

Early proposals expressing preferences on rules focus on Default Logic [4, 11], whereas more recently the emphasis has been given to logic programs and different proposals have been developed for representing and reasoning about user preferences such as ordered logic programs [10, 21, 23], preferred answer sets of extended logic programs [3] and logic programs with ordered disjunction [5]. Most of the approaches propose an extension of Gelfond and Lifschitz's extended logic programming by adding preference information [12, 27, 28], others attempt to extend the well founded semantics to logic programs with preferences [2, 20] and an extension of van-Gelder's alternating fixpoint theory for logic programs with priority is proposed in [27].

In [16] Gelfond and Son a methodology of reasoning with prioritized default in the language of logic programming under the answer set semantics is investigated. The approach admits the specification of preferences among rules and allows default rules which must be strictly obeyed and default rules which may be ignored if reasonable in a given context.

Delgrande et al. in [10] define an *ordered logic program* as an extended logic program whose rules are subject to a strict partial order with both static and dynamic preferences. The approach is fully prescriptive as it enforces the ordering information during the construction of the answer set. The original program is transformed into a second extended logic program in which preferences are respected in that the answer sets obtained by evaluating the transformed theory correspond to the preferred answer sets of the original theory.

In [12] it is proposed a methodology in which logic programs containing preferences on the set of rules can be translated into logic programs under stable model semantics.

5 Conclusions

In this paper the case of dynamic preferences involving atoms in logic programming has been studied. In particular, the behavior of the technique proposed by Sakama and Inoue [19] and Brewka et al. [6] has been analyzed and a semantics, interpreting each preference rule as a tool for representing a choice over alternative options, has been proposed. Specifically, the proposed approach extends the semantics proposed in [19] by considering a refinement of the natural order among preferences and introduces the concept of comparable models. Preferences and logic programs are examined together in order to determine the choice order and the sets of models which can be compared. The proposed framework forces user to introduce preferences on alternative choices so that they can be used to compare stable models on the base of alternative choices. Complexity analysis has been also performed showing that the use of additional information, regarding the preference order and the sets of non comparable models, does not increase the complexity of computing preferred stable models. Although the approach here proposed has the same expressivity of other approaches, proposed in the literature, the benefit relies in the fact it seems to better catch the intuitive meaning of prioritized programs by also considering structural information of preference rules.

References

1. Boutilier, C., Brafman, R., Domshlak, C., Hoos, H., Poole, D., CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *JAIR*, 21:135191, 2004.
2. Brewka, G., Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *JAIR*, 4: 19-36, 1996.
3. Brewka, G., Eiter, T., Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1-2), 297-356, 1999.
4. Brewka, G., Eiter, T., Prioritizing Default Logic. *Intellectics and Computational Logic*, 27-45, 2000.
5. Brewka, G., Logic programming with ordered disjunction. *AAAI/IAAI*, 100-105, 2002.
6. Brewka, G., Niemela, I., Truszczynski, M., Answer Set Optimization. *IJCAI*, 867-872, 2003.
7. Brewka, G., Complex Preferences for Answer Set Optimization, *KR*, 213-223, 2004.
8. Brewka, G., Niemela, I., Truszczynski, M., Prioritized Component Systems. *AAAI*, 596-601, 2005.
9. Buccafurri, F., Faber, W., Leone, N., Disjunctive deductive databases with inheritance. *ICLP*, 79-93, 1999.
10. Delgrande, J., P., Schaub, T., Tompits, H., Logic Programs with Compiled Preferences. *ECAI*, 464-468, 2000.
11. Delgrande, J., P., Schaub, T., Tompits, H., A Compilation of Brewka and Eiter's Approach to Prioritization. *JELIA*, 376-390, 2000.
12. Delgrande, J., P., Schaub, T., Tompits, H., A Framework for Compiling Preferences in Logic Programs. *Theory and Practice of Logic Programming*, 3(2), 129-187, 2003.
13. Eiter, T., Gottlob, G., Mannila, H., Disjunctive Datalog. *ACM Transaction On Database Systems*, 22(3), 364–418, 1997.
14. Gelfond, M., Lifschitz, V., The Stable Model Semantics for Logic Programming, *ICLP*, 1070–1080, 1988.
15. Gelfond, M., Lifschitz, V., Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, 9:365–385, 1991.
16. Gelfond, M., Son, T.C., Reasoning with prioritized defaults. *LPKR*, 164-223, 1997.
17. Grell, S., Konczak, K., Torsten Schaub, T., nomore$<$: A System for Computing Preferred Answer Sets. *LPNMR*, 394-398, 2005.
18. Papadimitriou, C. H., *Computational Complexity*. Addison-Wesley, 1994.
19. Sakama, C., Inoue, K., Priorized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123, 185-222, 2000.
20. Schaub, T., Wang , K., A Comparative Study of Logic Programs with Preference. *IJCAI*, 597-602, 2001.
21. Van Nieuwenborgh, D., Vermeir, D., Preferred Answer Sets for Ordered Logic Programs. *JELIA*, 432-443, 2002.
22. Van Nieuwenborgh, D., Vermeir, D., Ordered Diagnosis, *LPAR*, 244-258, 2003.
23. Van Nieuwenborgh, D., Heymans, S., Vermeir, D., On Programs with Linearly Ordered Multiple Preferences. *ICLP*, 180-194, 2004.
24. Wakaki, T., Inoue, K., Sakama, C., Nitta, K., Computing Preferred Answer Sets in Answer Set Programming. *LPAR*, 259-273, 2003.
25. Wakaki, T., Inoue, K., Sakama, C., Nitta, K., The PLP System. *JELIA*, 706-709, 2004.
26. Wang, X., You, J. H., Yuan, L. Y., Nonmonotonic reasoning by monotonic inferences with priority conditions. *NMELP*, 91-109, 1996.
27. Wang, K., Zhou, L., Lin, F., Alternating Fixpoint Theory for Logic Programs with Priority. *Computational Logic*, 164-178, 2000.
28. Zhang, Y., Foo, N., Answer sets for prioritized logic programs. *ILPS*, 69-83, 1997.

A Modularity Approach for a Fragment of $\mathcal{ALC}$

Andreas Herzig and Ivan Varzinczak

IRIT – 118 route de Narbonne
31062 Toulouse Cedex – France
{herzig,ivan}@irit.fr
http://www.irit.fr/LILaC

Abstract. In this paper we address the principle of modularity of ontologies in description logics. It turns out that with existing accounts of modularity of ontologies we do not completely avoid unforeseen interactions between module components, and modules designed in those ways may be as complex as whole theories. We here give a more fine-grained paradigm for modularizing descriptions. We propose algorithms that check whether a given terminology is modular and that also help the designer making it modular, if needed. Completeness, correctness and termination results are demonstrated for a fragment of $\mathcal{ALC}$. We also present the properties that ontologies that are modular in our sense satisfy w.r.t. reasoning services.

Keywords: Knowledge representation, description logics, modularity.

1 Motivation

Imagine an automatic passport control system in an airport such that all passengers must be controlled. Besides other software components, such a system is built on a passenger ontology. Suppose that the ontology is made up of statements like "a passenger has a passport", "EU citizens have EU passports", and "foreigners have non-EU passports". Such a knowledge can be encoded in description logics like $\mathcal{ALC}$ [1] by the following terminological axioms: $\mathsf{Passenger} \sqsubseteq \exists \mathsf{passport}.\top$, $\mathsf{EUcitizen} \equiv \forall \mathsf{passport}.\mathsf{EU}$, and $\mathsf{Foreigner} \equiv \forall \mathsf{passport}.\neg \mathsf{EU}$. Moreover, let the axiom $\mathsf{DoubleCitizen} \equiv \mathsf{Foreigner} \sqcap \mathsf{EUcitizen}$ define a foreigner that also has got a second citizenship of some EU country. It is not that hard to see that this description is consistent. Now, from such an ontology it follows $\mathsf{DoubleCitizen} \equiv \forall \mathsf{passport}.\bot$, and from this and the axiom $\mathsf{Passenger} \sqsubseteq \exists \mathsf{passport}.\top$ we conclude $\mathsf{DoubleCitizen} \sqsubseteq \neg \mathsf{Passenger}$, i.e., a person with double citizenship is not a passenger. Hence, if we have the assertion $\mathsf{DoubleCitizen}(\mathsf{BINLADEN})$, regarding the system behavior, this means that the concerned individual does not necessarily need to be controlled!

Despite the simplicity of such a scenario, problems like this are very likely to happen, especially if the knowledge base gets huge and hence more difficult to control. An alternative to ease maintainability of large ontologies is decomposing it into modules. Starting with [6], where modularity is assessed in logical theories in general, this issue has been investigated for ontologies in the recent

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 216–228, 2006.

literature on the subject [15, 5]. Nevertheless, it turns out that these methods for modularizing descriptions, i.e., creating independent partitions of a knowledge base, do not take into account internal interactions of components of the description that can lead to unintuitive conclusions like the one above, even if the ontology is consistent. Here we go further and propose a more fine-grained modularity principle with which we get a decomposition of the ontology so that interactions between and inside their components are limited and controlled.

Ontologies are usually represented by DL knowledge bases containing multiple roles $R_1, R_2, \ldots$ Such roles are used to formalize attributes of a concept. Then we naturally have modularity whenever a given ontology description Σ can be partitioned into sub-descriptions relative to each role:

$$\Sigma = \Sigma^{\emptyset} \cup \Sigma^{R_1} \cup \Sigma^{R_2} \cup \ldots$$

such that

- $\Sigma^{\emptyset}$ contains no role references, and
- the only role of Σ^{R_i} is R_i.

We call these sub-descriptions *modules* (some modules might be empty). Examples of such modules can easily be found in design of DL ontologies, where each Σ^{R_i} contains axioms involving only the role R_i, and $\Sigma^{\emptyset}$ is the sub-description whose axioms mention no role at all, i.e., contains only boolean combinations of concepts.

For example, for our passport control system we have the description:

$$\Sigma^{\mathsf{passport}} = \left\{ \begin{array}{c} \mathsf{Passenger} \sqsubseteq \exists \mathsf{passport}.\top, \\ \mathsf{EUcitizen} \equiv \forall \mathsf{passport}.\mathsf{EU}, \\ \mathsf{Foreigner} \equiv \forall \mathsf{passport}.\neg \mathsf{EU} \end{array} \right\}$$

$$\Sigma^{\emptyset} = \{\mathsf{DoubleCitizen} \equiv \mathsf{Foreigner} \sqcap \mathsf{EUcitizen}\}$$

Such a description is composed of two sub-descriptions, one for expressing the *attributive* part of the theory, $\Sigma^{\mathsf{passport}}$, and one to formalize the role-free constraints of the domain, $\Sigma^{\emptyset}$. $\Sigma^{\mathsf{passport}}$ formalizes the restrictions on the attributes of the concepts of the domain, in this case that a passenger must have a passport, that an EU citizen has an EU passport, and so on. $\Sigma^{\emptyset}$ establishes the *boolean* constraint according to which a double citizen is a foreigner and an EU citizen, with no regard to his attributes.

A similar partitioning of descriptions can be found in reasoning about actions, where each Σ^a contains descriptions of the atomic action a in terms of preconditions and effects, and $\Sigma^{\emptyset}$ is the set of static laws (alias domain constraints), i.e., those formulas that hold in every possible state of a dynamic system, and are thus global axioms. Another example is when mental attitudes such as knowledge, beliefs or goals of several independent agents are represented: then each module Σ^{α} contains the respective mental attitudes of agent α.

Let Σ denote a description logic ontology and suppose we want to know whether $\Sigma \models C \sqsubseteq D$, i.e., whether an axiom $C \sqsubseteq D$ follows from the description in Σ. Then it is natural to expect that we only have to consider those modules of Σ which concern the alphabet of $C \sqsubseteq D$, more specifically the roles occurring in $C \sqsubseteq D$. For instance, deductions concerning the role passport should not involve axioms for role hasDisease; querying the ontology of the passport control system should not require bothering with that of the fast-food in the airport hall. This is the problem we address in this paper.

The present work is structured as follows: in Section 2 we recall some logical definitions that we will use throughout this paper. In Section 3 we present a role-based decomposition of ontologies, which will serve as guideline for the definition of modularity in description logics we give in Section 4. We then define a fragment of $\mathcal{ALC}$ for which we have a sound and complete modularity test (Section 5). Before concluding, we show some of the benefits we get from ontologies that are modular in our sense (Section 6).

2 Description Logic $\mathcal{ALC}$

Here we briefly present the basic definitions of the description logic $\mathcal{ALC}$. For more details, see [1].

The basic syntactic building blocks of $\mathcal{ALC}$ as of any other description logics are atomic *concepts*, atomic *roles*, and *individuals*. We call atomic concepts and atomic roles elementary descriptions. Complex descriptions are built from them with concept constructors. We use A to denote atomic concepts, R for atomic roles, and $C, D, \ldots$ for complex concept descriptions.

Complex concept descriptions are recursively defined in the following way:

$$
\begin{array}{rll}
C ::= & A \mid & \text{(an atomic concept)} \\
& \top \mid & \text{(universal concept)} \\
& \bot \mid & \text{(contradiction concept)} \\
& \neg C \mid & \text{(complement)} \\
& C \sqcap C \mid & \text{(conjunction)} \\
& C \sqcup C \mid & \text{(disjunction)} \\
& \forall R.C \mid & \text{(value restriction)} \\
& \exists R.C \mid & \text{(existential restriction)}
\end{array}
$$

where A ranges over atomic concepts, R over atomic roles, and C over complex concepts. Recalling our running example, the statements $\mathsf{Foreigner} \sqcap \mathsf{EUcitizen}$, $\exists \mathsf{passport}.\top$, $\forall \mathsf{passport}.\mathsf{EU}$, and $\forall \mathsf{passport}.\neg \mathsf{EU}$ are complex concepts in $\mathcal{ALC}$.

We use individuals to describe a specific state of affairs in terms of concepts and roles. We use $a, b, \ldots$ to denote individuals. In our example, JAN and POLAND are individuals of which we can assert, respectively, the properties EUcitizen and EU. The intended meaning of such assertions is that JAN has EU citizenship and POLAND is a member of the European community. Individuals and assertions about them allow us to give a description of the world.

Definition 1. *An* interpretation $\mathcal{I}$ *is a tuple* $\langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ *such that* $\Delta^{\mathcal{I}}$ *is a nonempty set and* $\cdot^{\mathcal{I}}$ *a function mapping:*

- *every concept to a subset of* $\Delta^{\mathcal{I}}$
- *every role to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
- *every individual to an element of* $\Delta^{\mathcal{I}}$

Given an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, $\Delta^{\mathcal{I}}$ is the interpretation *domain*, and $\cdot^{\mathcal{I}}$ the associated interpretation function. If a is an individual name, A an atomic concept, R an atomic role, and C, D concepts, we have:

$$
\begin{aligned}
a^{\mathcal{I}} &\in \Delta^{\mathcal{I}} \\
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\
R^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \ : \ \forall b.(a,b) \in R^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \ : \ \exists b.(a,b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}
\end{aligned}
$$

In $\mathcal{ALC}$ we also have *terminological axioms* (axioms, for short). These are statements of the form $C \equiv D$ and $C \sqsubseteq D$. Axioms of the first kind are called *concept definitions* (alias *equalities*). Those of the second kind are called *concept inclusion axioms* (alias *inclusions* or *subsumptions*). If C and D are both complex concepts, then $C \sqsubseteq D$ is called a *general concept inclusion* axiom (GCI).

An interpretation $\mathcal{I}$ satisfies a concept definition $C \equiv D$ (noted $\models^{\mathcal{I}} C \equiv D$) if $C^{\mathcal{I}} = D^{\mathcal{I}}$. Intuitively, $C \equiv D$ establishes a definition for concept C in terms of D. In our example, we have $\mathsf{DoubleCitizen} \equiv \mathsf{Foreigner} \sqcap \mathsf{EUcitizen}$, which gives both necessary and sufficient conditions to be a person with double citizenship.

An interpretation $\mathcal{I}$ satisfies a subsumption $C \sqsubseteq D$ (noted $\models^{\mathcal{I}} C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Intuitively, $C \sqsubseteq D$ means that concept C is more specific than concept D. In our example we have $\mathsf{DoubleCitizen} \sqsubseteq \mathsf{EUcitizen}$, which says that a person with double citizenship is a specialization of a European citizen. We also have $\mathsf{Passenger} \sqsubseteq \exists \mathsf{passport}.\top$, saying that a necessary condition to be a passenger is having a passport. Concept inclusion axioms are used when one is not able to completely define a concept: in the last example, a passenger may have many other properties of which the knowledge engineer was not necessarily aware when modeling the description.

We call a (finite) set of terminological axioms a *terminology*, alias TBox. We denote TBoxes by $\mathcal{T}$. An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ (noted $\models^{\mathcal{I}} \mathcal{T}$) if $\models^{\mathcal{I}} C \sqsubseteq D$ for all $C \sqsubseteq D \in \mathcal{T}$. An axiom $C \sqsubseteq D$ is a *consequence* of a TBox $\mathcal{T}$ (noted $\mathcal{T} \models C \sqsubseteq D$) if for every interpretation $\mathcal{I}$, $\models^{\mathcal{I}} \mathcal{T}$ implies $\models^{\mathcal{I}} C \sqsubseteq D$.

Henceforth we can suppose w.l.o.g. that TBoxes are *linearized*, i.e., $\mathcal{T}$ only contains inclusion axioms (no concept definitions), and see $C \equiv D$ as just as an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$.

A *concept assertion* is a statement about an individual with respect to some concept. We denote by $C(a)$ the fact that a belongs to (the interpretation of) concept C. In our example, the assertion $\mathsf{Foreigner}(\mathsf{JOHN})$ says that JOHN is a non-European citizen, and that all properties a foreigner has (e.g. possessing a non-EU passport) apply to JOHN as well.

A *role assertion* establishes a relationship between two individuals. If a, b are individuals and R is a role name, then $R(a, b)$ asserts that b is a *filler* of the role R for a. In our example, the role assertion $\mathsf{refund}(\mathsf{JOHN}, \mathsf{VAT})$ states that JOHN can claim the refund of the value added tax when leaving the airport.

An interpretation $\mathcal{I}$ satisfies a concept assertion $C(a)$ (noted $\models^{\mathcal{I}} C(a)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and a role assertion $R(a, b)$ (noted $\models^{\mathcal{I}} R(a, b)$) if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A (finite) set of concept and role assertions define an ABox. We denote ABoxes by $\mathcal{A}$. An interpretation $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ (noted $\models^{\mathcal{I}} \mathcal{A}$) if $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$. A concept assertion $C(a)$ (resp. a role assertion $R(a, b)$) is a *consequence* of an ABox $\mathcal{A}$, noted $\mathcal{A} \models C(a)$ (resp. $\mathcal{A} \models R(a, b)$), if for every interpretation $\mathcal{I}$, $\models^{\mathcal{I}} \mathcal{A}$ implies $\models^{\mathcal{I}} C(a)$ (resp. $\models^{\mathcal{I}} R(a, b)$).

A *knowledge base* is a tuple $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox. An interpretation $\mathcal{I}$ is a model of $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ if $\models^{\mathcal{I}} \mathcal{T}$ and $\models^{\mathcal{I}} \mathcal{A}$. Logical consequence of an axiom $C \sqsubseteq D$, of a concept assertion $C(a)$ and of a role assertion $R(a, b)$ from a knowledge base Σ is defined in the standard way.

In the rest of this paper we are going to restrict ourselves only to the TBox component of knowledge bases.

3 Role-Based Decomposition

Here we give a novel way of decomposing ontologies. Let $\mathfrak{Roles} = \{R_1, R_2, \ldots\}$ be the set of all role names of a domain. Let $\mathit{roles}(C \sqsubseteq D)$ return the set of role names occurring in an axiom $C \sqsubseteq D$. For instance $\mathit{roles}(C \equiv \exists R_1.D \sqcap \forall R_2.E) = \{R_1, R_2\}$. Moreover, for a TBox $\mathcal{T}$, let $\mathit{roles}(\mathcal{T}) = \bigcup_{C \sqsubseteq D \in \mathcal{T}} \mathit{roles}(C \sqsubseteq D)$.

With that we define a role-based classification of axioms.

Definition 2. *A* boolean axiom *is an axiom $C \sqsubseteq D$ such that $\mathit{roles}(C \sqsubseteq D) = \emptyset$. If $\mathit{roles}(C \sqsubseteq D) \neq \emptyset$, $C \sqsubseteq D$ is a* non-boolean axiom.

If $\mathscr{R} \subseteq \mathfrak{Roles}$, $\mathscr{R} \neq \emptyset$, then we define

$$\mathcal{T}^{\mathscr{R}} = \{C \sqsubseteq D \in \mathcal{T} \ : \ \mathit{roles}(C \sqsubseteq D) \cap \mathscr{R} \neq \emptyset\}$$

Hence, $\mathcal{T}^{\mathscr{R}}$ contains all non-boolean axioms of the terminology $\mathcal{T}$ whose roles appear in $\mathscr{R}$. For $\mathscr{R} = \emptyset$, $\mathcal{T}^{\emptyset} = \{C \sqsubseteq D \in \mathcal{T} \ : \ \mathit{roles}(C \sqsubseteq D) = \emptyset\}$ is the set of all boolean axioms of a knowledge base.

For example, if

$$\mathcal{T} = \left\{ \begin{array}{c} \mathsf{Passenger} \sqsubseteq \exists \mathsf{passport}.\top,\ \mathsf{EUcitizen} \equiv \forall \mathsf{passport}.\mathsf{EU}, \\ \mathsf{Foreigner} \equiv \forall \mathsf{passport}.\neg \mathsf{EU},\ \mathsf{Foreigner} \sqsubseteq \exists \mathsf{refund}.\mathsf{Tax}, \\ \mathsf{DoubleCitizen} \equiv \mathsf{Foreigner} \sqcap \mathsf{EUcitizen} \end{array} \right\}$$

then we have

$$\mathcal{T}^{\{\mathsf{refund}\}} = \{\mathsf{Foreigner} \sqsubseteq \exists \mathsf{refund}.\mathsf{Tax}\}$$

and

$$\mathcal{T}^{\emptyset} = \{\mathsf{DoubleCitizen} \equiv \mathsf{Foreigner} \sqcap \mathsf{EUcitizen}\}$$

For parsimony's sake, we write $\mathcal{T}^R$ instead of $\mathcal{T}^{\{R\}}$.

Given these fundamental concepts, we are able to formally define modularity for ontologies in description logics.

4 Modular TBoxes

We can suppose from now on that $\mathcal{T}$ is *partitioned*, in the sense that $\{\mathcal{T}^{\emptyset}\} \cup \{\mathcal{T}^{R_i} \ : \ R_i \in \mathfrak{Roles}\}$ is a partition[1] of $\mathcal{T}$. We thus exclude $\mathcal{T}^{R_i}$ containing more than one role name, which means that complex concepts with nested roles are not allowed. We thus make it a hypothesis:

$$\{\mathcal{T}^{\emptyset}\} \cup \{\mathcal{T}^{R_i} \ : \ R_i \in \mathfrak{Roles}\} \text{ partitions } \mathcal{T} \tag{H}$$

We are interested in the following principle of modularity:

Definition 3. *A terminology $\mathcal{T}$ is* modular *if and only if for every $C \sqsubseteq D$,*

$$\mathcal{T} \models C \sqsubseteq D \text{ implies } \mathcal{T}^{roles(C \sqsubseteq D)} \cup \mathcal{T}^{\emptyset} \models C \sqsubseteq D.$$

Modularity means that when investigating whether $C \sqsubseteq D$ is a consequence of $\mathcal{T}$, the only axioms in $\mathcal{T}$ that are relevant are those whose role names occur in $C \sqsubseteq D$ and the boolean axioms in $\mathcal{T}^{\emptyset}$.

This is reminiscent of interpolation [4], which for the case of roles says:

Definition 4. *A terminology $\mathcal{T}$ has the* interpolation property *if and only if for every axiom $C \sqsubseteq D$, if $\mathcal{T} \models C \sqsubseteq D$, then there is a terminology $\mathcal{T}_{C \sqsubseteq D}$ such that*

- $roles(\mathcal{T}_{C \sqsubseteq D}) \subseteq roles(\mathcal{T}) \cap roles(C \sqsubseteq D)$
- $\mathcal{T} \models C' \sqsubseteq D'$ *for every* $C' \sqsubseteq D' \in \mathcal{T}_{C \sqsubseteq D}$
- $\mathcal{T}_{C \sqsubseteq D} \models C \sqsubseteq D$

[1] Remembering, $\{\mathcal{T}^{\emptyset}\} \cup \{\mathcal{T}^{R_i} \ : \ R_i \in \mathfrak{Roles}\}$ partitions $\mathcal{T}$ if and only if $\mathcal{T} = \mathcal{T}^{\emptyset} \cup \bigcup_{R_i \in \mathfrak{Roles}} \mathcal{T}^{R_i}$, and $\mathcal{T}^{\emptyset} \cap \mathcal{T}^{R_i} = \emptyset$, and $\mathcal{T}^{R_i} \cap \mathcal{T}^{R_j} = \emptyset$, if $i \neq j$. Note that $\mathcal{T}^{\emptyset}$ and $\mathcal{T}^{R_i}$ might be empty.

Our definition of modularity is a strengthening of interpolation because it requires $\mathcal{T}_{C \sqsubseteq D}$ to be a subset of $\mathcal{T}$.

Contrary to interpolation however, modularity does not generally hold. Clearly if the Hypothesis (H) is not satisfied, then modularity fails. To witness, consider

$$\mathcal{T} = \{C \equiv \forall R_1.\forall R_2.C', \forall R_1.\forall R_2.C' \equiv D\}$$

Then $\mathcal{T} \models C \equiv D$, but $\mathcal{T}^{\emptyset} \not\models C \equiv D$.

Nevertheless even under our hypothesis modularity may fail to hold. For example, let

$$\mathcal{T} = \{C \sqcup \forall R.\bot \equiv \top, C \sqcup \exists R.\top \equiv \top\}$$

Then $\mathcal{T}^{\emptyset} = \emptyset$, and $\mathcal{T}^{R} = \mathcal{T}$. Now $\mathcal{T} \models C$, but clearly $\mathcal{T}^{\emptyset} \not\models C$.

How can we know whether a given TBox $\mathcal{T}$ is modular? The following criterion is simpler:

Definition 5. *A terminology $\mathcal{T}$ is* boolean-modular *if and only if for every boolean axiom $C \sqsubseteq D$,*

$$\mathcal{T} \models C \sqsubseteq D \text{ implies } \mathcal{T}^{\emptyset} \models C \sqsubseteq D.$$

With that we guarantee modularity:

Theorem 1 ([12]). *Let $\mathcal{T}$ be a partitioned terminology. If $\mathcal{T}$ is boolean-modular, then $\mathcal{T}$ is modular.*

In the rest of the paper we investigate how it can be automatically checked whether a given terminology $\mathcal{T}$ is modular and how to make it modular, if needed. We do this for a version of $\mathcal{ALC}$ with a restriction on the form of the axioms we can state in a TBox.

5 Soundness and Completeness for a Fragment of $\mathcal{ALC}$

Definition 6. *A concept C is a* boolean concept *if $roles(C) = \emptyset$.*

We here make a syntactical restriction on the form of non-boolean axioms in our TBoxes.

Definition 7. *If C is a boolean concept, then $\forall R.C$ is a* boolean value restriction*, and $\exists R.C$ is a* boolean existential restriction.

In this section we suppose that:

All value/existential restrictions in a knowledge base are boolean value/existential restrictions. (H2)

Our fragment differs from $\mathcal{ALC}$ just in the sense that only boolean concepts are allowed in the scope of a quantification over a role. We observe however that we could allow for axioms with nested roles like $C \equiv \forall R_1.\forall R_2.D$ and GCIs

like $\forall R_3.E \sqsubseteq \forall R_4.F$. For that it would suffice to adapt an existing technique of *subformula renaming* [17] in the literature on classical logic [14, 2, 3] to recursively replace complex concepts with some new concepts, stating definitions for these as global axioms. For instance, $C \equiv \forall R_1.\forall R_2.D$ should then be rewritten as $C \equiv \forall R_1.C'$ and $C' \equiv \forall R_2.D$, and $\forall R_3.E \sqsubseteq \forall R_4.F$ could be replaced by $E' \sqsubseteq \forall R_4.F$ and $E' \equiv \forall R_4.E$, where C', E' are new concept names. It is known that subformula renaming is satisfiability preserving and can be computed in polynomial time [13]. However it remains to assess the impact the introduction of new concept names can have on the intuition about the original ontology.

Our central hypothesis here is that the different types of axioms in a given terminology should be neatly separated and only interfere in one sense: boolean axioms together with non-boolean axioms for role R may have consequences that do not follow from the non-boolean axioms for R alone. The other way round, non-boolean axioms should not allow to infer new boolean axioms. That is what we expect modularity of TBoxes to establish and we develop it in the sequel.

Definition 8. *A boolean inclusion axiom $C \sqsubseteq D$ is an* implicit boolean inclusion axiom *of a terminology $\mathcal{T}$ if and only if $\mathcal{T} \models C \sqsubseteq D$ and $\mathcal{T}^\emptyset \not\models C \sqsubseteq D$.*

In our running example, $\mathsf{DoubleCitizen} \sqsubseteq \neg\mathsf{Passenger}$ is an example of an implicit boolean inclusion axiom.

With Algorithm 1 below we can check whether a TBox has such implicit axioms. The idea is as follows: for each pair of axioms $C \sqsubseteq \exists R.D$ and $E \sqsubseteq \forall R.F$ in $\mathcal{T}$ such that F conflicts with D, i.e., $\mathcal{T} \models D \sqcap F \sqsubseteq \bot$, if $\mathcal{T}^\emptyset \cup \{C \sqcap E\}$ is satisfiable and $\mathcal{T}^\emptyset \not\models C \sqsubseteq \neg E$, mark $C \sqsubseteq \neg E$ as an implicit boolean inclusion axiom.

Algorithm 1. Deciding existence of implicit boolean inclusion axioms

input: a TBox $\mathcal{T}$
output: a set of implicit boolean inclusion axioms $\mathcal{T}^\emptyset_{imp}$

$\mathcal{T}^\emptyset_{imp} := \emptyset$
for all $R \in \mathfrak{Roles}$ **do**
  for all $\{C_1 \sqsubseteq \exists R.D_1, \ldots, C_n \sqsubseteq \exists R.D_n\} \subseteq \mathcal{T}$ **do**
    for all $\{E_1 \sqsubseteq \forall R.F_1, \ldots, E_m \sqsubseteq \forall R.F_m\} \subseteq \mathcal{T}$ **do**
      if $\mathcal{T}^\emptyset \not\models \bigsqcap_{1\leq i\leq n} C_i \sqcap \bigsqcap_{1\leq j\leq m} E_i \sqsubseteq \bot$ **and**
        $\mathcal{T}^\emptyset \models \bigsqcap_{1\leq i\leq n} D_i \sqcap \bigsqcap_{1\leq j\leq m} F_i \sqsubseteq \bot$ **then**
        $\mathcal{T}^\emptyset_{imp} := \mathcal{T}^\emptyset_{imp} \cup \{\bigsqcap_{1\leq i\leq n} C_i \sqsubseteq \bigsqcup_{1\leq j\leq m} \neg E_i\}$

Theorem 2. *Algorithm 1 terminates.*

Proof. Straightforward from finiteness of $\mathcal{T}$.

Lemma 1. *Let $\mathcal{T}^{\emptyset}_{imp*}$ be the output of Algorithm 2 on input $\mathcal{T}$. Then every $C \sqsubseteq D \in \mathcal{T}^{\emptyset}_{imp}$ is an implicit boolean inclusion axiom of $\mathcal{T}$.*

Converse of Lemma 1 does not hold. Indeed, consider the quite simple TBox:

$$\mathcal{T} = \left\{ \begin{array}{c} C_n \sqsubseteq \bot, \\ C_{i-1} \sqsubseteq \forall R.C_i,\ 1 \leq i \leq n, \\ \top \sqsubseteq \exists R.\top \end{array} \right\}$$

Thus, $\mathcal{T} \models C_i \sqsubseteq \bot$, for $0 \leq i \leq n$, but running Algorithm 1 returns only $\mathcal{T}^{\emptyset}_{imp} = \{C_{n-1} \sqsubseteq \bot\}$. This suggests that it is necessary to iterate the algorithm in order to find all implicit boolean inclusion axioms. Before doing that we observe that:

Theorem 3. *A terminology $\mathcal{T}$ is modular if and only if $\mathcal{T}^{\emptyset}_{imp} = \emptyset$.*

Considering the example just above, we can see that running Algorithm 1 on $\mathcal{T} \cup \{C_{n-1} \sqsubseteq \bot\}$ will give us $\mathcal{T}^{\emptyset}_{imp} = \{C_{n-2} \sqsubseteq \bot\}$. This means that some of the implicit boolean inclusion axioms of a terminology may be needed in order to derive others. Hence, Algorithm 1 must be iterated to get $\mathcal{T}$ modular. This is achieved with the following algorithm, which iteratively feeds the set of boolean axioms considered into the **if**-test of Algorithm 1:

Algorithm 2. Finding all implicit boolean inclusion axioms

input: a TBox $\mathcal{T}$
output: $\mathcal{T}^{\emptyset}_{imp*}$, the set of all implicit boolean inclusion axioms of $\mathcal{T}$

$\mathcal{T}^{\emptyset}_{imp*}$:= $\emptyset$
repeat
  $\mathcal{T}^{\emptyset}_{imp}$:= *find_imp_bia*($\mathcal{T} \cup \mathcal{T}^{\emptyset}_{imp*}$) {a call to Algorithm 1}
  $\mathcal{T}^{\emptyset}_{imp*}$:= $\mathcal{T}^{\emptyset}_{imp*} \cup \mathcal{T}^{\emptyset}_{imp}$
until $\mathcal{T}^{\emptyset}_{imp} = \emptyset$

Theorem 4. *Algorithm 2 terminates.*

Theorem 5. *Let $\mathcal{T}^{\emptyset}_{imp*}$ be the output of Algorithm 2 on input $\mathcal{T}$. Then*

1. *$\mathcal{T} \cup \{\mathcal{T}^{\emptyset}_{imp*}\}$ is modular;*
2. *$\mathcal{T} \models \bigcap\{\mathcal{T}^{\emptyset}_{imp*}\}$.*

Corollary 1. *For all boolean inclusion axioms $C \sqsubseteq D$, $\mathcal{T} \models C \sqsubseteq D$ if and only if $\mathcal{T} \cup \{\mathcal{T}^{\emptyset}_{imp*}\} \models C \sqsubseteq D$.*

This establishes that Algorithm 2 finds all implicit boolean inclusion axioms of a given terminology $\mathcal{T}$. Hence, adding such axioms to the original set of boolean axioms $\mathcal{T}^{\emptyset}$ guarantees modularity of $\mathcal{T}$.

We want to point out, however, that the algorithm only catches implicit boolean inclusion axioms. Deciding whether they are intuitive remains the knowledge engineer's task, and only she can carry out changes in the knowledge base in order to accommodate them in or discard them from the description. In our running example, the inclusion $\mathsf{DoubleCitizen} \sqsubseteq \neg\mathsf{Passenger}$ is not intuitive and should then be contracted from the terminology.

Algorithms 1 and 2 are generalizations/extensions of the method for PDL given in [12] where (in terms of description logics) only existential restrictions of the form $C \sqsubseteq \exists R.\top$ were allowed.

6 The Role of Modularity in Reasoning Services

The following result is important in the ontology building phase:

Theorem 6. *Let $\mathcal{T}$ and $C \sqsubseteq D$ be such that $\mathcal{T} \not\models \top \sqsubseteq \bot$. If $\mathcal{T}$ is modular, then $\mathcal{T} \cup \{C \sqsubseteq D\} \models \top \sqsubseteq \bot$ if and only if $\mathcal{T}^{\emptyset} \cup \mathcal{T}^{roles(C \sqsubseteq D)} \cup \{C \sqsubseteq D\} \models \top \sqsubseteq \bot$.*

This theorem says that under modularity consistency of a new learned axiom $C \sqsubseteq D$ w.r.t. a consistent TBox reduces to consistency check of the axioms that are relevant to $C \sqsubseteq D$.

Theorem 7. *If $\mathcal{T}$ is modular, then $\mathcal{T} \models \top \sqsubseteq \bot$ if and only if $\mathcal{T}^{\emptyset} \models \top \sqsubseteq \bot$.*

Hence, if there are no implicit boolean inclusion axioms, then consistency of the whole terminology can be checked by just checking consistency of $\mathcal{T}^{\emptyset}$.

It turns out that checking whether a concept C is the *least common subsumer* (*lcs*) of a set of concepts, i.e., the minimal concept that subsumes all other concepts in question [1], is also optimized under modularity:

Theorem 8. *Let Γ be a set of concepts. If $\mathcal{T}$ is modular, then C is the lcs of Γ w.r.t. $\mathcal{T}$ if and only if C is the lcs of Γ w.r.t. $\mathcal{T}^{\emptyset} \cup \mathcal{T}^{roles(C)}$.*

For $\mathcal{T}$ a TBox, we define $\mathcal{T}^R_{\forall} = \{C \sqsubseteq \forall R.D \ : \ C \sqsubseteq \forall R.D \in \mathcal{T}\}$, i.e., $\mathcal{T}^R_{\forall}$ contains all non-boolean axioms in the TBox $\mathcal{T}$ with value restrictions for role R.

Theorem 9. *If $\mathcal{T}$ is modular, then*

$$\mathcal{T} \models C \sqsubseteq \forall R.D \text{ if and only if } \mathcal{T}^{\emptyset} \cup \mathcal{T}^R_{\forall} \models C \sqsubseteq \forall R.D.$$

This means that under our modularity principle we have modularity inside the module for non-boolean axioms, too: when deducing an axiom with value restrictions we do not need to consider axioms with existential restrictions.

The existential restriction counterpart of Theorem 9, however, does not hold. To witness, from the modular description $\{\forall R.C \sqcup D, \exists R.\neg C\}$ we conclude $\exists R.D$, but $\{\exists R.\neg C\} \not\models \exists R.D$. Nevertheless, we can establish a result if only the universal concept ($\top$) is allowed in the scope of existential restrictions. For that we define $\mathcal{T}^R_{\exists} = \{C \sqsubseteq \exists R.\top \ : \ C \sqsubseteq \exists R.\top \in \mathcal{T}\}$.

Theorem 10. *If $\mathcal{T}$ is modular, then*

$$\mathcal{T} \models C \sqsubseteq \exists R.\top \text{ if and only if } \mathcal{T}^{\emptyset} \cup \mathcal{T}_{\exists}^{R} \models C \sqsubseteq \exists R.\top.$$

Let $\mathcal{T}_{\forall}^{R_1,\ldots,R_n} = \bigcup_{1 \leq i \leq n} \mathcal{T}_{\forall}^{R_i}$. The following theorem shows that under modularity deduction of an axiom based on nested value restrictions does not need the axioms based on existential restrictions:

Theorem 11. *If $\mathcal{T}$ is modular, then $\mathcal{T} \models C \sqsubseteq \forall R_1 \ldots \forall R_n.D$ if and only if $\mathcal{T}^{\emptyset} \cup \mathcal{T}_{\forall}^{R_1,\ldots,R_n} \models C \sqsubseteq \forall R.D$.*

The same result holds for deductions of axioms based on existential restrictions under the assumption that only $\top$ is allowed in the scope of $\exists$. Let $\mathcal{T}_{\exists}^{R_1,\ldots,R_n} = \bigcup_{1 \leq i \leq n} \mathcal{T}_{\exists}^{R_i}$.

Theorem 12. *If $\mathcal{T}$ is modular, then $\mathcal{T} \models C \sqsubseteq \exists R_1 \ldots \exists R_n.\top$ if and only if $\mathcal{T}^{\emptyset} \cup \mathcal{T}_{\exists}^{R_1,\ldots,R_n} \models C \sqsubseteq \exists R.\top$.*

7 Concluding Remarks

We defined here a modularity paradigm for ontologies in description logics and pointed out some of the problems that arise if it is not satisfied, even if the ontology is consistent. In particular we have argued that the boolean part of a description could influence but should not be influenced by the role-based one.

We have seen that the presence of implicit boolean inclusion axioms is a sign that we possibly have slipped up in designing the ontology in question. We showed how to detect this problem in a fragment of $\mathcal{ALC}$ with a syntactical restriction on its formulas. With Algorithm 2 we have a sound and complete decision procedure for such a task. Moreover, the output of the algorithm gives us guidelines that can help correcting the ontology.

We could also use full $\mathcal{ALC}$, in this case our method is sound but not complete. As an example, let $\mathcal{T} = \{C \equiv \forall R_1.\forall R_2.D, C' \equiv \forall R_1.\exists R_2.\neg D, \top \equiv \exists R_1.\top\}$. We have $\mathcal{T} \models C \sqsubseteq \neg C'$, but running Algorithm 2 on $\mathcal{T}$ gives $\mathcal{T}_{imp^*}^{\emptyset} = \emptyset$.

It could be argued that unintuitive consequences in ontologies are mainly due to badly written axioms and not to lack of modularity. True enough, but what we presented here is the case that making an ontology modular gives us a tool to detect some of such problems and correct it. (But note that we do not claim to correct badly written axioms automatically and once for all.) Besides this, having separate entities in the ontology and controlling their interaction help us to localize where the problems are, which is crucial for real world applications.

As our theorems show (proofs were omitted due to lack of space), being modular is a useful feature of terminologies w.r.t. reasoning: beyond being a reasonable principle of design that helps structuring data, it clearly restricts the search space, and thus makes reasoning easier.

The first work on formalizing modularity in logical systems in general seems to be due to Garson [6]. Modularity of theories in reasoning about actions was originally defined in [10] and extensively developed in [12, 9]. A different viewpoint

of that can be found in [11], where modularity of action theories is assessed from a more software engineering oriented perspective. The present work has been inspired by ideas in the referred approaches. Following [6], a modularization technique for ontologies in DL different from ours is addressed in [5].

Our notion of modularity is related to uniform interpolation for TBoxes [7]. Let $\mathit{concepts}(\mathcal{T})$ denote the concept names occurring in a TBox $\mathcal{T}$. Given $\mathcal{T}$ and a signature $\mathcal{S} \subseteq \mathit{concepts}(\mathcal{T}) \cup \mathit{roles}(\mathcal{T})$, a TBox $\mathcal{T}^{\mathcal{S}}$ over $(\mathit{concepts}(\mathcal{T}) \cup \mathit{roles}(\mathcal{T})) \setminus \mathcal{S}$ is a *uniform interpolant* of $\mathcal{T}$ outside $\mathcal{S}$ if and only if:

- $\mathcal{T} \models \mathcal{T}^{\mathcal{S}}$;
- $\mathcal{T}^{\mathcal{S}} \models C \sqsubseteq D$ for every $C \sqsubseteq D$ that has no occurrences of symbols from $\mathcal{S}$.

It is not difficult to see that a partition $\{\mathcal{T}^{\emptyset}\} \cup \{\mathcal{T}^{R_i} : R_i \in \mathfrak{Roles}\}$ is modular if and only if every $\mathcal{T}^{R_i}$ is a uniform interpolant of $\mathcal{T}$ outside $\mathit{roles}(\mathcal{T}) \setminus \{R_i\}$. In [16] there are complexity results for computing uniform interpolants in $\mathcal{ALC}$.

In [7] a notion of conservative extension is defined that is similar to our modularity. There, $\mathcal{T}_1 \cup \mathcal{T}_2$ is a *conservative extension* of $\mathcal{T}_1$ if and only if for all concepts C, D built from $\mathit{concepts}(\mathcal{T}_1) \cup \mathit{roles}(\mathcal{T}_1)$, $\mathcal{T}_1 \cup \mathcal{T}_2 \models C \sqsubseteq D$ implies $\mathcal{T}_1 \models C \sqsubseteq D$.

Given our Theorem 1, we can show that checking for modularity can be reduced to checking for conservative extensions of $\mathcal{T}^{\emptyset}$. Indeed, supposing that the signature of $\mathcal{T}^{\emptyset}$ is the set of all concept names, we have that $\mathcal{T}$ is modular if and only if for every role R_i, $\mathcal{T}^{R_i} \cup \mathcal{T}^{\emptyset}$ is a conservative extension of $\mathcal{T}^{\emptyset}$.

We plan to pursue further work on extensions of our method to more expressive description logics. Another extension that we foresee is generalizing modularity to also take into account ABoxes. In this case our algorithms should be adapted so that implicit interactions between terminologies and assertions can be caught.

Because interactions between TBoxes and ABoxes may lead to inconsistency, ontology update and revision should be considered, too. We are currently investigating update of terminologies based on the method given in [8], for which satisfaction of modularity shows to be fruitful.

Acknowledgments

We are grateful to the anonymous referees for useful comments on an earlier version of this paper. Thanks to Bernardo Cuenca Grau for useful and interesting discussions about the topic of this work. We also would like to thank Meghyn Bienvenu for comments about DL terminology and notation.

Ivan Varzinczak has been supported by a fellowship from the government of the Federative Republic of Brazil. Grant: CAPES BEX 1389/01-7.

References

1. F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, *Description Logic Handbook*, chapter 2, pages 47–100. Cambridge University Press, 2003.

2. T. Boy de la Tour. Minimizing the number of clauses by renaming. In M.E. Stickel, editor, *Proc. 10th International Conference on Automated Deduction (CADE'90)*, pages 558–572, London, 1990. Springer-Verlag. LNCS, vol. 449.
3. T. Boy de la Tour. An optimality result for clause form translation. *J. of Symbolic Computation*, 14(4):283–301, 1992.
4. W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. of Symbolic Logic*, 22:250–268, 1957.
5. B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In *Proc. 10th Intl. Conf. on Knowledge Representation and Reasoning (KR'2006)*, pages 198–208, Lake District, 2006. Morgan Kaufmann Publishers.
6. J. Garson. Modularity and relevant logic. *Notre Dame J. of Formal Logic*, 30(2):207–223, 1989.
7. S. Ghilardi, C. Lutz, and F. Wolter. Did I damage my ontology? a case for conservative extensions in description logic. In *Proc. 10th Intl. Conf. on Knowledge Representation and Reasoning (KR'2006)*, pages 187–197, Lake District, 2006. Morgan Kaufmann Publishers.
8. A. Herzig, L. Perrussel, and I. Varzinczak. Elaborating domain descriptions. In *Proc. 17th Eur. Conf. on Artificial Intelligence (ECAI'06)*, Riva del Garda, 2006. IOS Press.
9. A. Herzig and I. Varzinczak. Metatheory of actions: beyond consistency. To appear.
10. A. Herzig and I. Varzinczak. Domain descriptions should be modular. In R. López de Mántaras and L. Saitta, editors, *Proc. 16th Eur. Conf. on Artificial Intelligence (ECAI'04)*, pages 348–352, Valencia, 2004. IOS Press.
11. A. Herzig and I. Varzinczak. Cohesion, coupling and the meta-theory of actions. In L. Kaelbling and A. Saffiotti, editors, *Proc. 19th Intl. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pages 442–447, Edinburgh, 2005. Morgan Kaufmann Publishers.
12. A. Herzig and I. Varzinczak. On the modularity of theories. In R. Schmidt, I. Pratt-Hartmann, M. Reynolds, and H. Wansing, editors, *Advances in Modal Logic*, volume 5, pages 93–109. King's College Publications, 2005. Selected papers of AiML 2004 (also available at `http://www.aiml.net/volumes/volume5`).
13. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In A. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, pages 335–367. 2001.
14. D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. of Symbolic Computation*, 2(3):293–304, 1986.
15. H. Stuckenschmidt and M. Klein. Integrity and change in modular ontologies. In V. Sorge, S. Colton, M. Fisher, and J. Gow, editors, *Proc. 18th Intl. Joint Conf. on Artificial Intelligence (IJCAI'03)*, pages 900–908, Acapulco, 2003. Morgan Kaufmann Publishers.
16. B. ten Cat, W. Conradie, M. Marx, and Y. Venema. Definitorially complete description logics. In *Proc. 10th Intl. Conf. on Knowledge Representation and Reasoning (KR'2006)*, pages 79–89, Lake District, 2006. Morgan Kaufmann Publishers.
17. G.S. Tseitin. On the complexity of derivation in propositional calculus. In A. Slisenko, ed., Studies in Constructive Mathematics and Mathematical Logics, Part II, 1968.

Whatever You Say

Luke Hunsberger

Vassar College
Poughkeepsie, NY 12604-0444, USA
hunsberg@cs.vassar.edu

Abstract. This paper addresses an important problem in multi-agent coordination: the formal representation of parameters in the content of agent intentions that are only partially specified (e.g., when the intended action has not yet been executed and values for the parameters have not yet been chosen or the authority for choosing such values has been delegated to others). For example, Abe might intend to rent "whatever car Zoe tells him to", in which case the problem is how to formally represent the quoted clause (i.e., the "whatever" content). The paper presents a two-pronged approach. First, it uses the event calculus to model declarative speech-acts which agents use to establish facts about parameters in a social context. Second, it partitions the content of agent intentions into (1) a condition that the agent should refrain from determining and (2) a goal that the agent should strive to achieve. The satisfaction conditions of such intentions treat these types of content differently; however they can share variables and, thus, are linked in a restricted sense.

1 Introduction

Since people have limited computational resources, they cannot, at each moment, instantaneously compute their optimal action for that moment; instead, they must plan ahead [3]. Thus, they adopt plans and intentions concerning their future activity which are only partially specified and which they subsequently elaborate over time [4, 10]. One common way for plans to be only partially specified is that their parameters may not be fully determined. For example, while having no particular car in mind, Abe might intend to rent *a car*. Later on, Abe might select a car—say, `Car39`—to rent. However, before he makes such a selection, there is no car about which we can say Abe intends to rent *that car*.

In addition to frequently being only partially specified, the plans and activities of different people are frequently interdependent, thus motivating people to coordinate their future-directed planning activity [9]. As a result, they must frequently negotiate about objects, such as the car mentioned above, that may be only partially specified. For example, suppose that Abe decides to let the rental agent Zoe select the car that he is going to rent. Abe must update his intention to reflect this delegation of parameter-binding authority; he now intends to rent *whatever* car Zoe selects for him. In this paper, intentions concerning this kind of partially specified content are called intentions with "whatever" content. For

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 229–241, 2006.

computer agents to participate effectively in these kinds of commonplace, multi-agent planning and coordination scenarios, they must be able to represent and reason about intentions with "whatever" content.

An intention is *satisfied* [17] if it successfully motivates the intending agent to eventually do the action or achieve the state of affairs stipulated in its content. Whereas Abe's original intention "to rent a car" might be satisfied by his renting any of perhaps a hundred different cars, his updated intention "to rent whatever car Zoe selects" can only be satisfied by his renting whatever car Zoe happens to eventually select for him. Thus, the satisfaction of Abe's updated intention depends on how its "whatever" content is eventually determined—by Zoe.

This paper presents an approach to modeling intentions with "whatever" content that is based on public names (or identifiers) that agents mutually agree to use. For example, Abe and Zoe might agree to use the name `C1` to refer to the *context* of their rental-car interaction.[1] Similarly, they might agree to use the name `Car` to represent whatever car Zoe eventually selects for him. In this paper, `C1` is called a *social context* and `CAR` is called a parameter within a social context (or, a *social parameter*). Linking parameters to social contexts helps to disambiguate scenarios in which different contexts might have identically named parameters. For example, Zoe might be selecting rental cars for several people in different contexts.

The first part of this paper formally models the processes whereby agents assign names to social contexts and parameters associated with those contexts. The assignment of names is established by group declarations—that is, declarations attributable to groups. Agents also use group declarations to bind parameters to values and to delegate to other agents the authority for binding parameters. In this paper, a group declaration is modeled as an abstract event that "happens" when a group of agents make a decision, as a group, to make such a declaration. The formal definitions are given in terms of the existing GDMM framework for formally specifying group decision-making mechanisms [11, 12] that is recast, in this paper, in terms of the event calculus [14]. The event calculus facilitates the expression of axioms governing propositions (e.g., those established by declarations) that hold only over certain temporal intervals.

The second part of the paper addresses the syntax and satisfaction conditions for intentions with "whatever" content. It is formulated in terms of the *stit theory* of Belnap et al. [1]. The content of intentions is augmented to include not just a goal that the intending agent is committed to achieving, but also a proposition that the agent refrains from determining (e.g., Abe might intend to rent whatever car Zoe selects, while refraining from determining which car she selects).

The rest of this paper is organized as follows. Section 2 recasts the pre-existing GDMM framework in terms of the event calculus. Section 3 models group declarations that agents can use to manage social contexts and social parameters. Section 4 presents the syntax and satisfaction conditions for intentions with "whatever" content. Section 5 discusses related work and presents conclusions.

[1] Grosz [8] highlights the importance of context in collaborative, multi-agent scenarios.

2 The GDMM Framework in the Event Calculus

The GDMM framework [11, 12] is a framework for formally specifying group decision-making mechanisms (GDMMs). In that framework, a GDMM (or interaction protocol) is defined in terms of declarative speech-acts and the *incremental accumulation of authority.* In a typical GDMM, each agent might be authorized (by the group) to initiate a GDMM instance (i.e., a run of a protocol) by making an appropriate declaration. Such a declaration authorizes other agents to make further declarations, thereby establishing facts that in certain combinations authorize other agents to make still more declarations, and so on, until, in successful instances, some agent is eventually authorized to declare, on behalf of the group, that they have made a decision. That final, authorized declaration establishes the group decision as a mutually believed fact among the group members—an example of what Searle calls *institutional facts* [18].

The GDMM framework was originally presented using a dynamic, deontic, temporal logic that enabled various properties to be formally proven. However, that logic can be somewhat cumbersome when dealing with propositions—like those established or terminated by declarative speech-acts—that hold only over certain temporal intervals. Thus, this section recasts the original GDMM framework in terms of the event calculus [14]. In the recast framework, declarative speech-acts are represented by events; authorization conditions by fluents (i.e., reified propositions); and an authorized speech-act establishing the truth of its propositional content is represented by a speech-act event initiating an appropriate fluent. In addition, protocol-specific axioms specify a protocol's method of incrementally accumulating authority. For example, an axiom might stipulate that certain combinations of fluents authorize certain speech-acts; or that certain speech-acts initiate or terminate certain fluents. The recast GDMM framework is demonstrated on a sample *propose-accept-reject* (PAR) protocol [11, 12]; the formal definitions were tested using Shanahan's abductive event-calculus planner [19].

2.1 A Quick Summary of the Event Calculus

The event calculus [14] is based on *events, fluents* and *time-points.* Events include actions such as buying a book or making a declarative speech-act. Fluents are reified propositional terms that are *initiated* or *terminated* by events. Time-points mark the occurrence of events and the initiation or termination of fluents. The most important event-calculus predicates are listed in Fig. 1. In the figure, `E` represents an arbitrary event, `F` an arbitrary fluent, and `T`, $\mathtt{T}_1$ and $\mathtt{T}_2$ arbitrary time-points. The predicates are governed by axioms such as SC1 and SC2 listed below. SC1 states that if `F` holds at time `0` and is not subsequently clipped, then it continues to hold. SC2 states that if `F` is initiated by `E` at time $\mathtt{T}_1$, and is not clipped between $\mathtt{T}_1$ and some later time $\mathtt{T}_2$, then `F` continues to hold at $\mathtt{T}_2$.

(SC1) $\mathtt{[initially(F)} \wedge \neg\mathtt{clip(0,F,T)]} \Rightarrow \mathtt{holds(F,T)}$

(SC2) $\mathtt{[happens(E,T_1)} \wedge \mathtt{inits(E,F,T_1)} \wedge (\mathtt{T_1}<\mathtt{T_2}) \wedge \neg\mathtt{clip(T_1,F,T_2)]} \Rightarrow \mathtt{holds(F,T_2)}$

`happens(E,T)`	– event `E` happens at time `T`.
`holds(F,T)`	– fluent `F` holds at time `T`.
`initially(F)`	– fluent `F` holds from time `0` onward (until clipped).
`inits(E,F,T)`	– if `E` happens at time `T`, then `F` is initiated at time `T`.
`terms(E,F,T)`	– if `E` happens at time `T`, then `F` is terminated at time `T`.
`clip(`T_1`,F,`T_2`)`	– `F` is terminated sometime between times T_1 and T_2.
`declip(`T_1`,F,`T_2`)`	– `F` is initiated sometime between times T_1 and T_2.

Fig. 1. Predicates used in the event calculus

2.2 Declarative Speech-Acts and Authorization Conditions

A declarative speech-act is represented by an event term of the form

`decl(G,Hs,Content)`, abbreviated as δ

where `G` is an agent (the speaker), `Hs` is a group of agents (the hearers), and `Content` is a fluent representing the propositional content of the declaration. Authorization for such a speech-act is represented by a fluent of the form

`auth(Gr, decl(G,Hs,Content))`, abbreviated as `auth(Gr,`δ`)`

That is, agent `G` is authorized by the group `Gr` to make a declaration with content `Content` to a set of hearers `Hs` $\subseteq$ `Gr`. Axiom A1, below, is the main axiom governing declarative speech-acts. It stipulates that any suitably authorized declaration establishes the truth of its propositional content.[2]

$$\texttt{holds(auth(Gr,}\delta\texttt{),T)} \;\Rightarrow\; \texttt{inits(}\delta\texttt{,Content,T)} \qquad \text{(A1)}$$

2.3 The PAR Protocol in the Recast Framework

In the sample PAR protocol [11, 12], agents use declarative speech-acts to make proposals, vote on proposals, and announce group decisions. Such speech-acts are represented by the event terms listed in Fig. 2. Axioms governing the incremental accumulation of authority in the PAR protocol are listed in Fig. 3.

Axiom E1 stipulates that each agent `G` in a group `Gr` is initially authorized to make proposals to `Gr`. In this axiom, the authorizing group and the set of hearers are the same (`Gr`); the content of the declarative speech-act, $\delta_{\texttt{MP}}$, is the fluent, `madeProp(G,Gr,Prop)`; and the predicate `proposable(Prop)` is used to restrict the range of allowable content for proposals. Axioms SC1, A1 and E1 together entail that any agent `G` is authorized to make a PAR proposal to any group `Gr` as long as: (1) `G` is a member of `Gr`; (2) the content of the proposal is "proposable"; and (3) the agent's authorization to make such proposals has not been "clipped" by an intervening event (e.g., a group decision to revoke it). To make a proposal, `G` simply declares that it has done so, whereupon (by Axiom A1) a fluent of the form, `madeProp(G,Gr,Prop)`, is initiated. Axiom E2

[2] In all axioms in this paper, all free variables are implicitly universally quantified.

- `decl(G,Gr,made(G,Gr,Prop))`, abbreviated $\delta_{\mathtt{MP}}$:

  "`G` declares to the group `Gr` that it has made a proposal `Prop`."

- `decl(G2,{G,G2},voted(G2,G,Gr,Prop,Vote))`, abbreviated $\delta_{\mathtt{V}}$:

  "`G2` declares to `G` that it has made a vote concerning the proposal `Prop`, where `Vote` $\in$ {`accept`,`reject`}."

- `decl(G2,{G,G2},voted(G2,G,Gr,Prop,accept))`, abbreviated $\delta_{\mathtt{VA}}$:

  "`G2` declares to `G` that it has voted to accept the proposal `Prop`."

- `decl(G,Gr,grAcc(Gr,Prop))`, abbreviated $\delta_{\mathtt{GA}}$:

  "`G` declares to the group `Gr` that they have accepted `Prop`."

Fig. 2. Event terms representing declarations in the PAR protocol

$$[(\mathtt{G} \in \mathtt{Gr}) \wedge \mathtt{proposable(Prop)}] \Rightarrow \mathtt{initially(auth(Gr,}\delta_{\mathtt{MP}}\mathtt{))} \quad \text{(E1)}$$

$$[(\mathtt{G2} \in \mathtt{Gr}) \wedge (\mathtt{G2} \neq \mathtt{G}) \wedge \mathtt{holds(auth(Gr,}\delta_{\mathtt{MP}}\mathtt{),T)}] \Rightarrow \mathtt{inits(}\delta_{\mathtt{MP}}\mathtt{,auth(Gr,}\delta_{\mathtt{V}}\mathtt{),T)} \quad \text{(E2)}$$

$$\mathtt{holds(auth(Gr,}\delta_{\mathtt{MP}}\mathtt{),T)} \Rightarrow \mathtt{inits(}\delta_{\mathtt{MP}}\mathtt{,accepters(G,Gr,Prop,}\emptyset\mathtt{),T)} \quad \text{(E3)}$$

$$[\mathtt{holds(accepters(G,Gr,Prop,Others),T)} \wedge \mathtt{holds(auth(Gr,}\delta_{\mathtt{VA}}\mathtt{),T)}] \Rightarrow \mathtt{inits(}\delta_{\mathtt{VA}}\mathtt{,accepters(G,Gr,Prop,\{G2\} \cup Others),T)} \quad \text{(E4)}$$

$$[(\mathtt{Gr} = \{\mathtt{G}\} \cup \mathtt{Others}) \wedge \mathtt{holds(accepters(G,Gr,Prop,Others),T)}] \Rightarrow \mathtt{inits(}\delta_{\mathtt{GA}}\mathtt{,grAcc(Gr,Prop),T)} \quad \text{(E5)}$$

Fig. 3. Axioms governing incremental accumulation of authority in the PAR protocol

stipulates that the making of a proposal authorizes each of the other agents in the group to vote on it—either to accept or reject it.

In the PAR protocol, if every agent votes to accept a proposal, then the originator of that proposal—here, `G`—becomes authorized to declare, on behalf of the group, that they have made a decision—namely, to accept the proposal. `G`'s authorization to make such a declaration is accumulated incrementally, over time, as each agent declares its own acceptance of the proposal, as governed by axioms E3, E4 and E5. Axiom E3 stipulates that the making of a proposal initiates a fluent of the form, `accepters(G,Gr,Prop,`$\emptyset$`)`, representing that no one in the group has (yet) voted to accept `G`'s proposal. Axiom E4 stipulates that an agent `G2`'s authorized vote to accept a proposal incrementally updates the list of "accepters" (by adding `G2`). Axiom E5 stipulates that if all of the other agents (`Others`) have voted to accept `G`'s proposal, then `G` becomes authorized to declare

on behalf of the group that they have accepted the proposal. If so authorized, then, by Axiom A1, G's declaration establishes the fluent, `grAcc(Gr,Prop)`.

These axioms were encoded in Prolog and fed as input to Shanahan's abductive event-calculus planner [19] which was able to come up with valid sequences of speech-acts to yield various group decisions under the PAR protocol. For example, the following sequence was generated in response to a query about how a group of agents {g,h,i} might decide to accept a proposal `prop`:

```
happens(decl(g,[g,h,i],madeProp(g,[g,h,i],prop)),t51)
happens(decl(i,[g,i],voted(i,g,[g,h,i],prop,accept)),t52)
happens(decl(h,[g,h],voted(h,g,[g,h,i],prop,accept)),t50)
happens(decl(g,[g,h,i],grAcc([g,h,i],prop)),t48)
```

where the time-points were subject to the constraints: `t51 < t52 < t50 < t48`. Although the PAR protocol is quite simple, the same approach can be used to specify arbitrarily complex protocols based on declarative speech-acts and the incremental accumulation of authority in the GDMM framework.

3 Group Declarations for Contexts and Parameters

For this paper, the most important types of group decisions are those that establish names for social contexts or parameters within those contexts, and those that bind parameters to values or delegate authority for binding parameters. Such decisions can be made using any GDMM; thus, it is convenient to abstract away the GDMM used to generate the group decision and focus instead on the proposition established by that decision. Toward that end, this section uses the GDMM framework to model *group declarations*—that is, declarations attributable to groups of agents. It then addresses the use of group declarations to manage social contexts and parameters within those contexts.

In the single-agent case, an agent might establish the binding of a parameter thusly: "I hereby declare that the parameter `P` in the context `C` shall be bound to the value `67`." By analogy, a group can establish such facts by making *group declarations.* In particular, a group declaration, if suitably authorized, has the power to establish the truth of its propositional content. However, a group declaration is not uttered; instead it "happens", by convention, when, at the successful culmination of a GDMM instance, one of the agents announces, on behalf of the group, that they have decided to make a declaration. For example, at the end of a complex group decision-making procedure, a member of Congress might announce that the Congress has decided, as a group, to declare war against some other country. In such a case, the declaration of war is attributed to the Congress as a whole, not to the individual making the announcement.

It is important to distinguish two kinds of authorization associated with group declarations: internal and external. Internal authorization is that which is incrementally accumulated during a run of whatever GDMM is being used to generate the group declaration. For example, the member of Congress announcing their decision to declare war must be suitably authorized by the Congress; otherwise,

no group declaration takes place. In contrast, the external authorization for a group declaration is independent of the GDMM used to generate it. Instead, external authorization, which frequently comes from outside the group making the declaration, is that which gives the group's declaration the power to establish the truth of its propositional content. In other words, the external authorization for group declarations is analogous to the authorization conditions for single-agent declarations. For example, a declaration of war by the Congress has the power to establish a state of war only because the people, via the Constitution, have authorized Congress to make such declarations.

A group declaration is represented by an event term of the form

$$\texttt{grDecl(Gr,Hs,Content)}, \text{ abbreviated as } \Delta$$

where `Gr` represents the group making the declaration, `Hs` represents the set of hearers, and `Content` is a fluent representing the content of the declaration.[3] A group declaration is not an action that is directly "executable" by the group. Instead, a group declaration "happens", by convention, when a group makes a group decision whose content has the form

$$\texttt{done(grDecl(Gr,Hs,Content))}, \text{ abbreviated as } \texttt{done}(\Delta).$$

In such a case, the group decision initiates (e.g., by Axiom E5) a fluent

$$\texttt{grAcc(Gr,done}(\Delta)\texttt{)}$$

which can be glossed as "`Gr` has decided to make a group declaration to `Hs` that `Content` holds." Axiom A2, below, stipulates that such a fluent "counts as" [18] a group having made the indicated group declaration.

$$\texttt{holds(grAcc(Gr,done}(\Delta)\texttt{),T)} \Rightarrow \texttt{happens}(\Delta,\texttt{T}) \qquad \text{(A2)}$$

Then, in direct analogy with Axiom A1, Axiom A3 below stipulates that an authorized group declaration establishes the truth of its propositional content.

$$\texttt{holds(auth(AuthGr,}\Delta\texttt{),T)} \Rightarrow \texttt{inits}(\Delta,\texttt{Content,T}) \qquad \text{(A3)}$$

In this axiom, `AuthGr` represents the (external) authorizing group.

The rest of this section focuses on how agents can use group declarations to establish names for social contexts and social parameters, and to bind such parameters or delegate the authority for binding them. In what follows, all contexts and parameters are presumed to be under the sole control of the group `Gr`—that is, `Gr` is its own "external" authorizing group. In addition, the set of hearers is presumed to be the entire group. Thus, `Gr` = `AuthGr` = `Hs`. In addition, for convenience, repeated arguments are omitted. Thus, a group declaration is represented by a term of the form, `grDecl(Gr,Content)`—abbreviated as Δ—and the corresponding authorization condition is represented by a fluent of the form,

[3] Δ denotes a group declaration; δ denotes a single-agent declaration.

- `grDecl(Gr,sContext(Gr,C))`, abbreviated $\Delta_{\mathtt{C}}$:
  "Group `Gr` declares a new social context named `C`."
- `grDecl(Gr,sParam(Gr,C,P))`, abbreviated $\Delta_{\mathtt{P}}$:
  "Group `Gr` declares a new parameter named `P` associated with context `C`."
- `grDecl(Gr,sBindParam(Gr,C,P,V))`, abbreviated $\Delta_{\mathtt{BP}}$:
  "Group `Gr` declares that parameter `P` in context `C` is bound to value `V`."
- `grDecl(Gr,sDelegParam(G,Gr,C,P))`, abbreviated $\Delta_{\mathtt{DP}}$:
  "Group `Gr` declares that the authority for binding the parameter `P` in the context `C` is delegated to the agent `G`."
- `decl(G,Gr,sBindParam(Gr,C,P,V))`, abbreviated $\delta_{\mathtt{BP}}$:
  "Agent `G` declares that parameter `P` in context `C` is bound to value `V`."

Fig. 4. Event terms representing declarations for social contexts and parameters

$$\mathtt{initially(auth}(\Delta_{\mathtt{C}}\mathtt{))} \tag{E6}$$

$$\mathtt{holds(auth}(\Delta_{\mathtt{C}}\mathtt{),T)} \Rightarrow \mathtt{inits}(\Delta_{\mathtt{C}}\mathtt{,auth}(\Delta_{\mathtt{P}}\mathtt{),T}) \tag{E7}$$

$$\mathtt{holds(auth}(\Delta_{\mathtt{P}}\mathtt{),T)} \Rightarrow \mathtt{inits}(\Delta_{\mathtt{P}}\mathtt{,auth}(\Delta_{\mathtt{BP}}\mathtt{),T}) \tag{E8}$$

$$\mathtt{holds(auth}(\Delta_{\mathtt{P}}\mathtt{),T)} \Rightarrow \mathtt{inits}(\Delta_{\mathtt{P}}\mathtt{,auth}(\Delta_{\mathtt{DP}}\mathtt{),T}) \tag{E9}$$

$$\mathtt{holds(auth}(\Delta_{\mathtt{DP}}\mathtt{),T)} \Rightarrow \mathtt{inits}(\Delta_{\mathtt{DP}}\mathtt{,auth(Gr,}\delta_{\mathtt{BP}}\mathtt{),T}) \tag{E10}$$

Fig. 5. Axioms pertaining to the declarations in Fig. 4

`auth(grDecl(Gr,Content))`—abbreviated as $\mathtt{auth}(\Delta)$. Fig. 4 lists the types of group declarations (and one single-agent declaration) to be discussed. Fig. 5 lists the axioms pertaining to the declarations in Fig. 4.

A group `Gr` creates a social context named `C` by making a declaration of the form $\Delta_{\mathtt{C}}$ in Fig. 4. By Axiom E6 in Fig. 5, any group is initially authorized to create arbitrary social contexts for itself. Thus, by Axiom A3, a group declaration, $\Delta_{\mathtt{C}}$, establishes a fluent of the form, `sContext(Gr,C)`.

A group `Gr` creates a social parameter named `P`, linked to a social context `C`, by making a declaration of the form $\Delta_{\mathtt{P}}$ in Fig. 4. By Axiom E7, a group's creation of a social context `(Gr,C)` automatically authorizes that group to create social parameters within that context. Similarly, a group's creation of a social parameter `(Gr,C,P)` automatically authorizes that group to bind that parameter to some value (Axiom E8) or *delegate* the authority for binding that parameter to some other agent (Axiom E9).

A group `Gr` binds a parameter `P` in the context `C` to the value `V` by making a declaration of the form $\Delta_{\mathtt{BP}}$ in Fig. 4. If suitably authorized, then, by Axiom A3,

such a declaration would initiate a fluent of the form `sBindParam(Gr,C,P,V)`.[4] Alternatively, a group might decide to delegate the authority for binding that parameter to some agent, say `G`, by making a declaration of the form $\Delta_{\texttt{DP}}$ in Fig. 4. By Axiom E10, such a declaration authorizes `G` to bind `P` to *any* value `V` by making a declaration of the form $\delta_{\texttt{BP}}$ in Fig. 4.[5] Should `G` make such a declaration, it would, by Axiom A1, initiate the fluent, `sBindParam(Gr,C,P,V)`. Thus, whether the group `Gr` binds `P` directly using a group decision or indirectly via the delegate `G`, the end result is the initiation of the same fluent: `sBindParam(Gr,C,P,V)`.

Example. Abe (`a`) intends to rent whatever car Zoe (`z`) selects for him. In this case, they make group declarations that initiate the following fluents:

`sContext({a,z},c)`	– `c` is a social context for them.
`sParam({a,z},c,p)`	– `p` is a social parameter for them in that context.
`sDelegParam(z,{a,z},c,p)`	– they have delegated the binding of `p` to Zoe.

By Axiom E10, the last fluent in the above list initiates the following fluent, which represents that Zoe is authorized to bind `p` to *any* value `V`.

`auth({a,z},decl(z,{a,z},sBindParam({a,z},c,p,V)))`

4 Intentions with "Whatever" Content

This section presents a novel representation for intentions with "whatever" content. The satisfaction conditions for such intentions clearly distinguish conditions that the intending agent seeks to achieve and those that it actively refrains from determining. The representation is expressed in terms of the "sees to it that" (*stit*) operator defined by Belnap et al. [1], which is briefly summarized below. Afterward, intentions with "whatever" content and their satisfaction conditions are defined and the definitions are illustrated with examples.

4.1 Seeing to It That

Belnap et al. [1] present a theory of "agents and choices in branching time" within which they formally define a modal "sees to it that" (*stit*) operator, which they use to represent *agentive* expressions. They argue that "[a proposition] Q is agentive for [an agent] α just in case Q may be usefully paraphrased as [α *stit*: Q]." For example, the sentence, "Abe sees to it that a car is rented", is agentive for Abe since it has the form, [A *stit*: ϕ], where A denotes Abe and ϕ denotes the proposition, "a car is rented".

[4] The binding of a parameter should also terminate that group's authority to subsequently bind that same parameter or to delegate the binding of that parameter. For space reasons, providing such axioms is left to the reader.

[5] A decision to delegate parameter-binding authority to an agent `G` would also entail an obligation on `G` to eventually bind that parameter; however, this paper focuses on authorization conditions, not obligations. Grosz and Hunsberger [9] address some of the obligations entailed by various kinds of group decision.

The semantics of the *stit* operator stipulate that $[\alpha\ stit{:}\ Q]$ holds now if and only if: (1) Q holds now due to a prior choice (or sequence of choices) made by α; and (2) α's choice was real in the sense that some other choice open to α might have resulted in Q *not* holding. For example, I might see to it that my office gets cold by opening a window, where my alternative, leaving the window closed, might have resulted in my office staying warm.

In their "Restricted Complement Thesis", Belnap et al. argue that "a variety of constructions concerned with agents and agency—including deontic statements, imperatives, and statements of intention, among others—must take agentives as *their* complements." For example, the expression, $Int{:}\ [\alpha\ stit{:}\ Q]$, would represent that the agent α intends to see to it that the proposition Q holds.

Belnap et al. define *active refraining* in terms of the *stit* operator, as follows:

$$refrain(\alpha, \psi) \ \equiv\ [\alpha\ stit{:}\ \neg[\alpha\ stit{:}\ \psi]]$$

That is, an agent α actively refrains from bringing about ψ if α sees to it that α does *not* see to it that ψ holds. In other words, some choice made by α, perhaps even a choice to do nothing, must guarantee that $\neg\psi$ remains an option—at least insofar as α's choices are concerned. Of course, the choice(s) of some other agent(s) might nonetheless establish ψ, despite α's refraining.

The following abbreviation will be useful later in this section:

$$refrain(\alpha, \pm\psi) \ \equiv\ refrain(\alpha, \psi) \wedge refrain(\alpha, \neg\psi)$$

That is, α *both* refrains from ψ and refrains from $\neg\psi$. Such an expression holds if some prior choice(s) made by α guarantee that both ψ and $\neg\psi$ remain options.

4.2 Intentions with "Whatever" Content

Definition 1. *An* intention with "whatever" content *is an expression of the form:* $Int_w(G, x, \psi(x), \phi(x))$*, where* G *is a term,* x *is a variable, and* $\psi(x)$ *and* $\phi(x)$ *are arbitrary propositions that may contain free occurrences of* x*.*

The intended interpretation of such an expression is that the agent G intends to see to it that the proposition $\phi(x)$ holds for whatever (unique) value of x the proposition $\psi(x)$ holds, while refraining from determining the choice of x for which $\psi(x)$ holds. The formal interpretation is given in Definition 2.

Definition 2. $Int_w(G, x, \psi(x), \phi(x))$ *is* satisfied *if:*

(1) $(\forall x) refrain(G, \pm\psi(x))$ *holds; and*
(2) **if** *there is a unique object* d *in the semantic domain for which the expression* $\psi(c_d)$ *holds, where* c_d *is a constant term denoting* d *and* $\psi(c_d)$ *is obtained from* $\psi(x)$ *by substituting* c_d *for each occurrence of the free variable* x *in* $\psi(x)$*,* **then** *the expression,* $stit(G, \phi(c_d))$*, also holds.*

Condition 1 stipulates that G should refrain from determining $\psi(x)$ or $\neg\psi(x)$ for any x—that is, choices made by G should guarantee that both $\psi(x)$ and $\neg\psi(x)$ remain options for any x. Condition 2 stipulates that *if* there is a unique value of x for which the expression $\psi(x)$ holds, then the agent G must see to it that the expression $\phi(x)$ holds *for that same value of* x.

Example. Recall Abe's intention to rent whatever car Zoe selects for him. Suppose that Abe (A) and Zoe (Z) have already established a name C for a social context and a parameter P for the car. Suppose further that they have delegated the binding of P to Zoe. Abe's intention can be represented by an intention with "whatever" content where:

$\psi(x) \equiv sBindParam(\{A, Z\}, C, P, x)$; and
$\phi(x) \equiv Rents(A, x)$.

According to Definition 2, for Abe's intention to be satisfied, he must refrain from both $\psi(x)$ and $\neg\psi(x)$ for all x. In other words, *his* choices must not constrain the possible values for the parameter P. In addition, if the condition $\psi(x)$ holds for some *unique* value of x (e.g., should Zoe declare P to have the value `Car39`), then Abe must see to it that $\phi(x)$ holds for that value of x (e.g., that he rents `Car39`). In short, if Abe refrains from determining which car is selected, and Zoe selects a unique car, then Abe must see to it that he rents *that* car; however, if no such car is selected, or more than one is selected, then Abe's intention is trivially satisfied.

Intentions with "whatever" content can be defined with multiple partially-determined objects by substituting $(x_1, x_2, \ldots, x_n)$ for x, $(\forall x_1, x_2, \ldots, x_n)$ for $(\forall x)$, and $\psi_1(x_1) \wedge \ldots \wedge \psi_n(x_n)$ for $\psi(x)$ in Definition 1; and, in addition, substituting $(d_1, \ldots, d_n)$ for d and $(c_{d_1}, \ldots, c_{d_n})$ for c in Definition 2. For example, Abe's intention to hammer in a nail using whatever hammer Zoe specifies and whatever nail Yao (Y) specifies could be represented by

$$Int_w(A, (x_1, x_2), \psi_1(x_1) \wedge \psi_2(x_2), \phi(x_1, x_2))$$

where $\psi_1(x_1) \equiv sBindParam(Z, C, \texttt{hamr}, x_1)$;
$\psi_2(x_2) \equiv sBindParam(Y, C, \texttt{nail}, x_2)$; and
$\phi(x_1, x_2) \equiv Pounds(A, x_1, x_2)$.

5 Related Work and Conclusions

Several researchers are actively investigating the use of the event calculus to model interaction protocols in normative settings. For example, Yolum and Singh [20] use it to model protocols as *commitment machines*. In that work, agents use various kinds of speech-acts to adopt or modify (one-on-one) social commitments. Pitt et al. [16] use the event calculus to formalize a complex voting protocol for general-purpose decision-making in virtual organizations. In their work, events such as proposing, voting, and so forth initiate or terminate various powers (authorizations), permissions and obligations. These approaches are complementary to the approach taken in this paper where the GDMM framework is based solely on declarative speech-acts and the incremental accumulation of authority, and is used to model declarations attributable to groups.

Other researchers are investigating contracts for multi-agent systems in terms of normative concepts. For example, Farrell et al. [6] define a contract language in

terms of obligation, power and permission and present an algorithm for tracking the normative state of a contract over its entire life-cycle. And Boella and van der Torre [2] view contracts as legal institutions based on Searle's *construction of social reality* [18]. In their work, mental states such as beliefs, desires and intentions are attributed not only to agents, but also to normative systems. In prior work, Grosz and Hunsberger [9] specify the obligations entailed by certain common types of group decision (e.g., binding a parameter, selecting a recipe for a complex task, or delegating a task) in the context of multi-agent coordination scenarios. Ongoing research is aimed at augmenting that work to include authorizations and permissions, as well as intentions with "whatever" content.

The most related work on delegation is that of Norman and Reed [15]. In their work, agents use imperative speech-acts to delegate tasks to other agents and to command others to refrain from further delegating those same tasks. They use a propositional logic and thus do not address intentions with "whatever" content, but they employ two *stit* operators, one for propositions and one for actions.

In the field of linguistics, Dekker and van Rooy [5] formally analyze so-called *Hob-Nob sentences* in which "a number of people ... have attitudes with a common focus, whether or not there actually is something at that focus" which is a broad category that seems to include intentions with "whatever" content.[6] In addition, Kamp and Reyle [13] use Discourse Representation Theory (DRT) to analyze sentences (or sequences of sentences) that include partially specified content in the form of indefinite noun phrases and pronouns that subsequently refer to that content—as in: "Every farmer who owns a donkey beats it" or "John owns a Porsche. It fascinates him." An investigation into the potential application of these methods to intentions with "whatever" content (i.e., partially specified content to which agents need to refer as they coordinate with others), is the subject of ongoing research.

The research presented in this paper is part of a long-term project aimed at developing collaboration-capable computer agents [9]. Current work is focused on providing a comprehensive logical foundation for intentions with "whatever" content that can accommodate other types of partially specified content (e.g., Bea intends to drive whatever car Abe rents) as well as the obligations that are entailed by group decisions in multi-agent planning and coordination scenarios.

References

1. Nuel Belnap, Michael Perloff, and Ming Xu. *Facing the Future.* Oxford University Press, 2001.
2. Guido Boella and Leendert van der Torre. Contracts as legal institutions in organizations of autonomous agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems.* ACM, 2004.
3. Michael E. Bratman. *Intentions, Plans, and Practical Reason.* Harvard University Press, Cambridge, MA, 1997.
4. Michael E. Bratman. *Faces of Intention: Selected Essays on Intention and Agency.* Cambridge University Press, 1999.

[6] The quotation is from Geach [7], cited in Dekker and Rooy [5].

5. Paul Dekker and Robert van Rooy. Intentional identity and information exchange. In R. Cooper and T. Gamkrelidze, editors, *Second Tbilisi Symposium on Language, Logic and Computation*, 1997.
6. Andrew D. H. Farrell, Marek Sergot, Mathias Salle, and Claudio Bartolini. Using the event calculus for tracking the normative state of contracts. *International Journal of Cooperative Information Systems*, 14(2–3), June–September 2005.
7. P. Geach. Intentional identity. *Journal of Philosophy*, 74:309–44, 1967.
8. Barbara J. Grosz. The contexts of collaboration. In E. Sosa K. Korta and X. Arrazola, editors, *Cognition, Agency and Rationality*, pages 175–188. Kluwer Press, Dordrecht, 1999.
9. Barbara J. Grosz and Luke Hunsberger. The dynamics of intention in collaborative activity. *Journal of Cognitive Systems Research*, 7:259–272, 2006.
10. Barbara J. Grosz and Sarit Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86:269–357, 1996.
11. Luke Hunsberger. *Group Decision Making and Temporal Reasoning*. PhD thesis, Harvard University, 2002. Available as Harvard Technical Report TR-05-02.
12. Luke Hunsberger. A framework for specifying group decision-making mechanisms (poster). In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-2005)*. ACM Press, 2005.
13. Hans Kamp and Uwe Reyle. *From Discourse to Logic*, volume 42 of *Studies in Linguistics and Philosophy*. Kluwer Academic Publishers, 1993.
14. R.A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.
15. Timothy J. Norman and Chris Reed. A model of delegation for multi-agent systems. In M. Fisher M. d'Inverno, M. M. Luck and C. Preist, editors, *Foundations and Applications of Multi-Agent Systems*, volume 2403 of *Lecture Notes in Artificial Intelligence*, pages 185–204. Springer-Verlag, 2002.
16. Jeremy Pitt, Lloyd Kamara, Marek Sergot, and Alexander Artikis. Formalization of a voting protocol for virtual organizations. In *Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2005)*, pages 373–380. ACM, 2005.
17. J. R. Searle. *Speech acts: An essay in the philosophy of language*. Cambridge University Press, Cambridge, UK, 1969.
18. J.R. Searle. *The Construction of Social Reality*. Allen Lane, London, 1995.
19. Murray Shanahan. *The Event Calculus Explained*, volume 1600 of *Lecture Notes in Computer Science*. 1999.
20. Pinar Yolum and Munindar P. Singh. Reasoning about commitments in the event calculus: An approach for specifying and executing protocols. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 2003.

Automatic Deductive Synthesis of Lisp Programs in the System ALISA

Yulia Korukhova*

Lomonosov Moscow State University, Computational Mathematics and Cybernetics Faculty
119992 Russia, Moscow, GSP-2, Leninskie Gory, MSU
yulia@cs.msu.su

Abstract. The work deals with deductive synthesis of functional programs. During this synthesis formal specification of a program is taken as a mathematical existence theorem and while proving it, we derive a program and prove that this program corresponds to given specification. Our method of synthesis is based on the deductive tableau method, that allows to derive three basic constructions of a functional program. But the full search of possible proof attempts in the deductive tableau induces a very large search space; the proof is needed to be guided. For using this method in the automatic mode additional heuristics are required. Some of such heuristics are proposed in this work. They consist in proof planning by using rippling and in the use of sorted logic with type hierarchy that reduces search space and blocks some branches of proof, corresponding to synthesis of incorrect functions. The proposed techniques are implemented in the system ALISA[1] and used for automatic synthesis of several functions on Lisp language.

1 Introduction

The problem of automatic program synthesis was treated since 1960s. It consists in construction of a program code from some description of desired program. Two approaches were distinguished: inductive synthesis (that assumes program construction from traces of its work) and deductive synthesis (that assumes program derivation from a specification that describes a relationship between input and output of a desired program). Deductive approach is particularly interesting because it allows to construct programs with proving its correctness with respect to given specification, so the constructed program does not require verification. If such synthesis is performed automatically and the correctness of chosen method of synthesis is proved, the problem of writing a program is reduced to writing a correct specification. Good specification is readable and clear for a user, because it describes the goal of a program and may not present a particular method for computation the required solution. Such specification is taken as a mathematical

* This work is a part of the author's PhD project, which was supervised by Assoc. Prof. Dr. V.N. Pilschikov. The work has been partially supported by RFBR grant 05-01-00948a.

[1] ALISA - **A**utomatic **LI**sp **S**ynthesizer.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 242 – 252, 2006.

existence theorem and we prove the existence of an object that satisfies the specified conditions. This proof is required to be constructive and each step of a proof corresponds to a step of synthesis.

There were several methods and realizations of deductive synthesis for some class of programs. First systems, such as Heuristic Compiler [17], PROW [12] were used to derive a sequence of operators and condition. For derivation of cycle or recursion special inference rule, based on using of mathematical induction in the proof was required. Programs containing cycle operator were derived in the system PRIZ [19], using the synthesis based on computational model, but the cyclic structures of a program were incorporated in this model as many other facts about the domain.

In [13],[14] the deductive tableau method was proposed. It is appropriate for synthesis of functional programs from formal logic-based specifications. The method gives the rules that allow to derive three basic constructions of functional program: function application, conditional term and recursion, but order of these rules application is not specified. The deductive tableau method was used for interactive synthesis. This synthesis was guided manually by user, as described in [8]. Also automated synthesis was implemented by embedding the deductive tableau method as a higher order logic rules into ISABELLE [15], where proofs were partially automated by tactics, written by user.

We took the deductive tableau method as a basic method for the automatic synthesis in the system ALISA. But when the fully automated synthesis in the deductive tableau (with the full search of possible proof attempts) is performed, the induced search space is very large even for simple problems. To reduce it, we can either use some additional heuristics that decrease the number of rules applicable in every step of the proof, or plan the proof before application of the deductive rules. In our system for the first goal we use sorted logic with type hierarchy; it allows to reduce the search space and also to avoid the synthesis of incorrect function branches. Second, we suggest here to use proof planning by using rippling heuristic [4], [7] for constructing the proof path corresponding to recursive branch synthesis.

The paper is organized as follows. Section 2 describes the synthesis in the deductive tableau. Section 3 precise the particular problems that should be resolved for performing synthesis fully automatically and describes the proposed solutions. Section 4 reports on practical results and presents some directions of future research. In the section 5 conclusions are drawn.

2 Synthesis in the Deductive Tableau

Deductive program synthesis begins with a specification which represents a relationship between input and output of a program. We use a specification language based on first order logic predicates and constructions. There are also specific predicates for lists and integers (such as `head(a)`, `tail(a)`, `atom(x)`, `number(x)`, etc.) For example, a function that calculates the value of integer square root for an integer number b has the following specification:

```
<sqrt(b)> <= for Number(b) find <z> such that          (1)
                if (b≥0) then (z²≤b) ∧ (b<(z+1)²)
```

After "for" there are type definitions for function parameters (b is an integer), z is the output variable and the expression after "such that" describes a relationship between inputs and outputs.

The specification (1) is taken as a mathematical existence theorem

$$(\forall b)(\exists z)(\ \text{if}\ (b \geq 0)\ \text{then}\ (z^2 \leq b) \wedge (b < (z+1)^2))$$

To synthesize an algorithm for computation of z we are trying to prove the theorem constructively. The structure of the proof determines the structure of the program we extract. In particular, a case analysis in the proof corresponds to the formation of a conditional term in the program; the use of principle of mathematical induction in the proof coincides with the appearance of recursion.

Suppose, we've found a term `t(b)` that meets the specified conditions. The program we produce is then

```
sqrt(b) = t(b)
```

For writing such a proof we use special framework called the deductive tableau. The tableau structure is shown in Table 1. A tableau corresponds to a first order sentence

$$\text{if}\ (A1 \wedge A2 \wedge \ldots \wedge An)\ \text{then}\ (G1 \vee G2 \vee \ldots \vee Gn)$$

where A1…An are assertions written in the tableau and G1...Gn are goals to be proved. Each row of a deductive tableau contains either an assertion or a goal. The table has one or more output columns, corresponding to outputs of functions. The goal functions appear in the output columns during synthesis. The output columns of one row are processed in the same manner.

The following properties are useful consequences of the above definition.

Duality. By removing an assertion (a goal) and adding its negation as a goal (an assertion) we obtain an equivalent tableau.
Variables. All free variables are assumed universally quantified in assertions and existentially quantified in goals. Free variables in different rows are independent and can be renamed if necessary. Bound variables are replaced by skolem constants or functions, using skolemization procedure [9]. During the proof skolem functions can only be unified with the same skolem function.
Instance. Let λ be a substitution and G a goal in a row with output s. Then adding or deleting a row with goal $G\lambda$ and output $s\lambda$ will produce the equivalent tableau.
Output. A row with no output entry is equivalent to one whose output entry is a new variable, that doesn't occur free in this row.

The synthesis starts with adding the expression written in the specification as a new goal in a deductive tableau. For the specification (1) the goal if $(b \geq 0)$ then $(z^2 \leq b)$ and $(b<(z+1)^2)$ is added. Then the deduction rules are applied. The deduction rules add new rows to a tableau, preserving validity of the

logical expression associated with the tableau. Some examples of deduction rules are the following:

The splitting rule. If a row contains a goal `(G1 or G2)` it can be decomposed into two rows with the goal `G1` and the goal `G2`. The output entries are inherited by the generated rows.

The resolution rule. If a tableau contains rows 1 and 2 (see Table 1), the row 3 can be added. Logical simplifications (such as, for example, `A and true` → `true`) should be done in the new row.

Table 1. The deductive tableau.The resolution rule. Here 1,2 are rows with no free variables in common (we rename variables to achieve it if necessary), P,Q – quantifier free subexpressions, such that Pλ=Qλ. We replace all occurrences of Pλ in G1λ with false and all occurrences of Qλ in G2λ with true and add a new row 3 with the conditional output.

	Assertions	Goals	Output
1		G1 [P]	z
2		G2[Q]	t
3		G1 λ [false] ∧ G2λ [true]	if Pλ then tλ else zλ

Remarks about outputs: if one of two rows, say the 2nd, has no output entry, then the output of the generated row will be zλ, where z is the output for G1; if rows 1 and 2 both have no output entries then the generated row will also have no output.

The equality rule. If the tableau contains rows 1 and 2 (see Table 2), the row 3 can be added to this table. Remarks about outputs are the same as for the resolution rule.

The analogous equality rule can be applied for rows, containing equal terms. We take the same rule as in the Table 2, but consider P,Q,R to be terms and take the equality for terms P=R and Pλ=Qλ instead of equality ≡ for logical expressions.

Table 2. The equality rule. Here 1,2 are rows with no free variables in common (we rename variables to achieve it if necessary), P,Q,R – quantifier free subexpressions, such that Pλ≡Qλ. We replace all occurrences of (P≡R)λ in G1λ with false and some occurrences of Qλ in G2λ with Rλ and add a new row 3.

1		G1[P≡R]	z
2		G2[Q]	t
3		G1λ [false] ∧ G2λ<Rλ>	if (P≡R)λ then tλ else zλ

Using these rules and the tableau properties we can derive a non-recursive branch of the goal function. For the specification (1), we can derive a row N, see Table 3.

Table 3. The sqrt example: the nonrecursive branch of the function. Row 1 is obtained by splitting the initial goal, row 2 represents one of the system axioms, useful for this synthesis. Row N – the nonrecursive branch of the sqrt function (it means "if we prove that b=0, then sqrt(b)=0").

	Assertions	Goals	sqrt(b)
1		$(z^2 \leq b) \wedge (b<(z+1)^2)$	z
2	if x=0 then $x^2=0$		
N		b=0	0

The induction rule. For introducing a recursive function call to the table we use the induction rule. For the initial goal of the tableau an induction hypothesis can be added, see Table 4.

Table 4. The induction rule. Here f(a) is the goal function, the induction is hold on a. The 2nd row is a general variant of the induction hypothesis, row 3 represents an induction hypothesis for lists and for the relation tail(a) $<_{wf}$ a determined for nonempty lists. The 4th row shows an example of induction hypothesis for integers for the relation a-1 $<_{wf}$ a for nonnegative integers.

	Assertions	Goals	f(a)
1		Q[a,z]	z
2	if x $<_{wf}$ a then Q[x,f(x)]		
3	if not(a=NIL) then Q[tail(a),f(tail(a))]		
4	if (a>0) then Q[a-1,f(a-1)]		

We use the Noetherian induction scheme:

$$\frac{\forall x\ (\forall y\ \ y <_{wf} x \rightarrow F(y)\ ) \rightarrow F(x)}{\forall x\ F(x)}$$

where $<_{wf}$ is a well-founded relation (a relation that can not derive infinite decreasing sequences for the objects of a current sort). In the proof construction particular variants of this hypothesis are used, they are formed by instantiating a particular relation instead of $<_{wf}$, see examples in Table 4. There are several relations embedded in the system, they are stored as usual axioms (assertions) of a kind `if A then y` $<_{wf}$ `x`, where A is a logical expression, specifying a particular conditions for `x` and `y` to be connected by $<_{wf}$ relation. Usually while proving these conditions the nonrecursive branch is derived. Other relations can be easily added by user to the system as an axiom, this facility allows to construct new hypotheses.

The synthesis is complete when a row with the true goal is derived. In the output column of this row the computational algorithm for the goal function can be found.

3 Automation of Synthesis in the Deductive Tableau

The deductive tableau method gives a very good instrument for synthesis, but for performing synthesis automatically the order of application should be determined. If

we are simply trying to apply one rule after another for all existing rows of the table the number of possible proof attempts grows exponentially. Additional heuristics are needed to limit the number of proof attempts and make the proof search directed. We are going to look at some of them here.

3.1 The Use of Sorted Logic

We propose to extend our logic by assigning sorts to constants, variables and assigning the domains for functions and predicates arguments and a sort of functions ranges. We are working now in the theory of lists and the theory of integers, so there are two basic sorts of objects in our system. In axiomatizing these theories we need some way of distinguishing between them; otherwise false inferences could result from applying axioms to objects of wrong types. The sort of an object is determined from the context by considering the functions applied to this object and from "for" section of the formal specification. For example, variable `x` in the expression `tail(x)` is considered to be list, but `x` is treated as integer in the expression `x+1=b`, the predicate `Number(b)` in "for" section of specification declares, that `b` should be treated as integer.

The idea of sorted logic was presented, for example in [18], but we extend it by specifying the hierarchy of subsorts for each sort. We determine the relation < for sorts: `t1<t2` means that `t2` contains all the objects of `t1`. For example, for integers we have: positive<not_zero<integer (where positive contains all integers greater than 0 and not_zero contains all integers except 0), but the types list and not_zero are not bounded by the hierarchic relation. Particular subsorts are also determined from the context from the information about known functions (for example, in the expression `1/a a` is considered to be not_zero).

The use of sorted logic with hierarchy has two advantages. First, it allows to reduce the number of applicable rules by blocking the instantiations of objects of a wrong sort (instantiation of a list variable by an integer expression and vice versa). Second, it allows to avoid some incorrectly synthesized functions. By the functions synthesized correctly we mean functions that meet the given specification, and can be computed in according to domain restrictions for this function's subterms. Others are considered to be incorrect. For example a function

```
f(a)= (b=3/a) ∧ not(a>0)
```

can not be computed for a=0, because operation / (division) is undetermined for a=0, whereas the same function written in the other form

```
f(a)= if (a>0) then b=3/a
```

can be computed even for `a=0`. The difference is in the order of computing, that is fixed in if-then expression and can be different in ∧ operation, because it is considered to be commutative.

Let's look at the example in Table 5. From the same initial rows (1, 2 and 3) we can derive two functions. First, we apply resolution rule to rows 3 and 1 and then for the result and the 2nd row. The output of the resulted row (see row 4 in the Table 5) can

not be computed for a=0. But if we change the order of resolution application (first, to rows 2 and 1 and then for result row and the 3rd row) we obtain the row 5 with the output, that can be computed even for a=0.

Table 5. An example of different resolution application. Rows 1-3 are initial rows and rows 4 and 5 are obtained by using the resolution rule in different order.

1		**not** (b=3/a) ∧ (a>0)	1
2		(b=3/a)	2
3		**not**(a>0)	3
4		**true**	if (b=3/a) then 2 else (if a>0 then 1 else 3)
5		**true**	if (a>0) then (if b=3/a then 2 else 1) else 3

To avoid the application of rules that derived constructions not appropriate for computing, we use the information about sorts of objects and their hierarchy. During resolution, equality or relation replacement rule application we should replace the expressions with "smaller" types by true (or false) earlier. In our example the expression `b=3/a` (where the sort of a is not_zero) should be replaced before the expression `(a>0)` (where the sort of a is integer), because not_zero < integer. By applying this restriction the row 4 (see Table 5) will not be added to the tableau.

The use of sorted logic with hierarchy can resolve the problem only in for functions, that meets the specification, it does not correct the problems in the specification if they exist. For example, if a specification is written in such a way that there are input values, for which the specification value is undetermined or erroneous, the same problem will nor be resolved by considering sort hierarchy during the synthesis. If f(a) is specified as

```
<f(a)><=for number(a) find<z>such that if (a>=0) then 1/a
```

both the specification and the derived program will have problems with a=0. We've assumed, that the given specification should not allow such incorrect functions applications.

3.2 Constructing the Proof Plan Using Rippling

The stage of recursion formation is particularly needed to be directed, because of a large number of rows participating in the proof attempts. We propose, first, to perform the induction step using rippling technique [4],[6], [7] and save the path of the proof. Then we perform the proof in the table according to the found proof path and simultaneously a function is derived in the output column. A strategy of using general rippling principles for planning the proof has been described in [10].

Rippling is a rewriting, based on the idea, that very often the induction hypothesis and conclusion are syntactically similar. We rewrite the conclusion in such a way that we can use the hypothesis. We use only the rules that move the differences through the induction conclusion in a way that makes progress in minimizing difference with the induction hypothesis. The minimization of differences will always stop (either because all differences will be removed, or none of the rules could be applied). We change the conclusion only, because the hypothesis is assumed to be true.

The rules that are used for rippling are called wave rules. They are special rewriting rules formed from the axioms, known in the system. For example the axiom

(A<B) → (A+C<B+C) (2)

produces a following wave rule

(A + C) < (B +C)) ==> (A<B) (3)

Note, that the direction of rewriting is opposite to the logical implication, because we use backward reasoning (from conclusion to the hypothesis).The underlined parts in (3) are wave-fronts, that should be removed. The unmarked parts form skeleton, which is preserved during rewriting. Wave fronts are marked using difference unification algorithm [3], and these rules should "decrease" the differences in the expression to which they are applied. Rewrite rules are formed once at the moment then a new axiom appears in the deductive tableau and then rules are stored in the system.

To perform the induction step we insert the induction hypothesis in the tableau using the induction rule. To form a concrete induction hypothesis we need a particular well founded relation. Such relations are stored in the system as axioms, containing $<_{wf}$ (and also a new well founded relation can be added as an axiom). A particular well founded relation is chosen from the relations known in the system. The choice is based on the information about induction parameter type (there are such relations appropriate for lists and relations for integers). If several well founded relations can be used they are tried one after another. The induction conclusion is the initial goal from the table.

When an induction hypothesis and conclusion are written, the differences are marked as wave fronts and rewriting using wave rules starts. During it we save the sequence of axioms numbers, corresponding to used wave rules. This sequence is called proof path. The proof using rippling is considered to be successfully finished if an instance of hypothesis is obtained in the conclusion, so we can use the fact that hypothesis is assumed to be true and instantiate it by true. That helps us to prove the conclusion. In other cases we search for another way of proof using backtracking: we try to use another annotation or form another hypothesis. If after that the proof still does not succeed we can continue the full search in the deductive tableau. So we do not lose a solution while applying rippling heuristic. But it helps us to derive programs faster in practice: if proof is successfully finished, we return to deductive tableau and restore the proof by deduction rules according to the found path; it is necessary for derivation of a function in the output column.

4 Implementation, Results and Future Work

The described techniques were implemented in the system ALISA. The system performs synthesis in the interactive and in the automatic mode. During interactive synthesis user can create a deductive tableau, add a row, in particular containing a goal to prove, and choose an appropriate deductive rule. In the automatic mode the specification of a function and, possibly, some axioms are given and the proof is performed by the system. Some of the functions that are derived automatically are the following:

1. **Integer square root**

Specification: <**sqrt**(b)> <= for number (b) find <z> such that
if b≥ 0 then (sqr(z)<=b) ∧ (b<sqr(z+1)) ∧ (z ≥ 0)

Derived function:

```
(DEFUN sqrt(b) (COND((NUMBERP b) (COND((< b 1) 0)
     (T(COND((<= (sqr(+(sqrt (+ b -1)) 1))b)(+(sqrt(+ b -1)) 1))
     (T (sqrt (+ b -1]
```

Number of rule applications in full search – 31000, in synthesis with heuristics - 76

2. **List partitioning** into last element and front (all elements, except the last one).

Specification:

```
<front(a),last(a)> <= for list(a) find <h,t> such that
       if ¬(a =()) then (tail(t)=()) ∧ (a=append(h,t))
```

Derived functions:

```
(DEFUN front(a)(COND ((LISTP a)
                (COND ((NULL (CDR a)) NIL)
                      (T (CONS (CAR a) (front(CDR a))))))))
(DEFUN last(a)(COND ((LISTP a)
                (COND ((NULL (CDR a)) a)
                       (T (last (CDR a)))))))
```

Number of rule applications in full search – 7200 , in synthesis with heuristics – 28.

When the hierarchy of subsorts was not implemented in the system, it could derive another version of front-last functions (they immediately return error, for example, for a list containing one element, which is correctly proceeded by the front-last functions presented above):

```
front(s) = if not(tail(last(tail(s)))=NIL) then NIL else
      if not(tail(s)=append(front(tail(s)),last(tail(s)))
          then NIL
       else addfirst(head(s),front(tail(s)))
last(s) = if not(tail(last(tail(s)))=NIL) then s else
     if not(tail(s)=append(front(tail(s)),last(tail(s))))
          then s
      else tail(last(tail(s)))
```

The use of hierarchy blocked certain rule applications and allowed to avoid of such synthesis.

3. **Sorting program**

Specification:

```
<sort(b)> <= for list(b) find<z> such that
                    perm(b,z)∧ord(z)
```

Derived program:

```
(DEFUN sort(b) (COND ((LISTP b) (COND ((NULL b) b)
     (T (COND ((NULL (sort (CDR b))) b)
     (T (COND ((<= (CAR (sort (CDR b))) (CAR b)) (CONS (CAR
(sort (CDR b))) (sort (CONS (CAR b) (CDR (sort (CDR b)))))))
      (T (COND ((NULL (CDR b)) b) (T (CONS (CAR b) (sort (CDR
b))))))))))))))
```

Number of rule applications: full search - greater than 10^6, with heuristics – 4947.

Rippling was already applied for automatic construction of programs, for example in [16] it was used for construction of a special tactic for proofs in the sequent calculus. In our approach we combine rippling with deductive tableau framework.

The examples mentioned in [16] (sqrt, quotient-remainder, append, last) were successfully derived in ALISA.

Also rippling techniques was used in automatic synthesis of logic programs, for example, in the system Periwinkle [11]. But this system failed to synthesize sorting and list partitioning programs. The possibility to construct program structure and induction scheme during synthesis dynamically by choosing a particular well founded relation (from the templates of given relations) allowed to derive front-last list partitioning and sorting program in ALISA.

The work in this paper has concentrated on the fully automatic synthesis of recursive functional programs. The system currently derives several programs, but does not prefer any one type of recursion to another and the question of the complexity of derived program is not analyzed. The first derived program is presented as a result and the synthesis stops. A useful extension to the system would be to specify additional restrictions as the type of recursion (for example, requiring the program be tail recursive). The number of resolved tasks can be greater after adding new sorts to the system, by enlarging the set of know axioms and well founded relations.

5 Conclusions

In this paper we have investigated the application of proof planning to the automatic synthesis of functional programs. The work developed out of existing work in synthesis by the deductive tableau method and in rippling.

The main goal was to develop some techniques that allow to plan the proof construction in the deductive tableau for performing fully automatic synthesis at a reasonable time. Our work made the following principal contributions:

- The particular information about sorts and their hierarchy in the theory of integers and in the theory of lists allowed to avoid incorrectly synthesized programs.
- We have applied rippling heuristic for planning the proof in the deductive tableau.
- The proposed techniques were implemented in the system ALISA, reducing number of rules applications by several orders.

Although we are still far away from automatically synthesizing complex programs from their formal specifications, but we have made progress towards that goal by reducing time required for automatic synthesis of some tasks and rippling heuristic has played a crucial role in this success.

References

1. Armando A., Smaill A. and Green I. Automatic Synthesis of Recursive Programs: The Proof-Planning Paradigm. *Automated Software Engineering,* 6(4): 329-356, 1999.
2. Ayari A., Basin D. A Higher-Order Interpretation of Deductive Tableau. *Journal of symbolic* computation*,* 31(5): 487-520, 2001
3. Basin D., Walsh T. Difference Unification // – Ruzena Bajcsy (Ed.): Proceedings of the 13th IJCAI, Morgan Kaufmann, p. 116-122,1993.

4. Basin, D., Walsh, T.: A Calculus for and Termination of Rippling.// Journal of Automated Reasoning, vol. 16 147-180, 1996
5. Bundy, A., Stevens A., van Harmelen,.F., Ireland, A., Smaill, A. Rippling: A heuristic for guiding inductive proofs.// Artificial Intelligence, vol 62, 1993
6. Bundy A. The Automation of Proof by Mathematical Induction. *Handbook of automated reasoning*, vol.1, Elsevier Science Publishers B.V, 2001
7. Bundy A., Basin D., Hutter D.,Ireland A. Rippling: Meta-level Guidance for Mathematical Reasoning. Cambridge University Press, 2005
8. Burback R., Manna Z. and Waldinger R. et al. Using the Deductive Tableau System. MacIntosh Educational Software Collection, Chariot Software Group, 1990
9. Chang C.-L., Lee R. *Symbolic Logic and Mechanical Theorem Proving*, Academic Press Inc., New York, San Francisco, London, 1973
10. Korukhova Y. Planning Proof in the Deductive Tableau Using Rippling. Proceedings of the 5th International Conference RASC, Nottingham Trent University, UK, 384-389, 2004
11. Kraan I., Basin D., Bundy A. Middle-Out Reasoning for Synthesis and Induction. Journal of Symbolic Computation, 16(1-2): 113-145, 1996.
12. Lee R.C.T., Chang C.L., Waldinger R.J. An Improved program synthesizing algorithm and its correctness // Communications ACM, 17(4): 211-217, 1974
13. Manna Z. and Waldinger R. A Deductive approach to Program Synthesis, ACM Trans. Programming Languages and Systems 2 (1): 90-121, 1980
14. Manna Z. and Waldinger R. Fundamentals of Deductive Program Synthesis. IEEE Transactions on Software Engineering, 18(8): 674-704, 1992
15. Paulson L.C. Isabelle: A generic theorem prover. Lecture Notes in Computer Science 828:xvii-321, 1994.
16. Pientka, B., Kreitz, C. Automating inductive specification proofs. // Fundamenta Informatica, vol. 39(1-2), IOS Press, p. 189-208, 1999.
17. Simon H.A. Experiments with a Heuristic Compiler // Journal ACM, 10(4):493-506, 1963.
18. Traugott J. Deductive Synthesis of Sorting Programs. Journal of Symbolic Computation, 7: 533-572, 1989.
19. Tyugu E.H. Konceptual'noe programmirovanie (Conceptual programming) – Moscow, Nauka, 1984 *(in Russian)*

A Fault-Tolerant Default Logic*

Zhangang Lin, Yue Ma, and Zuoquan Lin

School of Mathematical Sciences
Peking University, Beijing 100871,China
{zglin, mayue, lz}@is.pku.edu.cn

Abstract. Reiter's default logic can not handle inconsistencies and incoherences and thus is not satisfactory enough in commonsense reasoning. In the paper we propose a new variant of default logic named FDL in which the existence of extension is guaranteed and the trivial extension is avoided. Moreover, Reiter's default extensions are reserved and can be identified from the other extensions in FDL. Technically, we develop a paraconsistent and monotonic reasoning system based on resolution as the underlying logic of FDL. The definition of extension is also modified in the manner that conflicts between justifications of the used defaults and the conclusions of the extension, which we call justification conflicts, are permitted, so that justifications can not be denied by "subsequent" defaults and the existence of extension is guaranteed. Then we select the desired extensions as preferred ones according to the criteria that justification conflicts should be minimal.

1 Introduction

Reiter's default logic[1] is a most advocated nonmonotonic reasoning system. It augments classical logic by *defaults* that differ from standard inference rules in sanctioning inferences that rely on given as well as absent information. Knowledge is represented in default logic by a *default theory* $\langle D, W\rangle$ consisting of a set of defaults D and a set of formulas W. Formulas in W are the *axioms* of the default theory. A default $\frac{\alpha:\beta_1,\cdots,\beta_n}{\gamma}$ has two types of antecedents: α is called the *prerequisite* and is established if α is derivable, and for $1 \leq i \leq n$, β_i is called a *justification* and is established if $\neg\beta_i$ can not be derived. If both conditions hold, the default is *applicable* and the *consequent* γ is concluded. An *extension* of a default theory which is a fixpoint of the belief revision operator w.r.t. the default theory can be viewed as a belief set described by the default theory. For clarity, we use the term *default logic* to refer to Reiter's default logic and call extension in Reiter's default logic *default extension*.

Despite its simple syntax and powerful expressivity, Reiter's default logic is not satisfactory enough in the following two aspects. On the one hand, a default theory has only a trivial default extension that includes every formula once the axioms in the default theory have contradictions (see Proposition 3). That is,

* This work is partially supported by NSFC (grant number 60373002 and 60496322) and NKBRPC (grant number 2004CB318000).

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 253–265, 2006.

contradictions in the axioms damage the meaningful information of a default theory. Such contradictions are called *inconsistencies* and are represented by the curve labeled with 1 in Figure 1. A default theory is *inconsistent* if it has inconsistencies, otherwise it is *consistent*.

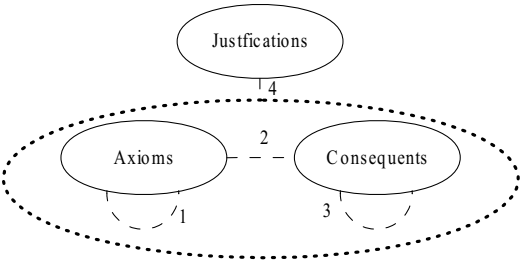


Fig. 1. Inconsistencies and Incoherences

On the other hand, some contradictions in a default theory cause the nonexistence of extension. We call such contradictions *incoherences*. A default theory is *incoherent* if it has no default extension, otherwise it is *coherent*. Incoherences may be categorized into three sorts shown in the following three examples respectively.

Example 1. $T = \langle D, W\rangle$, where $D = \{\frac{:B}{\neg A}\}$ and $W = \{A\}$. In T incoherences occur between W and the consequents of applicable defaults, which are represented by the line labeled with 2 in Figure 1.

Example 2. $T = \langle D, W\rangle$, where $D = \{\frac{:B}{C}, \frac{:D}{\neg C}\}$ and $W = \emptyset$. In T incoherences occur in the consequents of applicable defaults, which are represented by the curve labeled with 3 in Figure 1.

Example 3. $T = \langle D, W\rangle$, where $D = \{\frac{:\neg B}{A}\}$ and $W = \{A \rightarrow B\}$. In T incoherences occur between the justifications of used defaults and the consequences of (W and the consequents of used defaults), which are represented by the line labeled with 4 in Figure 1.

Some researchers hold the view that triviality and the nonexistence of extension of a given default theory are not shortcomings of default logic, and sometimes they are useful when default logic is used as a problem solver. For instance, a planning problem may be represented by a default theory whose default extensions correspond to the solutions of the problem and the incoherences of this default theory tell us that the problem has no solution. With this viewpoint, they have made efforts to find characterizations of default theories that have default extensions[1, 2, 3, 4].

The above viewpoint is reasonable in some aspects but does not seem to be sound when it turns to commonsense reasoning, since commonsense is always inconsistent and incoherent and we expect that some meaningful conclusions can still be reached even the knowledge is inconsistent or incoherent.

According to the above analysis, some researchers believe that Reiter's default logic is not fault-tolerant enough. Some of them refer to paraconsistent logic[5, 6, 7, 8], multi-valued logic[9, 10, 11, 12] in particular, to overcome inconsistencies, such as Bi-default logic[13] and four-valued default logic[14]. To deal with incoherences, some researchers modify Reiter's definition of extension to guarantee the existence of extension, among which are justified default logic[15], constrained default logic[16] and cumulative default logic[17].

The above five default reasoning systems are good attempts to extract meaningful information from a default theory with inconsistencies or incoherences, but they fail to handle inconsistencies and incoherences simultaneously. Moreover, justified default logic and constrained default logic can not identify default extensions from the other extensions and therefore they can not solve some problems that Reiter's default logic can, e.g. they are not suitable as problem solvers.

In the paper we propose a default reasoning system called *FDL*(shorted for Fault-tolerant Default Logic) in which every default theory has at least one extension and the trivial extension is avoided. Thus FDL can extract meaningful information from a default theory with inconsistencies and incoherences, which indicates that it can perform better than other default reasoning systems in commonsense reasoning. Besides its fault-tolerance, we also show that Reiter's default extensions are reserved and can be identified from the other extensions, which makes FDL able to solve the problems that Reiter's default logic can.

To overcome inconsistencies, it is natural that the underlying logic of FDL should be paraconsistent. But a paraconsistent logic is not adequate, since if it is nonmonotonic, the existence of extension can not be ensured—using an applicable default may make the prerequisite of a used default not derivable and thus inapplicable. Besides, to reserve Reiter's default extensions, the underlying logic should coincide with classical logic as to consistent sets of formulas. Since most existing paraconsistent logics which coincide with classical logic as to consistent sets of formulas are nonmonotonic, we need to develop a paraconsistent and monotonic one. To guarantee the existence of extension, we have to go further—a paraconsistent and monotonic underlying logic can only resolve contradictions represented by 1, 2 and 3 in Figure 1. To resolve contradictions represented by 4 in Figure 1, the definition of extension needs to be modified. In FDL, the modification is in the manner that justifications of the used defaults should be most consistent with the conclusions of the default theory, i.e. justification conflicts which are minimized are permitted.

To sum up, our work may be divided into the following steps:

1. Develop a paraconsistent and monotonic reasoning system as the underlying logic to handle inconsistencies.
2. Modify the definition of default extension so that FDL can tolerate incoherences and the existence of extension is guaranteed.
3. Select desired extensions.

Step 1 and 2 make FDL able to tolerate inconsistencies and incoherences, but at the cost that some counter-intuitive extensions may appear. With Step 3, these counter-intuitive extensions are excluded and desired ones are reserved. The trick—first guarantee the existence, then select desired ones—is also involved in preferred models[18, 19, 11] of multi-valued logic.

The rest of the paper is structured as follows. In Section 2 we briefly review Reiter's default logic. In Section 3 we introduce the paraconsistent and monotonic underlying reasoning system of FDL. Our system is represented in Section 4. Some properties of FDL are also studied in this section. We compare our work with others in Section 5 and conclude the paper in Section 6.

2 Default Logic

We start by completing our initial introduction to Reiter's default logic.

Throughout this paper, let $\mathcal{L}$ be a propositional language. We use Greek and uppercase letters to represent the formulas and the atoms in $\mathcal{L}$ respectively. Each atom A and its negation $\neg A$ are called *literals* which are represented by lowercase letters. The connectives in $\mathcal{L}$ are defined in the canonical manner. We write $\vdash$ for the provability relation in classical logic. The set of consequences of S is defined as $Cn(S) = \{\alpha \in \mathcal{L} \,|\, S \vdash \alpha\}$, where S is a set of formulas in $\mathcal{L}$.

A default is *normal* if it is of the form $\frac{\alpha:\beta}{\beta}$. Let D be a set of defaults. By $Pre(D)$, $Just(D)$ and $Con(D)$, we denote the sets of prerequisites, justifications and consequents of the defaults in D respectively. A set of defaults D and a set of formulas W form a *default theory* $\langle D, W\rangle$, which is *normal* if all defaults in D are normal. For simplicity, we assume that W and D are both finite. A default theory may induce one, multiple or even no default extensions in the following way.

Definition 1 ([1]). *Let $T = \langle D, W\rangle$ be a default theory. For any set of formulas S, $\Gamma(S)$ is the smallest set of formulas such that*

1. $\Gamma(S) = Cn(\Gamma(S))$.
2. $W \subseteq \Gamma(S)$.
3. *If* $\frac{\alpha:\beta_1,\cdots,\beta_n}{\gamma} \in D$, $\alpha \in \Gamma(s)$, *and* $\neg\beta_1 \notin S$, $\cdots$, $\neg\beta_n \notin S$, *then* $\gamma \in \Gamma(S)$.

A set of formulas E is a default extension *of $\langle D, W\rangle$ if $\Gamma(E) = E$.*

The *set of generating defaults* for E w.r.t. default theory T is defined as

$$GD(E,T) = \left\{ \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \in D \,|\, \alpha \in E, \neg\beta_1 \notin E, \cdots, \neg\beta_n \notin E \right\}$$

Proposition 1 ([1]). *If E is a default extension of default theory $T = \langle D, W\rangle$, then $E = Cn(W \cup Con(GD(E,T)))$.*

Proposition 2 ([1]). *Let $T = \langle D, W\rangle$ be a default theory. For any set of formulas E, define $E_0 = W$ and for each $i \geq 0$*

$$E_{i+1} = Cn(E_i) \cup \left\{ \gamma \;\middle|\; \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \in D, \text{ where } \alpha \in E_i, \neg\beta_1 \notin E, \cdots, \neg\beta_n \notin E \right\}$$

Then E is a default extension of T iff $E = \bigcup_{i=0}^{\infty} E_i$.

Proposition 3 ([1]). *Default theory $T = \langle D, W \rangle$ has only a trivial default extension iff W is inconsistent.*

Proposition 4 ([1]). *Each normal default theory has at least one default extension.*

3 The Underlying Logic

Since we base the underlying logic on resolution, we have to convert formulas into clauses. If a formula contains inconsistencies, there would be more than one set of clauses corresponding to it and the result of resolution is different. Hence a "normal" form of clauses is necessary.

Definition 2. *Let α be a formula which is not a tautology in $\mathcal{L}$ and $P_1, \cdots, P_n$ be all atoms occurring in α. We say formula $\beta = (l_{11} \vee \cdots \vee l_{1n}) \wedge \cdots \wedge (l_{m1} \vee \cdots \vee l_{mn})$ is a* principal conjunctive normal form *of α if*

1. *α is equivalent to β.*
2. *l_{ij} is P_j or $\neg P_j$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.*

If α is a tautology, define the principal conjunctive normal form of α to be t.

Definition 3. *Let α be a formula in $\mathcal{L}$. The* set of principal clauses *of α is defined as $C(\alpha) = \{\{l_1, \cdots, l_n\} | l_1 \vee \cdots \vee l_n$ is a conjunct of a principal conjunctive normal form of $\alpha\}$. Let S be a set of formulas in $\mathcal{L}$. The set of principal clauses of S is $C(S) = \bigcup_{\alpha \in S} C(\alpha)$.*

Lemma 1. *Each set of formulas has exactly one set of principal clauses.*

Definition 4. *A set of clauses S is* resolution closed *if $\{A\} \cup C_1 \in S$ and $\{\neg A\} \cup C_2 \in S$ imply $C_1 \cup C_2 \in S$, provided that $C_1 \cup C_2$ is not empty and A is an atom. The smallest set that includes S and is resolution closed is called the* resolution closure *of S written as $RC(S)$.*

Definition 5. *A set of clauses S is* appending closed *if*

1. *$\{t\} \in S$.*
2. *If $C_1 \in S$ and $C_1 \subseteq C_2$, then $C_2 \in S$.*

The appending closure *of S written as $AC(S)$ is the smallest set that includes S and is appending closed.*

Definition 6. *Let α be a formula and S be a set of formulas in $\mathcal{L}$. If $C(\alpha) \subseteq AC(RC(C(S)))$, we say that α is* monotonically derivable *from S written as $S \vdash_{mc} \alpha$. The* monotonic closure *of S is defined as $MC(S) = \{\alpha \in \mathcal{L} | S \vdash_{mc} \alpha\}$.*

Example 4. Assume S=$\{A, A \to B\}$. Then $C(S)$=$\{\{A\}, \{\neg A, B\}\}$, $RC(C(S))$= $\{\{A\}, \{\neg A, B\}, \{B\}\}$, $AC(RC(C(S)))$ =$\{\{t\}, \{A\}, \{\neg A, B\}, \{B\}, \cdots\}$. Since $C(A \wedge B)$=$\{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}\} \subseteq AC(RC(C(S)))$, $S \vdash_{mc} A \wedge B$. It can be verified that $MC(S) = Cn(S)$. Now let $S' = S \cup \{\neg A\}$. Then $C(S')$=$\{\{\neg A\}, \{A\}, \{\neg A, B\}\}$, $RC(C(S'))$=$\{\{\neg A\}, \{A\}, \{\neg A, B\}, \{B\}\}$, $AC(RC(C(S')))$= $\{\{t\}, \{\neg A\}, \{A\}, \{\neg A, B\}, \{B\}, \cdots\}$. Therefore $MC(S')$=$\{\neg A, A, A \to B, B, A \vee B, \neg A \vee B, A \wedge \neg A, \cdots\}$. It is readily to verify that $MC(S) \subseteq MC(S')$. Also $S' \nvdash_{mc} C$ for any atom C not occurring in S', which indicates that MC is paraconsistent.

From the above example, we notice that inconsistencies can be conquered with MC. Moreover, MC is monotonic, as stated by the following proposition.

Proposition 5 (Monotonicity of MC). *If $S \subseteq S'$, then $MC(S) \subseteq MC(S')$.*

Proposition 6. *If S is classically consistent, then $MC(S) \equiv Cn(S)$.*

Proposition 6 implies that, although MC is strictly weaker than Cn(since MC is paraconsistent), they are identical as to consistent sets of formulas.

In [20], Lehmann suggests that the reasoning relation $\vdash_p$ in a plausibility logic should satisfy the following conditions:

1. Inclusion: $\Gamma \vdash_p \alpha$ if $\alpha \in \Gamma$.
2. Right Monotonicity: If $\Gamma \vdash_p \alpha$, then $\Gamma \vdash_p \alpha \vee \beta$ for any formula β.
3. Cautious Left Monotonicity: If $\Gamma \vdash_p \alpha$ and $\Gamma \vdash_p \beta$, then $\Gamma \cup \{\alpha\} \vdash_p \beta$.
4. Cautious cut: If $\Gamma \cup \{\alpha\} \vdash_p \beta$ and $\Gamma \vdash_p \alpha$, then $\Gamma \vdash_p \beta$.

It can be verified that $\vdash_{mc}$ satisfies all of the above conditions but cautious cut. It means that a derived formula can not be used as a lemma to infer new formulas, i.e. cumulativity does not hold as to $\vdash_{mc}$. Therefore $\vdash_{mc}$ is not a plausibility logic in the above sense.

4 Fault-Tolerant Default Logic

4.1 Alternative Extension

To this point, we modify the definition of default extension.

Definition 7. *Let $T = \langle D, W\rangle$ be a default theory. For the sets of formulas E, J and E', we say that default $\delta = \frac{\alpha:\beta_1,\cdots,\beta_n}{\gamma}$ is* applicable *to E' w.r.t. E and J if*

1. *$\alpha \in E'$.*
2. *$\neg\beta_i \notin E$ or there is a formula equivalent to $\neg\beta_i$ in $E \cap J$ for each $1 \leq i \leq n$.*

For sets of formulas E and J, we say pair $\langle E, J\rangle$ is *smaller* than $\langle E', J'\rangle$ written as $\langle E, J\rangle \leq \langle E', J'\rangle$ if $E \subseteq E'$ and $J \subseteq J'$. $\langle E, J\rangle$ is *consistent* if E is consistent.

Definition 8. *Let $T = \langle D, W\rangle$ be a default theory. $\Gamma(\langle E, J\rangle)$ is the smallest pair $\langle E', J'\rangle$ such that*

1. $W \subseteq E'$
2. $E' = MC(E')$
3. *If $\frac{\alpha:\beta_1,\cdots,\beta_n}{\gamma} \in D$ is applicable to E' w.r.t. E and J, then $\gamma \in E'$, $\neg\beta_1 \in J'$, $\cdots$, $\neg\beta_n \in J'$.*

A pair $\langle E, J\rangle$ is an alternative extension *of $T = \langle W, D\rangle$ if $\langle E, J\rangle = \Gamma(\langle E, J\rangle)$.*

With the similar approach in [21], we can verify that Γ and alternative extension are well-defined.

Theorem 1. *Let $T = \langle D, W\rangle$ be a default theory. For the sets of formulas E and J, define $E_0 = MC(W)$, $J_0 = \emptyset$, and for $i \geq 0$*

$$E_{i+1} = MC(E_i) \cup \left\{ \gamma \,\middle|\, \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \in D \text{ is applicable to } E_i \text{ w.r.t. } E \text{ and } J \right\}$$

$$J_{i+1} = J_i \cup \left\{ \neg\beta_1, \cdots, \neg\beta_n \,\middle|\, \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \in D \text{ is applicable to } E_i \text{ w.r.t. } E \text{ and } J \right\}$$

Then $\langle E, J\rangle$ is an alternative extension of T iff $E = \bigcup_{i=0}^{\infty} E_i$ and $J = \bigcup_{i=0}^{\infty} J_i$.

Definition 9. *The* set of generating defaults *for pair $\langle E, J\rangle$ w.r.t. default theory T written as $GD(E, J, T)$ is*

$$GD(E, J, T) = \left\{ \delta \,\middle|\, \delta = \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \in D \text{ is applicable to } E \text{ w.r.t. } E \text{ and } J \right\}$$

Theorem 2. *For the sets of formulas E and J, if $\langle E, J\rangle$ is an alternative extension of default theory $T = \langle D, W\rangle$, then $E = MC(W \cup Con(GD(E, J, T)))$ and $J = \{\neg\beta | \beta \in Just(GD(E, J, T))\}$.*

The following two examples indicate that alternative extension could tolerate all contradictions shown in Figure 1.

Example 5. Consider default theory $T = \langle D, W\rangle$, where $D = \{\frac{B:C}{D}\}$ and $W = \{A, \neg A, A \rightarrow B\}$. According to Proposition 3, T has only a trivial default extension. Contrarily, since the underlying logic of FDL is paraconsistent, it has a nontrivial alternative extension $\langle MC(\{A, \neg A, A \rightarrow B, D\}), \{\neg C\}\rangle$.

Example 6. Consider the default theories in Example 1, 2 and 3. They have $\langle MC(\{A, \neg A\}), \{\neg B\}\rangle$, $\langle MC(\{C, \neg C\}), \{\neg B, \neg D\}\rangle$ and $\langle MC(\{A, A \rightarrow B, B\}), \{\neg\neg B\}\rangle$ as their alternative extensions respectively.

4.2 Operational Considerations

In [22], an operational semantics is assigned to Reiter's default logic. We can do the similar thing to alternative extension. Let $T = \langle D, W\rangle$ be a default theory and $\Pi = \langle d_1, d_2, \cdots\rangle$ be a sequence of defaults from D. We denote the initial segment of Π of length k by Π_k, provided that the length of Π is at least k, and define $In(\Pi_i) = MC(W \cup Con(\Pi_i))$, $Out(\Pi_i) = \{\neg\beta | \beta \in Just(\Pi_i)\}$.

Definition 10. *A default* $d = \alpha : \beta_1, \cdots, \beta_n/\gamma$ *is* applicable *to a sequence* Π *if*

1. $\alpha \in In(\Pi)$.
2. $\neg\beta_i \notin In(\Pi)$ *or there is a formula equivalent to* $\neg\beta_i$ *in* $In(\Pi) \cap Out(\Pi)$ *for each* $1 \leq i \leq n$.

Definition 11. *A sequence* $\Pi = \langle d_1, d_2, \cdots\rangle$ *in* D *is a* process *if* d_k *is applicable to* Π_{k-1} *for every* $k \geq 1$. *Process* Π *is* closed *if every* $d \in D$ *that is applicable to* Π *already occurs in* Π.

Lemma 2. *Each default theory has at least one closed process.*

Since justifications of used defaults in a process can not be invalidated by subsequent defaults, we have Lemma 3 and Theorem 3.

Lemma 3. *If* Π *is a closed process of default theory* T, *then* $\langle In(\Pi), Out(\Pi)\rangle$ *is an alternative extension of* T.

Theorem 3 (Semimonotonicity). *If* $T = \langle D, W\rangle$ *and* $T' = \langle D', W\rangle$ *are two default theories, where* $D \subseteq D'$, *and* $\langle E, J\rangle$ *is an alternative extension of* T, *then* T' *must have an alternative extension* $\langle E', J'\rangle$ *such that* $\langle E, J\rangle \leq \langle E', J'\rangle$.

From Lemma 2 and 3, we immediately have

Theorem 4 (Existence of Alternative Extension). *Each default theory has at least one alternative extension.*

For an alternative extension $\langle E, J\rangle$, if there is a closed process Π such that $In(\Pi) = E$ and $Out(\Pi) = J$, we say $\langle E, J\rangle$ has Π *corresponding* to it. Although Definition 11 and Lemma 3 imply that each closed process corresponds to an alternative extension, there are some alternative extensions that have no process corresponding to them, i.e. some alternative extensions are not constructive.

Example 7. Consider default theory $T = \langle D, W\rangle$, where $D = \{\frac{:A}{B}, \frac{:\neg B}{\neg A}\}$ and $W = \emptyset$. T has just two closed processes: $\langle\frac{:A}{B}\rangle$ and $\langle\frac{:\neg A}{\neg B}\rangle$. There is no closed process corresponding to alternative extension $\langle MC(\{\neg A, B\}), \{\neg A, \neg\neg B\}\rangle$.

4.3 The Largest and the Minimal Alternative Extensions

Minimality does not hold as to alternative extension.

Example 8. Let $D = \{\frac{:A}{B}, \frac{:\neg B}{C}\}$ and $W = \emptyset$. It can be verified that default theory $T = \langle D,\ W\rangle$ has two alternative extensions: $\langle E_1,\ J_1\rangle$=$\langle MC(\{B\}),\ \{\neg A\}\rangle$ and $\langle E_2,\ J_2\rangle$=$\langle MC(\{B,\ C\}),\ \{\neg A,\ \neg\neg B\}\rangle$. Since $E_1 \subset E_2$, $J_1 \subset J_2$, $\langle E_1,\ J_1\rangle < \langle E_2,\ J_2\rangle$.

This is not occasional. As a matter of fact, we have

Theorem 5. *If* $\langle E_k, J_k\rangle$ *are alternative extensions of default theory* T *for each* $k = 1, 2, \cdots$, *then there must exist the smallest alternative extension* $\langle E, J\rangle$ *of* T

that is bigger than $\langle E_k, J_k\rangle$ *for each* k *such that* $E = \bigcup_{i=0}^{\infty} F_i$ *and* $J = \bigcup_{i=0}^{\infty} K_i$, *where* $F_0 = \bigcup_k E_k$, $K_0 = \bigcup_k J_k$, *and for* $i \geq 0$

$$F_{i+1} = MC(F_i) \cup \left\{ \gamma \,\middle|\, \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \text{ is applicable to } F_i \text{ w.r.t. } F_i \text{ and } K_i \right\}$$

$$K_{i+1} = K_i \cup \left\{ \neg\beta_1, \cdots, \neg\beta_n \,\middle|\, \frac{\alpha : \beta_1, \cdots, \beta_n}{\gamma} \text{ is applicable to } F_i \text{ w.r.t. } F_i \text{ and } K_i \right\}$$

The above theorem indicates that all the alternative extensions of a default theory form a *complete upper semilattice* w.r.t. $\leq$.

Corollary 1 (Existence of the Largest Alternative Extension). *Each default theory* T *has an alternative extension* $\langle E, J\rangle$ *such that for each alternative extension* $\langle E', J'\rangle$ *of* T, $\langle E', J'\rangle \leq \langle E, J\rangle$.

Definition 12. *For an alternative extension* $\langle E, J\rangle$ *of default theory* T, *if there is no alternative extension of* T *that is smaller than* $\langle E, J\rangle$, *then* $\langle E, J\rangle$ *is a* minimal alternative extension *of* T.

Theorem 6. *If* $\langle E, J\rangle$ *is a minimal alternative extension of default theory* T, *then there must be a closed process* Π *such that* $E = In(\Pi)$ *and* $J = Out(\Pi)$.

The above theorem does not hold vice versa. See the following example.

Example 9. Consider default theory $T = \langle D, W\rangle$, where $D = \{\frac{:A}{B}, \frac{:C}{\neg A}\}$ and $W = \emptyset$. Sequence $\langle \frac{:A}{B}, \frac{:C}{\neg A}\rangle$ is a closed process of T. Therefore $\langle E, J\rangle = \langle MC(\{\neg A, B\}), \{\neg A, \neg C\}\rangle$ is an alternative extension. However, it is not a minimal alternative extension(alternative extension $\langle MC(\{\neg A\}), \{\neg C\}\rangle$ is smaller than $\langle E, J\rangle$).

4.4 Preferred Extension

In Example 8, we note that some alternative extensions are counter-intuitive. In this subsection, we exclude those counter-intuitive ones according to the criteria that the justification conflicts should be minimal.

Definition 13. *Let* T *be a default theory. An alternative extension* $\langle E', J'\rangle$ *of* T *is called a* preferred extension *if* $E' \cap J'$ *is minimal in* $\{E \cap J | \langle E, J\rangle$ *is an alternative extension of* $T\}$.

Example 10. Let $T = \langle D, W\rangle$ be a default theory, where $D = \{\frac{:A}{B}, \frac{:\neg B}{\neg A}, \frac{B:C}{C}, \frac{\neg A:B}{B}\}$ and $W = \emptyset$. T has three alternative extensions: $\langle E_1, J_1\rangle = \langle MC(\{B, C\}), \{\neg A, \neg C\}\rangle$, $\langle E_2, J_2\rangle = \langle MC(\{\neg A, B, C\}), \{\neg B, \neg\neg B, \neg C\}\rangle$ and $\langle E_3, J_3\rangle = \langle MC(\{\neg A, B, C\}), \{\neg A, \neg\neg B, \neg C\}\rangle$. Since $E_1 \cap J_1 = \emptyset$, $E_2 \cap J_2 = \{\neg\neg B\}$ and $E_3 \cap J_3 = \{\neg A, \neg\neg B\}$, $\langle E_1, J_1\rangle$ is a preferred extension of T, while the other two are not.

Example 11. Let $T = \langle D, W\rangle$ be a default theory, where $D = \{\frac{:A}{\neg A}\}$ and $W = \emptyset$. T has only one alternative extension $\langle MC(\{\neg A\}), \{\neg A\}\rangle$ which is also the only preferred extension of T.

Theorem 7 (Existence of Preferred Extension). *Each default theory has at least one preferred extension.*

Theorem 8. *Each preferred extension is a minimal alternative extension.*

The above theorem does not hold vice versa, as shown by the following example.

Example 12. Let $D = \{\frac{:A}{B}, \frac{B:C}{\neg A}, \frac{:\neg B}{\neg A \wedge \neg C}\}$ and $W = \emptyset$. Default theory $T = \langle D, W\rangle$ has $\langle E_1, J_1\rangle = \langle MC(\{\neg A, B\}), \{\neg A, \neg C\}\rangle$, $\langle E_2, J_2\rangle = \langle MC(\{\neg A, \neg C\}), \{\neg\neg B\}\rangle$ and $\langle E_3, J_3\rangle = \langle MC(\{\neg A, B, \neg C\}), \{\neg A, \neg\neg B, \neg C\}\rangle$ as its alternative extensions, where $\langle E_1, J_1\rangle$ and $\langle E_2, J_2\rangle$ are minimal alternative extensions, but $\langle E_1, J_1\rangle$ is not a preferred extension, whereas $\langle E_2, J_2\rangle$ is.

Corollary 2 (Minimality of Preferred Extension). *If $\langle E, J\rangle$ and $\langle E', J'\rangle$ are both preferred extensions of default theory T and $\langle E, J\rangle \leq \langle E', J'\rangle$, then $E = E'$ and $J = J'$.*

Theorem 9. *E is a consistent default extension of default theory T iff T has a consistent preferred extension $\langle E, J\rangle$ such that $E \cap J = \emptyset$.*

Semimonotonicity does not hold as to preferred extension(see the following example). Therefore semimonotonicity implies the existence of extension, but not vice versa.

Example 13. Let $W = \emptyset$ and $D = \{\frac{:A}{B}, \frac{:\neg B}{\neg A}\}$, $D' = \{\frac{:A}{B}, \frac{:\neg B}{\neg A}, \frac{B:\neg A}{\neg A}\}$. Default theory $\langle D, W\rangle$ has two preferred extensions: $\langle E_1, J_1\rangle = \langle MC(\{B\}), \{\neg A\}\rangle$ and $\langle E_2, J_2\rangle = \langle MC(\{\neg A\}), \{\neg\neg B\}\rangle$, while $\langle D', W\rangle$ has only one preferred extension: $\langle E, J\rangle = \langle MC(\{\neg A\}), \{\neg\neg B\}\rangle$. Although $D \subseteq D'$, $\langle D', W\rangle$ has no preferred extension bigger than $\langle E_1, J_1\rangle$.

Definition 14. *If $\langle E, J\rangle$ and $\langle E', J'\rangle$ are preferred extensions of default theory T such that $E \cap J = E' \cap J'$ and $\{\alpha | \alpha \wedge \neg\alpha \in E\} \subset \{\alpha | \alpha \wedge \neg\alpha \in E'\}$, we say $\langle E, J\rangle$ is* more consistent *than $\langle E', J'\rangle$. If T has no preferred extension more consistent than $\langle E, J\rangle$, $\langle E, J\rangle$ is a* most consistent preferred extension *of T.*

Theorem 10 (Existence of Most Consistent Preferred Extension). *Each default theory has at least one most consistent preferred extension.*

From Theorem 9, we have

Corollary 3. *E is a default extension of consistent and coherent default theory T iff there is a set of formulas J such that $\langle E, J\rangle$ is a most consistent preferred extension of T and $E \cap J = \emptyset$.*

To this point, we have discussed a set of extensions, the inclusion relations among which are shown in Figure 2. The figure also indicates that to compute all the preferred extensions and most consistent preferred extensions, it suffices to consider only closed processes which are constructive.

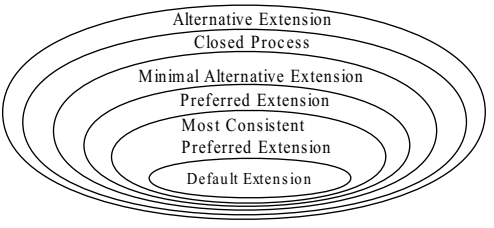


Fig. 2. Inclusion Relations

Theorem 11. *If $T = \langle D, W\rangle$ is a normal default theory where W is consistent, then E is a default extension of T iff T has a closed process Π such that $In(\Pi) = E$ and $In(\Pi) \cap Out(\Pi) = \emptyset$.*

The above theorem implies that as to a consistent normal default theory, closed processes, minimal alternative extensions, preferred extensions, most consistent preferred extensions and default extensions are identical. Theorem 9 and Corollary 3 imply that as to a consistent and coherent default theory, default extensions are identical with consistent preferred extensions as well as most consistent preferred extensions.

5 Related Work

The main idea of the paper is inspired by Reiter's seminal paper[1], some variants of it[15, 17, 16] and Linke *et al*'s work on default logic[4]. The formal definition of default extension is more delicate than could have been expected. This is due to the context-sensitive nature of justifications. In fact, a default's justifications can be refuted only when all default consequents contributing to a default extension are known. This is why the non-refutability of a justification is verified w.r.t. the final default extension. In such an approach, default extensions are not constructive and one is obliged to inspect the entire set of defaults to decide whether a default can be applied.

Reiter[1], Lukaszewicz[15] and Linke[4] have tried to reduce this kind of global checks to local ones to make extensions constructive. In normal default logic, defaults are restricted to be normal so that local checks are adequate. Linke *et al* replace such global checks by the strictly necessary ones. In justified default logic, a default is applicable only if its prerequisite is derivable and its justifications and consequent are consistent with used defaults. Therefore global checks are unnecessary in justified default logic.

By avoiding this kind of global checks, normal default logic and justified default logic are seminormal and the existence of extension is guaranteed. But since the underlying logic of the two are not paraconsistent, the trivial extension can not be avoided.

Taking no account of the existence of extension, bi-default logic[13] adopts the approach of signed system[8]. It translates a default theory into two related

default theories which are both consistent. The two related default theories comprise a bi-default theory. By dividing the inconsistencies into two parts, bi-default logic overcomes triviality. Compared with bi-default logic, the approach taken by Yue *et al*[14] seems to be more natural. They define four-valued models for an arbitrary default theory. Since four-valued logic is paraconsistent, all four-valued models are nontrivial. Similar to bi-default logic, a transformation is introduced to translate the original default theory to a consistent signed one. What is interesting is, it is proved that four-valued models of the original default theory can be gained by computing the default extensions of the translated default theory. Unfortunately, the relationship between four-valued models and extension is not clear.

The paper is an attempt to resolving inconsistencies and incoherences simultaneously. Thus it needs to go further than the above default logics. In FDL, we also try to avoid global checks: the condition of justification establishing is weaker than in justified default logic and the original default logic, which makes "subsequent" defaults would never invalidate used defaults. Moreover, since the syntax is not modified and default extensions are reserved, FDL retains the simplicity and powerful expressivity of Reiter's default logic.

6 Conclusion

Our main contribution in the paper is, based on a paraconsistent and monotonic reasoning system, we generalize Reiter's default logic, i.e., each default theory has at least one extension in FDL and Reiter's default logic coincides with FDL(when most consistent preferred extension is used) as to a consistent and coherent default theory.

Although FDL has some nice properties, the computation of alternative extensions and preferred extensions is under discussion. Besides, the relationship between FDL and other default reasoning systems is not so clear, and more research will be devoted to it.

Not only do inconsistencies and incoherences occur in default logic, but also they exist in other reasoning systems, in which logic programming is a case in point. In the future work, we will apply the idea of FDL to other reasoning systems.

References

1. Reiter, R.: A logic for default reasoning. Artificial Intelligence **13** (1980) 81–132
2. Papadimitriou, C.H., Sideri, M.: Default theories that always have extensions. Artificial Intelligence **69**(1-2) (1994) 347 – 357
3. Cholewinski, P.: Reasoning with stratified default theories. Logic Programming and Nonmonotonic Reasoning **928** (1995) 273–286
4. Linke, T., Schaub, T.: Alternative foundations for Reiter's default logic. Artificial Intelligence **124**(1) (2000) 31–86
5. daCosta, N.: Theory of inconsistent formal systems. Notre Dame Journal of Formal Logic **15** (1974) 497–510

6. Lin, Z., Li, W.: On logic of paradox. In: Proceedings of the 25th IEEE International Symposium on Multipl-Valued Logic. (1995) 248–255
7. Lin, Z.: Paraconsistent circumscription. Journal of Pattern Recognition and Artificial Intelligence (**10**(6)) 679–686
8. Besnard, P., Schaub, T.: Signed system for paraconsistent reasoning. Journal of Automated Reasoning **20**(1-2) (1998) 191–213
9. Belnap, N.: How computer should think. In: Contemporary Aspects of Philosophy. (1977) 7–37
10. Belnap, N.: A useful four-valued logic. In: Modern uses of multiple-valued logic. (1977) 30–56
11. Arieli, O., Avron, A.: The value of the four values. Artificial Intelligence **102**(1) (1998) 97–141
12. Ginsberg, M.L.: Multivalued logics: a uniform approach to reasoning in artificial intelligence. Computational Intelligence **4** (1988) 265–316
13. Han, Q., Lin, Z.: Paraconsistent default reasoning. In: 10th International Workshop on Non-Monotonic Reasoning. (2004) 197–203
14. Yue, A., Lin, Z.: Default logic based on four valued semantics. Chinese journal of computer **28**(9) (2005) 1447–1458
15. Lukaszewicz, W.: Considerations on default logic: an alternative approach. Computational Intelligence **4**(1) (1988) 1–16
16. Schaub, T.: On constrained default theories. In: ECAI. (1992) 304–308
17. Brewka, G.: Cumulative default logic: in defense of nonmonotonic inference rules. Artificial Intelligence **50**(2) (1991) 183–205
18. Shoham, Y.: A semantical approach to nonmonotonic logics. In Ginsberg, M.L., ed.: Readings in Nonmonotonic Reasoning. Kaufmann, Los Altos, CA (1987) 227–250
19. Shoham, Y.: Reasoning about change: time and causation from the standpoint of artificial intelligence. MIT Press, Cambridge, MA, USA (1988)
20. Lehmann, D.J.: Plausibility logic. In: CSL. (1991) 227–241
21. Marek, W., Truszczyński, M.: Nonmonotonic logic: Context-dependent reasoning. Springer-Verlag, Berlin (1994) 61–62
22. Antoniou, G., Sperschneider:, V.: Operational concepts of nonmonotonic logics, part 1: Default logic. Artificial Intelligence **8**(1) (1994) 3–16

Reasoning About Actions Using Description Logics with General TBoxes

Hongkai Liu[1], Carsten Lutz[1], Maja Miličić[1], and Frank Wolter[2]

[1] Institut für Theoretische Informatik
TU Dresden, Germany
lastname@tcs.inf.tu-dresden.de
[2] Department of Computer Science
University of Liverpool, UK
frank@csc.liv.ac.uk

Abstract. Action formalisms based on description logics (DLs) have recently been introduced as decidable fragments of well-established action theories such as the Situation Calculus and the Fluent Calculus. However, existing DL action formalisms fail to include general TBoxes, which are the standard tool for formalising ontologies in modern description logics. We define a DL action formalism that admits general TBoxes, propose an approach to addressing the ramification problem that is introduced in this way, show that our formalism is decidable and perform a detailed investigation of its computational complexity.

1 Introduction

Action theories such as the Situation Calculus (SitCalc) and the Fluent Calculus aim at describing actions in a semantically adequate way [10, 12]. They are usually formulated in first- or higher-order logic and do not admit decidable reasoning. For reasoning about actions in practical applications, such theories are thus not directly suited. There are two obvious ways around this problem: the first one is to accept undecidability and replace reasoning by programming. This route is taken by the inventors of action-oriented programming languages such as Golog [5] and Flux [13], whose semantics is based on the SitCalc and Fluent Calculus, respectively. The second one is to try to identify fragments of action theories such as SitCalc that are sufficiently expressive to be useful in applications, but nevertheless admit decidable reasoning. For example, a simple such fragment is obtained by allowing only propositional logic for describing the state of the world and pre- and post-conditions of actions. A much more expressive formalism was identified in our recent paper [2], where we define action formalisms that are based on description logics (DLs) [3]. More precisely, we use DL ABoxes to describe the state of the world and pre- and post-conditions of actions and prove that reasoning in the resulting formalism is decidable [2]. We also show in [2] that, in this way, we actually get a decidable fragment of SitCalc.

In description logic, TBoxes are used as an ontology formalism, i.e., to define concepts and describe relations between them. For example, a TBox may describe

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 266–279, 2006.

relevant concepts from the domain of universities such as lecturers, students, courses, and libraries. From the reasoning about actions perspective, TBoxes correspond to state constraints. For example, a TBox for the university domain could state that every student that is registered for a course has access to a university library. If we execute an action that registers the student Dirk for a computer science course, then after the action Dirk should also have access to a university library to comply with the state constraint imposed by the TBox. Thus, general TBoxes as state constraints induce a ramification problem.

Regarding TBoxes/state constraints, the DL action formalism defined in [2] has two major limitations: first, we only admit *acyclic TBoxes* which are a much more lightweight ontology formalism than the *general TBoxes* that can be found in all state-of-the-art DL reasoners [17]. For example, the DL formulation of the above ontology statement regarding access to libraries requires a general concept inclusion (GCIs) as offered by general TBoxes. Second, we allow only concept names (but no complex concepts) in post-conditions and additionally stipulate that these concept names are *not* defined in the TBox. In the present paper, we present a pragmatic approach to overcoming these limitations while retaining decidability of reasoning. In particular, we show how to incorporate general TBoxes into DL action formalisms which also means to drop the second restriction since there is no clear notion of a concept name "being defined" in a general TBox.

The main reason for adopting the mentioned restrictions in [2] was that they disarm the ramification problem that is introduced by more general TBoxes and post-conditions, c.f. the above example. Attempts to *automatically* solve the ramificiation problem, e.g. by adopting a Winslett-style PMA semantics [16], lead to semantic and computational problems: we show in [2] that counter-intuitive results and undecidability of reasoning are the consequence of adopting such a semantics. Since there appears to be no general automated solution to the ramification problem introduced by general TBoxes unless resorting to very inexpressive DLs [4], we propose to leave it to the designer of an action description to fine-tune the ramifications of the action. This is similar to the approach taken in the SitCalc and the Fluent Calculus to address the ramification problem. There, the designer of an action description can control the ramifications of the action by specifying causal relationships between predicates [6, 11]. While causality appears to be a satisfactory approach for addressing the ramification problem that is induced by Boolean state constraints, it seems not powerful enough for attacking the ramifications introduced by general TBoxes, which usually involve complex quantification patterns. We therefore advocate a different approach for DL action formalisms with general TBoxes: when describing an action, the user can specify the predicates that can change by executing the action, as well as those that cannot change. To allow an adequate fine-tuning of ramifications, we admit rather complex statements about the change of predicates such as "the concept name A can change from positive to negative only at the individual a, and from negative to positive only where the complex concept C was satisfied before the action was executed".

Name	Syntax	Semantics
inverse role	r^-	$(r^{\mathcal{I}})^{-1}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
at-least restriction	$(\geqslant n\ r\ C)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$
at-most restriction	$(\leqslant n\ r\ C)$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in C^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$

Fig. 1. Syntax and semantics of $\mathcal{ALCQIO}$

The family of action formalisms introduced in this paper can be parameterised with any description logic. We show that, for many standard DLs, the reasoning problems *executability* and *projection* in the corresponding action formalism are decidable. We also pinpoint the exact computational complexity of these reasoning problems. As a rule of thumb, our results show that reasoning in the action formalism instantiated with a description logic $\mathcal{L}$ is of the same complexity as standard reasoning in $\mathcal{L}$ extended with nominals (which correspond to first-order constants [1]). For fine-tuning ramifications, deciding the consistency of actions is of prime importance. We introduce two notions of consistency (weak and strong) and show that one of them is of the same complexity as deciding projection while the other one is undecidable even when the action formalism is instantiated with the basic DL $\mathcal{ALC}$. Details regarding the technical results can be found in the report [7].

2 Description Logics

In DLs, *concepts* are inductively defined with the help of a set of *constructors*, starting with a set $\mathsf{N_C}$ of *concept names*, a set $\mathsf{N_R}$ of *role names*, and (possibly) a set $\mathsf{N_I}$ of *individual names*. In this section, we introduce the DL $\mathcal{ALCQIO}$, whose concepts are formed using the constructors shown in Figure 1. There, the inverse constructor is the only role constructor, whereas the remaining six constructors are concept constructors. In Figure 1 and throughout this paper, we use $\#S$ to denote the cardinality of a set S, a and b to denote individual names, r and s to denote roles (i.e., role names and inverses thereof), A, B to denote concept names, and C, D to denote (possibly complex) concepts. As usual, we use $\top$ as abbreviation for an arbitrary (but fixed) propositional tautology, $\bot$ for $\neg\top$, $\rightarrow$ and $\leftrightarrow$ for the usual Boolean abbreviations, $\exists r.C$ (*existential restriction*) for $(\geqslant 1\ r\ C)$, and $\forall r.C$ (*universal restriction*) for $(\leqslant 0\ r\ \neg C)$.

The DL that allows only for negation, conjunction, disjunction, and universal and existential restrictions is called $\mathcal{ALC}$. The availability of additional constructors is indicated by concatenation of a corresponding letter: $\mathcal{Q}$ stands for number restrictions; $\mathcal{I}$ stands for inverse roles, and $\mathcal{O}$ for nominals. This explains

the name $\mathcal{ALCQIO}$ for our DL, and also allows us to refer to its sublanguages in a simple way.

The semantics of $\mathcal{ALCQIO}$-concepts is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set of individuals and the *interpretation function* $\cdot^{\mathcal{I}}$ maps each concept name $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name $r \in \mathsf{N_R}$ to a binary relation $r^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, and each individual name $a \in \mathsf{N_I}$ to an individual $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to inverse roles and arbitrary concepts is inductively defined as shown in the third column of Figure 1.

A *general concept inclusion axiom (GCI)* is an expression of the form $C \sqsubseteq D$, where C and D are concepts. A *(general) TBox* $\mathcal{T}$ is a finite set of GCIs. An ABox is a finite set of *concept assertions* $C(a)$ and *role assertions* $r(a, b)$ and $\neg r(a, b)$ (where r may be an inverse role). An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, a concept assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, a role assertion $r(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, and a role assertion $\neg r(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$. We denote satisfaction of a GCI $C \sqsubseteq D$ by an interpretation $\mathcal{I}$ with $\mathcal{I} \models C \sqsubseteq D$, and similar for ABox assertions. An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ (written $\mathcal{I} \models \mathcal{T}$) iff it satisfies all GCIs in $\mathcal{T}$. It is a *model* of an ABox $\mathcal{A}$ (written $\mathcal{I} \models \mathcal{A}$) iff it satisfies all assertions in $\mathcal{A}$.

A concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff $C^{\mathcal{I}} \neq \emptyset$ for some model $\mathcal{I}$ of $\mathcal{T}$. An ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$ iff $\mathcal{A}$ and $\mathcal{T}$ have a common model.

3 Describing Actions

The action formalism proposed in this paper is not restricted to a particular DL. However, for our complexity results we consider the DL $\mathcal{ALCQIO}$ and its sublogics. In the following, we use $\mathcal{LO}$ to denote the result of extending the DL $\mathcal{L}$ with nominals. A *concept literal* is a concept name or the negation thereof, and a *role literal* is defined analogously.

Definition 1 (Action). *Let $\mathcal{L}$ be a description logic. An* $\mathcal{L}$-action $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ *consists of*

- *a finite set* pre *of $\mathcal{L}$ ABox assertions, the* pre-conditions*;*
- *the* occlusion pattern occ *which is a set of mappings* $\{\mathsf{occ}_{\varphi_1}, \ldots, \mathsf{occ}_{\varphi_n}\}$ *indexed by $\mathcal{L}$ ABox assertions $\varphi_1, \ldots, \varphi_n$ such that each* occ_{φ_i} *assigns*
  - *to every concept literal B an $\mathcal{LO}$-concept* $\mathsf{occ}_{\varphi_i}(B)$*,*
  - *to every role literal s a finite set* $\mathsf{occ}_{\varphi_i}(s)$ *of pairs of $\mathcal{LO}$-concepts.*
- *a finite set* post *of* conditional post-conditions *of the form φ/ψ, where φ and ψ are $\mathcal{L}$ ABox assertions.*

Intuitively, the pre-conditions specify under which conditions the action is applicable. The post-condition φ/ψ says that, if φ is true before executing the action, then ψ should be true afterwards. The purpose of the occlusion pattern is to control ramifications: they provide a description of where concept and role names may change during the execution of an action. More precisely, suppose

$\mathsf{occ} = \{\mathsf{occ}_{\varphi_1}, \ldots, \mathsf{occ}_{\varphi_n}\}$ and $\varphi_{i_1}, \ldots, \varphi_{i_m}$ are the assertions which are true before the action was executed. If A is a concept name, then instances of the concept

$$\mathsf{occ}_{\varphi_{i_1}}(A) \sqcup \cdots \sqcup \mathsf{occ}_{\varphi_{i_m}}(A)$$

may change from A to $\neg A$ during the execution of the action provided, but instances of $\neg(\mathsf{occ}_{\varphi_{i_1}}(A) \sqcup \cdots \sqcup \mathsf{occ}_{\varphi_{i_m}}(A))$ may not. Likewise, instances of

$$\mathsf{occ}_{\varphi_{i_1}}(\neg A) \sqcup \cdots \sqcup \mathsf{occ}_{\varphi_{i_m}}(\neg A)$$

may change from $\neg A$ to A. For role names, $(C, D) \in \mathsf{occ}_{\varphi_{i_k}}(r)$ means that pairs from $C^{\mathcal{I}} \times D^{\mathcal{I}}$ that have been connected by r before the action may lose this connection through the execution of the action, and similarly for the occlusion of negated role names. Before giving more details on how occlusions relate to ramifications, we introduce the semantics of actions. To this end, it is convenient to introduce the following abbreviation. For an action α with $\mathsf{occ} = \{\mathsf{occ}_{\varphi_1}, \ldots, \mathsf{occ}_{\varphi_n}\}$, an interpretation $\mathcal{I}$, a concept literal B, and a role literal s, we set

$$\mathsf{occ}(B)^{\mathcal{I}} := \bigcup_{\mathcal{I} \models \varphi_i} (\mathsf{occ}_{\varphi_i}(B))^{\mathcal{I}} \qquad \mathsf{occ}(s)^{\mathcal{I}} := \bigcup_{(C,D) \in \mathsf{occ}_{\varphi_i}(s), \mathcal{I} \models \varphi_i} (C^{\mathcal{I}} \times D^{\mathcal{I}}).$$

Definition 2 (Action semantics). *Let $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ be an action and $\mathcal{I}, \mathcal{I}'$ interpretations sharing the same domain and interpretation of all individual names. We say that α* may transform *$\mathcal{I}$ to $\mathcal{I}'$ w.r.t. a TBox $\mathcal{T}$ ($\mathcal{I} \Rightarrow_\alpha^{\mathcal{T}} \mathcal{I}'$) iff the following holds:*

- *$\mathcal{I}, \mathcal{I}'$ are models of $\mathcal{T}$;*
- *for all $\varphi/\psi \in \mathsf{post}$: $\mathcal{I} \models \varphi$ implies $\mathcal{I}' \models \psi$ (written $\mathcal{I}, \mathcal{I}' \models \mathsf{post}$);*
- *for each $A \in \mathsf{N_C}$ and $r \in \mathsf{N_R}$, we have*

$$A^{\mathcal{I}} \setminus A^{\mathcal{I}'} \subseteq (\mathsf{occ}(A))^{\mathcal{I}} \qquad \neg A^{\mathcal{I}} \setminus \neg A^{\mathcal{I}'} \subseteq (\mathsf{occ}(\neg A))^{\mathcal{I}}$$
$$r^{\mathcal{I}} \setminus r^{\mathcal{I}'} \subseteq (\mathsf{occ}(r))^{\mathcal{I}} \qquad \neg r^{\mathcal{I}} \setminus \neg r^{\mathcal{I}'} \subseteq (\mathsf{occ}(\neg r))^{\mathcal{I}}$$

Let us explain how occlusions provide a way to control the ramifications induced by general TBoxes by reconsidering the example from the introduction. The TBox $\mathcal{T}$ contains the following GCIs which say that everybody registered for a course has access to a university library, and that every university has a library:

$$\exists \mathsf{registered\_for}.\mathsf{Course} \sqsubseteq \exists \mathsf{access\_to}.\mathsf{Library}$$
$$\mathsf{University} \sqsubseteq \exists \mathsf{has\_facility}.\mathsf{Library}$$

This GCI cannot be expressed in terms of an acyclic TBox and is thus outside the scope of the formalism in [2]. The ABox $\mathcal{A}$ which describes the current state of the world says that computer science is a course held at TU Dresden, SLUB is the library of TU Dresden, and Dirk is neither registered for a course nor has access to a library:

$\mathsf{Course}(\mathsf{cs})$	$\mathsf{held\_at}(\mathsf{cs}, \mathsf{tud})$	$\neg\exists \mathsf{registered\_for}.\mathsf{Course}(\mathsf{dirk})$
$\mathsf{University}(\mathsf{tud})$	$\mathsf{has\_facility}(\mathsf{tud}, \mathsf{slub})$	$\neg\exists \mathsf{access\_to}.\mathsf{Library}(\mathsf{dirk})$
$\mathsf{Library}(\mathsf{slub})$		

The action

$$\alpha := (\emptyset, \mathsf{occ}, \{\mathsf{taut}/\mathsf{registered\_for}(\mathsf{dirk}, \mathsf{cs})\})$$

describes the registration of Dirk for the computer science course. For simplicity, the set of pre-conditions is empty and taut is some ABox assertion that is trivially satisfied, say $\top(\mathsf{cs})$. To obtain occ, we may start by strictly following the law of inertia, i.e., requiring that the only changes are those that are explicitly stated in the post-condition. Thus, occ consists of just one mapping $\mathsf{occ}_{\mathsf{taut}}$ such that

$$\mathsf{occ}_{\mathsf{taut}}(\neg\mathsf{registered\_for}) := \{(\{\mathsf{dirk}\}, \{\mathsf{cs}\})\}$$

and all concept and role literals except $\neg$registered_for are mapped to $\bot$ and $\{(\bot, \bot)\}$, respectively. This achieves the desired effect that only the pair (dirk, cs) can be added to "registered_for" and nothing else can be changed.

It is not hard to see that this attempt to specify occlusions for α is too strict. Intuitively, not allowing any changes is appropriate for Course, Library, University, held_at, has_facility and their negations since the action should have no impact on these predicates. However, not letting $\neg$access_to change leads to a problem with the ramifications induced by the TBox: as Dirk has no access to a library before the action and $\neg$access_to is not allowed to change, he cannot have access to a library after execution of the action as required by the TBox. Thus, the action is inconsistent in the following sense: there is no model $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$ and model $\mathcal{I}'$ of $\mathcal{T}$ such that $\mathcal{I} \Rightarrow_\alpha^{\mathcal{T}} \mathcal{I}'$. To take care of the TBox ramifications and regain consistency, we can modify occ. One option is to set

$$\mathsf{occ}_{\mathsf{taut}}(\neg\mathsf{access\_to}) := \{(\{\mathsf{dirk}\}, \mathsf{Library})\}$$

and thus allow Dirk to have access to a library after the action. Another option is to set

$$\mathsf{occ}_{\mathsf{taut}}(\neg\mathsf{access\_to}) := \{(\{\mathsf{dirk}\}, \mathsf{slub})\}$$

which allows Dirk to have access to SLUB after the action, but not to any other library.

Two remarks regarding this example are in order. First, the occlusion occ consists only of a single mapping $\mathsf{occ}_{\mathsf{taut}}$. The reason for this is that there is only a single post-condition in the action. If we have different post-conditions φ/ψ and φ'/ψ such that φ and φ' are not equivalent, there will usually be different occlusion mappings (indexed with φ and φ') to deal with the ramifications that the TBox induces for these post-conditions. Second, the example explains the need for extending $\mathcal{L}$ to $\mathcal{LO}$ when describing occlusions (c.f. Definition 1): without nominals, we would not have been able to properly formulate the occlusions although all other parts of the example are formulated without using nominals (as a concept-forming operator).

As illustrated by the example, it is important for the action designer to decide consistency of actions to detect ramification problems that are not properly addressed by the occlusions. In the following, we propose two notions of consistency.

Definition 3 (Consistency). *Let α be an action, $\mathcal{T}$ a TBox, and $\mathcal{A}$ an ABox. We say that*

- *α is* weakly consistent with $\mathcal{T}$ and $\mathcal{A}$ *iff there is a model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$, and a model $\mathcal{I}'$ of $\mathcal{T}$ such that $\mathcal{I} \Rightarrow_\alpha^\mathcal{T} \mathcal{I}'$.*
- *α is* strongly consistent with $\mathcal{T}$ and $\mathcal{A}$ *iff for all models $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$, there is a model $\mathcal{I}'$ of $\mathcal{T}$ such that $\mathcal{I} \Rightarrow_\alpha^\mathcal{T} \mathcal{I}'$.*

Clearly, weak consistency implies strong consistency but not vice versa. In the example above, the first attempt to define the occlusions results in an action that is not even weakly consistent. After each of the two possible modifictions, the action is strongly consistent. We will see later that weak consistency is decidable while strong consistency is not.

To check whether an action can be applied in a given situation, the user wants to know whether it is executable, i.e., whether all pre-conditions are satisfied in the states of the world considered possible. If the action is executable, he wants to know whether applying it achieves the desired effect, i.e., whether an assertion that he wants to make true really holds after executing the action. These two problems are called executability and projection [10, 2].

Definition 4 (Executability and projection). *Let $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ be an action, $\mathcal{T}$ a TBox, and $\mathcal{A}$ an ABox.*

- *Executability: α is* executable in $\mathcal{A}$ w.r.t. $\mathcal{T}$ *iff $\mathcal{I} \models \mathsf{pre}$ for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$;*
- *Projection: The assertion φ is a* consequence of applying α in $\mathcal{A}$ w.r.t. $\mathcal{T}$ *iff for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$ and for all $\mathcal{I}'$ with $\mathcal{I} \Rightarrow_\alpha^\mathcal{T} \mathcal{I}'$, we have $\mathcal{I}' \models \varphi$.*

It is not difficult to see that the action formalism just introduced is a generalisation of the one introduced in [2] when composite actions are disallowed, for details see [7]. Clearly, executability can be polynomially reduced to *ABox consequence* which is defined as follows: given an ABox $\mathcal{A}$ and an assertion φ, decide whether $\mathcal{I}$ satisfies φ in all models $\mathcal{I}$ of $\mathcal{A}$. The complexity of this problem is extensively discussed in [2]. For example, it is NExpTime-complete for $\mathcal{ALCQIO}$ and ExpTime-complete for $\mathcal{ALC}$ extended with at most two of $\mathcal{Q}$, $\mathcal{I}$, and $\mathcal{O}$.

It can also be seen that (i) an action α is weakly consistent with a TBox $\mathcal{T}$ and ABox $\mathcal{A}$ iff $\bot(a)$ is not a consequence of applying α in $\mathcal{A}$ w.r.t. $\mathcal{T}$; (ii) φ is a consequence of applying $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ in $\mathcal{A}$ w.r.t. $\mathcal{T}$ iff the action $(\mathsf{pre}, \mathsf{occ}, \mathsf{post} \cup \{\top(a)/\neg\varphi\})$ is not weakly consistent with $\mathcal{T}$ and $\mathcal{A}$. Thus, weak consistency can be reduced to (non-)projection and vice versa and complexity results carry over from one to the other. In this paper, we will concentrate on projection.

4 Projection in ExpTime

We show that projection and weak consistency are ExpTime-complete for DL actions formulated in $\mathcal{ALC}$, $\mathcal{ALCO}$, $\mathcal{ALCI}$, $\mathcal{ALCIO}$. Thus, in these DLs reasoning about actions is not more difficult than the standard DL reasoning problems

such as concept satisfiability and subsumption w.r.t. TBoxes. The complexity results established in this section are obtained by proving that projection in $\mathcal{ALCIO}$ is in ExpTime. We use a Pratt-style type elimination technique as first proposed in [8].

In the following, we assume that the set occ of occlusions of an action consists of only one mapping $\mathsf{occ}_{\mathsf{taut}}$, where taut is $\top(a)$. We will identify occ with the mapping $\mathsf{occ}_{\mathsf{taut}}$ and write $\mathsf{occ}(X)$ instead of $\mathsf{occ}_{\mathsf{taut}}(X)$. Proofs are easily extended to actions containing general occlusions, see [7].

Let $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ be an action, $\mathcal{T}$ a TBox, $\mathcal{A}_0$ an ABox and φ_0 an assertion. We want to decide whether φ_0 is a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$. In what follows, we call α, $\mathcal{T}$, $\mathcal{A}_0$ and φ_0 the *input*. W.l.o.g., we make the following assumptions:

- concepts used in the input are built only from the constructors $\{a\}$, $\neg$, $\sqcap$, and $\exists r.C$;
- φ_0 is of the form $\varphi_0 = C_0(a_0)$, where C_0 is a (complex) concept;
- $\mathcal{A}_0$ and α contain only concept assertions.

The last two assumptions can be made because every assertion $r(a, b)$ can be replaced with $(\exists r.\{b\})(a)$, and every $\neg r(a, b)$ with $(\neg\exists r.\{b\})(a)$.

Before we can describe the algorithm, we introduce a series of notions and abbreviations. With Sub, we denote the set of subconcepts of the concepts which occur in the input. With Ind, we denote the set of individual names used in the input, and set $\mathsf{Nom} := \{\{a\} \mid a \in \mathsf{Ind}\}$.

The algorithm for deciding projection checks for the existence of a countermodel witnessing that φ_0 is *not* a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$. Such a countermodel consists of interpretations $\mathcal{I}$ and $\mathcal{I}'$ such that $\mathcal{I} \models \mathcal{A}_0$, $\mathcal{I} \Rightarrow^{\mathcal{T}} \mathcal{I}'$, and $\mathcal{I}' \not\models \varphi_0$. To distinguish the extension of concept and role names in $\mathcal{I}$ and $\mathcal{I}'$, we introduce concept names A' and role names r' for every concept name A and role name r used in the input. For a concept $C \in \mathsf{Sub}$ that is not a concept name, we use C' to denote the concept obtained by replacing all concept names A and role names r occurring in C by A' and r', respectively. We define the set of concepts Cl as:

$$\mathsf{Cl} = \{C, \neg C, C', \neg C' \mid C \in \mathsf{Sub} \cup \mathsf{Nom}\}.$$

The notion of a type plays a central role in the projection algorithm to be devised.

Definition 5. *A set of concepts $t \subseteq \mathsf{Cl}$ is a* type *for* Cl *iff it satisfies the following conditions:*

- *for all $\neg D \in \mathsf{Cl}$: $\neg D \in t$ iff $D \notin t$;*
- *for all $D \sqcap E \in \mathsf{Cl}$: $D \sqcap E \in t$ iff $\{D, E\} \subseteq t$;*
- *for all $C \sqsubseteq D \in \mathcal{T}$, $C \in t$ implies $D \in t$ and $C' \in t$ implies $D' \in t$;*
- *for all concept names A, $\{A, \neg A'\} \subseteq t$ implies that $\mathsf{occ}(A) \in t$ and $\{\neg A, A'\} \subseteq t$ implies that $\mathsf{occ}(\neg A) \in t$.*

A type is anonymous *if it does not contain a nominal. Let* $\mathfrak{T}_{\mathsf{ano}}$ *be the set of all anonymous types.*

Intuitively, a type describes the concept memberships of a domain element in the interpretations $\mathcal{I}$ and $\mathcal{I}'$. Our algorithm starts with a set containing (almost) all types, then repeatedly eliminates those types that cannot be realized in a countermodel witnessing that φ_0 is not a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$, and finally checks whether the surviving types give rise to such a countermodel. The picture is slightly complicated by the presence of ABoxes and nominals. These are treated via core type sets to be introduced next.

Definition 6. $\mathfrak{T}_S$ *is a* core type set *iff* $\mathfrak{T}_S$ *is a minimal set of types such that, for all* $a \in \mathsf{Ind}$*, there is a* $t \in \mathfrak{T}_S$ *with* $\{a\} \in \mathfrak{T}_S$*.*

A core type set $\mathfrak{T}_S$ *is called* proper *if the following conditions are satisfied:*

1. *for all* $C(a) \in \mathcal{A}_0$*,* $\{a\} \in t \in \mathfrak{T}_S$ *implies* $C \in t$*;*
2. *for all* $C(a)/D(b) \in \mathsf{post}$*: if there is a* $t \in \mathfrak{T}_S$ *with* $\{\{a\}, C\} \subseteq t$ *then there is a* $t' \in \mathfrak{T}_S$ *with* $\{\{b\}, D'\} \subseteq t'$*.*

Intuitively, a core type set carries information about the "named" part of the interpretations $\mathcal{I}_0$ and $\mathcal{I}_1$, where the named part of an interpretation consists of those domain elements that are identified by nominals. Let m be the size of the input. It is not difficult to check that the number of core type sets is exponential in m. Also, checking whether a core type set is proper can be done in linear time.

The following definition specifies the conditions under which a type is eliminated. For a role name r, we set $\mathsf{occ}(r^-) := \{(Y, X) \mid (X, Y) \in \mathsf{occ}(r)\}$, and analogously for $\mathsf{occ}(\neg r^-)$. For role names r, we set $\mathsf{Inv}(r) := r^-$ and $\mathsf{Inv}(r^-) := r$.

Definition 7. *Let* $\mathfrak{T}$ *be a set of types for* CI*. Then a type* $t \in \mathfrak{T}$ *is* good in $\mathfrak{T}$ *iff the following condition is satisfied for all roles* r*: if* $\exists r.C_1, \ldots, \exists r.C_k$ *and* $\exists r'.D'_1, \ldots, \exists r'.D'_m$ *are all concepts of this form in* t*, then there exist types* $t_1, \ldots, t_n \in \mathfrak{T}$ *and sets* $\rho_1, \ldots, \rho_n \subseteq \{0, 1\}$*,* $n \leq k + m$*, such that*

- *for* $1 \leq j \leq k$*, there is an* $\ell \in \{1, \ldots, n\}$ *such that* $C_j \in t_\ell$ *and* $0 \in \rho_\ell$*;*
- *for* $1 \leq j \leq m$*, there is an* $\ell \in \{1, \ldots, n\}$ *such that* $D'_j \in t_\ell$ *and* $1 \in \rho_\ell$*;*
- *if* $\neg\exists r.C \in t$ *and* $0 \in \rho_j$*, then* $\neg C \in t_j$*;*
- *if* $\neg\exists r'.D' \in t$ *and* $1 \in \rho_j$*, then* $\neg D' \in t_j$*;*
- *if* $\neg\exists \mathsf{Inv}(r).C \in t_j$ *and* $0 \in \rho_j$*, then* $\neg C \in t$*;*
- *if* $\neg\exists \mathsf{Inv}(r').D' \in t_j$ *and* $1 \in \rho_j$*, then* $\neg D' \in t$*;*
- *if* $0 \in \rho_j$ *and* $1 \notin \rho_j$ *then there exists a pair* $(X, Y) \in \mathsf{occ}(r)$ *such that* $X \in t$ *and* $Y \in t_j$*,*
- *if* $0 \notin \rho_j$ *and* $1 \in \rho_j$ *then there exists a pair* $(X, Y) \in \mathsf{occ}(\neg r)$ *such that* $X \in t$ *and* $Y \in t_j$*;*

Intuitively, the above definition checks whether there can be any instances of t in an interpretation in which all domain elements have a type in $\mathfrak{T}$. More precisely, $t_1, \ldots, t_n$ are the types of r-successors that are needed to satisfy the existential restrictions in t. The sets $\rho_1, \ldots, \rho_n$ determine the extension of the role r: if

```
ALCIO-elim(A_0, T, α, φ_0)
    for all proper core type sets T_S do
        i := 0;
        T_0 := T_S ∪ T_ano
        repeat
            T_{i+1} := {t ∈ T_i | t is good in T_i};
            i := i + 1;
        until T_i = T_{i-1};
        if T_S ⊆ T_i and there is a t ∈ T_i with {{a_0}, ¬C_0'} ⊆ t then
            return false
        endif
    endfor
    return true
```

Fig. 2. The type elimination algorithm

$0 \in \rho_j$, then the instance of t is connected to the r-successor of type t_j in $\mathcal{I}$, and similarly for $1 \in \rho_j$ and $\mathcal{I}'$.

The type elimination algorithm is given in a pseudo-code notation in Figure 2, where C_0 is the concept from the ABox assertion $\varphi_0 = C_0(a_0)$. A proof of the following lemma can be found in [7].

Lemma 1. *$\mathcal{ALCIO}$-elim$(\mathcal{A}_0, \mathcal{T}, \alpha, \varphi_0)$ returns true iff φ_0 is a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$.*

The algorithm runs in exponential time: first, we have already argued that there are only exponentially many core type sets. Second, the number of elimination rounds is bounded by the number of types, of which there are only exponentially many. And third, it is easily seen that it can be checked in exponential time whether a type is good in a given type set. Since concept satisfiability w.r.t. TBoxes is ExpTime-hard in $\mathcal{ALC}$ [3] and concept satisfiability can be reduced to (non-)projection [2], we obtain the following result.

Theorem 1. *Projection and weak consistency are* ExpTime*-complete in $\mathcal{ALC}$, $\mathcal{ALCO}$, $\mathcal{ALCI}$, and $\mathcal{ALCIO}$.*

It is not too difficult to adapt the algorithm given in this section to the DL $\mathcal{ALCQO}$. Therefore, we conjecture that the reasoning problems from Theorem 1 are also ExpTime-complete for $\mathcal{ALCQ}$ and $\mathcal{ALCQO}$.

5 $\mathcal{ALCQI}$ and $\mathcal{ALCQIO}$: Beyond ExpTime

In the previous section, we have identified a number of DLs for which both reasoning about actions and standard DL reasoning are ExpTime-complete. Another candidate for a DL with such a behaviour is $\mathcal{ALCQI}$, in which satisfiability and subsumption are ExpTime-complete as well [15]. However, it follows from results in [2] that projection in $\mathcal{ALCQI}$ is co-NExpTime-hard. In the following, we show that it is in fact co-NExpTime-complete, and that the same holds for

the DL $\mathcal{ALCQIO}$. Note that, for the latter DL, also concept subsumption is co-NExpTime-complete.

It is shown in [7] that Lemma 8 of [2] implies the following.

Theorem 2. *Projection (weak consistency) in $\mathcal{ALCQI}$ is co-*NExpTime*-hard (*NExpTime*-hard) even if occlusions for role literals are restricted to $(\bot, \bot)$ and occlusions of concept literals are restricted to $\bot$ and nominals.*

In the following, we establish a co-NExpTime upper bound for projection in $\mathcal{ALCQIO}$ (and thus also $\mathcal{ALCQI}$). The proof proceeds by reducing projection in $\mathcal{ALCQIO}$ to ABox (in)consistency in $\mathcal{ALCQIO}^{\neg,\cup,\cap}$, the extension of $\mathcal{ALCQIO}$ with the Boolean role constructors complement, union, and intersection.

Let α be an action, $\mathcal{T}$ a TBox, $\mathcal{A}_0$ an ABox and φ_0 an assertion. We are interested in deciding whether φ_0 is a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$. We use the same notions and abbreviations as in Section 4. As in that section, we also assume that φ_0 is of the form $C_0(a_0)$ and that occlusions are of a restricted form.

The idea for the following reduction is to define an ABox $\mathcal{A}_{\mathsf{red}}$ and a TBox $\mathcal{T}_{\mathsf{red}}$ such that each model of $\mathcal{A}_{\mathsf{red}}$ and $\mathcal{T}_{\mathsf{red}}$ encodes interpretations $\mathcal{I}$ and $\mathcal{I}'$ with $\mathcal{I} \models \mathcal{A}_0$ and $\mathcal{I} \Rightarrow_\alpha^{\mathcal{T}} \mathcal{I}'$, and $\mathcal{I}' \not\models \varphi_0$. The encoding of the two interpretations $\mathcal{I}$ and $\mathcal{I}'$ into a single model of $\mathcal{A}_{\mathsf{red}}$ and $\mathcal{T}_{\mathsf{red}}$ is similar to what was done in the previous section: we introduce a non-primed and a primed version of each concept and role name to distinguish the extension in $\mathcal{I}$ from that in $\mathcal{I}'$. We start by assembling the reduction ABox $\mathcal{A}_{\mathsf{red}}$. First, introduce abbreviations:

$$\begin{aligned} \mathsf{p}(C(a)) &:= \forall U.(\{a\} \to C), \\ \mathsf{p}(r(a,b)) &:= \forall U.(\{a\} \to \exists r.\{b\}), \\ \mathsf{p}(\neg r(a,b)) &:= \forall U.(\{a\} \to .\forall r.\neg\{b\}), \end{aligned}$$

where U denotes the universal role, i.e. $r \cup \neg r$ for some $r \in \mathsf{N_R}$. Now we can define the components of $\mathcal{A}_{\mathsf{red}}$ that take care of post-condition satisfaction. We define:

$$\mathcal{A}_{\mathsf{post}} := \{\big(\mathsf{p}(\varphi) \to \mathsf{p}(\psi')\big)(a_0) \mid \varphi/\psi \in \mathsf{post}\},$$

where ψ' is obtained from ψ by replacing concepts C and role names r in ψ by C' and r' respectively. We assemble $\mathcal{A}_{\mathsf{red}}$ as

$$\mathcal{A}_{\mathsf{red}} := \mathcal{A}_0 \cup \mathcal{A}_{\mathsf{post}}.$$

We continue by defining the components of the TBox $\mathcal{T}_{\mathsf{red}}$. The first component ensures that auxiliary role names $r_{\mathsf{Dom}(C)}$ and $r_{\mathsf{Ran}(D)}$ are interpreted as $C \times \top$ and $\top \times D$, respectively. For every $(C, D) \in \mathsf{occ}(s)$ for some role literal s from the input, the TBox $\mathcal{T}_{\mathsf{aux}}$ contains the following GCIs :

$$\begin{aligned} C &\sqsubseteq \forall \neg r_{\mathsf{Dom}(C)}.\bot & \top &\sqsubseteq \forall r_{\mathsf{Ran}(D)}.D \\ \neg C &\sqsubseteq \forall r_{\mathsf{Dom}(C)}.\bot & \top &\sqsubseteq \forall \neg r_{\mathsf{Ran}(D)}.\neg D \end{aligned}$$

The following component describes the behaviour of concept names and role names in parts of the domain where they are *not* allowed to vary. The TBox $\mathcal{T}_{\mathsf{fix}}$

contains for every concept name A in the input,

$$\neg\mathsf{occ}(A) \sqcap A \sqsubseteq A'$$
$$\neg\mathsf{occ}(\neg A) \sqcap \neg A \sqsubseteq \neg A'$$

and for every role name r in the input,

$$\top \sqsubseteq \forall\neg\Big(\bigcup_{(C,D)\in\mathsf{occ}(r)} (r_{\mathsf{Dom}(C)} \cap r_{\mathsf{Ran}(D)})\Big) \cap (r \cap \neg r').\bot$$
$$\top \sqsubseteq \forall\neg\Big(\bigcup_{(C,D)\in\mathsf{occ}(\neg r)} (r_{\mathsf{Dom}(C)} \cap r_{\mathsf{Ran}(D)})\Big) \cap (\neg r \cap r').\bot$$

Finally, we can construct $\mathcal{T}_{\mathsf{red}}$ as

$$\mathcal{T}_{\mathsf{red}} := \mathcal{T}_{\mathsf{aux}} \cup \mathcal{T}_{\mathsf{fix}} \cup \mathcal{T} \cup \{C' \sqsubseteq D' \mid C \sqsubseteq D \in \mathcal{T}\}.$$

The last two components of $\mathcal{T}_{\mathsf{red}}$ ensure that $\mathcal{I}$ and $\mathcal{I}'$ are models of the input TBox $\mathcal{T}$. It is not difficult to show that the following holds:

Lemma 2. *$C_0(a_0)$ is a consequence of applying α in $\mathcal{A}_0$ w.r.t. $\mathcal{T}$ iff $\mathcal{A}_{\mathsf{red}} \cup \{\neg C_0'(a_0)\}$ is inconsistent w.r.t. $\mathcal{T}_{\mathsf{red}}$.*

Since $\mathcal{ALCQIO}^{\cup,\cap,\neg}$ is a fragment of $\mathcal{C}^2$ (the 2-variable fragment of first-order logic with counting), we have that ABox inconsistency in $\mathcal{ALCQIO}^{\cup,\cap,\neg}$ is in co-NExpTime, even if numbers are coded in binary [9]. Since $\mathcal{A}_{\mathsf{red}}$ and $\mathcal{T}_{\mathsf{red}}$ are polynomial in the size of the input ABox $\mathcal{A}_0$, TBox $\mathcal{T}$, and action α, Lemma 2 gives us the same upper complexity bound for projection in $\mathcal{ALCQIO}$ and $\mathcal{ALCQI}$. Theorem 2 implies that this is a tight complexity bound:

Theorem 3. *In $\mathcal{ALCQIO}$, projection is co-*NExpTime*-complete and weak consistency is* NExpTime*-complete.*

6 Undecidability of Strong Consistency

We show that strong consistency is undecidable already in $\mathcal{ALC}$. The proof consists of a reduction of the undecidable *semantic consequence problem* from modal logic. Before formulating the DL version of this problem, we need some preliminaries. We use $\mathcal{ALC}$ concepts with only one fixed role name r, which we call $\mathcal{ALC}_r$-concepts. Accordingly, we also assume that interpretations interpret only concept names and the role name r. A *frame* is a structure $\mathcal{F} = (\Delta^{\mathcal{F}}, r^{\mathcal{F}})$ where $\Delta^{\mathcal{F}}$ is a non-empty set and $r^{\mathcal{F}} \subseteq \Delta^{\mathcal{F}} \times \Delta^{\mathcal{F}}$. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is *based* on a frame $\mathcal{F}$ iff $\Delta^{\mathcal{I}} = \Delta^{\mathcal{F}}$ and $r^{\mathcal{I}} = r^{\mathcal{F}}$. We say that a concept C is *valid* on $\mathcal{F}$ (written $\mathcal{F} \models C$) iff $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ based on $\mathcal{F}$.

Definition 8 (Semantic consequence problem). *Let D and E be $\mathcal{ALC}_r$-concepts. We say that E is a* semantic consequence *of D iff for every frame $\mathcal{F} = (\Delta^{\mathcal{F}}, r^{\mathcal{F}})$ such that $\mathcal{F} \models D$, it holds that $\mathcal{F} \models E$.*

In [14], it is proved that for $\mathcal{ALC}_r$-concepts D and E, the problem "Is E a semantic consequence of D?" is undecidable. We now show that the semantic consequence problem can be reduced to strong consistency. For $\mathcal{ALC}_r$-concepts D and E, we define the ABox $\mathcal{A}_E := \{\neg E(a)\}$ and the atomic action $\alpha_D = (\emptyset, \{\mathsf{occ_{taut}}\}, \mathsf{post})$ with $\mathsf{post} := \{\top(a)/(\exists u.\neg D)(a)\}$ where u is an arbitrary role name and $\mathsf{occ_{taut}}$ maps r and $\neg r$ to $\{(\bot, \bot)\}$, all other role literals to $\{(\top, \top)\}$, and all concept literals to $\top$. Then the following holds:

Lemma 3. *The action α_D is strongly consistent with the empty TBox and the ABox $\mathcal{A}_E$ iff E is a semantic consequence of D.*

Proof. "$\Rightarrow$" We show the contraposition. Assume that E is not a semantic consequence of D. Then there exists a frame $\mathcal{F} = (\Delta^{\mathcal{F}}, r^{\mathcal{F}})$ such that $\mathcal{F} \models D$ and there is an interpretation $\mathcal{I}$ based on $\mathcal{F}$ such that $E^{\mathcal{I}} \neq \Delta^{\mathcal{I}}$. We take $\mathcal{I}$ based on $\mathcal{F}$ such that $a^{\mathcal{I}} \notin E^{\mathcal{I}}$, thus $\mathcal{I} \models \mathcal{A}_E$. But every $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow^{\emptyset}_{\alpha_D} \mathcal{I}'$ must be based on $\mathcal{F}$ (since $r^{\mathcal{I}'} = r^{\mathcal{I}} = r^{\mathcal{F}}$) and must satisfy $D^{\mathcal{I}'} \neq \Delta^{\mathcal{I}'}$ (by post-condition of α). Since $\mathcal{F} \models D$, there is no such $\mathcal{I}'$. Thus, α_D is not strongly consistent with the empty TBox and the ABox $\mathcal{A}_E$.

"$\Leftarrow$" Assume that E is a semantic consequence of D. Let $\mathcal{I} \models \mathcal{A}_E$. By definition of $\mathcal{A}_E$, we have that $a^{\mathcal{I}} \notin E^{\mathcal{I}}$, and thus $\mathcal{I}$ is not based on a frame $\mathcal{F} = (\Delta^{\mathcal{F}}, r^{\mathcal{F}})$ validating E. Since E is a semantic consequence of D, $\mathcal{F}$ is not validating D either, and there is an interpretation $\mathcal{I}'$ based on $\mathcal{F}$ such that $D^{\mathcal{I}'} \neq \Delta^{\mathcal{I}'}$. Take $y \in \Delta^{\mathcal{I}'}$ such that $y \notin D^{\mathcal{I}'}$. Since D is an $\mathcal{ALC}_r$- concept, we may assume that $u^{\mathcal{I}'} = \{(a^{\mathcal{I}'}, y)\}$. Obviously, we have that $\mathcal{I} \Rightarrow^{\emptyset}_{\alpha_D} \mathcal{I}'$, and, consequently, α_D is strongly consistent with the empty TBox and $\mathcal{A}_E$.

As an immediate consequence, we obtain the following theorem.

Theorem 4. *Strong consistency of $\mathcal{ALC}$-actions is undecidable, even with the empty TBox.*

7 Discussion

We have introduced an action formalism based on description logics that admits general TBoxes and complex post-conditions. To deal with ramifications induced by general TBoxes, the formalism includes powerful occlusion patterns that can be used to fine-tune the ramifications. We believe that undecidability of strong consistency is no serious obstacle for the feasibility of our approach in practice. Although deciding strong consistency would provide valuable support for the designer of an action, it could not replace manual inspection of the ramifications. For example, occluding all concept names with $\top$ and all role names with $\{(\top, \top)\}$ usually ensures strong consistency but does not lead to an intuitive behaviour of the action. With weak consistency, we offer at least some automatic support to the action designer for detecting ramification problems.

Future work will include developing practical decision procedures. A first step is carried out in [7], where we show that in the following special (but natural) case, projection can be reduced to standard reasoning problems in DLs that are

implemented in DL reasoners such as RACER and FaCT++: (i) role occlusions in actions are given by $\mathsf{occ}_{\mathsf{taut}}$; (ii) $\mathsf{occ}_{\mathsf{taut}}(r) = \mathsf{occ}_{\mathsf{taut}}(\neg r)$; and (iii) concepts used in $\mathsf{occ}_{\mathsf{taut}}(r)$ are Boolean combinations of nominals,

Acknowledgements. We would like to thank Giuseppe De Giacomo for ideas and discussions. The second author is partially supported by the EU funded TONES project.The third author is supported by the DFG Graduiertenkolleg 334. The fourth author is partially supported by UK EPSRC grant no. GR/S63182/01.

References

1. C. Areces, P. Blackburn, and M. Marx. A road-map on complexity for hybrid logics. *Proc. of CSL-99*, number 1683 in LNCS, pages 307–321. Springer, 1999.
2. F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In *Proc. of AAAI-05*, AAAI Press, 2005.
3. F. Baader, D. L. McGuiness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2003.
4. G. de Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. *Proc. of AAAI-06*, AAAI Press, 2006.
5. H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.
6. F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI-95*, pages 1985–1991, Morgan Kaufmann, 1995.
7. H. Liu, C. Lutz, M. Milicic, and F. Wolter. Description logic actions with general TBoxes: a pragmatic approach. LTCS-Report 06-03, TU Dresden, Germany, 2006. See http://lat.inf.tu-dresden.de/research/reports.html.
8. V. R. Pratt. Models of program logics. In *Proc. of the Twentieth FoCS*, San Juan, Puerto Rico, 1979.
9. I. Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *Journal of Logic, Language and Information*, 14(3):369–395, 2005.
10. R. Reiter. *Knowledge in Action*. MIT Press, 2001.
11. M. Thielscher. Ramification and causality. *Artificial Intelligence Journal*, 89(1–2):317–364, 1997.
12. M. Thielscher. Introduction to the Fluent Calculus. *Electronic Transactions on Artificial Intelligence*, 2(3–4):179–192, 1998.
13. M. Thielscher. FLUX: A logic programming method for reasoning agents. *TPLP*, 5(4-5):533–565, 2005.
14. S. K. Thomason. The logical consequence relation of propositional tense logic. *Z. Math. Logik Grundl. Math.*, 21:29–40, 1975.
15. S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *JAIR*, 12:199–217, 2000.
16. M. Winslett. Reasoning about action using a possible models approach. In *AAAI-88*, pages 89–93, 1988.
17. A list of DL reasoners: http://www.cs.man.ac.uk/∼sattler/reasoners.html

Introducing *Attempt* in a Modal Logic of Intentional Action*

Emiliano Lorini[1], Andreas Herzig[2], and Cristiano Castelfranchi[1]

[1] Institute of Cognitive Sciences and Technologies-CNR, Rome, Italy
[2] Institut de Recherche en Informatique de Toulouse (IRIT), France

Abstract. The main objective of this work is to develop a multi-modal logic of Intention and Attempt. We call this logic *LIA*. All formal results are focused on the notion of *attempt*. We substitute the dynamic molecular notion *action* by his atomic constituent *attempt* and define the former from the latter. The relations between attempts, goals, beliefs and present-directed intentions are studied. A section of the paper is devoted to the analysis of the relations of our modal logic with a situation calculus-style approach.

1 Introduction

BDI (belief, desire, intention) logics [21, 18, 4, 12] are conceived as explicit formal models of the intentional pursuit. If this is true then they should be able to take into account notions such as the notion of *attempt* and *trying*. These two notions have been mainly discussed in the philosophical field and taken into account in some logics of agency[1] but few models exist that are able to integrate in the same formal framework a precise description of practical reasoning (motivational dynamics and functional properties of mental states) with a description of its external and physical counterpart: the executive phase of intentional action. The main objective of this work is to develop a multi-modal logic which enables us to deal with the notion of *attempt* inside the more general framework of Intentional Action theory. We call this logic *LIA*: Logic of Intention and Attempt. The main difference between *LIA* and standard dynamic logic is that the dynamic primitives are not atomic actions, but atomic attempts. In our view a model of intentional action should explicitly represent the process of action execution, from the agent "triggering" the action to the successful execution of the action (when the preconditions for action execution hold). The axioms of the logic will be presented and discussed in Section 3. In section 4 the notions of *attempt* and *action* will be compared. It will be shown how action theories can be specified starting from the primitive notion of *attempt*. In section 5 additional properties of *attempt* will be discussed and the formal definition of present-directed intention will be introduced.

2 Properties of Attempts and Basic Actions

With "agent *i attempts to do an action* α", we mean that "agent *i triggers the execution of action* α", "agent *i exerts himself to do action* α". In our view the *attempt* is the core element of the causal process which leads from the present-directed intention [1] to the

* We thank the anonymous referees of this paper for their helpful comments.

[1] See for example [23] where attempts are defined as "not necessarily successful actions".

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 280–292, 2006.

successful execution of the action in the external world. Several authors have emphasized the importance of this concept for a theory of Intentional action (see [3], [13], [19]). The Psycho-Psycho Law proposed by O'Shaughnessy [19] stresses the "bridging role" of the *attempt* between the present-directed intention and the execution of the action: "if an agent at an instant in time realizes that that instant is an instant at which he intends to perform action x, then logically necessarily he begins trying to do x at that very moment of realization". In our view the property relating *attempts* with *actions* is the following one: *an action is effectively performed if and only if the performing agent triggers the action under the appropriate preconditions for action execution. The fact that the preconditions for action execution hold, guarantees that the attempt will be successful: it will succeed in producing the associated action (i.e. in causing the intrinsic result of the associated action*[2]). In the present analysis we deal with *basic actions* of a given agent *i* and leave aside the issue of *complex actions* (therefore every time we use the term *action* we mean *basic action*). According to [10, 5] an agent can perform a basic action α without necessarily performing some other action β and without necessarily believing that β must be performed in order to perform α. On the other hand a complex action (for example *Jack killing Joe*) depends on some other action (*Jack shooting Joe*) which in turn depends on some other action (*Jack pulling the trigger*) and so on... Thus we assume that basic actions are precisely those actions which are always executed when the agent attempts to perform them and the preconditions for action execution hold. Complex actions do not have this property. There could be an agent *i*'s complex action α whose execution also depends on some external event or some agent *j*'s action β which must happen after the initiation of action α (for example Jack can not perform the complex action of *killing Joe* if Joe does not perform the action of *drinking the poisoned soda* after that Jack *has poured some poison into the glass of soda*). Moreover we focus in this paper on "intentional attempts" assuming that an attempt to do α is always produced by the goal to attempt to do action α (see also [28] with respect to this hypothesis). Finally let us observe that there are two ways to conceive the notion of *attempt* (or *trying*). Some authors [16, 19] conceive the attempt as a purely mental event. If we adopt this perspective we should postulate that *unsuccessful attempts*[3] (differently from actions) never change the physical (external) world and are not perceivable by other agents. Other authors [13] are more prone to accept that *attempt* already refers to the physical realization of the basic action. In this analysis we adhere to the latter view and assume that the category *unsuccessful attempt* includes all those cases of "partial" execution of a basic action not producing the intrinsic result of the action (for example an agent who attempts to *raise the hand above the head* and only moves the arm of few millimeters since the arm is blocked). In [10, 5, 13] it is assumed that basic actions include only bodily movements such as *raising the arm*, *moving the leg*, *turning the sensor* etc... Thus in the examples given for supporting our analysis we will often refer to basic actions by using names denoting human bodily movements.

[2] According to [26, 29] the intrinsic result of an action is "the result which logically must occur if the action is to have been done". For instance the agent cannot have *opened his eye* unless *his eye is open.*

[3] With *unsuccessful attempt* we mean that the action that the attempt should produce is not performed due to the fact that the preconditions for executing the action do not hold.

Our analysis can be extended to realistic applications where the agent would be a robot with an artificial body (artificial limbs, rotating wheels, moving sensors etc...).

3 Formal Logic: Syntax, Semantics, Axiom System

LIA is a multi-modal logic of time, attempts, actions, goals and beliefs.[4] The logic is based on a combination of an enhanced version of linear temporal logic where it is possible to talk about actions and Cohen and Levesque's logic of goal and intention [4]. The main difference with respect to standard dynamic logic [11] is that the notion of *atomic (basic) action* is substituted with the more primitive notion *basic attempt*. We will show that the former can be defined from the latter.

The syntactic primitives of the logic are the following: -a set of atomic (basic) actions $ACT = \{\alpha, \beta, ...\}$; -a set of agents $AGT = \{i, j, ...\}$; -a set of propositional atoms $\Pi = \{p, q, ...\}$. The set of propositional formulas of our language is denoted by $PROP$ (elements in $PROP$ are denoted by $\Phi, \Omega, \Psi, ...$). The set FOR of well formed formulas φ of our modal action language *L* is defined by the following BNF:

$$\varphi := p|\top|\neg\varphi|\varphi \wedge \psi|\,[[i,\alpha]]\,\varphi|G\varphi|X\varphi|\varphi Until\psi|Bel_i\varphi|Goal_i\varphi$$

where *p* ranges over Π, *i* ranges over *AGT* and α ranges over *ACT*.

$[[i,\alpha]]\,\varphi$ is read "φ holds after any agent *i*'s *attempt* to do α". Hence $[[i,\alpha]]\,\bot$ expresses "agent *i* does not attempt to do α". Three abbreviations are used. $\langle\langle i,\alpha\rangle\rangle\,\varphi$ abbreviates $\neg\,[[i,\alpha]]\,\neg\varphi$, $F\varphi$ abbreviates $\neg G\neg\varphi$ and $\varphi Before\psi$ abbreviates $\neg(\neg\varphi\ Until\psi)$. Hence $\langle\langle i,\alpha\rangle\rangle\,\varphi$ has to be read "agent *i* attempts to do α and φ holds after this attempt " and $\langle\langle i,\alpha\rangle\rangle\,\top$ has to be read "agent *i* attempts to do α". For example $\langle\langle Bill, raiseArm\rangle\rangle\,\top$ is read "Bill attempts to raise the arm". We briefly go into the basic semantics.

A model for *LIA* is defined by the tuple M = (W, R_X, R^{att}, B, G, V).

- W is a set of worlds.
- R_X is a mapping $R_X : W \longrightarrow 2^W$ associating sets of possible worlds $R_X(w)$ to each possible world *w*. We suppose that R_X is a total function.
- R^{att} is a mapping $R^{att} : AGT \times ACT \longrightarrow (W \longrightarrow 2^W)$ associating sets of possible worlds $R^{att}_{i:\alpha}(w)$ to each possible world *w*. We assume that every $R^{att}_{i:\alpha}$ is a partial function.
- *B* is a mapping $B : AGT \longrightarrow (W \longrightarrow 2^W)$ associating sets of possible worlds $B_i(w)$ to each possible world *w*. We suppose that every B_i is serial, transitive and euclidean.[5]
- *G* is a mapping $G : AGT \longrightarrow (W \longrightarrow 2^W)$ associating sets of possible worlds $G_i(w)$ to each possible world *w*. We suppose that also every G_i is serial, transitive and euclidean.
- *V* is a mapping $V : \Pi \longrightarrow 2^W$ associating sets of possible worlds to propositional atoms.

[4] The logic is described more extensively in [15] where also formal proofs of the theorems are presented.

[5] We use a modal logic KD45 as the logic for Belief and Goals, i.e. an agent does not entertain inconsistent Beliefs (and inconsistent Goals) and is aware of his Beliefs and disbeliefs (and of his Goals and non-Goals).

After defining R^*_X as the reflexive and transitive closure of R_X, we look at truth conditions.

- $M, w \models p$ iff $w \in V(p)$.
- $M, w \models \neg\varphi$ iff not $M, w \models \varphi$
- $M, w \models \varphi \wedge \psi$ iff $M, w \models \varphi$ and $M, w \models \psi$
- $M, w \models X\varphi$ iff $\forall w'$ such that $w' \in R_X(w)$ it holds that $M, w' \models \varphi$
- $M, w \models G\varphi$ iff $\forall w'$ such that $w' \in R^*_X(w)$ it holds that $M, w' \models \varphi$.
- $M, w \models \varphi Until \psi$ iff $\exists\ w' \in R^*_X(w)$ such that $M, w' \models \psi$ and $\forall w''$ if $w'' \in R^*_X(w)$ and $w' \in R^*_X(w'')$ and $w'' \notin R^*_X(w')$ then $M, w'' \models \varphi$
- $M, w \models [[i, \alpha]]\,\varphi$ iff $\forall w'$ such that $w' \in R^{att}_{i:\alpha}(w)$ it holds that $M, v \models \varphi$
- $M, w \models Bel_i\varphi$ iff $\forall w' \in B_i(w)$ it holds that $M, w' \models \varphi$.
- $M, w \models Goal_i\varphi$ iff $\forall w' \in G_i(w)$ it holds that $M, w' \models \varphi$.

We use a complete axiomatization of linear temporal logic (axioms 0a-7a plus inference rules R1-R3) [8, 9] and the axioms and inference rules of the basic normal modal logic for *belief* modal operator, *goal* modal operator and *attempt* modal operator plus axioms 1b-12b (table 1).[6]

Table 1. Axiom system

0a. All tautologies of propositional calculus	1b. $\neg(Bel_i\varphi \wedge Bel_i\neg\varphi)$
1a. $G(\varphi \to \psi) \to (G\varphi \to G\psi)$	2b. $Bel_i\varphi \to Bel_iBel_i\varphi$
2a. $X\neg\varphi \leftrightarrow \neg X\varphi$	3b. $\neg Bel_i\varphi \to Bel_i\neg Bel_i\varphi$
3a. $X(\varphi \to \psi) \to (X\varphi \to X\psi)$	4b. $\neg(Goal_i\varphi \wedge Goal_i\neg\varphi)$
4a. $G\varphi \to \varphi \wedge XG\varphi$	5b. $Goal_i\varphi \to Bel_iGoal_i\varphi$
5a. $G(\varphi \to X\varphi) \to (\varphi \to G\varphi)$	6b. $\neg Goal_i\varphi \to Bel_i\neg Goal_i\varphi$
6a. $\varphi Until \psi \to F\psi$	7b. $Bel_i\varphi \to Goal_i\varphi$
7a. $\varphi Until \psi \leftrightarrow \psi \vee (\varphi \wedge X(\varphi Until \psi))$	8b. $Bel_i\,[[j,\alpha]]\,\psi \wedge \neg Bel_i\,[[j,\alpha]]\,\bot \to [[j,\alpha]]\,Bel_i\psi$
Inference Rules:	9b. $[[j,\alpha]]\,Bel_i\psi \wedge \neg\,[[j,\alpha]]\,\bot \to Bel_i\,[[j,\alpha]]\,\psi$
R1. $\frac{\vdash\varphi \quad \vdash\varphi\to\psi}{\vdash\psi}$ (*modus ponens*)	10b. $Bel_i(GBel_i\psi \leftrightarrow Bel_iG\psi)$
R2. $\frac{\vdash\varphi}{\vdash G\varphi}$ (*G-necessitation*)	11b. $Goal_i\,\langle\langle i,\alpha\rangle\rangle\,\top \leftrightarrow \langle\langle i,\alpha\rangle\rangle\,\top$
R3. $\frac{\vdash\varphi}{\vdash X\varphi}$ (*X-necessitation*)	12b. $X\varphi \to [[i,\alpha]]\,\varphi$

Semantic characterizations (model correspondence) of the previous axioms and inference rules are given in [15]. In [15] it is also proved that *LIA* is *sound* with respect to the set of *LIA models* satisfying all the semantic constraints imposed by the previous axioms and inference rules. With $\models_{LIA}\ \varphi$ we mean that φ is valid in all *LIA models* and with $\vdash_{LIA}\ \varphi$ we mean that φ is a theorem of *LIA*. Moreover we say that a formula φ is a consequence of the set of global assumptions $\{\Phi_1, ..., \Phi_n\}$ in the class of models LIA (noted $\{\Phi_1, ..., \Phi_n\} \models_{LIA} \varphi$) if and only if for all models $M \in LIA$ if $\models_M \Phi_i$ for every Φ_i, then $\models_M \varphi$.

Axiom 11b deserves some comments. This axiom has not been analyzed before in the literature on modal logic of intentional action. It establishes that an agent attempts

[6] Notice that from our axiomatic system (axiom 2a and axiom 12b) it follows that: $\langle\langle i,\alpha\rangle\rangle\,\varphi \to [[i,\alpha]]\,\varphi$, i.e. action are deterministic.

to do some action α if and only if the agent has the goal to attempt to do action α. The axiom relates mental attitudes with the executive and behavioral element. [7] In our view it is relevant for a formal theory of action to account for the role of intention in producing a given performance. This fundamental issue is not enough stressed in formal models of intentional action. Axiom 11b has exactly this role. It accounts for the conditions for passing from a pure mental and motivational level to the executive and physical reality. We will show later (in section 5) that axiom 11b is central for the notion of *present-directed intention*. In this analysis we do not introduce at the formal level *sequential composition* of basic actions. We prefer to work with the simplest formal language for actions, leaving the problem of sequential composition to future work. Let us only stress that axiom 11b should also be applied to sequences of basic actions $\alpha;\beta;...$ performed by the same agent and which do not involve perception (epistemic actions). For example consider a football playing robot having the goal to perform the sequence of basic actions *turn-right;advance;shoot*. The goal to attempt to perform the sequence of actions triggers the performance and even if the robot is blocked by another player, he will attempt to execute the three basic actions in sequence, without realizing that the first action fails.[8]

4 Attempts and Action Theories

4.1 Definition of Action and Execution Preconditions[9]

Our definition of action is built on the special formula $Pre(\alpha)$ denoting the *physical preconditions for executing action* α (*execution preconditions*). We assume that $Pre(\alpha)$ is a function returning some classical formula Φ[10] that is $Pre: \ ACT \rightarrow PROP$. For example we might have: $FreeLeg = Pre(kickBall)$.

Definition 1. *Action.* $\langle i, \alpha \rangle \varphi =_{def} \langle\langle i, \alpha \rangle\rangle \varphi \wedge Pre(\alpha)$

Definition 1 relates the notion of *action* with the primitive notion of *attempt*. It says that action executions are attempts whose execution preconditions hold. An instance of definition 1 is: $\langle i, \alpha \rangle \top =_{def} \langle\langle i, \alpha \rangle\rangle \top \wedge Pre(\alpha)$. It says that a given action α is executed

[7] In [20] a similar axiom is proposed where actions are related with knowledge (the axiom says that if an agent *i* can do a certain action α from his repertoire then the agent knows that he can do it).

[8] We are supposing here some kind of *persistence* in the intentional execution of those sequences of actions which do not involve perception that is, when an agent has the goal to attempt to perform a sequence of actions which does not involve perception then the agent attempts to perform the complete (intended) sequence of actions (the agent cannot stop in the middle of the sequence and revise his pushing intentions) and when the agent attempts to perform a complete sequence of actions which does not involve perception then the agent has the goal to attempt to perform the complete sequence.

[9] Given that time is linear in our logic, we use the terms "execution precondition" and "execution law" instead of the terms "executability precondition" and "executability law".

[10] In realistic applications the function *Pre* should have an agent argument: the preconditions of kicking a ball may differ from agent to agent, for example for a lame agent $Pre(kickBall) = \bot$. Here we make the assumption that *execution preconditions* of an action are the same for all agents.

by agent *i* if and only if agent *i* attempts to do α and the preconditions for executing action α are holding. This is an explicit way to relate actions with attempts and to express execution laws. On the basis of definition 1 execution laws are defined by referring to the *attempt* notion. This kind of solution is quite different from standard solutions (see for example [2] and [22]) where execution laws are expressed by taking actions as primitive elements, without deconstructing them into more elementary constituents (viz. attempts). Moreover, from definition 1 it follows that a consequence of an attempt to perform α is also a consequence of the successful performance of basic action α and if the *execution preconditions* hold then the consequences of the attempt are equivalent to the consequences of the associated basic action. Indeed: a) $[[i,\alpha]]\,\varphi \rightarrow [i,\alpha]\,\varphi$ and b) $Pre(\alpha) \rightarrow ([[i,\alpha]]\,\varphi \leftrightarrow [i,\alpha]\,\varphi)$ are two valid formulas in our logic. In the next sections we analyze effect laws by substituting the notion of action with the more primitive notion of attempt and show that by distinguishing attempts from actions we get some important conceptual refinements.

4.2 Effect Preconditions

Similarly to Situation Calculus [22] we take for each propositional atom $p \in \Pi$ and basic action $\alpha \in ACT$ a propositional formula $\gamma^{+}(\alpha, p)$ describing the *positive effect preconditions* of the attempt to do α with respect to *p* and $\gamma^{-}(\alpha, p)$ describing the *negative effect preconditions* of the attempt to do α with respect to *p*. For example the following *positive effect preconditions* can be associated to the attempt to perform the actions *loading*, *pulling* and *picking up*.[11]

$\gamma^{+}(load, loadedGun) = freeHand \wedge holdsGun$
$\gamma^{+}(pull, wounded) = holdsGun \wedge loadedGun \wedge pointedGun \wedge freeHand$
$\gamma^{+}(pull, pulledTrigger) = holdsGun \wedge freeHand$
$\gamma^{+}(pull, scared) = holdsGun \wedge pointedGun$
$\gamma^{+}(pickUp, holdsGun) = gunOnTable \wedge freeArm \wedge freeHand$

Effect laws are specified accordingly in terms of global assumptions in Fitting's sense [7] of the form: $\gamma^{+}(\alpha, p) \rightarrow [[\alpha]]\,p$ and $\gamma^{-}(\alpha, p) \rightarrow [[\alpha]]\,\neg p$.

For instance positive effect axioms for the previous actions are specified by the following global assumptions:[12]

$holdsGun \wedge pointedGun \rightarrow [[pull]]\,scared$
$holdsGun \wedge loadedGun \wedge pointedGun \wedge freeHand \rightarrow [[pull]]\,wounded$
$freeHand \wedge holdsGun \rightarrow [[load]]\,loadedGun$
$holdsGun \wedge freeHand \rightarrow [[pull]]\,pulledTrigger$
$gunOnTable \wedge freeArm \wedge freeHand \rightarrow [[pickUp]]\,holdsGun$

We could assume as in [22] that (*positive* and *negative*) effects *preconditions* are complete. *Completeness assumption* can be formulated by means of global assumptions of the form: $\neg\gamma^{+}(\alpha, p) \wedge \neg p \rightarrow [[\alpha]]\,\neg p$ and $\neg\gamma^{-}(\alpha, p) \wedge p \rightarrow [[\alpha]]\,p$.

[11] To simplify our exposition we suppose in our examples that for each action α and possible effect p we have $\gamma^{-}(\alpha, p) = \bot$. Thus we do not need to specify *negative effect preconditions.*

[12] Notice that in effect laws actions are not indexed by agents. Indeed we assume that effects laws do not depend on the performing agent.

For instance, given the effect law $holdsGun \wedge pointedGun \rightarrow [[pull]]\, scared$ for the action *pulling*, we can establish that: $\neg(holdsGun \wedge pointedGun) \wedge \neg scared \rightarrow [[pull]]\, \neg scared$.

We make a *Consistency assumption* saying that negative effect preconditions and positive effect preconditions must be consistent that is: $\gamma^{+}(\alpha, p) \rightarrow \neg\gamma^{-}(\alpha, p)$.[13]

Finally we need to specify *execution preconditions* for our three actions *loading*, *pulling* and *picking-up*: $Pre(pull) = freeHand$, $Pre(load) = freeHand$, $Pre(pickUp) = freeArm \wedge freeHand$.

Given effect preconditions and appropriate assumptions successor state axioms can be specified as standard Situation Calculus requires.

Indeed suppose that $\gamma^{-}(\alpha, p)$, $\gamma^{+}(\alpha, p)$ are given and that the *completeness assumption* and *consistency assumption* are made then the following equivalences holds:

$$[[i, \alpha]]\, p \leftrightarrow \neg Goal_i\, \langle\langle i, \alpha\rangle\rangle\, \top \vee \gamma^{+}(\alpha, p) \vee (p \wedge \neg\gamma^{-}(\alpha, p))$$

In this paper we do not investigate any modal regression technique for our logic.[14] Let us only notice that according to the previous logical equivalence the effects of an attempt to do some action α are completely specified by positive effect preconditions ($\gamma^{+}-preconditions$), negative effect preconditions ($\gamma^{-}-preconditions$) and the goal to attempt to do α. Execution preconditions are not mentioned in the successor state axiom. Thus we can argue that under our logical framework every planning task can in principle be reduced to the task of finding the correct sequence of attempts for reaching a given result. Given successor state axioms built on the primitive notion of attempt, for every planning problem there is no need to verify whether execution preconditions hold. This implies that in *LIA* the notion of *execution precondition* is not necessary for formulating action theories.

4.3 Discussion

Being able to characterize attempts and actions, we can provide a further relevant distinction: the distinction between *stable effects* and *successful effects* of an attempt.

Let us go back to our previous example. We have identified the execution preconditions for *pulling* with $Pre(pull) = freeHand$. Moreover we have specified the following effect laws: $holdsGun \wedge loadedGun \wedge pointedGun \wedge freeHand \rightarrow [[pull]]\, wounded$ and $holdsGun \wedge pointedGun \rightarrow [[pull]]\, scared$.

Given definition 1 the first effect law can be rewritten as:
$holdsGun \wedge loadedGun \wedge pointedGun \rightarrow [pull]\, wounded$.

On one side a *stable positive effect* of an attempt to do some action α is a result that an attempt to perform α can produce even if the execution preconditions of action α do not hold. For instance *scared* is a stable positive effect of the attempt to *pull*. Indeed I can scare you simply by pointing a gun toward you and attempting to pull the trigger.[15]

[13] Due to our hypothesis that $\gamma^{-}(\alpha, p) = \bot$ for each action α and possible effect p such consistency is always the case.

[14] On the problem of how handling regression in dynamic logic see [6].

[15] We are assuming that you become aware of the risk of being killed only if you can perceive that I am attempting to pull the trigger (fear is not simply triggered by your seeing that I am pointing the gun toward you).

On the other side a *successful positive effect* of an attempt to do some action α is a result that an attempt to perform α causes only if the execution preconditions of action α hold. For instance *wounded* is a *successful positive effect* of the attempt to *pull.* Indeed I can wound you if after pointing the gun toward you and attempting to pull the trigger, I correctly execute the pulling movement (the execution preconditions of *pulling* hold) and the gun is loaded.

Formally:

- *p* is a *successful positive effect* of the attempt to perform the basic action α if and only if $\models_{LIA} \gamma^{+}(\alpha, p) \rightarrow Pre(\alpha)$.[16]
- *p* is a *stable positive effect* of the attempt to perform the basic action α if and only if there is a model $M \in LIA$ such that $\gamma^{+}(\alpha, p) \wedge \neg Pre(\alpha)$ is *satisfiable* in M.[17]

In our view there is always some stable effects associated with attempts. Even assuming that the *attempt* to do a basic action is a mere mental process, we can still identify stable effects of the attempt. Indeed under some appropriate preconditions attempting to do something can cause some modification of the mental states of the performing agent (and these modifications do not depend on the fact that the attempt is successful). For example if I believe that after raising my arm my arm goes up and I believe that the preconditions for raising my arm are holding (for instance I believe that my arm is not blocked) then after attempting to raise my arm I believe that my arm is up. This is made explicit by the next theorem of our logic: $Bel_i Pre(\alpha) \wedge Bel_i [i, \alpha] \varphi \rightarrow [[i, \alpha]] Bel_i \varphi$. We can also write plausible effect laws which mention stable effects of attempts at the level of mental attitudes and dispositions of the performing agent. For example if in the morning I am still half-awake and I attempt to stand up then I am awake after this attempt: $asleep \rightarrow [[stand - up]] awake$.

Application to "count as" scenarios. In the context of institutions, actions may "count as" implementations of others. Many actions in the social world acquire a different meaning when some institutional fact holds in that world.[18]

[16] Notice that the class of *successful positive effects* of an attempt to do some basic action α also includes the *intrinsic effect* of (basic) action α [29, 26]. Indeed the *intrinsic effect* of some (basic) action α is the state of affairs that it is guaranteed to hold when α is attempted and the execution preconditions of action α hold. For instance the intrinsic effect of the (basic) action of *raising the arm* is *raised arm,* the intrinsic effect of the (basic) action of *opening the mouth* is *open mouth* and so on... Formally: *p* is a *intrinsic effect* of some basic action α if and only if $\models_{LIA} \gamma^{+}(\alpha, p) \leftrightarrow Pre(\alpha)$. According to Stoutland also complex actions have intrinsic results. For instance the (complex) action of *opening the door* (opening the door is performed *by* moving the arm in a certain way) has the *door is open* as intrinsic effect.

[17] From the two definitions it follows that the category *stable positive effects* and the category *successful positive effects* are disjoint. Moreover the same kind of definitions apply to *successful negative effects* and *stable negative effects* of an attempt, that is: 1) $\neg p$ is a *successful negative effect* of some basic action α if and only if $\models_{LIA} \gamma^{-}(\alpha, p) \rightarrow Pre(\alpha)$; 2) $\neg p$ is a *stable negative effect* of some basic action α if and only if it exists a model $M \in LIA$ such that $\gamma^{-}(\alpha, p) \wedge \neg Pre(\alpha)$ is *satisfiable* in M.

[18] See also [14] for a formal approach to institutional actions.

For instance take the action of *signing a document* (or the action of *voting*). This action has the same physical realization of the action *writing*, but it differentiates from a simple writing since it is performed under some particular institutional preconditions. It is not the aim of this paper to investigate the exact institutional preconditions which are needed in order to make some physical action an institutional action.[19] Indeed several kinds of conditions concerning social roles, norms etc... must be satisfied: for example the performing agent needs to be entitled to perform the institutional action (he must play some institutional role)[20] and there should be some other agent with institutional power who verifies the correct execution of the action[21] etc... Just consider the following simple example.

$\gamma^{+}(write, closedHand) = freeHand$
$\gamma^{+}(write, written) = hasDoc \wedge holdsPen \wedge freeHand$
$\gamma^{+}(write, signed) = hasDoc \wedge holdsPen \wedge lastPage \wedge freeHand \wedge director$
$\gamma^{+}(write, voted) = election \wedge citizen \wedge holdsPen \wedge VotingPaper \wedge freeHand$

According to the previous formulations of positive effect preconditions and negative effect preconditions, if an agent has the hand free and attempts to write then the hand gets closed; if the agent has a document in front of him and a pen is in his hand and his hand is free and he attempts to write then the document gets written on;[22] if the agent is the director of the organization, has the last page of the document in front of him and a pen in the hand, his hand is free, and attempts to write then the document gets signed. Finally if it is election day, the agent is a citizen of the country, a pen is in his hand, his hand is free, has a voting paper in front of him and attempts to write then the agent gives his vote. We do not specify here the completeness laws (their specification is straightforward). Finally we formulate the execution preconditions of the *writing* action: $Pre(write) = freeHand$. Let us only use two abbreviations for indicating the institutional version of the attempt to do action α and the institutional version of action α:

$\langle\langle Ist - \alpha\rangle\rangle\, \varphi =_{def} \langle\langle \alpha\rangle\rangle\, \varphi \wedge Ist(\alpha)$;
$\langle Ist - \alpha\rangle\, \varphi =_{def} \langle \alpha\rangle\, \varphi \wedge Ist(\alpha)$ which can be rewritten as

$\langle Ist - \alpha\rangle\, \varphi =_{def} \langle\langle \alpha\rangle\rangle\, \varphi \wedge Pre(\alpha) \wedge Ist(\alpha))$ where $Ist(\alpha)$ denotes all conditions which make α become an *institutional action* or to "count as" an *institutional action* (we call them *institutional preconditions*).[23]

Since the institutional version of a basic action α is physically identical to α we can safely assume that the execution preconditions of α and the execution preconditions of the institutional version of α are identical.

[19] On this point see [24, 27].

[20] In order to marry a couple the agent must be a priest.

[21] In signing a contract is not enough to sign the document at the correct place. An institutional witness (the notary) is needed who verifies the correct execution of the procedure.

[22] Notice that in common sense language *writing* is not a proper basic action. Indeed agent generally *writes by* performing a *certain movement with the hand.* Thus our label "write" denotes rather the basic action (bodily movement) on which the complex action of *writing* is based.

[23] We assume that $Ist(\alpha)$ is a function returning some classical formula Φ that is: $Ist : ACT \rightarrow PROP$.

Notice that basic actions might have more than one institutional version. For instance the basic action of *writing* "counts as" the institutional action of *signing* under some institutional preconditions whereas it "counts as" the institutional action of *voting* under some different institutional preconditions. In order to account for different kinds of institutional actions based on the same basic action, $Ist(\alpha)$ must denote several alternative groups of institutional preconditions. For instance in order to distinguish the institutional action of *signing* from the institutional action of *voting* we must operate at the level of institutional preconditions and identify different subsets of institutional preconditions corresponding to each institutional version of the basic action. Let us consider the simple scenario where the action of *writing* has only two institutional versions (*signing* and *voting*). $Ist(\alpha)$ denotes only two subsets of institutional preconditions:

$$Ist(write) = (lastPage \wedge director) \vee (election \wedge citizen \wedge VotingPaper).$$

Having introduced $Ist(write)$, the institutional version of the action *writing* and the institutional version of the attempt to *write* can be specified:

$\langle\langle Ist - write\rangle\rangle\, \varphi =_{def} \langle\langle write\rangle\rangle\, \varphi \wedge (lastPage \wedge director) \vee (election \wedge citizen \wedge VotingPaper)$;
$\langle Ist - write\rangle\, \varphi =_{def} \langle write\rangle\, \varphi \wedge (lastPage \wedge director) \vee (election \wedge citizen \wedge VotingPaper)$.

The action *signing* (the attempt to *sign*) and the action *voting* (the attempt to *vote*) are specific institutional versions of the action *writing* (the attempt to *write*), that is they are defined as instances of *writing* under some specific subsets of the set of institutional preconditions of *writing*. Indeed:

$\langle sign\rangle\, \varphi =_{def} \langle write\rangle\, \varphi \wedge lastPage \wedge director$;
$\langle\langle sign\rangle\rangle\, \varphi =_{def} \langle\langle write\rangle\rangle\, \varphi \wedge lastPage \wedge director$;
$\langle vote\rangle\, \varphi =_{def} \langle write\rangle\, \varphi \wedge election \wedge citizen \wedge VotingPaper$
$\langle\langle vote\rangle\rangle\, \varphi =_{def} \langle\langle write\rangle\rangle\, \varphi \wedge election \wedge citizen \wedge VotingPaper$.

This means that: 1) getting φ after my attempt to perform the action of *signing* (or after performing the action of *signing*) means being the director and getting φ after the attempt to write my name (or after the action of *writing* my name) on the last page of the document; 2) getting φ after my attempt to perform the action of *voting* (or after performing the action of *voting*) means being a citizen of the country on the election day and getting φ after the attempt to write (or after the action of *writing*) on the voting paper.

The fact that *signing* and *voting* are specific institutional versions of the basic action *writing* is made explicit by the following four formal consequences of our definitions: a) $\langle sign\rangle\, \varphi \rightarrow \langle Ist - write\rangle\, \varphi$; b) $\langle vote\rangle\, \varphi \rightarrow \langle Ist - write\rangle\, \varphi$; c) $\langle\langle sign\rangle\rangle\, \varphi \rightarrow \langle\langle Ist - write\rangle\rangle\, \varphi$ and d) $\langle\langle vote\rangle\rangle\, \varphi \rightarrow \langle\langle Ist - write\rangle\rangle\, \varphi$.

Indeed if I attempt to sign (or to vote) and φ holds after this attempt then I also attempt to perform the institutional version of *writing* and φ holds after the attempt, and if I sign (or vote) and φ holds after this action then I also attempt to perform the institutional version of *writing* and φ holds afterward.

Moreover on the basis of the previous definitions of institutional action and institutional attempt we get (besides the validity $[[i, \alpha]]\, \varphi \rightarrow [i, \alpha]\, \varphi$) the following four

validities: a) $[[\alpha]]\,\varphi \rightarrow [[Ist-\alpha]]\,\varphi$; b) $[\alpha]\,\varphi \rightarrow [Ist-\alpha]\,\varphi$; c) $[[\alpha]]\,\varphi \rightarrow [Ist-\alpha]\,\varphi$ and d) $[[Ist-\alpha]]\,\varphi \rightarrow [Ist-\alpha]\,\varphi$.

It is evident that the relation among attempt and physical (basic) action is symmetrical to the relation among physical (basic) action and institutional action. Indeed if φ is a consequence of the attempt to do action α then φ is also a consequence of the successful execution of the basic action α and if φ is a consequence of performing the basic action α then φ is also a consequence of performing the institutional version of action α (moreover if φ is a consequence of the attempt to do action α then φ is also a consequence of the attempt to do the institutional version of action α; if φ is a consequence of the attempt to do the institutional version of action α then φ is also a consequence of doing the institutional version of action α).

Besides the distinction between *stable effects* and *successful effects* of an attempt we can provide the distinction among *institutional effects* and *natural effects* of an attempt. We define *institutional positive effects* of a given attempt to perform action α all those positive effects that the attempt to perform α causes only if the institutional preconditions of α hold. We distinguish these effects from *natural positive effects* of a given attempt to perform action α which are those positive effects that the attempt to perform α causes even if the institutional preconditions of α do not hold.

Formally:

- *p* is a *institutional positive effect* of the attempt to perform the basic action α if and only if $\models_{LIA} \gamma^{+}(\alpha, p) \rightarrow Ist(\alpha)$.
- *p* is a *natural positive effect* of the attempt to perform the basic action α if and only if there is a model $M \in LIA$ such that $\gamma^{+}(\alpha, p) \wedge \neg Ist(\alpha)$ is *satisfiable* in M.[24]

To sum up, we can distinguish four different sub-categories of effects of an attempt:

1. Institutional and stable effects of an attempt.
2. Natural and stable effects of an attempt.
3. Institutional and successful effects of an attempt.
4. Natural and successful effects of an attempt.

Going back to our initial example of *pulling* action, *scared* is a natural and stable effect of the attempt to *pull*, *wounded* (or *dead*) is a natural and successful effect of the attempt to *pull*. Finally *attempted homicide* (or *attempted capital punishment*) is an institutional and stable effect of the attempt to *pull* and *homicide* (or *capital punishment)* is an institutional and successful effect of the attempt to *pull.*[25]

[24] From the definition it follows that the category *institutional positive effects* and the category *natural positive effects* are also disjoint. Again the same kind of definitions apply to *natural negative effects* and *institutional negative effects* of an attempt, that is : 1) $\neg p$ is a *institutional negative effect* of some basic action α if and only if $\models_{LIA} \gamma^{-}(\alpha, p) \rightarrow Ist(\alpha)$; 2) $\neg p$ is a *natural negative effect* of some basic action α if and only if it exists a model $M \in LIA$ such that $\gamma^{-}(\alpha, p) \wedge \neg Ist(\alpha)$ is *satisfiable* in M.

[25] *Attempted homicide* and *homicide* are recognized as violations of the law in every civil society when a private person kills someone and is not entitled to do so (therefore they are institutional effects of a given action). On the other hand a firing squad executing a death penalty is entitled to kill and the effect of its action is recognized by the institution either as a *capital execution* or as an *attempted capital execution*.

5 Concluding Remarks: Attempt and Present-Directed Intention

In this final section we discuss additional properties of attempts, introduce the notion of present-directed intention and present some formal relations between the two concepts.

The formula $\langle\langle i,\alpha\rangle\rangle \top \leftrightarrow Bel_i \langle\langle i,\alpha\rangle\rangle \top$ is valid and establishes that our notion of *attempt* is related with agent's awareness (when agent *i* attempts to do some action α, he believes to be attempting and viceversa).

The valid formula $Goal_i [[i,\alpha]] \bot \rightarrow [[i,\alpha]] \bot$ establishes that if agent *i* has the goal to *avoid* to attempt to perform action α then action α is not attempted by agent *i*. We introduce next the notion of *present-directed intention* [1, 25].

Definition 2. *Present-directed Intention.* $PDI_i(\alpha) =_{def} Goal_i \langle i,\alpha\rangle \top$

The definition of present-directed intention is intimately related with axiom 11b stating the logical equivalence of the goal to attempt to do action α and attempt itself. Indeed the present-directed intention stage coincides with the stage at which the agent triggers the motor intentional behaviour. According to the present model when agent has the present-directed intention to do some action α: 1) he can attempt to do α (he can send the action to execution); 2) he is aware of this possibility; 3) he has the goal to attempt to do α, 4) he has the goal that the preconditions for executing action α hold, 5) he cannot believe that the preconditions for executing action α do not hold. [26] The previous statements are formally expressed by the following valid formulas of our logic:

a) $PDI_i(\alpha) \rightarrow \langle\langle i,\alpha\rangle\rangle \top$;
b) $PDI_i(\alpha) \rightarrow Bel_i \langle\langle i,\alpha\rangle\rangle \top$;
c) $PDI_i(\alpha) \rightarrow Goal_i \langle\langle i,\alpha\rangle\rangle \top$;
d) $PDI_i(\alpha) \rightarrow Goal_i Pre(i,\alpha)$;
e) $PDI_i(\alpha) \rightarrow \neg Bel_i \neg Pre(i,\alpha)$.

Finally just pay attention to the distinction among *the goal to attempt to do a certain action* α ($Goal_i \langle\langle i,\alpha\rangle\rangle \top$) and the notion of *present-directed intention to do a certain action* α. Notice that the inverse direction of previous formula c) is not a valid statement: in our logic an agent can have the goal to attempt to do α without having the present-directed intention to do α.[27]

References

1. Bratman, M. E. (1987). *Intentions, plans and practical reason*. Cambridge, MA: Harvard University Press.
2. Castilho, M. A., Gasquet, O., Herzig, A. (1999). Formalizing action and change in modal logic I: the frame problem. *Journal of Logic and Computation*, 9(5), pp. 701-735.

[26] Notice that an agent can have the goal to attempt to do α believing that the preconditions for executing α do not hold ($Goal_i \langle\langle i,\alpha\rangle\rangle \top \wedge Bel_i \neg Pre(i,\alpha)$ is satisfiable).

[27] Suppose that "Brett promises to pay Belton fifty dollars if Belton *attempts* to solve a certain chess problem within five minutes". Imagine that Brett assures Belton that he need not actually solve the problem for getting the fifty dollars. According to [17] it is plausible to say that Belton is motivated to attempt to solve problem even if he does not intend to solve the problem.

3. Chisholm, R. M. (1966). Freedom and Action. In Keith Lehrer (Ed.), *Freedom and Determinism*, Random House; New York, NY, pp. 105-39.
4. Cohen, P. R. , Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42, pp. 213-261.
5. Danto, A (1965). What we can do. *The Journal of Philosophy,* 60, pp. 435-445.
6. Demolombe, R., Herzig, A., Varzinczak, I. (2003). Regression in modal logic. *Journal of Applied Non-Classical Logics,* 13, pp. 165-185.
7. Fitting, M. (1983). *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel, Dordrecht.
8. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J. (1980). On the temporal analysis of fairness. In *Proceedings 7th ACM Symposium on Principles of Programming Languages*, pp. 163-173.
9. Goldblatt, R. (1992). *Logics of Time and Computation, 2nd edition*. CSLI Lecture Notes, Stanford, California.
10. Goldman, A. (1970). *A Theory of Human Action*. Englewood Cliffs: Prentice-Hall.
11. Harel, D., Kozen D., Tiuryn, J.(2000). *Dynamic Logic*. Cambridge, MA: MIT Press.
12. Herzig, A., Longin, D. (2004). C&L Intention Revisited. In *Proceedings of KR2004*, pp. 527-535.
13. Hornsby, J. (1980). *Actions*. Routledge & Kegan Paul, London.
14. Jones, A., Sergot, M. J. (1996). A formal characterisation of institutionalised power. *Journal of the IGPL*, 4(3), pp. 429–445.
15. Lorini, E. (2006). A logic of Intention and Attempt. *Technical Report*, Institute of Cognitive Science and Technologies-CNR, Rome.
16. McCann, H. J. (1974). Volition and Basic Action. *The Philosophical Review*, 83, pp. 451 473.
17. Mele, A. R. (1992). *Springs of action.* Oxford University Press, New York.
18. Meyer, J.J. Ch., van der Hoek, W., van Linder, B. (1999). A Logical Approach to the Dynamics of Commitments. *Artificial Intelligence*, 113(1-2), pp. 1-40.
19. O'Shaughnessy, B. (1973). Trying (as the Mental Pineal Gland.) *The Journal of Philosophy*, 70, pp. 365-86.
20. Pacuit, E., Parikh, R., Cogan, E. (2005). The Logic of Knowledge Based Obligation. To appear in *Knowledge, Rationality and Action*.
21. Rao, A. S., Georgeff M. P. (1991). Modelling rational agents within a BDI-architecture. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, San Mateo, CA.
22. Reiter, R. (2001). *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. Cambridge, MA: MIT Press.
23. Santos, F., Carmo, J., Jones, A. (1997). Action concepts for describing organised interaction. In *Proceedings Thirtieth Annual Hawai International Conference on System Sciences*, pp. 373-382.
24. Searle, J. R. (1995). *The construction of social reality*. Free Press, New York.
25. Searle, J. R. (1983). *Intentionality*. Cambridge University Press.
26. Stoutland, F. (1968). Basic Actions and Causality. *Journal of Philosophy*, 65, pp. 467-475.
27. Tummolini, L., Castelfranchi, C. (in press). The cognitive and behavioral mediation of institutions: Towards an account of institutional actions. *Cognitive Systems Research*, 7(2-3).
28. Vanderveken, D. (2003). Attempt and action generation: towards the foundations of the logic of action. *Cahiers d'pistmologie*, 293.
29. Von Wright, G. H. (1963). *Norm and Action*. London: Routledge and Kegan Paul.

On Herbrand's Theorem for Intuitionistic Logic*

Alexander Lyaletski[1] and Boris Konev[2]

[1] Faculty of Cybernetics, Kiev National Taras Shevchenko University, Ukraine
lav@unicyb.kiev.ua
[2] Department of Computer Science, University of Liverpool, United Kingdom
B.Konev@csc.liv.ac.uk

Abstract. In this paper we reduce the question of validity of a first-order intuitionistic formula without equality to generating ground instances of this formula and then checking whether the instances are deducible in a propositional intuitionistic tableaux calculus, provided that the propositional proof is compatible with the way how the instances were generated. This result can be seen as a form of the Herbrand theorem, and so it provides grounds for further theoretical investigation of computer-oriented intuitionistic calculi.

1 Introduction

In its classical formulation, Herbrand's theorem [9] relates the question of validity of a first-order formula in Skolem prenex form, $\forall x_1 \ldots \forall x_n\, \phi(x_1, \ldots, x_n)$, with the question of validity of one of its *Herbrand extensions*: The formula $\forall x_1 \ldots \forall x_n\, \phi(x_1, \ldots, x_n)$ is valid if, and only if, $\bigwedge_i^m \phi(t_{i,1}, \ldots, t_{i,n})$ is valid for some $m \geq 1$ and some collection of ground Herbrand terms $t_{i,j}$. Since every classical first-order formula can be reduced preserving satisfiability, through the Skolemisation, to this Skolem prenex form, Herbrand's theorem, essentially, provides a way to reduce the question of validity of first-order formulae to propositional logic. Even though the required Herbrand extension and the terms $t_{i,j}$ cannot be computed recursively (for otherwise first-order logic would be decidable), this result is particularly interesting for the automated reasoning community as it gives birth to a number of highly efficient proof methods such as resolution [21] and the inverse method [14]. Availability of similar results for other logics would also be of significant interest.

Yet, there is no general Herbrand-like theorem for intuitionistic logic, where formulae cannot in general be preprocessed into a prenex normal form, and the construction of a proof is often sensitive to the order in which the connectives and quantifiers are analysed. The biggest obstacle is that the Skolemisation does not preserve intuitionistic satisfiability. Consider, for example, formulae $\neg\forall x P(x) \supset \exists y \neg P(y)$ and $\exists x \neg P(x) \supset \exists y \neg P(y)$. The first of them is not intuitionistically valid while the other one, obviously, is; however, the Skolemised forms of the two coincide. These complications lead to the existence of limited forms of Herbrand's theorem for particular classes of intuitionistic formulae only [15, 16, 3].

While classical Herbrand's theorem is often proved semantically, it can also be obtained as a direct consequence of Gentzen's cut elimination theorem [7]: The question

* Supported by the Nuffield foundation grant NAL/00841/G.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 293–305, 2006.

of the *deducibility* of a first-order formulae in Skolem prenex form can be reduced to the *deducibility* of a Herbrand extension, and then the necessary Herbrand terms can be extracted from the cut-free proof. In fact, a similar idea is used in free-variable tableau methods [8], where quantifiers are dealt with separately from dealing with the propositional proof skeleton. Since free-variable tableau techniques are also available for intuitionistic logic [22, 23], one can hope to obtain *deductive* forms of intuitionistic Herbrand's theorem, in which the question of the deducibility of intuitionistic first-order formulae is reduced to the deducibility of a Herbrand extension.

We base the investigations presented in this paper on our earlier results in [10], where we introduce a tableau-based calculus without explicit rules dealing with quantifiers. Prior to proof search, we replace in a given formula bound variables with free variables and parameters depending on the polarity of the bounding quantifiers. Then, an *admissible substitution* suggests the correct order of quantifier rule applications, and a ground tableau proof of the given formula can be reconstructed, making it unnecessary to backtrack over different orders of quantifier rule applications. The soundness of the resulting calculus is provided by ordering restrictions in the way similar to the one considered in [20, 13, 22, 23] for calculi with quantifier rules.

The method presented in [10] is similar, to some extent, to the connection method for intuitionistic logic pioneered by Wallen [25] and developed further in [18, 19, 11, 24]. The key difference is in the way how we define admissibility of substitutions. In [25], the notion of admissibility is used to model proof search in a particular sequent intuitionistic calculus, and, therefore, this calculus is tightly integrated into the method. One has to search for an admissible substitution even for propositional intuitionistic formulae, and it is not easy to replace the chosen sequent calculus with a different proof system. In our approach, we try to separate dealing with quantifiers, impermutabilities, and propositional intuitionistic reasoning. To do that, we use admissibility to check eigenvariable conditions, we use a propositional proof system to check the deducibility, and, finally, we check that the propositional proof agrees with quantifiers. From the implementation point of view, some research in this direction was done in [17], where a tableau-like search for connections is implemented; however, they still use string unification in admissibility checks. Our admissibility checks are based on much simpler term unification; the price we have to pay is more complex proof search on the ground level.

It is this separation of propositional proof search and the search for admissible substitutions (with further check that the two agree), what allows us to formulate an analog of Herbrand's theorem. We reduce the question of the deducibility of an intuitionistic first-order formula to the deducibility of an analog of the Herbrand extension in an intuitionistic propositional calculus. We solve the problems with impermutabilities by imposing restrictions on derivations in the propositional calculus.

2 Preliminaries

We use the standard terminology of first-order logic without equality. The first order language is constructed over a *signature Sig* containing a finite (possibly empty) set of functional symbols, and a finite (nonempty) set of predicate symbols, the logical

connectives: the universal quantifier symbol $\forall$, the existential quantifier symbol $\exists$, and the propositional connectives for the implication ($\supset$), disjunction ($\vee$), conjunction ($\wedge$), and negation ($\neg$).

As for the set of variables Vr, we assume that Vr consists of two disjoint countable sets: mVr (original variables) and iVr (indexed variables) so that $\mathrm{Vr} = \mathrm{mVr} \cup \mathrm{iVr}$, where the following holds: for any $v \in \mathrm{mVr}$ and any positive integer (*index*) k ($k = 1, 2, \ldots$), iVr contains the indexed variable ${}^k v$.

Additionally, we extend the signature *Sig* in the following way: for any natural number (*index*) k ($k = 1, 2, \ldots$) and any symbol s from *Sig*, we add the *indexed symbol* ${}^k s$ to *Sig* denoting the constructed extension by *eSig*. For example, ${}^1\vee$, ${}^3\supset$, and ${}^5\forall$ are symbols of the extended signature. These left upper indices are used to distinguish connectives in different copies of the same formula, stemming from multiplicities, to encode impermutabilities.

The notions of terms, atomic formulae, literals, formulae, free and bound variables, and scopes of quantifiers over both *Sig* and *eSig* are defined in the usual way [8] and assumed to be known to the reader. We assume that no two quantifiers in any formula have a common variable, which can be achieved by renaming bound variables.

If the formula F' is constructed by renaming (some or all) bound variables in a formula F, we call F' a *variant* of F.

An *equation* is an unordered pair of terms s and t written as $s \approx t$. Assume L is a literal of the form $R(t_1, \ldots, t_n)$ (or $\neg R(t_1, \ldots, t_n)$) and M is a literal of the form $R(s_1, \ldots, s_n)$ (or $\neg R(s_1, \ldots, s_n)$, respectively), where R is a predicate symbol and $t_1, \ldots, t_n, s_1, \ldots, s_n$ are terms. Then $\Sigma(L, M)$ denotes the set of equations $\{t_1 \approx s_1, \ldots, t_n \approx s_n\}$. In this case, L and M are said to be *equal modulo* $\Sigma(L, M)$ ($L \approx M$ *modulo* $\Sigma(L, M)$).

A *substitution*, σ, is a finite mapping from variables to terms denoted by $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where variables $x_1, \ldots, x_n$ are pairwise different and $x_i \neq t_i$ for all $i = 1 \ldots n$. For an expression Ex and a substitution σ, the result of the application of σ to Ex is denoted by $Ex \cdot \sigma$. For any set Ξ of expressions, $\Xi \cdot \sigma$ denotes the set obtained by the application of σ to every expression in Ξ. If Ξ is a set of (at least two) expressions and $\Xi \cdot \sigma$ is a singleton, then σ is called a *unifier* of Ξ.

Expressions of the form $\mathbf{T}\phi$ or $\mathbf{F}\phi$, where ϕ is a formula, are termed *signed formulae*, and $\mathbf{T}$ and $\mathbf{F}$ are called *signs*. A *sequent* is a non-empty multiset of signed formulae having no common bound variables in pairs. Capital Greek letters $\Gamma, \Delta, \ldots$ denote multisets of signed formulae, and we write $\mathbf{T}\Gamma$ (or $\mathbf{F}\Delta$) to express the fact that all formulae in Γ (in Δ) are of the form $\mathbf{T}\phi$ (of the form $\mathbf{F}\psi$, respectively). We denote by $\mathrm{sf}(\Gamma)$ the multiset of all *sign free* formulae obtained from the formulae in Γ by deleting signs. For example, $\mathrm{sf}(\{\mathbf{T}p, \mathbf{F}q\}) = \{p, q\}$.

We say that an occurrence of a subformula ϕ in ψ is

- *positive* if ϕ is ψ;
- *positive* (*negative*) if ψ is of the form $(\chi \wedge \xi)$, $(\xi \wedge \chi)$, $(\chi \vee \xi)$, $(\xi \vee \chi)$, $(\chi \supset \xi)$, $\forall x \xi$, or $\exists x \xi$ and ϕ is positive (negative) in ξ;
- *negative* (*positive*) if ψ is of the form $(\chi \supset \xi)$ or $\neg\chi$ and ϕ is positive (negative) in χ.

The polarity of an occurrence of a subformula ϕ in a sequent $S = \mathbf{T}\Gamma, \mathbf{F}\Delta$ is determined by the polarity of the corresponding occurrence of ϕ in the formula $(\bigwedge \mathrm{sf}(\mathbf{T}\Gamma)) \supset$

$$\frac{}{\Gamma, \mathbf{T}A, \mathbf{F}A}\ (Ax)$$

$$\frac{\Gamma, \mathbf{T}A, \mathbf{T}B}{\Gamma, \mathbf{T}A \wedge B}\ (\mathbf{T}\wedge) \qquad \frac{\Gamma, \mathbf{F}A \quad \Gamma, \mathbf{F}B}{\Gamma, \mathbf{F}A \wedge B}\ (\mathbf{F}\wedge)$$

$$\frac{\Gamma, \mathbf{T}A \quad \Gamma, \mathbf{T}B}{\Gamma, \mathbf{T}A \vee B}\ (\mathbf{T}\vee) \qquad \frac{\Gamma, \mathbf{F}A}{\Gamma, \mathbf{F}A \vee B}\ (\mathbf{F}\vee_1) \qquad \frac{\Gamma, \mathbf{F}B}{\Gamma, \mathbf{F}A \vee B}\ (\mathbf{F}\vee_2)$$

$$\frac{\Gamma, \mathbf{T}A \supset B, \mathbf{F}A \quad \Gamma, \mathbf{T}B, \mathbf{F}\phi}{\Gamma, \mathbf{T}A \supset B, \mathbf{F}\phi}\ (\mathbf{T}\supset) \qquad \frac{\Gamma, \mathbf{T}A, \mathbf{F}B}{\Gamma, \mathbf{F}A \supset B}\ (\mathbf{F}\supset)$$

$$\frac{\Gamma, \mathbf{T}\neg A, \mathbf{F}A}{\Gamma, \mathbf{T}\neg A, \mathbf{F}\phi}\ (\mathbf{T}\neg) \qquad \frac{\Gamma, \mathbf{T}A}{\Gamma, \mathbf{F}\neg A}\ (\mathbf{F}\neg)$$

$$\frac{\Gamma, \mathbf{T}\forall x A(x), \mathbf{T}A(t)}{\Gamma, \mathbf{T}\forall x A(x)}\ (\mathbf{T}\forall) \qquad \frac{\Gamma, \mathbf{F}A(y)}{\Gamma, \mathbf{F}\forall x A(x)}\ (\mathbf{F}\forall)$$

$$\frac{\Gamma, \mathbf{T}A(y)}{\Gamma, \mathbf{T}\exists x A(x)}\ (\mathbf{T}\exists) \qquad \frac{\Gamma, \mathbf{F}A(t)}{\Gamma, \mathbf{F}\exists x A(x)}\ (\mathbf{F}\exists)$$

No sequent contains more than one formula of the form $\mathbf{F}\xi$. In the $\mathbf{T}\supset$ and $\mathbf{T}\neg$ rules, the expression $\mathbf{F}\phi$ might be empty (that is, the sequent contains no formula of the form $\mathbf{F}\psi$). In the rule (Ax), A is an atomic formula. In the rules $(\mathbf{F}\forall)$ and $(\mathbf{T}\exists)$ the variable y has no free occurrences in the conclusions of the rule.

Fig. 1. Tableau calculus TJ for intuitionistic logic

$(\bigvee \mathrm{sf}(\mathbf{F}\Delta))$: If a subformula ϕ occurs positively in $\mathrm{sf}(\mathbf{F}\Delta)$ or negatively in $\mathrm{sf}(\mathbf{T}\Gamma)$, we say that the occurrence of ϕ is *positive* in S, otherwise, the occurrence of ϕ is *negative*.

If an occurrence of a subformula $\forall x\psi$ is positive (or an occurrence of $\exists x\psi$ is negative) in a formula ϕ (or in a sequent S), we say that the quantifier $\forall x$ (respectively, $\exists x$) is *strong* in the formula ϕ (in the sequent S); otherwise, the quantifier $\forall x$ (respectively, $\exists x$) is *weak* in the formula ϕ (in the sequent S). If a quantifier Qx, where Q is $\forall$ or $\exists$, is strong (weak) in a sequent S, the variable x is called *strong* (*weak*) in S.

An indexed variable ${}^k v$ can be a *free variable* or *parameter* depending on v being weak or strong, respectively in a sequent S. For technical reasons only, if v is a weak (strong) variable, then ${}^k\underline{v}$ (${}^k\overline{v}$) denotes its free variable (parameter) 'copy'.

A formula ϕ is intuitionistically valid if, and only if, the sequent $\mathbf{F}\phi$ can be derived, for example, in the calculus TJ adapted from [23], with the sole difference that we use the tableau notation whereas [23] uses the sequential one.

3 Calculus TJ^*

The results of this paper are based on our research published in [10]. In order to make this paper self-contained, we repeat the necessary notions and definitions in this section.

Let $\mu(\phi)$ be the quantifier-free result of removing all quantifiers from ϕ. Let for a formula ϕ fix the one-to-one function ω mapping *strong* in ϕ variable $x \in \mathrm{mVr}$ into the parameter ${}^1\overline{x} \in \mathrm{iVr}$ and a weak in ϕ variable $x \in \mathrm{mVr}$ into the free variable ${}^1\underline{x} \in \mathrm{iVr}$

(mind the left upper indices!). We also assign left upper indices to the occurrences of logical connectives in $\omega(\phi)$—originally, the left upper index of all logical connectives is 1; in the process of derivation other indices are also assigned, see **Convention** below.

For example, if ϕ is $\forall x(\exists y P(x,y) \supset P(x,x))$, then $\mu(\phi) = P(x,y) \supset P(x,x)$, $\omega(\phi) = {}^1\forall^1\underline{x}({}^1\exists^1\overline{y}P({}^1\underline{x},{}^1\overline{y})^1 \supset P({}^1\underline{x},{}^1\underline{x}))$, and $\mu(\omega(\phi)) = P({}^1\underline{x},{}^1\overline{y})^1 \supset P({}^1\underline{x},{}^1\underline{x})$.

We extend the definition of μ and ω to sequents and arbitrary sets of formulae in the obvious way. (There is no ambiguity in the definition of ω since all the formulae of any sequent have no common variables in pairs.)

In any tableaux-style calculus one has to deal with the necessity to apply quantifier rules. A distinctive feature of our approach is that we remove quantifiers from given formulae; and multiple quantifier rule applications can be modelled by means of the (**T**Copying) rule defined below.

If ψ is a formula, Q is one of $\forall$ or $\exists$, and $Qx\phi$ is its subformula, we call $Qx\phi$ a *maximal Q-subformula* of ψ if $Qx\phi$ is not an immediate subformula of another Q-subformula of ψ and we call x a *principal variable* of $Qx\phi$. In addition, all variables bounded by quantifiers within $Qx\phi$ are called *latent* in $Qx\phi$.

For example, both $\phi_1 = \forall x \neg \forall y \forall z P(x,y,z)$ and $\phi_2 = \forall y \forall z P(x,y,z)$ are maximal $\forall$-subformulae of $\psi = \forall x \neg \forall y \forall z P(x,y,z)$. The variables x, y, and z are all latent in ϕ_1, but only y and z are latent variables in ϕ_2. Note that $\forall z P(x,y,z)$ is not a maximal subformula of ψ.

Convention. If ϕ is a maximal Q-subformula containing indexed variables, and j is an index, then ${}^j\phi$ denotes the result of replacing the indices of *all* logical connectives and the indexes of all *latent* variables in ϕ with j. For example, if $\phi = P({}^3\underline{x},{}^1\overline{y})\ {}^3\wedge Q({}^3\underline{x})$, $j = 5$, and both x and y are latent, then ${}^5\phi = P({}^5\underline{x},{}^5\overline{y})\ {}^5\wedge Q({}^5\underline{x})$. If, however, only y is latent in ϕ, then ${}^5\phi = P({}^3\underline{x},{}^5\overline{y})\ {}^5\wedge Q({}^3\underline{x})$.

The notion of a maximal Q-subformula is extended to the case of $\mu\omega$-images of sequents in the following way: If a formula $Qx\phi$ is a maximal Q-subformula of a (usual) sequent S, then for every $j \in \mathbb{N}$, the formula ${}^j\mu(\omega(Qx\phi))$ is a maximal Q-subformula of $S_{\mu\omega}$. Besides, ${}^j x$ is called a *principal variable* of ${}^j\mu(\omega(Qx\phi))$. Moreover, if a variable y is latent in $Qx\phi$, then for every $j \in \mathbb{N}$, the variable ${}^j y$ is latent in ${}^j\mu(\omega(Qx\phi))$.

Assume we are interested in the validity of a closed formula ϕ. In our calculus TJ^*, proof search begins with the starting sequent $S_{\mu\omega} = \mathbf{F}\mu(\omega(\phi))$. The rules of TJ^* are given in Fig. 2. Note that the quantifier rules became redundant and are absent from the calculus.

(Quasi)-proof. A sequent is said to be *closed* if it contains occurrences of both $\mathbf{T}A$ and $\mathbf{F}A$, where A is an atomic formula. A sequent is *quasi-closed* if it contains occurrences of both $\mathbf{T}A$ and $\mathbf{F}B$, where A and B are atomic formulae and $A \approx B$ modulo $\Sigma(A,B)$. Applying the above-mentioned rules 'from top to bottom' to a starting sequent and afterwards to its 'consequences', and so on, we construct a so-called *inference tree* for the starting sequent. An inference tree is called a *quasi-proof (proof) tree for a starting sequent* if all its leaves are quasi-closed (closed).

$$\frac{\Gamma, \mathbf{T}A, \mathbf{T}B}{\Gamma, \mathbf{T}A\,{}^{k}\!\wedge B}\ (\mathbf{T}\wedge) \qquad \frac{\Gamma, \mathbf{F}A \quad \Gamma, \mathbf{F}B}{\Gamma, \mathbf{F}A\,{}^{k}\!\wedge B}\ (\mathbf{F}\wedge)$$

$$\frac{\Gamma, \mathbf{T}A \quad \Gamma, \mathbf{T}B}{\Gamma, \mathbf{T}A\,{}^{k}\!\vee B}\ (\mathbf{T}\vee) \qquad \frac{\Gamma, \mathbf{F}A}{\Gamma, \mathbf{F}A\,{}^{k}\!\vee B}\ (\mathbf{F}\vee_1) \qquad \frac{\Gamma, \mathbf{F}B}{\Gamma, \mathbf{F}A\,{}^{k}\!\vee B}\ (\mathbf{F}\vee_2)$$

$$\frac{\Gamma, \mathbf{T}A\,{}^{k}\!\supset B, \mathbf{F}A \quad \Gamma, \mathbf{T}B, \mathbf{F}\phi}{\Gamma, \mathbf{T}A\,{}^{k}\!\supset B, \mathbf{F}\phi}\ (\mathbf{T}\supset) \qquad \frac{\Gamma, \mathbf{T}A, \mathbf{F}B}{\Gamma, \mathbf{F}A\,{}^{k}\!\supset B}\ (\mathbf{F}\supset)$$

$$\frac{\Gamma, \mathbf{T}\,{}^{k}\!\neg A, \mathbf{F}A}{\Gamma, \mathbf{T}\,{}^{k}\!\neg A, \mathbf{F}\phi}\ (\mathbf{T}\neg) \qquad \frac{\Gamma, \mathbf{T}A}{\Gamma, \mathbf{F}\,{}^{k}\!\neg A}\ (\mathbf{F}\neg)$$

$$\frac{\Gamma, \mathbf{T}\phi, \mathbf{T}\,{}^{l}\!\phi}{\Gamma, \mathbf{T}\phi}\ (\mathbf{T}\text{Copying})$$

No sequent contains more than one formula of the form $\mathbf{F}\xi$. In the $(\mathbf{T}\supset)$ and $(\mathbf{T}\neg)$ rules, the expression $\mathbf{F}\phi$ might be empty (that is, the sequent contains no formula of the form $\mathbf{F}\psi$).
In the rule (**T**Copying):

- ϕ is a maximal $\forall$-subformula of $S_{\mu\omega}$;
- l is a new index, that is, ${}^{l}\!\phi$ does not have common latent variables with other formulae of the sequent.

Fig. 2. Calculus TJ^*

3.1 Main Result for TJ^*

Let ϕ be a formula. By (i,ϕ) we denote the i-th occurrence of a logical connective (which could be a propositional connective or a quantifier) in ϕ when the formula ϕ is read from left to right. If (i,ϕ) is the occurrence of a logical connective $\odot$, we also refer to this occurrence as ${}_i\!\odot$. Moreover, in what follows, any occurrence ${}_i\!\odot$ of a symbol $\odot$ in a formula F is treated as a new symbol. Therefore, ${}_i\!\odot$ and ${}_j\!\odot$ are different symbols which denote the same logical "operation" $\odot$. We also refer to a bound variable x bound by the quantifier ${}_iQx$ as ${}_ix$.

For a formula ϕ, we write ${}_i\!\odot \prec_\phi {}_j\!\odot'$ if, and only if, in ϕ, the selected occurrence ${}_j\!\odot'$ of the logical connective $\odot'$ is in the scope of the selected occurrence ${}_i\!\odot$ of $\odot$. For example, if ϕ is $({}_1\neg\,\psi\ {}_2\wedge\,(\xi\ {}_3\vee\,\chi))$, then ${}_2\wedge \prec_\phi {}_1\neg$ and ${}_2\wedge \prec_\phi {}_3\vee$. We extend the relation $\prec_\phi$ to bound variables: $x \prec_\phi y$ if the quantifier Qy is in the scope of the quantifier $Q'x$ (recall that no two quantifiers in any formula have a common variable, which can be achieved by renaming bound variables). For example, for the formula $\phi = {}_1\forall x\ {}_2\exists y P(x,y)$, we have: $x \prec_\phi y$.

We also extend the (transitive and irreflexive) relation $\prec_\phi$ to the case of indexed logical connectives in the following way: for any i and j and for any formula ϕ, we have ${}^{k}_{i}\!\odot \prec_\phi {}^{l}_{j}\!\odot'$ if, and only if, ${}_i\!\odot \prec_\phi {}_j\!\odot'$. Similarly for indexed variables, ${}^{i}x \prec_\phi {}^{j}y$ if, and only if, $x \prec_\phi y$.

Let Tr be an inference tree in the calculus TJ^* then the union of the relations $\prec_\phi$, defined for all formulae ϕ from Tr, is a transitive and irreflexive relation denoted by $\prec_{Tr}$.

Any substitution σ induces a (possibly empty) relation $\ll_\sigma$ as follows: $y \ll_\sigma x$ if, and only if, there exists $x \mapsto t \in \sigma$ such that x is a free variable, the term t contains y, and y

is a parameter. For example, consider the substitution $\sigma = \{{}^1\underline{x} \mapsto f({}^2\overline{y}, {}^1\underline{y}, {}^1\overline{z})\}$. Then, ${}^2\overline{y} \ll_\sigma {}^1\underline{x}$ and ${}^1\overline{z} \ll_\sigma {}^1\underline{x}$ (note that ${}^1\underline{x}$ and ${}^1\underline{y}$ are not in the relation $\ll_\sigma$).

A substitution σ is *admissible* for a formula ϕ (for an inference tree Tr) if, and only if, for every $x \mapsto t \in \sigma$, x is a free variable, and the transitive closure $\lhd_{\phi,\sigma}$ of $\prec_\phi \cup \ll_\sigma$ ($\lhd_{Tr,\sigma}$ of $\prec_{Tr} \cup \ll_\sigma$) is an irreflexive relation.

Let Tr be an inference tree for a starting sequent $S_{\mu\omega}$ in TJ^*. Suppose ${}^{l_1}_{j_1}\odot_1, \ldots, {}^{l_r}_{j_r}\odot_r$ is the sequence of propositional connectives occurrences in formulae from $S_{\mu\omega}$, which are eliminated in Tr by applying inference rules, written in the rules applications order leading to the construction of Tr. Then the sequence of such rules applications is called *proper* for Tr. We denote such sequence by $\alpha_{Tr}({}^{l_1}_{j_1}\odot_1), \ldots, \alpha_{Tr}({}^{l_r}_{j_r}\odot_r)$. (It must be clear that there can exist more than one proper sequence for an inference tree Tr.)

Further, an inference tree Tr for $S_{\mu\omega}$ is called *compatible* with the substitution σ if, and only if, there exists a proper sequence $\alpha_{Tr}({}^{l_1}_{j_1}\odot_1), \ldots, \alpha_{Tr}({}^{l_r}_{j_r}\odot_r)$ for Tr such that for any natural numbers m and n, the property $m < n$ implies that the ordered pair $\langle {}^{l_n}_{j_n}\odot_n, {}^{l_m}_{j_m}\odot_m\rangle$ does not belong to $\lhd_{Tr,\sigma}$.

The following result can easily be extracted from [10].

Proposition 1. *A sequent* $\mathbf{F}\phi$ *is deducible in the calculus TJ if, and only if, a quasi-proof tree Tr for* $\mathbf{F}(\mu(\omega(\phi)))$ *can be constructed in the calculus* TJ^**, and there exists a substitution* σ *such that (i)* $Tr \cdot \sigma$ *is a proof tree, (ii)* σ *is an admissible substitution for Tr, and (iii) the tree Tr is compatible with* σ*.*

4 Herbrand's Theorem

This section develops the ideas suggested in [12]: Given a first-order intuitionistic formula, we generate ground instances of this formula and then check whether the instances are deducible in a propositional intuitionistic tableaux calculus, provided that the propositional proof is compatible with the way how the instances were generated.

First, we introduce a specialised *convolution calculus*, which allows one to "gather" the required multiple occurrences of subformulae. Then, we introduce the notion of a *Herbrand quasi-universe* and formulate the main result of this paper.

4.1 The Convolution Calculus

We reduce the deducibility of first-order sequents to the deducibility of quantifier-free sequents. Let Tr be an inference tree for a starting sequent S (of the form $\mathbf{F}\phi$) in the calculus TJ^*. To every sequent Sq in Tr, we assign the sequent $\iota(Sq)$, termed the *spur* of Sq, as follows.

- If Sq is a leaf of Tr, having the form $\mathbf{T}F_1, \ldots, \mathbf{T}F_n, \mathbf{F}G$, then $\iota(Sq)$ is $\mathbf{T}F_1, \ldots, \mathbf{T}F_n, \mathbf{F}G$.
- If Sq is not a leaf node and spurs are assigned to all its successors, we assign $\iota(Sq)$ to Sq in accordance with the rules of the convolution calculus given in Fig. 3: If a rule R of the calculus TJ^* is applied to the sequent Sq in Tr, the spur is assigned to Sq as prescribed by the rule $\uparrow R$ of the convolution calculus applied "bottom up".

$$\frac{\Gamma, \mathbf{T}\iota(A), \mathbf{T}\iota(B)}{\Gamma, \mathbf{T}\iota(A) \wedge \iota(B)}\ (\uparrow \mathbf{T}\wedge) \qquad \frac{\Gamma, \mathbf{F}\iota(A) \quad \Gamma, \mathbf{F}\iota(B)}{\Gamma, \mathbf{F}\iota(A) \wedge \iota(B)}\ (\uparrow \mathbf{F}\wedge)$$

$$\frac{\Gamma, \mathbf{T}\iota(A) \quad \Gamma, \mathbf{T}\iota(B)}{\Gamma, \mathbf{T}\iota(A) \vee \iota(B)}\ (\uparrow \mathbf{T}\vee) \qquad \frac{\Gamma, \mathbf{F}\iota(A)}{\Gamma, \mathbf{F}\iota(A) \vee \iota(B)}\ (\uparrow \mathbf{F}\vee_1) \qquad \frac{\Gamma, \mathbf{F}\iota(B)}{\Gamma, \mathbf{F}\iota(A) \vee B}\ (\uparrow \mathbf{F}\vee_2)$$

$$\frac{\Gamma, \mathbf{T}\iota(A) \supset \iota(B), \mathbf{F}\iota(A) \quad \Gamma, \mathbf{T}\iota(B), \mathbf{F}\iota(\phi)}{\Gamma, \mathbf{T}(\iota(A) \supset \iota(B)) \wedge (\iota(A) \supset \iota(B)), \mathbf{F}\iota(\phi)}\ (\uparrow \mathbf{T}\supset) \qquad \frac{\Gamma, \mathbf{T}\iota(A), \mathbf{F}\iota(B)}{\Gamma, \mathbf{F}\iota(A) \supset \iota(B)}\ (\uparrow \mathbf{F}\supset)$$

$$\frac{\Gamma, \mathbf{T}\neg\iota(A), \mathbf{F}\iota(A)}{\Gamma, \mathbf{T}\neg\iota(A) \wedge \neg\iota(A), \mathbf{F}\phi}\ (\uparrow \mathbf{T}\neg) \qquad \frac{\Gamma, \mathbf{T}\iota(A)}{\Gamma, \mathbf{F}\neg\iota(A)}\ (\uparrow \mathbf{F}\neg)$$

$$\frac{\Gamma, \mathbf{T}\iota(\phi), \mathbf{T}\,{}^{l}\iota(\phi)}{\Gamma, \mathbf{T}\iota(\phi) \wedge {}^{l}\iota(\phi)}\ (\uparrow \mathbf{T}\text{Copying})$$

The rule $(\uparrow \mathbf{T}\neg)$ assigns the spur $\Gamma, \mathbf{T}\,\neg\iota(A) \wedge \neg\iota(A), \mathbf{F}\phi$ to the sequent $\Gamma, \mathbf{T}\,{}^{k}\neg A, \mathbf{F}\phi$.

Fig. 3. Convolution Calculus

The result of the assignment of spurs to all sequents in Tr is called the *spurred image* of Tr and is denoted by $\iota(Tr)$. The top node of $\iota(Tr)$ is denoted by $\iota(S)$, where S is a starting sequent. It should be clear that $\iota(Tr)$ consists of quantifier-free formulae only. Moreover, any formula in $\iota(Tr)$ consists of the symbols of the original signature only.

Next, we are going to reduce the deducibility of sequences to the deducibility of spurs in a tableau propositional calculus. Note that, since all the necessary multiple occurrences of subformulae are introduced to the spur, the propositional calculus is contraction free.

Let pTJ be the calculus obtained from TJ by deleting all its quantifier rules and replacing the rules $(\mathbf{T}\supset)$ and $(\mathbf{T}\neg)$ with

$$\frac{\Gamma, \mathbf{F}A \quad \Gamma, \mathbf{T}B, \mathbf{F}\phi}{\Gamma, \mathbf{T}A \supset B, \mathbf{F}\phi}\ (p\mathbf{T}\supset) \qquad \frac{\Gamma, \mathbf{F}A}{\Gamma, \mathbf{T}\neg A, \mathbf{F}\phi}\ (p\mathbf{T}\neg)\,,$$

respectively. We extend the definitions of admissible substitutions and compatibility of inference trees and substitutions to the case of pTJ.

The following properties of proof trees can be easily proved by induction on the number of rules applications.

Proposition 2. *Let Tr be an inference tree for a starting sequent S in the calculus TJ^* and σ a substitution. Then the following properties hold w.r.t. $\iota(Tr)$, $\iota(S)$, and σ:*

1) $\iota(Tr)$ and Tr contain the same variables;

2) $\iota(Tr)$ is an inference tree in TJ^* for $\iota(S)$ (up to multiple applications of the $\mathbf{F}\wedge$);

3) $\iota(Tr) \cdot \sigma$ is an inference tree in the calculus pTJ for the initial sequent $\iota(S) \cdot \sigma$;

4) $\iota(Tr) \cdot \sigma$ is a proof tree in pTJ if, and only if, $Tr \cdot \sigma$ is a proof tree in TJ^*;

5) σ is admissible for $\iota(Tr)$ if, and only if, σ is admissible for Tr;

6) $\iota(Tr)$ is compatible with σ if, and only if, Tr is compatible with σ.

Taking this proposition into account, we can reformulate Proposition 1 as follows.

Proposition 3. *A sequent $\mathbf{F}\phi$ is deducible in the calculus TJ if, and only if, a quasi-proof tree Tr for $\mathbf{F}(\mu(\omega(\phi)))$ can be constructed in the calculus TJ^*, and there exists a*

substitution σ *such that (i)* $\iota(Tr)\cdot\sigma$ *is deducible in pTJ, (ii)* σ *is an admissible substitution for* $\iota(Tr)$*, and (iii) the tree* $\iota(Tr)$ *is compatible with* σ*.*

4.2 Intuitionistic Herbrand Theorem

Now we introduce our modification of the notions of the multiplicity [2, 25] and the Herbrand quasi-universe [12].

Let ϕ be a formula and $\phi_1,\ldots,\phi_n$ its variants. If $\phi,\phi_1,\ldots,\phi_n$ does not have any bound and latent variables in pairs, then $\phi_1\wedge\ldots\wedge\phi_n$ $(n>0)$ is called a *variant n-fold* $\wedge$*-duplication*. The formula $\phi\wedge\ldots\wedge\phi$ is called an *identical n-fold* $\wedge$*-duplication* of ϕ.

Herbrand extension. Let ψ be a formula and ξ a $\wedge$-duplication of ϕ. The result of the replacement of ϕ in ψ with ξ is called a *one-step Herbrand extension of* ψ if one of the following condition is satisfied:

(i) ϕ is a maximal negative $\forall$-subformula of ψ and ξ is a variant duplication of ϕ, which has no common bound and latent variables with ψ;
(ii) ϕ is a negative $\supset$-subformula (negative $\neg$-subformula) of ψ, and ξ is a identical duplication of ϕ.

Finally, the result of a finite sequence of one-step extensions consequently applied to a given formula ψ is called a *Herbrand extension* of ψ.

The notion of a Herbrand quasi-universe, introduced in [12] and modified here for the intuitionistic case, plays the same role in our research as the usual Herbrand universe in the case of classical logic. Unlike the "usual" Herbrand universe, the quasi-universe also contains parameters in the case where strong variables occur in an initial sequent.

Herbrand quasi-universe. Let S be a sequent. Then $HQ(S)$ denotes the following minimal set of terms called a *Herbrand quasi-universe*: (i) every constant and every parameter, occurring in S, belong to $HQ(S)$ (if there is no constant in S then the special constant $c_0\in HQ(S)$); (ii) if f is a k-ary functional symbol and terms $t_1,\ldots,t_k\in HQ(S)$ then $f(t_1,\ldots,t_k)\in HQ(S)$.

Theorem 1. *A sequent* $\mathbf{F}\phi$ *is deducible in the calculus TJ if, and only if, there exist a Herbrand extension* $HE(\phi)$ *of* ϕ *and a substitution* σ *of terms from the Herbrand quasi-universe* $HQ(\mathbf{F}\mu(HE(\phi)))$ *for all free variables in* $\mu(HE(\phi))$ *such that (i)* σ *is an admissible substitution for* $\mu(HE(\phi))$ *and (ii) there exists a proof tree for* $\mathbf{F}\mu(HE(\phi))\cdot\sigma$ *in pTJ, compatible with* σ*.*

Proof. Note that we can assume that σ substitutes terms from a Herbrand quasi-universe only, which is followed from the subformula property of the calculi given above and from the fact that we can restrict ourselves with the consideration of substitutions being simultaneous most general unifiers of certain sets of terms.

Necessity. Let $\mathbf{F}\phi$ be deducible in the calculus TJ. By Proposition 3, there exist a quasi-proof tree Tr for the sequent $\mathbf{F}\mu(\omega(\phi))$ in TJ^* and a substitution σ such that (i) $\iota(Tr)\cdot\sigma$ is a proof tree in pTJ, (ii) σ is admissible for Tr, and (iii) $\iota(Tr)$ is compatible with σ.

Consider the top sequent $\iota(\mathbf{F}\phi)$ of $\iota(Tr)$. By definitions of the convolution calculus and a Herbrand extension, there exists a sequence of one-step Herbrand extensions such that the sequents $\iota(\mathbf{F}\phi)$ and $\mathbf{F}\mu(HE(\phi))$ coincide. This proves both items (i) and (ii).

Sufficiency. As in the case on the necessity, it follows from Proposition 3 and the properties of the convolution calculus and $HE(\phi)$.

We demonstrate our approach on a series of examples.

Example 1 (The rôle of admissibility). Let ϕ be the following formula $\forall x \exists y P(x,y) \supset \exists y' \forall x' P(x',y')$. All Herbrand extensions ξ_k $(= HE(\phi))$ of ϕ are of the form:

$$(\forall\, {}^1x_1 \exists\, {}^1y_1 P({}^1x_1, {}^1y_1) \wedge \ldots \wedge \forall\, {}^1x_k \exists\, {}^1y_k P({}^1x_k, {}^1y_k)) \supset \exists\, {}^1y' \forall\, {}^1x' P({}^1x', {}^1y').$$

The Herbrand quasi-universe for this case is $QH(HE(\mu(\phi))) = \{c_0, {}^1y_1, \ldots, {}^1y_k, {}^1x'\}$. It easy to see that the substitution σ_i of the form $\{{}^1x_i \mapsto {}^1x', {}^1y' \mapsto {}^1y_i, {}^jx \mapsto c_0 : j \neq i\}$ transforms $\mathbf{F}\mu(\xi_k)$ into the sequent

$$\mathbf{F}\mu(\xi_k)\cdot\sigma_i = \mathbf{F}(P(c_0, {}^1y_1) \wedge \ldots \wedge P(c_0, {}^1y_{i-1}) \wedge P({}^1x', {}^1y_i)) \wedge P(c_0, {}^1y_{i+1}) \wedge \ldots \wedge P(c_0, {}^1y_k)) \supset P({}^1x', {}^1y_i),$$

which is deducible in pTJ.

However, σ_i is not admissible for ξ_k since we have ${}^1x_i \prec_{\xi_k} {}^1y_i$, ${}^1y' \prec_{\xi_k} {}^1x'$, ${}^1x' \ll_{\sigma_i} {}^1x_i$, and ${}^1y_i \ll_{\sigma_i} {}^1y'$, and therefore, ${}^1x_i \lhd_{\xi_k,\sigma_i} {}^1x_i$. As a consequence of Theorem 1, $\mathbf{F}\phi$ is not deducible in in TJ. Note $\mathbf{F}\phi$ is also not classically deducible as shown in [12].

Next, consider the formula $\phi = \exists y \forall x P(x,y) \supset \forall x' \exists y' P(x',y')$. Then similarly to the case considered above, $HE(\phi) = \exists\, {}^1y_1 \forall\, {}^1x_1 P({}^1x_1, {}^1y_1) \supset \forall\, {}^1x' \exists\, {}^1y' P({}^1x', {}^1y')$ and the substitution $\sigma = \{{}^1x_1 \mapsto {}^1x', {}^1y' \mapsto {}^1y_1\}$, is admissible for $HE(\phi)$. Moreover, the sequent $\mathbf{F}\mu(HE(\phi))\cdot\sigma$ is deducible in pTJ and any its proof tree is compatible with σ. Thus, the sequent $\mathbf{F}\phi$ is deducible in TJ, i.e. it is intuitionistically valid.

Example 2 (The rôle of multiplicities). Let ϕ be the formula $\exists y \forall x B(x,y) \supset \exists z (B(a,z) \wedge B(b,z))$ and S be the sequent $\mathbf{F}\phi$, where x, y, and z are variables; a and b are constants; and B is a predicate symbol. We show that ϕ is an intuitionistically valid formula.

If we do not introduce copies of subformulae, there is no substitution σ of a term from $HQ(\mu(S)) = \{a,b,y\}$ for the free variable x such that $HQ(\mu(S))\cdot\sigma$ is deducible in pTJ. Therefore, Theorem 1 is not applicable.

If, however, we consider a Herbrand extension of ϕ,

$$HE(\phi) = \exists y (\forall\, {}^1x B({}^1x, y) \wedge \forall\, {}^2x B({}^2x, y)) \supset \exists z (B(a,z) \wedge B(b,z)),$$

and substitution $\sigma = \{{}^1x \mapsto a, {}^2x \mapsto b, z \mapsto y\}$, then the sequent $\mathbf{F}HQ(\mu(S))\cdot\sigma$ is deducible in pTJ. Obviously, σ substitutes terms from $HQ(\mu(HE(\phi)))$ and it is admissible for $\mu(HE(\phi))$. Moreover, any proof tree for $\mathbf{F}\mu(HE(\phi))\cdot\sigma$ in pTJ is compatible with σ. By Theorem 1, we come to the conclusion that the sequent $\mathbf{F}\phi$ is deducible in TJ and, therefore, ϕ is an intuitionistically valid formula.

Example 3 (The rôle of compatibility). Let us consider the formula $\phi = (\neg\forall x P(x) \supset \exists y \neg P(y))$. There exists no Herbrand extension of ϕ but ϕ itself. Therefore, $QH(HE(\phi)) = \{c_0, x\}$ and the only possible admissible substitution $\sigma = \{y \mapsto x\}$ transforms $\mathbf{F}\mu(\phi)$ into the sequent $S = \mathbf{F}\neg P(x) \supset \neg P(x)$, which is deducible in pTJ.

The only possible proof tree Tr for S in pTJ is as follows.

1. $\mathbf{F}\neg P(x) \supset \neg P(x)$ (starting sequent)
2. $\mathbf{T}\neg P(x), \mathbf{F}\neg P(x)$ (from (1), by $(\mathbf{F}\supset)$-rule)
3. $\mathbf{T}\neg P(x), \mathbf{T}P(x)$ (from (2), by $(\mathbf{F}\neg)$-rule)
4. $\mathbf{T}P(x), \mathbf{F}P(x)$ (from (3), by $(\mathbf{T}\neg)$-rule: Axiom)

Notice that Tr is not compatible with σ: the $(\mathbf{F}\supset)$-rule application (the 2nd step) precedes to the $(\mathbf{T}\supset)$-rule application (the 3rd step), although the compatibility condition requires the inverse order of rule applications. Since there is no other way to construct a proof tree for S, we conclude that the sequent $\mathbf{F}\phi$ is not deducible in TJ, as implied by Theorem 1, and the formula ϕ is not intuitionistically valid.

5 Conclusions and Future Work

Herbrand's theorem in intuitionistic context is inherently complex. The ultimate goal is to be able to compute given a first-order formula ϕ a series of its ground Herbrand extensions $\widehat{\phi}_1, \widehat{\phi}_2, \ldots$ in such a way that

$$\phi \text{ is intuitionistically valid if, and only if, for some } n \geq 1, \widehat{\phi}_n \text{ is intuitionistically valid.} \quad (1)$$

To our best knowledge, nobody yet succeeded to do that for arbitrary intuitionistic formulae. We are aware of three approaches to this problem.

Classical Herbrand extension for fragments of intuitionistic logic. For certain fragments of intuitionistic logic, (1) still holds for the classical Skolemisation and Herbrand extensions. The simplest case of intuitionistic formulae in prenex form is considered in [15, 3] while [15, 16] gives a full characterisation of intuitionistic formulae for which the classical Skolemisation and Herbrand extensions do the job. As the example considered in the Introduction shows, the classical Skolemisation does not work for all intuitionistic formulae.

Reduction to a different language. Alternatively, one can reduce validity of first-order formulae to validity of formulae in logics "between" propositional and first-order. This approach was pursued by Fitting through predicate abstractions [5] and by Baaz&Iemhoff through existence predicates and the eSkolemisation [1]. The main disadvantage of such an approach for the automated reasoning community is the necessity to develop specialised provers for these "intermediate" logics.

Constraining proofs. Finally, one can extract an admissible substitution and a Herbrand extension from a sequent, tableau, or connection method proof

[20, 13, 22, 23, 25, 18, 19, 11, 24]. Then a given formula is valid if, and only if, there exists a sequent, tableau, or connection method (resp.) *restricted* proof for the Herbrand extension. Note that the resulting Herbrand extension and the proof search process are tightly integrated with each other, and it is not possible, say, to use Herbrand extension from one approach and proof method from the other.

This papers extends further the third approach. Our main contribution is that we separate the generation of ground instances from checking the propositional deducibility and these two processes are only connected through compatibility check. We can formulate the notion of compatibility with a substitution *regardless* of the particular set of tableaux inference rules used. For example, one can introduce *pLJ*, a variant of Gentzen's intuitionistic calculus *LJ*, which differs from *LJ*, in only that it lacks quantifier rules and the thinning rule. The definition of an inference tree compatible with a substitution can be easily transferred to *LJ*. Then, the following proposition can be proved. (We use tableau form of *LJ*.)

Proposition 4. *A sequent* $\mathbf{F}\phi$ *is deducible in the calculus LJ if, and only if, there exist a Herbrand extension* $HE(\phi)$ *of* ϕ *and a substitution* σ *of terms from the Herbrand quasi-universe* $HQ(\mathbf{F}\mu(HE(\phi)))$ *for all free variables in* $\mu(HE(\phi))$ *such that (i)* σ *is an admissible substitution for* $\mu(HE(\phi))$ *and (ii) there exists a proof tree for* $\mathbf{F}\mu(HE(\phi))\cdot\sigma$ *in pLJ, compatible with* σ.

This result suggests the following conjecture.

Conjecture 1. For any propositional tableau intuitionistic calculus *pC* it is possible to formulate the notion of compatibility with a substitution such that any formula ϕ is intuitionistically valid if, and only if, there are an Herbrand extension $HE(\phi)$ of ϕ and a substitution σ of terms from the Herbrand quasi-universe $HQ(\mathbf{F}\mu(HE(\phi)))$ for all free variables in $\mu(HE(\phi))$ such that (i) $\mu(HE(\phi))\cdot\sigma$ is intuitionistically valid as a propositional formula, (ii) σ is an admissible substitution for $\mu(HE(\phi))$, and (iii) for $\mathbf{F}\mu(HE(\phi))\cdot\sigma$, there exists a proof tree in pC compatible with σ.

We are planning to further explore this conjecture especially for contraction-free [4], multi-succedent [24], and labelled [6] tableau proof systems.

References

1. M. Baaz and R. Iemhoff. On the skolemization of existential quantifiers in intuitionistic logic. *Annals of Pure and Applied Logic*, 2006. to appear.
2. W. Bibel. *Automated Theorem Proving*. Vieweg, Braunschweig, second edition, 1987.
3. K. A. Bowen. An Herbrand theorem for prenex formulas of LJ. *Notre Dame Journal of Formal Logic*, 17(2):263–266, 1976.
4. Roy Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log.*, 57(3):795–807, 1992.
5. Melvin Fitting. A modal herbrand theorem. *Fundam. Inform.*, 28(1-2):101–122, 1996.
6. D. Gabbay. *Labelled deductive systems*. Oxford university press, 1996.
7. G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–433, 1934.

8. R. Hähnle. Tableaux and related methods. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 3, pages 101–178. Elsevier, 2001.
9. J. Herbrand. Recherches sur la théorie de la démonstration, (thesis). In W. Goldfarb, editor, *Logical writings*. Cambridge, 1977.
10. B. Konev and A. Lyaletski. Tableau method with free variables for intuitionistic logic. In *Proceedings of the International IIS: IIPWM'06 Conference*, Advances in Soft Computing, 2006. To appear.
11. C. Kreitz and J. Otten. Connection-based theorem proving in classical and non-classical logics. *J. UCS*, 5(3):88–112, 1999.
12. A. Lyaletski. Sequent forms of Herbrand theorem and their applications. *Annals of Mathematics and Artificial Intelligence*. In press.
13. A. V. Lyaletski. Gentzen calculi and admissible substitutions. In *Actes Préliminaieres du Symposium Franco-Sovietique "Informatika-91"*, pages 99–111, Grenoble, France, 1991.
14. S. Y. Maslov. An inverse method for establishing deducibility of nonprenex formulas of the predicate calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 48–54. Springer, Berlin, Heidelberg, 1983.
15. G. Mints. Herbrand theorem. In *Mathematical Theory of Logical Inference*, pages 311–350. Nauka, Moscow, 1967.
16. G. Mints. The Skolem method in intuitionistic calculi. In *Proc. Steklov Inst. Math.*, volume 121, pages 73–109. 1972.
17. J. Otten. ileanTAP: An intuitionistic theorem prover. In *Proc. TABLEAUX'97*, volume 1227 of *LNCS*, pages 307–312, 1997.
18. J. Otten and C. Kreitz. A connection based proof method for intuitionistic logic. In *Proc. TABLEAUX'95*, volume 918 of *LNCS*, pages 122–137, 1995.
19. J. Otten and C. Kreitz. A uniform proof procedure for classical and non-classical logics. In *KI-96*, volume 1137 of *LNCS*, pages 307–319, 1996.
20. S. Reeves. Semantic tableaux as framework for automated theorem-proving. In C. S. Mellish and J. Hallam, editors, *Proc. AISB-87*, pages 125–139, 1987.
21. J. A. Robinson. A machine oriented logic based on the resolution principle. *J. Assoc. Comput. Mach*, 12:23–41, 1965.
22. N. Shankar. Proof search in the intuitionistic sequent calculus. In *Proc. CADE'92*, volume 607 of *LNCS*, pages 522–536, 1992.
23. A. Voronkov. Proof search in intuitionistic logic based on constraint satisfaction. In *Proc. TABLEAUX'96*, volume 1071 of *LNCS*, pages 312–329, 1996.
24. A. Waaler and L. Wallen. Tableaux for intuitionistic logics. In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 255–296. Kluwer, Dordrecht, 1999.
25. L. Wallen. *Automated Deduction in Nonclassical Logics*. MIT Press: Cambridge, 1990.

Ambiguity Propagating Defeasible Logic and the Well-Founded Semantics

Frederick Maier and Donald Nute

Department of Computer Science and Artificial Intelligence Center
The University of Georgia
Athens, GA 30602
fmaier@uga.edu dnute@uga.edu

Abstract. The most recent version of defeasible logic (Nute, 1997) is related to the well-founded semantics by translating defeasible theories into normal logic programs using a simple scheme proposed in (Brewka, 2001). It is found that by introducing ambiguity propagation into this logic, the assertions of defeasible theories coincide with the well-founded models of their logic program translations. Without this addition, the two formalisms are found to disagree in important cases.

A translation in the other direction is also provided. By treating default negated atoms as presumptions in defeasible logic, normal logic programs can be converted into equivalent defeasible theories.

1 Introduction

This paper relates the most recent version of defeasible logic described in (Nute, 1997) to the well-founded semantics for normal logic programs via a translation scheme. After the scheme is presented, it is shown that in important cases the conclusions of a given defeasible theory do not correspond with the logic program's well-founded model. However, by modifying the proof system of defeasible logic, a new ambiguity propagating variant is created, and for this new variant the correspondence holds.

The translation scheme used here was first proposed in (Brewka, 2001) and was used to compare logic programs under a prioritized well-founded semantics to a variant of defeasible logic presented in (Antoniou *et al.*, 2000). For the purposes of this paper, we will call the defeasible logic presented in (Nute, 1997) NDL to distinguish it from other variants. The ambiguity propagating variant presented here will be called ADL. NDL goes beyond the variants presented in (Antoniou *et al.*, 2000) by including a more extensive treatment of the ways that defeasible rules may conflict and by explicitly considering failures of proofs due to cycles in rules. Both of these are important improvements in defeasible logic, and the impact of these changes are discussed in detail in (Nute, 2001).

Another translation scheme for NDL is found in (Maier & Nute, 2006). Defeasible theories are translated into logic programs via the introduction of new literals explicitly representing rules and conflicts. The scheme has the virtue of allowing one to successfully embed defeasible theories of NDL into logic programs, but it does not reveal any deep connection between defeasible logic and logic programming. The translation

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 306–318, 2006.

there is in turn based upon one found in (Antoniou & Maher, 2002) which uses a notational variant of an earlier version of defeasible logic described in (Nute 1992, 1994). Antoniou and Maher show a relationship between this variant and stable models, but the result is limited to defeasible theories without cycles in their rules. They establish a more general relationship between all defeasible theories and Kunen's three-valued semantics (Kunen, 1987).

The main contribution of this paper is the development of the ambiguity propagating defeasible logic ADL and a demonstration of a correspondence between the assertions of ADL theories and the well-founded models of the translations. The well-founded semantics can thus be viewed as indirectly providing a semantics for ADL. This result allows us to use the translation method together with a proof method that is sound with respect to the well-founded semantics as an implementation of ADL.

Furthermore, it is shown here that normal logic programs can be translated into ADL theories in such a way that the well-founded model of a given logic program corresponds to the assertions of its translation. Each default negated literal '$not\ p$' of the program is treated as a defeasible rule $\emptyset \Rightarrow \neg p$ of the defeasible theory (stating, in effect, that $\neg p$ presumably holds).

2 The Defeasible Logics NDL and ADL

This section presents the language of NDL and its ambiguity propagating counterpart ADL as well as proof systems for each. NDL is presented first, and ADL is presented as a modification of NDL's proof system. For a fuller discussion of NDL, see (Nute, 2001).

The well-formed formulas in the common language of NDL and ADL are literals (atomic sentences and their negations). The language also contains *strict* rules, written $A \rightarrow p$ (where p is a literal and A a finite set of literals), *defeasible* rules (written $A \Rightarrow p$), and *undercutting defeaters* or simply *defeaters* (written $A \rightsquigarrow p$). In the following we will use $\dashrightarrow$ to stand for any rule, whether strict, defeasible, or defeater.

The basic idea behind the proof theory for NDL is that we can derive a literal p from a defeasible theory just in case p is the head of some strict or *undefeated* defeasible rule in the theory and all of the literals in the body of the rule are also derivable. A strict rule with an empty body is called a *fact* and the head of such a rule is by definition always derivable. A defeasible rule with an empty body is called a *presumption* and may be defeated by other rules. The role of defeaters is solely to defeat other arguments that might otherwise establish a literal. E.g., the defeater $q \rightsquigarrow \neg p$ can be used to prevent proving p, but it cannot be used to directly prove $\neg p$.

Definition 1. *A* defeasible theory D *is a triple* $\langle R, C, \prec \rangle$*, where* R *a set of rules, each with a possibly empty antecedent,* C *a set of finite sets of literals in the language of* D *such that for any literal* p *in the language of the theory,* $\{p, \neg p\} \in C$*, and* $\prec$ *an acyclic binary superiority relation over the non-strict rules in* R*.*

Let $D = \langle R, C, \prec \rangle$ be a defeasible theory. The sets of literals in C are called *conflict sets*. Each conflict set $cs \in C$ specifies a set of literals that cannot simultaneously consistently hold. If C only contains sets of the form $\{p, \neg p\}$, we say D has a minimal

conflict set. We say that the conflict set C is closed under strict rules if, for all $cs \in C$, if $A \rightarrow p$ is a rule and $p \in cs$, then $\{A \cup (cs - \{p\})\} \in C$. It is not a necessary condition that a defeasible theory be closed under strict rules, but it is certainly an attractive condition. We will call a defeasible theory *closed* if its conflict set is closed under strict rules.

The proof theory for NDL is based upon argument trees.

Definition 2. *Let D be a defeasible theory and p a literal in the language of D. The expression $D \mid\sim p$ is called a* positive defeasible assertion*, while $D \sim\mid p$ is called a* negative defeasible assertion.

Informally, $D \mid\sim p$ and $D \sim\mid p$ are interpreted to mean that a demonstration (respectively, a refutation) exists for p from D. Note that $D \sim\mid p$ is equivalent to neither $D \mid\sim \neg p$ nor $D \not\mid\sim p$. $D \sim\mid p$ means that there is a demonstration that there is no defeasible proof of p from D.

Definition 3. *τ is an* argument tree *for D iff τ is a finite tree such that every node of τ is labeled either $\mid\sim p$ or $\sim\mid p$ (for some literal p appearing in D).*

Definition 4. *The* depth *of a node n is k iff n has k ancestors in τ. The depth of a tree is taken to be the greatest depth of any of its nodes.*

Definition 5. *Let A be a set of literals, and n a node of a defeasible argument tree τ:*

1. *A* succeeds *at n iff for all $q \in A$, there is a child m of n such that m is labeled $\mid\sim q$.*
2. *A* fails *at n iff there is a $q \in A$ and a child m of n such that m is labeled $\sim\mid q$.*

Definition 6. *τ is a defeasible proof in NDL iff τ is an argument tree for D, and for each node n of τ, one of the following obtains:*

1. *n is labeled $\mid\sim p$ and either:*
   a. *there is a strict rule $r : A \rightarrow p \in R$ such that A succeeds at n, or*
   b. *there is a defeasible rule $r : A \Rightarrow p \in R$ such that A succeeds at n and for all $cs \in C$, if $p \in cs$, then there is a $q \in cs - \{p\}$ such that for all rules $s : B \dashrightarrow q \in R$, either B fails at n or else $s \prec r$.*
2. *n is labeled $\sim\mid p$ and:*
   a. *for all strict rules $r : A \rightarrow p \in R$, A fails at n, and*
   b. *for all defeasible rules $r : A \Rightarrow p \in R$, either*
      i. *A fails at n, or*
      ii. *there is a $cs \in C$ such that $p \in cs$, and for all $q \in cs - \{p\}$, there is a rule $s : B \dashrightarrow q \in R$ such that B succeeds at n and $s \not\prec r$.*
3. *n is labeled $\sim\mid p$ and m has an ancestor m in τ such that m is labeled $\sim\mid p$ and all nodes between n and m are negative defeasible assertions.*

Condition 6.3 is called *failure-by-looping*. Since conclusions cannot be established by circular arguments, failure-by-looping can help to show that a literal cannot be derived from a defeasible theory.

Definition 7. *Where D is a defeasible theory and p is a literal in the language of D, $D \mid\sim_{NDL} p$ iff there is a proof τ in NDL such that the root of τ is labeled $\mid\sim p$, and $D \sim\!\mid_{NDL} p$ iff there is a proof tree τ such that the root of τ is labeled $\sim\!\mid p$. If S is a set of literals in the language of D, $D \mid\sim_{NDL} S$ if and only if for all $p \in S$, $D \mid\sim_{NDL} p$. $D \sim\!\mid_{NDL} S$ iff for some $p \in S$, $D \sim\!\mid_{NDL} p$.*

Some important formal properties of NDL are established in the following theorems.

Theorem 1 (Coherence). *If D is a defeasible theory and $D \mid\sim_{NDL} p$, then $D \not\sim\!\mid_{NDL} p$.*

Theorem 2 (Consistency). *If $D = \langle R, C, \prec\rangle$ is a defeasible theory, C is closed under strict rules, $S \in C$, and $D \mid\sim_{NDL} S$, then $\langle\{A \rightarrow p : A \rightarrow p \in R\}, C, \prec\rangle \mid\sim_{NDL} S$.*

Theorem 3 (Cautious Monotony). *If $D = \langle R, C, \prec\rangle$ is a defeasible theory, $D \mid\sim_{NDL} p$, and $D \mid\sim_{NDL} q$, then $\langle R \cup \{\rightarrow p\}, C, \prec\rangle \mid\sim_{NDL} q$.*

Theorem 1 assures us that we cannot both prove and demonstrate the absence of any proof for the same literal. Theorem 2 says that when conflict sets are closed under strict rules, any incompatible set of literals in NDL derivable from a defeasible theory must be derivable from the strict rules alone. In other words, the defeasible rules of a theory can never introduce any new incompatibilities. Of course, this interpretation of Theorem 2 assumes that all possible incompatibilities are captured in the conflict set of the theory. Cautious Monotony is a principle which many authors working on nonmonotonic reasoning propose as a necessary feature for any adequate nonmonotonic formalism.

The advantages of adding failure-by-looping to our proof theory should be obvious. Consider the simple defeasible theory D_1 below.

Example 1

R_{D_1}: 1) $\rightarrow mammal$
2) $furry, has\_wings \Rightarrow bat$
3) $bat \Rightarrow furry$
4) $bat \Rightarrow has\_wings$
5) $bat \Rightarrow flies$
6) $mammal \Rightarrow \neg flies$

C_{D_1}: $\{bat, \neg bat\}$
$\{furry, \neg furry\}$
$\{has\_wings, \neg has\_wings\}$
$\{mammal, \neg mammal\}$
$\{flies, \neg flies\}$

$\prec_{D_1}$: $\emptyset$

In earlier versions of defeasible logic that lacked failure-by-looping, although we could easily see that there was no way to show $D_1 \mid\sim bat$, we could not demonstrate this in the proof theory, that is, we could not show $D_1 \sim\!\mid bat$. Consequently, neither could we show $D_1 \mid\sim \neg flies$. Failure-by-looping provides a mechanism for showing $D \sim\!\mid_{NDL} bat$, which then allows us to show $D_1 \mid\sim_{NDL} \neg flies$.

When we later define a translation of defeasible theories into standard logic programs in such a way that the consequences of a theory correspond to the well-founded model for the logic program, this example will also serve to show that failure-by-looping is necessary to get this correspondence. Where the theory D_1 above is translated into Π_{D_1}, the literals bat, $furry$, and has_wings are all unfounded, but $\neg flies$ is well-founded. The corresponding literals are undetermined in versions of defeasible logic without

failure-by-looping. Where a defeasible theory has cyclic rules, failure-by-looping is needed to capture within the proof theory the concept of a literal being unfounded.

Adding explicit conflict sets and closing them under strict rules provides an alternative solution to a class of examples that have always seemed odd to the authors. Consider the defeasible theory

Example 2

$$D_2 = \langle \{\Rightarrow p, \Rightarrow q, p \Rightarrow r, q \Rightarrow r, p \rightarrow \neg q, q \rightarrow \neg p\}, C, \emptyset \rangle$$

where C is closed under the strict rules in R. The corresponding default theory is

$$R_2 = \langle \{\neg(p \wedge q)\}, \{\tfrac{:p}{p}, \tfrac{:q}{q}, \tfrac{p:r}{r}, \tfrac{q:r}{r}\} \rangle$$

The classical part of R_2 containing the sentence $\neg(p \wedge q)$, acts like the conflict set in the defeasible theory and ensures that p and q do not both belong to the same extension. R_2 has two extensions containing $\{p, r\}$ and $\{q, r\}$, and the intersection of these extensions contains $\{r\}$. r is a "floating conclusion" of R_2 because it belongs to every extension even though there is no sentence that supports it that belongs to every extension.

This seems unintuitive to us. In NDL, the rules $\Rightarrow p$ and $\Rightarrow q$ conflict with each other since $\{p, q\}$ is a conflict set in the theory. Neither rule takes precedence over the other; so neither consequent is defeasibly derivable. Thus neither of the rules to establish r is satisfied, and r is also not defeasibly derivable. Our proof theory avoids these floating conclusions in an intuitively reasonable way.

2.1 Ambiguity Propagation and ADL

Consider the defeasible theory below in which conflict sets are minimal (from Brewka 2001).

Example 3

$$D_3 = \langle \{\Rightarrow p, \Rightarrow \neg p, \Rightarrow q, p \Rightarrow \neg q\}, C, \emptyset \rangle$$

We have $D_3 \sim\!|_{NDL}\ p$, $D_3 \sim\!|_{NDL}\ \neg p$, $D_3 \sim\!|_{NDL}\ \neg q$, and $D_3 \not\sim\!|_{NDL}\ q$. This shows that NDL is *ambiguity blocking*. An argument exists for p, and hence $\neg q$, but this does not prevent us in NDL from concluding q. Many feel that *ambiguity propagating* systems, where such conclusions are forbidden, are intuitively more reasonable. An ambiguity propagating defeasible logic is presented in (Antoniou & Maher, 2002), and it is this variant that Brewka in (2001) considers for translation into logic programs. He dismisses an earlier, ambiguity blocking variant.

It turns out that a very minor modification to the proof system of NDL produces an ambiguity propagating defeasible logic (ADL), and it is precisely this modification that ensures that the results of a defeasible theory match those of its logic program counterpart under Brewka's translation (so that $D \not\sim\!|_{ADL}\ p$ if and only if $p \in wfm(\Pi_D)$ and $D \sim\!|_{ADL}\ p$ if and only if $\neg p \in wfm(\Pi_D)$). The modification creating ADL affects only part 2.b.ii of Definition 6. It specifies that p is defeated only if every defeasible rule in support of p fails or else is defeated by a satisfied strict rule or a defeasible rule of strictly *higher precedence* for each element $q \in cs - \{p\}$. In NDL, a rule of equal precedence can be used. The modification (Definition 8) is shown below.

Definition 8. *τ is a defeasible proof in ADL iff τ is an argument tree for D, and for each node n of τ, one of the following obtains:*

1. *n is labeled $\mid\sim p$ and either:*
   a. *there is a strict rule $r : A \rightarrow p \in R$ such that A succeeds at n, or*
   b. *there is a defeasible rule $r : A \Rightarrow p \in R$ such that A succeeds at n and for all $cs \in C$, if $p \in cs$, then there is a $q \in cs - \{p\}$ such that for all rules $s : B \dashrightarrow q \in R$, either B fails at n or else $s \prec r$.*
2. *n is labeled $\sim\mid p$ and:*
   a. *for all strict rules $r : A \rightarrow p \in R$, A fails at n, and*
   b. *for all defeasible rules $r : A \Rightarrow p \in R$, either*
      i. *A fails at n, or*
      ii. *there is a $cs \in C$ such that $p \in cs$, and for all $q \in cs - \{p\}$, there is a rule $s : B \dashrightarrow q \in R$ such that B succeeds at n and s is strict or else $r \prec s$.*
3. *n is labeled $\sim\mid p$ and m has an ancestor m in τ such that m is labeled $\sim\mid p$ and all nodes between n and m are negative defeasible assertions.*

Apart from this modification, all other aspects of the proof system are left alone. In example 3 above in ADL, one sees that since the rules for p and $\neg p$ are of the same precedence, neither $D_3 \sim\mid_{ADL} p$ nor $D_3 \sim\mid_{ADL} \neg p$ can be shown. Because $D_3 \sim\mid_{ADL} p$ cannot be shown, $D_3 \sim\mid_{ADL} \neg q$ cannot be shown, and so neither can $D_3 \mid\sim_{ADL} q$. Each of these literals is underdetermined in ADL, neither defeasibly proven nor refuted.

The default theory corresponding to D_3 is

$$R_3 = \langle \emptyset, \{ \tfrac{:p}{p}, \tfrac{:\neg p}{\neg p}, \tfrac{:q}{q}, \tfrac{p:\neg q}{\neg q} \} \rangle$$

R_3 has three default extensions containing $\{p, q\}$, $\{p, \neg q\}$, and $\{\neg p, q\}$. Notice that the intersection of these three extensions is "empty". So the "sceptical" interpretation of R_3 agrees with ADL in this case.

By examining the definition of proof trees for both NDL and ADL, it can be seen that every valid proof in ADL is a valid proof in NDL.

Theorem 4. *If D is a defeasible theory and $D \mid\sim_{ADL} p$, then $D \mid\sim_{NDL} p$. If $D \sim\mid_{ADL} p$, $D \sim\mid_{NDL} p$.*

ADL versions of Theorems 1, 2, and 3 also hold.

3 The Well-Founded Semantics for Normal Logic Programs

A logic program Π consists of a set of rules having the form

$$p \text{ :- } q_1, q_2, \ldots, q_n.$$

where p and each subgoal q_i is an atomic formula, or else such a formula preceded by the symbol not (often called negation-as-failure or negation-by-default). We will call a subgoal in which not occurs negative. The other subgoals are positive.

If a program contains no negative subgoals, then it is called *definite* and has a unique Herbrand model (Emden & Kowalski, 1976). This is often taken to be the intended meaning of the program. Programs in which not appears are called *normal* programs. Importantly, normal programs need not have a single least Herbrand model. E.g., the program $p :- not\ q$ has two minimal Herbrand models: $\{p\}$ and $\{q\}$. The *well-founded semantics* (Gelder, Ross,R) was developed to provide a reasonable interpretation of logic programs containing negation. For every program, a unique well-founded model exists. The well-founded model for the previous program is $\{p, \neg q\}$.

Let Π be a normal logic program containing only ground atoms and B_Π the set of atoms appearing in Π. An interpretation $\mathcal{I}$ of Π is any consistent set of positive and negative literals whose atoms are taken B_Π. If p appears in $\mathcal{I}$, then p is said to be $true$ in $\mathcal{I}$. If $\neg p$ appears in $\mathcal{I}$, then p is *false* in $\mathcal{I}$. If neither p nor $\neg p$ appears in $\mathcal{I}$, then p is said to be *undefined* in $\mathcal{I}$.

A set of atoms $S \subseteq B_\Pi$ is said to be *unfounded* with respect to an interpretation $\mathcal{I}$ iff for each $p \in S$ and for each rule r of Π with head p, there exists a positive or negative subgoal q of r such that

1. q is false in $\mathcal{I}$, or
2. q is positive and $q \in S$.

Unfounded sets are closed under union, and so for any Π and $\mathcal{I}$, there exists a *greatest unfounded set* of Π wrt $\mathcal{I}$, denoted $U_\Pi(\mathcal{I})$:

$$U_\Pi(\mathcal{I}) = \{\bigcup A \mid A \text{ is an unfounded set of } \Pi \text{ with respect to } \mathcal{I}\}.$$

$U_\Pi(\mathcal{I})$ can be viewed as a monotone operator and is used to derive the negative consequences of a program. The *immediate consequence operator* T, defined below, is used to derive the positive consequences of a program.

$$T_\Pi(\mathcal{I}) = \{p \mid r \text{ is a rule of } \Pi \text{ with head } p, \text{ and each } q_i \text{ in the body of } r \text{ is in } \mathcal{I}\}.$$

These two operators are combined to form a third:

$$W_\Pi(\mathcal{I}) = T_\Pi(\mathcal{I}) \cup \neg \cdot U_\Pi(\mathcal{I})$$

where $\neg \cdot U_\Pi(\mathcal{I})$ is the element-wise negation of $U_\Pi(\mathcal{I})$. $U_\Pi(\mathcal{I})$, $T_\Pi(\mathcal{I})$, and $W_\Pi(\mathcal{I})$ are all monotonic, and can be used to define the sequence

1. $\mathcal{I}_0 = \emptyset$
2. $\mathcal{I}_{k+1} = W_\Pi(\mathcal{I}_k)$

The well-founded model of Π, $wfm(\Pi)$, is the least fixed-point of this sequence.

4 Translating Defeasible Theories into Logic Programs

(Brewka, 2001) provides a simple and natural translation scheme to compare a version of defeasible logic (Antoniou & Maher, 2002) to logic programs using a prioritized well-founded semantics. Several examples are presented to demonstrate that the two

systems do not agree, and it is argued there that the results of the defeasible theory are less reasonable.

We have altered the translation scheme to account for extended conflict sets and will use it to compare ADL to the WFS for normal programs. Only finite grounded defeasible theories and programs are considered. Since conflict sets are sufficient to encode negation, we will assume all literals appearing in a defeasible theory are positive ($\neg p$ is represented as p'). The translation thus necessarily yields a normal logic program. Furthermore, since (as discussed in the next section) defeaters and the precedence relation do not add to the expressiveness of defeasible logic, we will assume that no defeaters occur in the theory and that the precedence relation is empty.

Let $D = \langle R, C, \prec \rangle$ be a defeasible theory such that no defeaters occur in R_D and $\prec_D = \emptyset$. We define the logic program Π_D as follows.

1. If $q_1, q_2, \ldots, q_n \rightarrow p \in D$, then $p :\!- q_1, q_2, \ldots, q_n \in \Pi_D$.
2. Let cs_1, cs_2, ..., cs_m be the conflict sets of D containing p and $(a_1, a_2, \ldots, a_m)$ any tuple in $cs_1 - \{p\} \times cs_2 - \{p\} \times \ldots \times cs_m - \{p\}$. If $q_1, q_2, \ldots, q_n \Rightarrow p \in D$, then $p :\!-\ not\ a_1, \ldots,\ not\ a_m, q_1, q_2, \ldots, q_n \in \Pi_D$.

Each a_i corresponds to some literal of a conflict set containing p, and in order for a defeasible rule to succeed at least one literal from each conflict set must fail. Significantly, if conflict sets are minimal, then the above translation of defeasible rules reduces to $p :\!-\ not\ p', q_1, q_2, \ldots, q_n$.

As the following examples show, the results of NDL are often incorrect with respect to the well-founded model of the Brewka inspired translation.

Example 4

$D_4 = \langle \{\rightarrow p, \rightarrow p'\}, \{p, p'\}, \emptyset \rangle$
$\Pi_{D_4} = \{p :\!-\ not\ p'., p' :\!-\ not\ p.\}$

Here, $D_4 \sim\!|_{NDL}\ p$ and $D_4 \sim\!|_{NDL}\ p'$ but the well founded model of Π_{D_4} is empty. According to the WFS, p is undefined.

Example 5

$D_5 = \langle \{p \rightarrow p\}, \{p, p'\}, \emptyset \rangle$
$\Pi_{D_5} = \{p :\!- p\}$

Both $D_5 \sim\!|_{NDL}\ p$ and $D_5 \sim\!|_{ADL}\ p$. Without failure-by-looping, neither of these results could be derived. The well-founded model of the translation is $\{\neg p\}$.

Example 6

The logic program corresponding to D_3 is

$$\Pi_{D_3} = \{p :\!-\ not\ p'., p' :\!-\ not\ p.\ , q' :\!-\ \ not\ q, p., q :\!-\ not\ q'.\}$$

P_4 and P_5 together show that $D \sim\!|_{NDL}\ p$ is equivalent to neither $\neg p \in wfm(\Pi_D)$ nor $\neg p$ undefined. In Example 6, we have $D_3 \sim\!|_{NDL}\ p$, $D_3 \sim\!|_{NDL}\ p'$, $D_3 \sim\!|_{NDL}\ q'$, and $D_3 \not\sim_{NDL}\ q$. NDL is *ambiguity blocking*. However, the well-founded model of logic program is empty. In general, for a defeasible theory D, $D \not\sim_{NDL}\ p$ does not imply $p \in wfm(\Pi_D)$. In some cases at least, the well-founded semantics is more conservative than NDL. In contrast, the assertions of ADL are correct with respect to the well-founded semantics.

Theorem 5. *Let $D = \langle R, C, \emptyset\rangle$ be a defeasible theory of ADL without defeaters. Then $D \not\sim_{ADL}\ p$ if and only if $p \in wfm(\Pi_D)$, and $D \sim\!|_{ADL}\ p$ if and only if $\neg p \in wfm(\Pi_D)$.*

In Example 4, no superior argument exists for either p or p', and so none of $D_4 \sim\!|_{ADL}\ p$, $D_4 \sim\!|_{ADL}\ p'$, $D_4 \not\sim_{ADL}\ p$, or $D_4 \not\sim_{ADL}\ p'$ can be shown. In Example 5, in agreement with both NDL and the WFS, $D \sim\!|_{ADL}\ p$ can be shown using failure-by-looping. In Example 6, arguments exist for both p and p', but no superior argument exists for either. They are both ambiguous, and this ambiguity prevents concluding anything about q or q'.

5 Eliminating Precedence and Defeaters from Defeasible Logic

Theorem 5 assumes that the defeasible theory to be translated does not contain any defeaters and that the precedence relation is empty. However, eliminating these from defeasible logic does not lessen its expressiveness.

For any defeasible theory $D = \langle R_D, C_D, \prec_D\rangle$ we can construct another, E, lacking defeaters and with an empty precedence relation, such that D and E agree on all the literals appearing in D.

The conflict sets in E are minimal and defined using literals from R_E, and $\prec_E$ is empty. R_E is constructed by explicitly representing the rules of D, as described below.

1. If r: $A \rightarrow p \in D$, then r_a = $A \rightarrow supported(r)$, r_b = $supported(r) \rightarrow fires(r)$, and r_c = $fires(r) \rightarrow p$ appear in R_E.
2. If r: $A \Rightarrow p \in D$, then r_a = $A \rightarrow supported(r)$, r_b = $supported(r) \Rightarrow fires(r)$, r_c = $fires(r) \rightarrow p$, appear in R_E:
3. If r: $A \rightsquigarrow p \in D$, then r_a = $A \rightarrow supported(r)$ and r_b = $supported(r) \Rightarrow fires(r)$ appear in R_E.
4. Let $cs = \{q_1, q_2, \ldots q_n, p\}$ be a conflict set of D, r a defeasible or defeater rule with head p and $s_1, s_2, \ldots, s_n$ rules such that s_i has head q_i and $s_i \not\prec r$. If for each s_i, s_i is strict or $r \prec s_i$, the rule $supported(s_1), \ldots supported(s_n) \rightarrow fires(r)'$ occurs in E. Otherwise, the rule $supported(s_1), \ldots supported(s_n) \Rightarrow fires(r)'$ occurs in E.

Theorem 6. *Let D and E be defeasible theories as defined and p a literal of D. Then $D \not\sim p$ if and only if $E \not\sim p$, and $D \sim\!| p$ if and only if $E \sim\!| p$.*

We have added new literals of the form $fires(r)$ and $supported(r)$ which explicitly encode when a rule r of D may fire (i.e., the head is derivable) and when it is supported

(ie., its body is derivable). Intuitively, if r is strict, then it may fire if and only if its body is supported. This is represented in r_a, r_b, and r_c, all of which are strict. If r is defeasible or a defeater, r_b is defeasible, meaning that a rule in E with head $fires(r)'$ can potentially defeat it. If r is a defeater, r_a and r_b bear no relation to p and cannot be used to derive it.

In the item $1.d$ above, we have explicitly represented when a set of rules $s_1, \ldots, s_m$ can be used to defeat another r. We need only consider an s_i if it is not inferior to r, for inferior rules cannot be used to defeat r.

6 Translating Normal Logic Programs into Defeasible Logic

Here we show that normal logic programs can also be translated into defeasible theories so that the assertions of the theory match the well-founded model. Let Π be a normal program and B_Π the set of atoms in Π. For each atom $b \in B_\Pi$, define b' to be a new atom not appearing in Π. Given these, we define a new program Φ as follows.

1. If $p :\!-a_1, \ldots, a_n,\ \ not\ b_1, \ldots,\ not\ b_m \in \Pi$, then $p :\!-a_1, \ldots, a_n, b'_1, \ldots, b'_m \in \Phi$.
2. For all b'_i, $b'_i :\!-\ not\ b_i \in \Phi$.

As each new atom b' has exactly one rule, and the only subgoal of that rule is $not\ b$, b' occurs in the well-founded model of Φ if and only if $\neg b$ does.

Lemma 1. *Let Π and Φ be programs as defined above. For any $b_i \in B_\Pi$, $b'_i \in wfm(\Phi)$ iff $\neg b_i \in wfm(\Phi)$. Also, $\neg b'_i \in wfm(\Phi)$ iff $b_i \in wfm(\Phi)$.*

We may view b' as the positive representation of '$\neg b$' (and for normal programs it is impossible for b' and b to both be in the well-founded model). Furthermore, with respect to the original atoms of B_Π, Π and Φ are equivalent under the WFS.

Lemma 2. *Let Π and Φ be programs as defined above. For all $p \in B_\Pi$, $p \in wfm(\Pi)$ iff $p \in wfm(\Phi)$, and $\neg p \in wfm(\Pi)$ iff $\neg p \in wfm(\Phi)$.*

The relationship between Π and Φ makes a translation of Π into a defeasible theory D_Π apparent. Let $r = p :\!-a_1, \ldots a_n,\ not\ b_1, \ldots,\ not\ b_m$ be a rule of Π. Define r_{D_Π} to be $a_1, \ldots a_n, b'_1 \ldots, b'_m \rightarrow p$. Let $Str = \{r_{D_\Pi} \mid r \in \Pi\}$, and $Pr = \{\emptyset \Rightarrow p' \mid\ not\ p$ occurs in some rule of $\Pi\}$. Given this, D_Π is defined as follows:

$$R_{D_\Pi} = \langle Str \cup Pr, C, \emptyset \rangle$$

where C is minimal. The default literals in the program have become presumptions in the defeasible theory. The rules of the original program are strict in the defeasible logic theory.

It is clear given the above definition that translating D_Π into a logic program yields Φ. From Lemma 1 and Theorem 5 it follows that p is provable in D_Π if and only if p' is refutable, and p' is provable if and only if p is provable. Furthermore, since according to Theorem 5 the assertions of D_Π correspond to the well-founded model of Φ, by Lemma 2 the assertions of D_Π agree with the well-founded model of Π wrt B_Π.

Theorem 7. *Let D_Π be defined as above. For any $p \in B_\Pi$, $D_\Pi \mid\sim_{ADL} p$ iff $D_\Pi \sim\!\!\mid_{ADL} p'$, and $D_\Pi \mid\sim_{ADL} p'$ iff $D_\Pi \sim\!\!\mid_{ADL} p$.*

Theorem 8. *Let Π be a normal program and D_Π its defeasible logic translation. For any atom $p \in B_\Pi$, $D_\Pi \mid\sim_{ADL} p$ iff $p \in wfm(\Pi)$, and $D_\Pi \sim\!\!\mid_{ADL} p$ iff $\neg p \in wfm(\Pi)$.*

Example 7

$$
\begin{array}{lllll}
\Pi = & \{p :- not\ q, & q :- not\ p\} & & \\
\Phi = & \{p :- q', & q :- p', & p' :- not\ p, & q' :- not\ q\} \\
D_\Pi = & \langle\{q' \rightarrow p, & p' \rightarrow q, & \Rightarrow p', \Rightarrow q'\}, & C, \emptyset\rangle
\end{array}
$$

Here we have replaced in the rules of Π each subgoal $not\ p$ (alternatively $not\ q$) with p' (alternatively q')and added the rules $p' :- not\ p$ and $q' :- not\ q$. The explicitly negative literals p' and q' occur nowhere in the original program Π and so it is safe to equate, e.g., $not\ p$ with p'. The well-founded model of both Π and Φ is empty. In D_Π, the presumption of p' prevents concluding q', but there is no superior argument for q, and so q' is not refuted, either. The same holds for p', and because of this nothing can be determined for p or q. The set of assertions of D_Π is empty.

In an extended logic program, an explicitly negative literal p' might already appear in Π, and so the manoeuver of replacing $not\ p$ with p' is no longer acceptable (the equivalence of Φ and Π would no longer hold. For instance, if $\Pi = \{p :- \emptyset, p' :- \emptyset, q :- not\ p\}$, then the well-founded model is $\{p, p', \neg q\}$. However, $\Phi = \{p :- \emptyset, p' :- \emptyset, q :- p', p' :- not\ p\}$, and the well-founded model of Φ is $\{p, p', q\}$.

If we instead replace $not\ p$ with some entirely new literal c_p, then the equivalence is restored. However, in the defeasible logic translation, p and p' do not conflict (the program rule $c_p :- not\ p$ would be interpreted as a $\emptyset \Rightarrow c_p$ and c_p and p would conflict). This at first seems very odd. However, strictly speaking, there really is no connection between p and p' in the WFS, either. The literal p' is just a rather strange looking atom.

7 Conclusion

The two versions of defeasible logic presented here differ in that one, NDL, blocks ambiguity while the other, ADL, does not. In a previous paper (Maier and Nute, 2006) we showed how to translate finite defeasible theories into normal logic programs in such a way that the consequences of the theory in NDL correspond to the well-founded model of the logic program. In this paper, we showed a similar result for ADL. Furthermore, we showed in this paper how to translate finite normal logic programs into defeasible theories in such a way that the correspondence between the well-founded model of the program and the consequences of the defeasible theory in ADL are preserved. The correspondence between logic programs and defeasible theories seems to depend on the fact that ADL is ambiguity preserving rather than ambiguity blocking. The translation from defeasible theories to normal logic programs seems to us simpler and more direct for the case of ADL than NDL. Some might consider this a reason to prefer ADL

over NDL. But different researchers have taken different positions on this issue, and our intuitions favor ambiguity blocking despite the closer correspondence of ADL with well-founded semantics.

In Example 3, the two presumptions $\Rightarrow p$ and $\Rightarrow \neg p$ defeat each other, but in ADL $\Rightarrow p$ and $p \Rightarrow \neg q$ are still available to defeat the presumption $\Rightarrow q$, creating a "zombie path" (Makinson & Schechta, 1991), an argument path that has been "killed" by a defeater but that still has the power to defeat or "kill" other arguments. So we get $D_1 \mid\sim_{NDL} p$ and $D_1 \sim|_{NDL} \{p, \neg p, \neg q\}$, while we get $D_1 \mid\sim_{ADL} \emptyset$ and $D_1 \sim|_{ADL} \emptyset$. In the corresponding default theory, the intersection of all extensions was empty, agreeing with ADL. Nevertheless, we think that lacking overriding reasons for accepting either p or $\neg p$, we should take neither into account when considering q, the approach taken in NDL.

So far as positive consequences are concerned, Example 2 is handled in the same way by both NDL and ADL: we get neither $D_2 \mid\sim_{NDL} r$ nor $D_2 \mid\sim_{ADL} r$. In both systems, the two presumptions $\Rightarrow p$ and $\Rightarrow q$ conflict because $\{p, q\}$ is a conflict set in D_2. We believe this is the correct result in examples of this sort. However, we get $D_2 \sim|_{NDL} \{p, q, r\}$, but $D_2 \sim|_{ADL} \emptyset$.

Our results for defeasible theories and logic programs assume that the theories and programs are finite, but they should also hold for theories that do not have infinite chains of dependency. In a theory like

$$D = \langle \{\Rightarrow p, q_1 \Rightarrow \neg p\} \cup \{q_{i+1} \Rightarrow q_i : i \text{ a positive integer } \}, C, \emptyset \rangle$$

our defeasible logics and well-founded semantics for logic programs diverge. Since proof trees in either NDL or ADL are finite, not even failure-by-looping will allow us to show that p is derivable in either system. However, p will be in the well-founded model for the corresponding logic program. Suppose we admit semi-infinite proofs in NDL and ADL. A semi-infinite proof would be a tree all of whose infinite branches have a node n such that all descendants of n in the infinite branch are labeled with negative defeasible assertions. We think that no additional conditions are needed for semi-infinite proofs, but we have not investigated this problem further. The question whether the consequences of an infinite defeasible theory in such a system corresponds to the well-founded model for the corresponding infinite logic program is interesting in principal, but finite, constructive proof procedures for such theories and logic programs would in the general case not exist.

Translating defeasible theories into logic programs offers one way to implement NDL and ADL. Whether this or some more direct method for computing consequences is the better method depends in large part on the cost of translating defeasible theories into logic programs. We have not done a complete analysis of the complexity issues yet, but at first glance it appears that when conflict sets are closed under strict rules, translating a defeasible theory into a logic program might require more than polynomial time. This might not be so bad if the translation only had to be performed once or if the translation procedure were modular. If new facts are added to a defeasible theory, then these facts can be translated into an already existing program in linear time. But adding new rules, and particularly new defeasible rules, looks like it will also require more than polynomial time. So in applications where new rules will not be added very often, then

translation into a logic program seems to be a reasonable approach. The more often new rules, including new presumptions, are added to a theory, the less attractive this approach appears.

References

Antoniou, G., and Maher, M. J. 2002. Embedding defeasible logic into logic programs. In *Proceedings of ICLP*, 393–404.

Antoniou, G.; Billington, D.; Governatori, G.; Maher, M. J.; and Rock, A. 2000. A family of defeasible reasoning logics and its implementation. In *ECAI*, 459–463.

Brewka, G. 2001. On the relationship between defeasible logic and well-founded semantics. In *LPNMR*, 121–132.

Emden, M. H. V., and Kowalski, R. 1976. The semantics of predicate logic as a programming language. *Journal of the ACM* 23:733–742.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. 1988. Unfounded sets and well-founded semantics for general logic programs. In *Proceedings 7th ACM Symposium on Principles of Database Systems*, 221–230.

Gelder, A. V.; Ross, K. A.; and Schlipf, J. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 221–23.

Kunen, K. 1987. Negation in logic programming. *Journal of Logic Programming* 4:289–308.

Maier, F., and Nute, D. 2006. Relating defeasible logic to the well-founded semantics for normal logic programs. In Delgrande, J. P., and Schaub, T., eds., *NMR*.

Makinson, D., and Schechta, K. 1991. Floating conclusions and zombie paths: two deep difficulties in the 'directly skeptical' approach to inheritance nets. *Artificial Intelligence* 48:199–209.

Nute, D. 1992. Basic defeasible logic. In del Cerro, L. F., and Penttonen, M., eds., *Intensional Logics for Programming*. Oxford University Press. 125–154.

Nute, D. 1994. Defeasible logic. In Gabbay, D., and Hogger, C., eds., *Handbook of Logic for Artificial Intelligence and Logic Programming, Vol. III*. Oxford University Press. 353–395.

Nute, D. 1997. Apparent obligation. In Nute, D., ed., *Defeasible Deontic Logic*, Synthese Library. Dordrecht, Netherlands: Kluwer Academic Publishers. 287–315.

Nute, D. 2001. Defeasible logic: Theory, implementation, and applications. In *Proceedings of INAP 2001, 14th International Conference on Applications of Prolog*, 87–114. Tokyo: IF Computer Japan.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Hierarchical Argumentation

S. Modgil

Advanced Computation Lab, CRUK, London WC2A 3PX
sm@acl.icnet.uk

Abstract. In this paper we motivate and formalise a framework that organises Dung argumentation frameworks into a hierarchy. Argumentation over preference information in a level n Dung framework is then used to resolve conflicts between arguments in a level n-1 framework. We then re-examine the issue of Dung's acceptability semantics for arguments from the perspective of hierarchical argumentation.

1 Introduction

Dung's influential theory of argumentation [8] evaluates the status of arguments by applying a 'calculus of opposition' to a framework $(Args, \mathcal{R})$. It is the abstract nature of Dung's theory that partly accounts for its wide ranging influence. The structure of arguments $Args$ and definition of the conflict based binary relation $\mathcal{R}$ on $Args$ is left unspecified. This enables different argumentation systems with their own defined language, construction of arguments, definition of conflict and relation $\mathcal{R}$, to instantiate a Dung framework in order to evaluate the status of the system's constructed arguments. Furthermore, it has been shown [8] that many of the major species of non-monotonic and logic programming systems turn out to be special forms of Dung's theory. More generally, Dung's theory has established foundations for formalising and analysing the handling of uncertainty and conflict in AI based systems. All the above systems require some notion of preference to resolve conflict. In argumentation terms this means that the defined $\mathcal{R}$ accounts for a preference ordering on arguments based on their relative strength. However, information relevant to establishing a preference ordering ('preference information') may itself be incomplete, uncertain or conflicting. Hence, in this paper we present what we believe to be the first framework for reasoning about, indeed **arguing** over, preference information about arguments. Starting with a Dung framework containing arguments $A1$ and $A2$ that conflict with each other, one could in some meta-logic reason that: 1) $A1$ is preferred to $A2$ because of c (= $B1$), **and** 2) $A2$ is preferred to $A1$ because of c' (= $B2$). Hence, to resolve the conflict between $A1$ and $A2$ requires 'meta-argumentation' to determine which of the conflicting *arguments* $B1$ or $B2$ is preferred. Of course, one may need to ascend to another level of argumentation if there are conflicting arguments $C1$ and $C2$ respectively justifying a preference for $B1$ over $B2$ and $B2$ over $B1$. We therefore propose a hierarchy of Dung frameworks in which level n arguments refer to level $n-1$ arguments, and conflict based relations and preferences between level $n-1$ arguments. The level 1 framework makes no commitment to the system instantiating it, and a minimal set of commitments are made to first order logic based argumentation systems instantiating frameworks at level $n > 1$.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 319–332, 2006.

The requirement for hierarchical argumentation arises from the fact that different principles and criteria ([1]) may be used to valuate the strength of arguments. For example, $A1$ may be preferred to $A2$ by the 'weakest link' principle (the argument's strength is the minimum of the strengths of the argument's constituent sentences) [1], whereas $A2$ may be preferred to $A1$ based on the 'last link' principle (the argument's strength is the strength of the sentence used to derive the argument's claim) [13]. One can then construct arguments justifying use of one principle in preference to the other. Also, for any given principle, the valuations of arguments may vary according to context or perspective. One perspective or source of information for valuating argument strength may indicate that $A1$ is preferred to $A2$, whereas from another perspective $A2$ is preferred to $A1$. To resolve the conflict requires arguing for a preference between perspectives. Furthermore, recent works (e.g., [2, 3, 11]) extending theories of argumentation over beliefs to argumentation over agents' desires and intentions, illustrate requirements for a context dependent account of agents' cognitive processes. For example, an argument $A1$ justifying an action is defined as being in conflict with an argument $A2$ for an alternative action realising the same goal [3, 11]. $A1$ may be preferred to $A2$ based on $A1$'s action involving less resource use. However, $A2$'s action may be more efficacious than $A1$'s action. Resolving the conflict requires a context specific argument justifying which of the resource or efficacy criteria is more important [11].

Reasoning about preferences is also explored in [6, 13], in which the object level language is extended with rules that allow context dependent inference of possibly conflicting relative prioritisations of *rules*. Thus, in these works argument strength is exclusively based on the priorities of their constituent sentences (rules), whereas the framework proposed here allows for argument strength to be based on any number of criteria. Furthermore, our framework gives a clean formal separation of meta and object level reasoning. This is necessary if one is to reason about strengths of arguments as opposed to their constituent sentences (e.g., consider argument strength based on the depth/length of the proof that constitutes the argument, or the value promoted by the argument [4]).

The remainder of this paper is structured as follows. Section 2 reviews the notion of an argumentation system and Dung's theory. We then discuss the semantics of conflict based argument interactions prior to formalisation of hierarchical argumentation frameworks. In section 3 we suggest desiderata by which to assess the suitability of argument acceptability semantics, in domains where hierarchical argumentation over preference information is used to resolve conflicts in argumentation systems formalising reasoning in the presence of logical contradiction. Section 4 concludes with a discussion of future work.

2 Formalising Hierarchical Argumentation

2.1 Preliminaries

Argumentation systems are built around a logical language and associated notion of logical consequence $\Gamma \vdash \alpha$. If $\Delta \subseteq \Gamma$ is the set of premises from which α is inferred, then an argument A claiming α can be represented as the tree or sequence of inferences deriving α from Δ, or as the pair (Δ, α). Whichever representation, we say:

- *support*(A) = Δ and *claim*(A) = α.
- A is *consistent* if *support*(A) is consistent (*support*(A) $\nvdash \bot$)
- A' is a *strict sub-argument* of A if *support*(A') $\subset$ *support*(A).

The conflict based *attack* relation is then defined amongst the constructed arguments, whereupon the *defeat* relation is defined by additionally accounting for the relative strength of (preferences between) the attacking arguments. A Dung framework can then be instantiated by the system's constructed arguments and their relations. Here, we define two notions of a Dung framework:

Definition 1. *Let Args be a finite set of arguments. An attack argumentation framework AF_{at} is a pair (Args, $\mathcal{R}_{at}$) where $\mathcal{R}_{at} \subseteq$ (Args $\times$ Args). A defeat argumentation framework AF_{df} is a pair (Args,$\mathcal{R}_{df}$) where $\mathcal{R}_{df} \subseteq$ (Args $\times$ Args)*

If (A, A'),$(A', A) \in \mathcal{R}_{at}$ then A and A' are said to symmetrically attack, denoted by $A \rightleftharpoons A'$. If only $(A, A') \in \mathcal{R}_{at}$, then A asymmetrically attacks A', denoted $A \rightharpoonup A'$. Where there is no possibility of ambiguity we also use $\rightleftharpoons$ and $\rightharpoonup$ to denote symmetric and asymmetric defeats. We also use this notation to denote frameworks, e.g., ($A \rightleftharpoons A'$, A'') denotes ($\{A, A', A''\}, \{(A, A'), (A', A)\}$).

The *acceptable extensions* (under different semantics $\mathcal{S}$) of a framework are then defined [8]. An argument is justified (rejected) if it belongs to all (no) extensions.

Definition 2. *Let E be a subset of $Args$ in AF = AF_{at} or AF_{df}, and let $\mathcal{R}$ denote either $\mathcal{R}_{at}$ or $\mathcal{R}_{df}$. Then:*

- *E is* conflict-free *iff $\nexists$ A, $A' \in E$ such that $(A, A') \in \mathcal{R}$*
- *An argument A is* collectively defended *by E iff $\forall A'$ such that $(A', A) \in \mathcal{R}$, $\exists$ $A'' \in E$ such that $(A'', A') \in \mathcal{R}$.*

Let E be a conflict-free *subset of $Args$, and let F: $2^{Args} \rightarrow 2^{Args}$ such that $F(E)$= $\{A \in Args \mid A$ is collectively defended by $E\}$.*

- *E is an an admissible extension of AF iff $E \subseteq F(E)$*
- *E is a complete extension of AF iff $E = F(E)$*
- *E is a preferred extension of AF iff E is a maximal (w.r.t set inclusion) admissible extension*

*For $\mathcal{S} \in \{$*complete, preferred*$\}$, let $\{E_1, \ldots, E_n\}$ be the set of all $\mathcal{S}$ extensions of AF. Let $A \in Args$. Then $A \in \mathcal{S}$-*justified*(AF) iff $A \in \bigcap_{i=1}^{n} E_i$; $A \in \mathcal{S}$-*rejected*(AF) iff $A \in Args$ - $(\bigcup_{i=1}^{n} E_i)$*[1]

The following is an example of a concrete argumentation system instantiating an attack framework.

Example 1. Let $\mathcal{L}_1$ be a logical propositional language closed under negation, $\mathcal{K}$ a knowledge base containing a set of named defeasible rules of the form $r : \phi_1 \ldots \phi_{n-1} \Rightarrow \phi_n$, where each ϕ_i is an element of $\mathcal{L}_1$, r is a unique propositional name for the rule, and

[1] We omit definition of the *stable* semantics (a special case of *preferred* semantics) since for some argumentation frameworks no stable extensions exist. We also omit the *grounded* extension since this is equivalent to the intersection of all the complete extensions [8].

ϕ_n is the head and $\phi_1 \ldots \phi_{n-1}$ the antecedent of the rule. Note that the antecedent may be empty, in which case $r :\Rightarrow \phi$ represents an assumption. An argument A constructed from $\mathcal{K}$ is a tree in which each node is of the form $r :\Rightarrow \varphi$, in which case it is a leaf node, or a rule of the form $r : \phi_1 \ldots \phi_n \Rightarrow \varphi$, in which case, for $i = 1 \ldots n$ there exists a child node consisting of a rule with head ϕ_i. If $r : \phi_1 \ldots \phi_n \Rightarrow \varphi$ is the root node of A then *claim*(A) = φ, and *support*(A) = $\{r | r : \phi_1 \ldots \phi_n \Rightarrow \varphi$ is a node in $A\}$. In the following definition of the attack relation we let conflict(ϕ, ϕ') iff $\phi \equiv \neg\phi'$.

Definition 3. *Let A be an argument with claim α, A' an argument with claim β. Then A attacks A' iff conflict(α, β) or there exists a strict sub-argument A'' of A', such that claim(A'') = γ and conflict(α, γ)*

Consider the following example $\mathcal{K}$, the arguments $Args_1$ shown here as support claim pairs, and the instantiated attack argumentation framework AF_{at_1} = ($Args_1$, $\mathcal{R}_{at_1}$) where $\mathcal{R}_{at_1}$ is defined in as in definition 3 :

$\mathcal{K} = \{r1 :\Rightarrow \neg a, r2 :\Rightarrow a, r3 : a \Rightarrow b, r4 :\Rightarrow \neg b\}$

$Args_1 = \{A1 = (\{r1\}, \neg a), A2 = (\{r2\}, a), A3 = (\{r2, r3\}, b), A4 = (\{r4\}, \neg b)\})$.

$$AF_{at_1} = \quad A1 \rightleftharpoons A2 \quad \downarrow \quad A3 \rightleftharpoons A4$$

The preferred/complete extensions of AF_{at_1} are $\{A2, A3\}$, $\{A1, A4\}$ and $\{A2, A4\}$. No argument is preferred/complete-justified.

2.2 Formalising Hierarchical Argumentation Frameworks

Hierarchical argumentation aims at argumentation over preference information so as to define the *defeat* relation on the basis of the *attack* relation and thus enable resolution of conflicts between attacking arguments. In general, A defeats A' if A attacks A', and A' does not 'individually defend' itself against A's attack, ie.:

$$\mathcal{R}_{df} = \mathcal{R}_{at} - \{(A, A') \mid defend(A', A)\}$$

where A' individually defends itself against A if A' is preferred to (and in some cases may be required to attack) A. Hence, given AF_{at_1} = (Args_1, $\mathcal{R}_{at_1}$) instantiated by some argumentation system, then to obtain AF_{df_1} = (Args_1, $\mathcal{R}_{df_1}$) we reason in some first order logic about the strengths and relative preferences of arguments in Args_1, in order to infer wff of the form $defend(\mathcal{A}', \mathcal{A})$ (where $\mathcal{A}'$ and $\mathcal{A}$ name arguments $A', A \in \text{Args}_1$). For example, suppose AF_{at_1} = ($A1 \rightleftharpoons A2$). Neither $A1$ or $A2$ are $\mathcal{S}$-justified. Inferring $defend(\mathcal{A}1, \mathcal{A}2)$ we obtain AF_{df_1} = ($A1 \rightharpoonup A2$). $A1$ is now $\mathcal{S}$-justified.

However, one might be able to infer that $A1$ is preferred to and so defends $A2$'s attack, **and** that $A2$ is preferred to and so defends $A1$'s attack. Hence the requirement that the first order logic itself be the basis for an argumentation system instantiating AF_{at_2} = (Args_2, $\mathcal{R}_{at_2}$) (practical systems for first order argumentation are described in [5]). Arguments B and B' in Args_2, with respective claims $defend(\mathcal{A}2, \mathcal{A}1)$ and $defend(\mathcal{A}1, \mathcal{A}2)$, attack each other. If B is $\mathcal{S}$-justified then $A2$ asymmetrically defeats $A1$, else if B' is $\mathcal{S}$-justified then $A1$ asymmetrically defeats $A2$ in AF_{df_1}. Of course,

to determine which of B and B' are $\mathcal{S}$-justified requires determining which asymmetrically defeats the other in AF_{df_2}, and so 'ascending' to a framework AF_{at_3}. If we can exclusively construct an AF_{at_3} argument C for $defend(\mathcal{B}, \mathcal{B}')$ (or $defend(\mathcal{B}', \mathcal{B})$) then we are done. Otherwise, we may need to ascend to AF_{at_4}, and so on.

Hence, a hierarchical argumentation framework (HAF) is of the form $(AF_{at_1}, \ldots, AF_{at_n})$. For $i > 1$, $AF_{at_i} = (Args_i, \mathcal{R}_{at_i})$ is instantiated by a first order logic based argumentation system where $Args_i$ are constructed from a theory Γ_i of wff in a first order language $\mathcal{L}_i$. Each Γ_i contains a set of formulae obtained by a function $\mathcal{M}_{i-1} : (Args_{i-1}, \mathcal{R}_{at_{i-1}}) \rightarrowtail \wp(\mathcal{L}_i)$, from which one can construct AF_{at_i} arguments valuating the strength of arguments in $Args_{i-1}$. Given these arguments one can also construct AF_{at_i} arguments with claims of the form $preferred(\mathcal{A}', \mathcal{A})$ and $defend(\mathcal{A}', \mathcal{A})$. The latter requires that each Γ_i ($i > 1$) also axiomatise the notion of individual defense. There exist two such notions in the argumentation literature:

$$preferred(\mathcal{A}', \mathcal{A}) \wedge attack(\mathcal{A}', \mathcal{A}) \rightarrow defend(\mathcal{A}', \mathcal{A}) \quad \text{(N1)}$$

or, A' is simply preferred to A:

$$preferred(\mathcal{A}', \mathcal{A}) \rightarrow defend(\mathcal{A}', \mathcal{A}) \quad \text{(N2)}$$

The choice of axiomatisation only makes a difference in the case of asymmetric attacks. If $A \rightharpoonup A'$, then assuming **N1**, A asymmetrically defeats A' irrespective of their relative strength (preference), since the latter does not attack the former and so one cannot infer $defend(\mathcal{A}', \mathcal{A})$. In this case we call $A \rightharpoonup A'$ a *preference independent attack*. Assuming **N2**, A asymmetrically defeats A' only if it is not the case that A' is preferred to A. In this case we call $A \rightharpoonup A'$ a *preference dependent attack*.

Definition 4. *Let $AF = (Args, R_{at})$ and Γ, Γ' first order theories.*

• *Let $\Gamma' = \{\textbf{N1}\} \cup \{attack(\mathcal{A}, \mathcal{A}') | (A, A') \in R_{at}\}$. Then Γ axiomatises preference independent attacks in AF, if $\Gamma' \subseteq \Gamma$ and neither predicate $attack/2$ or $defend/2$ appear in $\Gamma - \Gamma'$*[2]

• *Γ axiomatises preference dependent attacks in AF if **N2** $\in \Gamma$*

We remark on the conditions under which the alternative axiomatisations of attack are appropriate. Argumentation systems in which the asymmetric $A1 \rightharpoonup A2$ arises include:

- logic programming systems (e.g., [13]) where $A1$ proves (claims) what was assumed non-provable (negation as failure) by $A2$. Intuitively, this is formalised as preference independent; $A1$ defeats $A2$ irrespective of their relative preference.

- systems where $A1$'s claim logically contradicts a premise in $A2$'s support (e.g.[2]), or $A1$ denies the link between support and claim of $A2$ (e.g.[12]). These are usually formalised as preference dependent. If $A2$ is preferred to, and so defends $A1$'s attack, then neither defeats the other and both appear in an acceptable extension despite $A1$ and $A2$ being inherently contradictory! We argue that this anomaly be resolved by either reformulating as symmetric attacks, or as preference independent attacks.[3]

[2] This restriction ensures that one can infer $defend(\mathcal{A}, \mathcal{A}')$ only if $(A, A') \in R_{at}$.

[3] In eg. 1, if $A1 \rightharpoonup A3$ is preference dependent and preferred($A3$, $A1$) then neither defeats the other and both appear in a conflict free set! If preference independent then $A1$ defeats $A3$, and if preferred($A2$, $A1$) and hence defeat($A2$, $A1$) then $A3$ will be appropriately reinstated.

- systems in which $A1$ responds to a 'critical question' and $A2$ instantiates an argument scheme [14]. For example, $A2$ instantiates a presumptive scheme justifying a course of action [3]. $A1$ is an argument indicating that $A2$'s action has an unsafe side-effect. This asymmetric attack is appropriately modelled as preference dependent since the arguments do not logically contradict. If *preferred*($A2$,$A1$) then $A1$ does not defeat $A2$ and so both can coexist in an acceptable extension; the action is justified while acknowledging that it has an unsafe side-effect.

We now formally define hierarchical argumentation frameworks and the defeat frameworks obtained from the attack frameworks in a HAF:

Definition 5. *A hierarchical argumentation framework is an ordered finite set of argumentation frameworks* $\Delta = ((Args_1, \mathcal{R}_{at_1}),\ldots,(Args_n, \mathcal{R}_{at_n}))$ *such that for* $i > 1$ *:*

- $\mathcal{L}_i$ *is a first order language whose signature contains the binary predicate symbols* 'preferred'*,* 'attack' *and* 'defend' *and a set of constants* $\{\mathcal{A}_1,\ldots,\mathcal{A}_n\}$ *naming arguments* $\text{Args}_{i-1} = \{A_1, \ldots, A_n\}$
- $Args_i$ *is the set of consistent arguments constructed from a first order theory* Γ_i *in the language* $\mathcal{L}_i$*, where* Γ_i *axiomatises preference dependent or independent attacks in* $AF_{at_{i-1}}$ *and* Γ_i *contains some set* $\mathcal{M}_{i-1}((Args_{i-1}, \mathcal{R}_{at_{i-1}}))$ *of* $\mathcal{L}_i$ *wff*
- $\{(A, A') | A, A' \in Args_i$*, claim(*$A$*) =* defend*(*$\mathcal{X}, \mathcal{Y}$*), claim(*$A'$*) =* defend*(*$\mathcal{Y}, \mathcal{X}$*)*$\} \subseteq \mathcal{R}_{at_i}$.

Definition 6. *(*$\text{AF}_{df_1},\ldots,\text{AF}_{df_n}$*) is obtained from* $\Delta = (\text{AF}_{at_1},\ldots,\text{AF}_{at_n})$ *as follows:*

1) *For* $i = 1\ldots n$*,* $Args_i$ *in* $AF_{df_i} = Args_i$ *in* AF_{at_i}

2) $\mathcal{R}_{df_n} = \mathcal{R}_{at_n}$

3) *For* $i = 1\ldots n\text{-}1$*,* $\mathcal{R}_{df_i} = \mathcal{R}_{at_i} - \{(A,A') \mid$ *defend(*$\mathcal{A}', \mathcal{A}$*) is the claim of a* $\mathcal{S}$*-justified argument of* $AF_{df_{i+1}}\}$

Let $\mathcal{S} \in \{$*complete,preferred*$\}$*. We say that* $A \in \mathcal{S}$*-justified(*Δ*) (*$\mathcal{S}$*-rejected(*Δ*)) iff* $A \in \mathcal{S}$*-justified(*AF_{df_1}*) (*$\mathcal{S}$*-rejected(*AF_{df_1}*))*

From hereon, if Γ_i ($i > 1$) axiomatises preference independent attacks in $AF_{at_{i-1}}$, then we call the HAF 'preference independent'. In what follows we give a concrete example of argumentation systems instantiating a HAF. We will make use of the following definition [5] of an argument constructed from a first order theory (from hereon we assume the usual axiomatisation of real numbers in any first order theory):

Definition 7. *An argument* A *constructed from a first order theory* Γ *is a pair* (Δ, α) *such that: i)* $\Delta \subseteq \Gamma$*; ii)* $\Delta \vdash_{FOL} \alpha$*; iii)* Δ *is consistent and set inclusion minimal. We say that* Δ *is the support and* α *the claim of* A*.*

Example 2. Let $\Delta = (AF_{at_1}, AF_{at_2}, AF_{at_3})$ be a preference independent HAF, where AF_{at_1} is the framework in example 1. We describe AF_{at_2} and AF_{at_3}.

AF$_{at_2}$ **= (Args**$_2$**,** $\mathcal{R}_{at_2}$**)**

Args_2 are constructed from Γ_2 where in addition to **N1** and $\{attack(\mathcal{A}, \mathcal{A}' | (A, A') \in \mathcal{R}_{at_1}\}$, Γ_2 also contains (r_s, h_r, ll and wl respectively denote 'rule_strength', 'head_rule', 'last link' and 'weakest link'):

1. the set $\mathcal{M}_1((Args_1, \mathcal{R}_{at_1}))$ =
   - $\{rule(\mathcal{A}, R) \mid A \in Args_1$, R names a rule in $A\}$ =
     $\{rule(\mathcal{A}1, r1), rule(\mathcal{A}2, r2), rule(\mathcal{A}3, r2), rule(\mathcal{A}3, r3), rule(\mathcal{A}4, r4)\}$
   - $\{h\_r(\mathcal{A}, R) \mid A \in Args_1$, R names the rule that is the root node of $A\}$ =
     $\{h\_r(\mathcal{A}1, r1), h\_r(\mathcal{A}2, r2), h\_r(\mathcal{A}3, r3), h\_r(\mathcal{A}4, r4)\}$
2. valuations of the strength of rules by agents 1 and 2 ($ag1$ and $ag2$) = $\{r\_s(ag1, r1, 0.3), r\_s(ag2, r1, 0.6), r\_s(ag1, r2, 0.4), r\_s(ag1, r3, 0.6), r\_s(ag1, r4, 0.5)\}$
3. (a) $h\_r(\mathcal{A}, R) \wedge r\_s(Source, R, X) \rightarrow val(\mathcal{A}, ll, X)$
   (b) $rule(\mathcal{A}, R) \wedge r\_s(Source, R, X) \wedge \forall R'\ (R' \neq R \wedge rule(\mathcal{A}, R') \wedge r\_s(Source', R', Y) \rightarrow Y \geq X) \rightarrow val(\mathcal{A}, wl, X)$
   (c) $val(\mathcal{A}, P, X) \wedge val(\mathcal{A}', P, Y) \wedge X > Y \rightarrow preferred(\mathcal{A}, \mathcal{A}')$

Let $Args_2$ be defined as in definition 7 and let $\mathcal{R}_{at_2}$ be defined as in definition 3 where the conflict relation is defined as follows: conflict(ϕ, ϕ') if:
- $\phi \equiv \neg\phi'$
- $\phi = r\_s(Source1, R, X)$, $\phi' = r\_s(Source2, R, Y)$, $X \neq Y$
- $\phi = val(\mathcal{A}, P, X)$, $\phi' = val(\mathcal{A}, P, Y)$, $X \neq Y$
- $\phi = defend(\mathcal{A}, \mathcal{A}')$, $\phi' = defend(\mathcal{A}', \mathcal{A})$.

To simplify the presentation we show a subset of the arguments and their attack relations in (Args$_2$, $\mathcal{R}_{at_2}$) noting that the attacks and arguments not shown do not change the final outcome when applying hierarchical argumentation:

$$\begin{array}{ccc} B1 \rightleftharpoons B2 & & B5 \rightleftharpoons B6 \\ \downarrow \quad \downarrow & & \\ B3 \rightleftharpoons B4 & & \end{array}$$

claim($B1$) = $r\_s(ag1, r1, 0.3)$, *claim*($B2$) = $r\_s(ag2, r1, 0.6)$

claim($B3$) = $defend(\mathcal{A}1, \mathcal{A}2)$. $B3$ is an argument based on the last link principle using $r\_s(ag2, r1, 0.6)$. *support*($B3$) also includes $r\_s(ag1, r2, 0.4)$, 3(a), 3(c) and **N1**.

claim($B4$) = $defend(\mathcal{A}2, \mathcal{A}1)$. $B4$ is an argument based on the last link principle using $r\_s(ag1, r1, 0.3)$. *support*($B4$) also includes $r\_s(ag1, r2, 0.4)$, 3(a), 3(c) and **N1**.

claim($B5$) = $defend(\mathcal{A}3, \mathcal{A}4)$. *support*($B5$) includes the last link valuations of $A3$ (= 0.6) and $A4$ (= 0.5), 3(a), 3(c) and **N1**.

claim($B6$) = $defend(\mathcal{A}4, \mathcal{A}3)$. *support*($B6$) includes the weakest link valuations of $A3$ (= 0.4) and $A4$ (= 0.5), 3(b), 3(c) and **N1**

AF$_{at_3}$ = (**Args**$_3$, $\mathcal{R}_{at_3}$)
Args$_3$ are constructed from Γ_3 where in addition to **N1** and $\{attack(\mathcal{B}, \mathcal{B}' | (B, B') \in \mathcal{R}_{at_2}\}$, Γ_3 also contains:

1. the set $\mathcal{M}_2((Args_2, \mathcal{R}_{at_2}))$ =
   a) $\{s\_val(\mathcal{B}, Source, R, X) | B \in Args_2, claim(B) = r\_s(Source, R, X)\}$ =
   $\{s\_val(\mathcal{B}1, ag1, r1, 0.3), s\_val(\mathcal{B}2, ag2, r1, 0.6)\}$
   b) $\{p\_val(\mathcal{B}, \mathcal{A}, \mathcal{A}', P) | B \in Args_2, claim(B) = defend(\mathcal{A}, \mathcal{A}'), (val(\mathcal{A}, P, X) \wedge val(\mathcal{A}', P, Y) \wedge (X > Y) \rightarrow preferred(\mathcal{A}, \mathcal{A}')) \in support(B)\}$ = $\{p\_val(\mathcal{B}3, \mathcal{A}1, \mathcal{A}2, ll), p\_val(\mathcal{B}4, \mathcal{A}2, \mathcal{A}1, ll), p\_val(\mathcal{B}5, \mathcal{A}3, \mathcal{A}4, ll), p\_val(\mathcal{B}6, \mathcal{A}4, \mathcal{A}3, wl)\}$

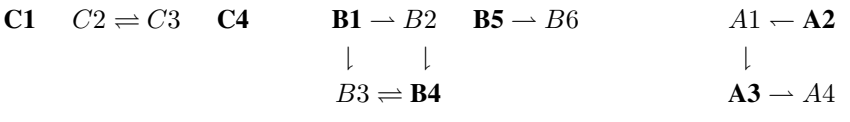
2. a set Π of named partial orderings[4], where if $\wp$ is the name of an ordering in Π, then this is represented by the usual first order reflexivity, antisymmetry and transitivity axioms, and formulae of the form $_{>}(\wp, J, K)$ interpreted as source/principle J is prioritised above source/principle K. In this example we simply have:
$\Pi = \{_{>}(ag\_order1, ag1, ag2)\ ,\ _{>}(princ\_order1, ll, wl)\}$
3. $s\_val(\mathcal{B}, Source1, R, X) \wedge s\_val(\mathcal{B}', Source2, R, Y) \wedge\ _{>}(O, Source1, Source2) \rightarrow preferred(\mathcal{B}, \mathcal{B}')$
4. $p\_val(\mathcal{B}, \mathcal{A}, \mathcal{A}', P1) \wedge p\_val(\mathcal{B}', \mathcal{A}', \mathcal{A}, P2) \wedge\ _{>}(O, P1, P2) \rightarrow preferred(\mathcal{B}, \mathcal{B}')$

Let $Args_3$ be defined as in definition 7 and $\mathcal{R}_{at_3}$ defined as in definition 3 where conflict(ϕ, ϕ') if $\phi \equiv \neg\phi'$ or $\phi = defend(\mathcal{B}, \mathcal{B}')$, $\phi' = defend(\mathcal{B}', \mathcal{B})$. We show a subset of the arguments and their attack relations in (Args$_3$, $\mathcal{R}_{at_3}$):

$$C1 \qquad C2 \rightleftharpoons C3 \qquad C4$$

claim($C1$) = $defend(\mathcal{B}1, \mathcal{B}2)$ where *support*($C1$) includes the formulae in 1(a), $_{>}(ag\_order1, ag1, ag2)$, rule 3 and **N1**.
claim($C2$) = $defend(\mathcal{B}3, \mathcal{B}4)$, *claim*($C3$) = $defend(\mathcal{B}4, \mathcal{B}3)$, *claim*($C4$) = $defend(\mathcal{B}5, \mathcal{B}6)$, where the support of each includes the formulae in 1b), rule 4 and **N1**. $C2$ and $C3$'s supports include $_{>}(princ\_order1, ll, ll)$ (by reflexivity). $C4$'s support includes $_{>}(princ\_order1, ll, wl)$.

Applying definition 6 to Δ obtains the following defeat frameworks with $\mathcal{S}$-justified arguments shown in bold (only a subset of AF_{df_2} and AF_{df_3} are shown):



$\{A2, A3\}$ is now the single preferred/complete extension of AF_{df_1} and set of preferred/complete-justified arguments of AF_{df_1} (and hence Δ). Note that if other orderings were available, e.g., an ordering ranking agent 2 higher than agent 1, then the resulting mutual attacks amongst AF_{at_3} arguments would require ascending to AF_{at_4} in which some contextual justification for preferring one ranking over another could be constructed.

We conclude this section by proving some properties of preference independent HAFs that will be referred to in the following section.

Proposition 1. *Let* $((Args_1, \mathcal{R}_{df_1}), \ldots, (Args_n, \mathcal{R}_{df_n}))$ *be obtained from the* preference independent *HAF* $((Args_1, \mathcal{R}_{at_1}), \ldots, (Args_n, \mathcal{R}_{at_n}))$. *Then:*

a) *If* $(C, B) \in \mathcal{R}_{at_i}$ *and* $(B, C) \notin \mathcal{R}_{at_i}$ *then* $(C, B) \in \mathcal{R}_{df_i}$
b) *If* $(B, C), (C, B) \in \mathcal{R}_{at_i}$ *then* (B, C) *and/or* $(C, B) \in \mathcal{R}_{df_i}$
c) *If* $(B, C) \in \mathcal{R}_{df_i}$ *then* $(B, C) \in \mathcal{R}_{at_i}$

[4] In practice, ordering information may itself be inferred, e.g., an ordering on agents inferred from data describing the relative positions of the agents in an organisation's hierarchy.

Proof. *c) holds since (by def.6-3) $\mathcal{R}_{df_i} \subseteq \mathcal{R}_{at_i}$. To show a) and b) we first show that:*

A $\in$ $\mathcal{S}$-justified($(Args_i, \mathcal{R}_{df_i})$) and claim(A) = $defend(\mathcal{X}, \mathcal{Y})$, implies (X,Y) $\in$ $\mathcal{R}_{at_{i-1}}$ (1)

By def.5, $\forall A \in Args_i$, A is consistent, and so given def.4, if claim(A) = $defend(\mathcal{X}, \mathcal{Y})$,

*then A must be constructed using **N1** and only if $(X, Y) \in \mathcal{R}_{at_{i-1}}$*

a) and b) hold for i = n since (by def.6-1 and def.6-2) $\mathrm{AF}_{df_n} = \mathrm{AF}_{at_n}$. For $i \neq n$:

*• Assume a) does not hold, i.e., (C,B) $\in \mathcal{R}_{at_i}$, (B,C) $\notin \mathcal{R}_{at_i}$ and (C,B) $\notin \mathcal{R}_{df_i}$. By def.6-3, if (C,B) $\notin \mathcal{R}_{df_i}$ then there must be an $\mathcal{S}$-justified argument of $\mathrm{AF}_{df_{i+1}}$ with claim $defend(\mathcal{B}, \mathcal{C})$. Hence, by **(1)**, (B,C) $\in \mathcal{R}_{at_i}$ contradicting the assumption.*

• Assume b) does not hold, i.e., (B,C),(C,B) $\in \mathcal{R}_{at_i}$, (B,C),(C,B) $\notin \mathcal{R}_{df_i}$ and so by def.6-3, $\exists$ D,D′ $\in$ $\mathcal{S}$-justified($\mathrm{AF}_{df_{i+1}}$) such that claim(D) = defend($\mathcal{C}$,$\mathcal{B}$), claim(D′) = defend($\mathcal{B}$,$\mathcal{C}$)

1. ***For i = n - 1**: By def.5, (D,D′), (D′,D) $\in \mathcal{R}_{at_n}$, and since $\mathcal{R}_{df_n} = \mathcal{R}_{at_n}$, (D,D′), (D′,D) $\in \mathcal{R}_{df_n}$, contradicting the assumption that D, D′ $\in$ $\mathcal{S}$-justified(AF_{df_n}).*
   ***Inductive hypothesis**: **(b)** holds for $m > i$.*
2. ***For arbitrary i**: By def.5, (D,D′), (D′,D) $\in \mathcal{R}_{at_{i+1}}$ and by inductive hypothesis, (D,D′) and/or (D′,D) $\in \mathcal{R}_{df_{i+1}}$, contradicting the assumption that D, D′ $\in$ $\mathcal{S}$-justified($\mathrm{AF}_{df_{i+1}}$)*

Corollary 1. *Let (AF_{df_1},. . .,AF_{df_n}) be obtained from the* preference independent *HAF (AF_{at_1},. . .,AF_{at_n}). Then E is a conflict free subset of $Args_i$ in AF_{at_i} iff E is a conflict free subset of $Args_i$ in AF_{df_i}.*

3 Assessing Acceptability Semantics from the Perspective of Hierarchical Argumentation

Comparative assessments of Dung's acceptability semantics against certain benchmark example frameworks (e.g., [10]) have been critiqued (e.g., [7]) on the grounds that they are inherently ad hoc. We suggest an assessment be made against desiderata of a more general nature than examples. However, it is unrealistic to expect a universal set of desiderata given the heterogeneity of domain to which argumentation theory has been applied. For example, in a legal context, burden of proof considerations might warrant semantics that commit to smaller sets of acceptable arguments. On the other hand, one would want to maximise - *within reason* - the number of arguments considered justified in an argumentation system formalising reasoning in the presence of logical contradiction (analogous to maximising persistence in theories of belief revision). In the latter case, α is a consequence of an inconsistent knowledge base $\mathcal{K}$ iff it is the claim of a justified argument (e.g., [1]). Applying hierarchical argumentation to such systems recognises that preference information may well be incomplete, uncertain and conflicting. With this in mind, we informally articulate the notion of maximising *within reason* the number of arguments considered justified: *A is a justified argument iff it is justified irrespective of the availability of further preference information*. We now assess

Dung's complete and preferred semantics w.r.t this requirement. In dealing with argumentation systems formalising reasoning in the presence of logical contradiction, we (as suggested in the previous section) focus on preference independent HAFs. In what follows we make use of the following concepts.

Definition 8. *Let AF = (Args,$\mathcal{R}$) be an attack or defeat framework.* AF′ *=(Args,$\mathcal{R}'$) is a* **partial resolution** *of* AF *iff* $\forall A, A' \in Args$*:*

1. *if* $(A, A') \in \mathcal{R}$ *and* $(A', A) \notin \mathcal{R}$ *then* $(A, A') \in \mathcal{R}'$
2. *if* (A, A'),$(A', A) \in \mathcal{R}$ *then* (A, A') *and/or* $(A', A) \in \mathcal{R}'$
3. *if* $(A, A') \in \mathcal{R}'$ *then* $(A, A') \in \mathcal{R}$.

• *We say that* AF′ *=(Args,$\mathcal{R}'$) is a* **resolution** *of* AF *iff it is a* **partial resolution** *of* AF*, and if* (A, A'),$(A', A) \in \mathcal{R}$ *then it is not the case that* (A, A') ***and*** $(A', A) \in \mathcal{R}'$.

• *Let* AF_{df_1} *be obtained from* $\Delta = (AF_{at_1}, \ldots, AF_{at_n})$*. Let* AF'_{df_1} *be obtained from some* $\Delta' = (AF_{at_1}, \ldots, AF'_{at_n})$ *such that* AF'_{df_1} *is a* **resolution** *of* AF_{df_1}*. Then we say that* Δ' **resolves** Δ.

Proposition 1 states that utilising the available preference information in a preference independent HAF (AF_{at_1},. . .,AF_{at_n}) results in AF_{df_i} that are partial resolutions of AF_{at_i}, i.e., AF_{df_i} differs from AF_{at_i} only in that symmetric attacks are replaced by asymmetric defeats (this is not the case if preference dependent attacks are axiomatised). Intuitively then, each Δ' resolving Δ represents a case in which the available preference information in Δ' is consistent and complete; in the sense that all symmetric attacks in AF_{at_1} that remain as symmetric defeats in AF_{df_1} obtained from Δ, are resolved in favour of asymmetric defeats in the resolution AF'_{df_1} (of AF_{df_1}) obtained from Δ'. We therefore state the following desideratum requiring that an argument be justified iff justified irrespective of how the preference information is consistently completed (recall that $A \in \mathcal{S}$-justified(Δ) iff $A \in \mathcal{S}$-justified(AF_{df_1}) where AF_{df_1} is obtained from Δ by def. 6):

$A \in \mathcal{S}$-justified(Δ) iff for all Δ' such that Δ' resolves Δ, $A \in \mathcal{S}$-justified(Δ') (D1)

Abstracting from the specific binary relation (be it attack or defeat), **D1** expresses a relationship between the justified arguments of a framework *AF* and those justified in every resolution *AF′* of *AF*. Hence, **D1** is satisfied if the following is satisfied:

$A \in \mathcal{S}$-justified(AF) iff for all resolutions AF' of AF, $A \in \mathcal{S}$-justified(AF') (D2)

We therefore assess Dung's preferred and complete semantics w.r.t **D2**. Theorem 1 implies that the left to right half of **D2** holds for the complete semantics. Consider the following counter-example for the preferred: *AF* = ($A \rightharpoonup B \rightharpoonup C \rightharpoonup A$, $A \rightleftharpoons D$, $C \rightharpoonup E$) - $\{B, D, E\}$ is the unique preferred extension. However, $\emptyset$ is the unique preferred extension of the resolution *AF′* in which $A \rightleftharpoons D$ is replaced by $A \rightharpoonup D$.

Theorem 1. *Let* AF′ *= $(Args, \mathcal{R}')$ be a partial resolution of* AF *= $(Args, \mathcal{R})$. Then* complete-justified*(*AF*)* $\subseteq$ complete-justified*(*AF′*).*

Proof. *The* complete-justified *arguments of an argumentation framework are the same as in the* grounded *extension [8]. Hence, we show that the grounded extension of* AF *is*

a subset of the grounded extension of AF′. *Dung makes use of iterative application of the operator F (in def.2) -* $F^0 = \emptyset$, $F^{i+1} = \{A \in Args | A$ *is collectively defended by* $F^i\}$ *- to show (if* $Args$ *is finite) that the grounded extension is given by* $\bigcup_{i=0}^{\infty}(F^i)$.
Let $G = F$ *where* G *applies to* AF′ *and* F *to* AF. *We need to show that if* $A \in F^i$ *then* $A \in G^i$*:*

- i = 1*:* $F^1 = F(F^0)$ *contains arguments* A *that are not attacked/defeated, and since (by def.8-3)* $\mathcal{R}' \subseteq \mathcal{R}$, *then* $A \in G^1$. (1)

- For $i > 1$, *to show* $A \in G^i$ *we need to show that for any* $A \in F^i$*:* $(B, A) \in \mathcal{R}$ *and (hence, given the definition of 'collectively defend' in def.2)* $\exists C.C \in F^{i-1}$ *and* $(C, B) \in \mathcal{R}$, *and* $(B, A) \in \mathcal{R}'$, ***implies*** $\exists C'.C' \in G^{i-1}$ *and* $(C', B) \in \mathcal{R}'$ (2)
Assume $(B, A) \in \mathcal{R}$, $\exists C.C \in F^{i-1}$ *and* $(C, B) \in \mathcal{R}, (B, A) \in \mathcal{R}'$*:*

- ***i = 2*** *(*$F^2 = F(F^1)$*): We have* $\exists C.C \in F^1$ *and by* ***(1)*** $C \in G^1$, *and since* $(C, B) \in \mathcal{R}$ *and* $\neg\exists D$ *s.t.* $(D, C) \in \mathcal{R}$, *then by definition 8-1)* $(C, B) \in \mathcal{R}'$
  Inductive hypothesis (IH)*: (2) holds for* $A \in F^j$ $j < i$
- ***i > 2****: Suppose* $(B, C) \notin \mathcal{R}$. *Then by definition 8-1),* $(C, B) \in \mathcal{R}'$, *and by* ***IH*** $C \in G^{i-1}$. *Suppose* $(B, C) \in \mathcal{R}$ *and* $(C, B) \notin \mathcal{R}'$. *By assumption of* $(C, B) \in \mathcal{R}$ *and definition 8-2),* $(B, C) \in \mathcal{R}'$. *By* ***IH****,* $C \in G^{i-1}$ *and we can substitute* C *for* A *in* ***(2)****. We have* $(B, C) \in \mathcal{R}$ *and (hence)* $\exists C''$. $C'' \in F^{i-2}$ *and* $(C'', B) \in \mathcal{R}$, *and* $(B, C) \in \mathcal{R}'$ *and so* $\exists C'$. $C' \in G^{i-2}$ *and* $(C', B) \in \mathcal{R}'$.

Theorem 2 states that the right to left half of **D2** holds for the preferred semantics. Consider the following counter-example for the complete semantics: *AF* = ($A \rightleftharpoons B, B \rightharpoonup C, A \rightharpoonup C$, $C \rightharpoonup D$). Then *complete-justified*(AF) = $\emptyset$, and yet D is a complete-justified argument of both resolutions ($A \rightharpoonup B, B \rightharpoonup C, A \rightharpoonup C$, $C \rightharpoonup D$) and ($B \rightharpoonup A, B \rightharpoonup C, A \rightharpoonup C$, $C \rightharpoonup D$). Proof of theorem 2 requires the following lemma.

Lemma 1. *E is an admissible extension of AF iff there exists a resolution AF' of AF such that E is an admissible extension of AF'.*

Proof. *Corollary 1 states the equivalence of conflict free subsets of arguments of* AF_{at} *in a HAF and the obtained partial resolution* AF_{df}. *Hence, E is a conflict free subset of $Args$ in AF iff there exists a resolution AF' of AF s.t. E is a conflict free subset of $Args$ in AF'.*
Left to right half: let A be any argument in E and $\{B_1, \ldots B_n\}$ the set s.t. for $i = 1 \ldots n$, $(B_i, A) \in \mathcal{R}$. By definition 2 there exists a $\{C_1, \ldots C_n\} \subseteq E$ s.t. for $i = 1 \ldots n$ $(C_i, B_i) \in \mathcal{R}$. Let AF' be a resolution s.t. for $i = 1 \ldots n$ $(C_i, B_i) \in \mathcal{R}'$. We have that E is a conflict free subset of $Args$ in AF'. By def.8-3), $\{(B, A) | (B, A) \in \mathcal{R}'\} \subseteq \{(B_1, A), \ldots (B_n, A)\}$. Hence, $\forall B$ s.t. $(B, A) \in \mathcal{R}'$, $\exists C$ s.t. $(C, B) \in \mathcal{R}'$, i.e., E is an admissible extension of AF'.
Right to left half: let A be any argument in E, $\{B_1, \ldots B_n\}$ the set s.t. for $i = 1 \ldots n$, $(B_i, A) \in \mathcal{R}'$, and $\{C_1, \ldots C_n\} \subseteq E$ s.t. for $i = 1 \ldots n$ $(C_i, B_i) \in \mathcal{R}'$. We have that E is a conflict free subset of $Args$ in AF. By def.8-3), for $i = 1 \ldots n$, (B_i, A),$(C_i, B_i) \in \mathcal{R}$. Assume a B s.t. $(B, A) \in \mathcal{R}$, $(B, A) \notin \mathcal{R}'$. By def.8-1) it must be the case that $(A, B) \in \mathcal{R}$. Hence, $\forall B$ s.t. $(B, A) \in \mathcal{R}$, $\exists C$ s.t. $(C, B) \in \mathcal{R}$, i.e., E is an admissible extension of AF.

Theorem 2. *If for all resolutions AF' of AF, $A \in$ preferred-justified(AF'), then $A \in$ preferred-justified(AF)*

Proof. *Proof is by contraposition. Assume $A \notin$ preferred-justified(AF), i,e, there exists a preferred extension E of AF s.t. $A \notin E$. Let E' be any superset of E and A any argument in $(E' - E)$. By definition of preferred extensions (def.2) and the monotonicity of F [8] in def.2, $\exists\, (B, A) \in \mathcal{R}$ and $\neg\exists C \in E'$ s.t. $(C, B) \in \mathcal{R}$. By lemma 1, there exists a resolution AF' of AF s.t. E is an admissible extension of AF'. It must be the case that $(B, A) \in \mathcal{R}'$. Suppose otherwise. Then by assumption of $(B, A) \in \mathcal{R}$ and def.8-1), it must be the case that $(A, B) \in \mathcal{R}$ contradicting $\neg\exists C \in E'$ s.t. $(C, B) \in \mathcal{R}$. Since AF' is a resolution then it remains the case that $\neg\exists C \in E'$ s.t. $(C, B) \in \mathcal{R}'$. Hence E is a preferred extension of AF', i.e., $A \notin$ preferred-justified(AF').*

To summarise, the complete semantics will never be 'in error' in that if A is justified in a framework, then it is justified irrespective of how the preference information is consistently completed (theorem 1). The trade of is that A may not be justified even though it 'should' be (in the sense that it is justified irrespective of how the preference information is consistently completed). The preferred semantics will accept as justified all such arguments (theorem 2). However they may be 'in error' in that A may be justified in a framework, but not justified irrespective of how the preference information is consistently completed. However, since we are interested in maximising the number of justified arguments, we suggest opting for the preferred rather than complete semantics. This is because if A is preferred-justified in a framework then every argument B attacking/defeating A is rejected in the framework and **in all consistent completions**. To see why, suppose $A \in$ preferred-justified(AF), in which case A is in every preferred extension of AF. Hence, $\forall B\ (B, A) \in \mathcal{R}$, B is not in any preferred extension, i.e., $B \in$ preferred-rejected(AF). Suppose a resolution AF' s.t. $A \notin$ preferred-justified(AF'). By def. 8-3) $\mathcal{R}' \subseteq \mathcal{R}$. Hence, for any B such that $(B, A) \in \mathcal{R}'$, $B \in$ preferred-rejected(AF) and by theorem 3 below, $B \in$ preferred-rejected(AF'). Recall the counter-example preceding theorem 1. Although B, D and E are preferred-justified in AF but not in a resolution, they are respectively attacked/defeated by A, A and C, where A and C are rejected in AF and all resolutions of AF. That the preferred semantics does not satisfy the left to right half of **D2** is related to the fact that A and C belong to a pathologically problematical odd cycle of attacks/defeats.

Theorem 3. *For $\mathcal{S} \in \{$complete,preferred$\}$, $A \in \mathcal{S}$-rejected(AF) iff for all resolutions AF' of AF, $A \in \mathcal{S}$-rejected(AF')*

Proof. *Left to right half: Proof is by contraposition. Assume a resolution AF' s.t. $A \notin \mathcal{S}$-rejected(AF'). Hence, there exists a $\mathcal{S}$ extension E of AF' s.t. $A \in E$. By def.2 E is admissible. By lemma 1, E is an admissible extension of AF. By def.2, $\exists\ E' \supseteq E$ s.t. E' is a $\mathcal{S}$ extension of AF. Since $A \in E'$, $A \notin \mathcal{S}$-rejected(AF).*
Right to left half: Proof is by contraposition. Assume $A \notin \mathcal{S}$-rejected(AF). Hence, there exists a $\mathcal{S}$ extension E of AF s.t. $A \in E$. E is admissible, and by lemma 1 there exists a resolution AF' s.t. E is an admissible extension of AF', and so there exists a $\mathcal{S}$ extension $E' \supseteq E$ of AF' s.t. $A \in E'$, i.e., $A \notin \mathcal{S}$-rejected(AF').

4 Conclusions

We have formalised hierarchical argumentation over preference information. Arguments in a level n Dung framework resolve conflicts between arguments in a $n-1$ framework. Our approach is applicable to a wide range of argumentation systems given that no commitments are made to the system instantiating the level 1 Dung framework, and that the two widely used notions of the relationship between attack and defeat are axiomatised. Future work will further substantiate the generality of our approach. In particular we aim to formalise and extend value based argumentation [4] as an instance of hierarchical argumentation in which preference dependent attacks are formalised. We believe that hierarchical argumentation can also address challenges raised by applications of argumentation theory in agent and multi-agent contexts [2, 3, 11, 9] in which interacting arguments over different epistemological categories will require different notions of conflict and conflict based interaction, and different principles by which the relative strengths of arguments are evaluated, all within a single system. For example, argumentation-based dialogues require that agents justify their preference for one argument over another, and have this justification itself challenged (e.g.,[9]).

In this paper we also contributed to a general understanding of the relative strengths and weaknesses of Dung's preferred and complete semantics, assessing them against desiderata motivated by application of hierarchical argumentation to argumentation systems for reasoning in the presence of logical contradiction. While neither semantics fully satisfy these desiderata, we argued in favour of the preferred semantics. Future work will also explore related issues raised by the application of hierarchical argumentation to preference information. For example, one could state that an argument is 'objectively' justified, if justified independently of a preference over principles by, and or perspectives from, which argument strength is valuated.

Finally, one of our basic aims has been to put the general idea of meta-argumentation on the map. We share this aim with [15] in which the focus is on reasoning about the construction of arguments rather than preference information.

Acknowledgements. This work was funded by the EU FP6-IST-002307 ASPIC project.

References

1. L. Amgoud and C. Cayrol. Inferring from inconsistency in preference-based argumentation frameworks. *International Journal of Automated Reasoning*, Volume 29 (2):125–169, 2002.
2. L. Amgoud and S. Kaci. On generation of bipolar goals in argumentation-based negotiation. In I. Rahwan, P. Moraitis, and C. Reed, editors, *Proc. 1st Int. Workshop on Argumentation in Multi-Agent Systems*, New York, 2004. Springer.
3. L. Atkinson. *What Should We Do?: Computational Representation of Persuasive Argument in Practical Reasoning*. PhD thesis, Dept. Computer Science, University of Liverpool, 2005.
4. T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *Journal of Logic and Computation*, 13(3):429–448, 2003.
5. P. Besnard and A. Hunter. Practical first-order argumentation. In *Proc. 20th American National Conference on Artificial Intelligence (AAAI'2005)*, pages 590–595, 2005.
6. G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19, 1996.

7. M. Caminada. *For the sake of the Argument. Explorations into argument-based reasoning*. PhD thesis, Department of Computer Science, Free University Amsterdam, 2004.
8. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.
9. D. Hitchcock, P. McBurney, and S. Parsons. A framework for deliberation dialogues. In H. V. Hansen et.al, editor, *Proc. Fourth Biennial Conference of the Ontario Society for the Study of Argumentation (OSSA 2001)*, Canada, 2001.
10. H. Jakobovits and D. Vermeir. Robust semantics for argumentation frameworks. *Journal of logic and computation*, 9(2):215–261, 1999.
11. S. Modgil. Nested argumentation and its application to decision making over actions. In *Proc. Second Int. Workshop on Argumentation in Multi-Agent Systems*, Netherlands, 2005.
12. J. L. Pollock. Defeasible reasoning. *Cognitive Science*, 11:481–518, 1987.
13. H. Prakken and G. Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, 7:25–75, 1997.
14. D. N. Walton. *Argument Schemes for Presumptive Reasoning*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1996.
15. M. Wooldridge, P. McBurney, and S. Parsons. On the meta-logic of arguments. In *AAMAS '05: Proc. Fourth international joint conference on Autonomous agents and multiagent systems*, pages 560–567, NY, USA, 2005. ACM Press.

Anti-prenexing and Prenexing for Modal Logics

Cláudia Nalon[1] and Clare Dixon[2]

[1] Departamento de Ciência da Computação, Universidade de Brasília
Caixa Postal 4466 – CEP:70.910-090 – Brasília – DF – Brazil
nalon@unb.br
[2] Department of Computer Science, University of Liverpool
Liverpool, L69 7ZF – United Kingdom
C.Dixon@csc.liv.ac.uk

Abstract. Efficient proof methods for normal modal logics are highly desirable, as such logical systems have been widely used in computer science to represent complex situations. Resolution-based methods are often designed to deal with formulae in a normal form and the efficiency of the method (also) relies on how efficient (in the sense of producing fewer and/or shorter clauses) the translation procedure is. We present a normal form for normal modal logics and show how the use of simplification, for specific normal logics, together with anti-prenexing and prenexing techniques help us to produce better sets of clauses.

1 Introduction

Beliefs, knowledge, intentions, desires, and obligations of agents as well as the behaviour of these (and possibly other) aspects over time are often used to describe complex situations in computer science. This is the case, for instance, in the specification of distributed [3] and multi-agent systems [11]. Normal modal logics are often chosen to model and reason about these situations. Given a logical specification, an automated tool such as, for instance, a theorem prover, can then be used for verifying properties of the system. However, in order to model the different aspects of a complex, particular situation, it may be necessary to combine different logical languages. When the combination is given by the *fusion* of logical systems, that is, when the components are independently axiomatisable, proofs can be obtained by combining the provers for each language. Combining those provers may require special care such that all relevant information is correctly handled and exchanged between the different tools. Also, this may require the use of tools which are based on different implementations (e.g. different input languages) or, worse, on different approaches (e.g. partially based on translation to first-order language × partially based on the modal language, resolution × tableau, etc), making this task harder.

We are currently investigating a uniform approach which deals with theorem proving for a variety of *propositional normal modal logics*, that is, logics in which the schema $\Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi)$ (the axiom **K**), where $\Box$ is the modality

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 333–345, 2006.

for *necessity* and φ and ψ are well-formed formulae, is valid. We are interested in multi-modal normal logics based on the following axioms

$$\begin{aligned}
&\mathbf{K} : \Box(\varphi \Rightarrow \psi) \Rightarrow (\Box\varphi \Rightarrow \Box\psi)\\
&\mathbf{T} : \Box\varphi \Rightarrow \varphi\\
&\mathbf{D} : \Box\varphi \Rightarrow \Diamond\varphi\\
&\mathbf{4} : \Box\varphi \Rightarrow \Box\Box\varphi\\
&\mathbf{5} : \Diamond\varphi \Rightarrow \Box\Diamond\varphi\\
&\mathbf{B} : \Diamond\Box\varphi \Rightarrow \varphi
\end{aligned}$$

where $\Diamond$ is the modality for *possibility*. We formally introduce the weakest of these logics, $\mathsf{K}_{(n)}$, in Section 2. The proof method for each logic is resolution-based and our approach is clausal: in order to prove that a formula, φ, is valid, we first transform its negation, $\neg\varphi$, into a clausal normal form. Here we do not focus on the proof method, which can be found in [8], but on the properties that a normal form for these logics should have in order to achieve efficiency. We discuss briefly some aspects that should be considered when designing a normal form for a *family* of logics.

In the classical propositional case there is only one resolution rule to apply to the set of clauses, whilst we often need several rules for the modal case. This happens because the semantics of modal logics are relative to a set of worlds, so we often need to perform reasoning tasks which are not local (to the actual world). Also, if we consider multi-agents contexts, the resolution rules must consider the different contexts (relative to each agent) in which the reasoning applies. Separating these contexts may facilitate the reasoning task. Thus, it should be taken into consideration when designing a normal form for these logics.

Also, we should think of strategies that could be used to reduce the proof search. Our work is based on that of [1], but the normal form differs slightly (where l_i are literals, i.e., propositions or their negations): clauses are separated into literal clauses (disjunctions of literals), positive modal clauses (an implication as $l_1 \Rightarrow \Box l_2$), and negative modal clauses (an implication as $l_1 \Rightarrow \neg\Box l_2$). This further separation potentially allows a better design of strategies to guide the selection of clauses to which apply the resolution method. For instance, complete strategies for (purely) propositional logic could be used when the parents are literal clauses. The set of support strategy could also be used when the parents are modal clauses, taking, for instance, the positive modal clauses in the usable set and the negative modal clauses in the set of support (or vice-versa).

The new normal form is given in Section 3. Transformation into clausal form is carried out by performing classical style rewriting, simplification, and renaming [10], a technique which may avoid combinatorial explosion on the size of the formula by replacing complex subformulae by new symbols, whose meaning are linked to the formula that they are replacing.

Efficient translation is crucial for practical use of the resolution method. By *efficient* we mean that the translation method produces fewer or shorter clauses. In first-order logic, it has been shown [2] that the transformation of a given problem into anti-prenex normal form (i.e. when quantifiers are moved inwards a formula) results in a better set of clauses. However, to the best of our

knowledge, there has not been a similar investigation for modal logics. We present an algorithm for anti-prenexing in Section 4. Experimentally, anti-prenexing together with simplification, given in Section 5, followed by transformation into the normal form performs better than both the transformation preceded by anti-prenexing and transformation alone.

We introduce the prenex normal form in Section 6 and show how this can also be used to reduce the nesting of modal operators *after* anti-prenexing. Simplification for specific logics is used in both steps, anti-prenexing and prenexing. Preliminary results show that the set of clauses is usually smaller than that obtained by translation into the normal form alone.

Experimental results are given in Section 7. We provide concluding remarks in Section 8.

2 The Basic Normal Logic

The basic normal modal logic that we present here is known as $\mathsf{K}_{(n)}$. This is the weakest of the normal modal systems, where only the distribution axiom (the axiom **K**) holds. There is no restriction on the accessibility relation over worlds. As the subscript in the name of the logic indicates, we consider the multi-agent version, given by the fusion of several copies of $\mathsf{K}_{(1)}$, one for each agent.

Formulae are constructed from a denumerable set of *propositional symbols*, $\mathcal{P} = \{p, q, p', q', p_1, q_1, \ldots\}$. The finite set of agents is defined as $\mathcal{A} = \{1, \ldots, n\}$. In addition to the standard propositional connectives $(\neg, \vee, \wedge, \Rightarrow)$, we introduce a set of unary modal operators $\boxed{1}, \ldots, \boxed{n}$, where $\boxed{i}\varphi$ is read as "agent i considers φ necessary". When $n = 1$, we may omit the index, that is, $\Box\varphi = \boxed{1}\varphi$. We do not define the operator $\Diamond$: the fact that an "agent i considers φ possible" is expressed by $\neg\boxed{i}\neg\varphi$. The language of $\mathsf{K}_{(n)}$ is defined as follows:

Definition 1. *The* ***set of well-formed formulae****,* $\mathsf{WFF}_{\mathsf{K}_{(n)}}$*:*

- *the propositional symbols are in* $\mathsf{WFF}_{\mathsf{K}_{(n)}}$*;*
- **true** *and* **false** *are in* $\mathsf{WFF}_{\mathsf{K}_{(n)}}$*;*
- *if* φ *and* ψ *are in* $\mathsf{WFF}_{\mathsf{K}_{(n)}}$*, then so are* $\neg\varphi$*,* $(\varphi \wedge \psi)$*,* $(\varphi \vee \psi)$*,* $(\varphi \Rightarrow \psi)$*, and* $\boxed{i}\varphi$ $(\forall i \in \mathcal{A})$*.*

The following definitions will also be used later.

Definition 2. *A* ***literal*** *is either a proposition or its negation.*

Definition 3. *A* ***modal literal*** *is either* $\boxed{i}l$ *or* $\neg\boxed{i}l$*, where* l *is a literal and* i *is in the set of agents,* $\mathcal{A} = \{1, \ldots, n\}$*.*

Definition 4. *A formula* χ *is* ***disjunctive*** *if, and only if, is of the form* $(\varphi \Rightarrow \psi)$*,* $(\varphi \vee \psi)$ *or* $\neg(\varphi \wedge \psi)$*. Otherwise,* χ *is said to be* ***conjunctive****.*

Polarity of a formula is defined as usual: if a formula is inside the scope of an even (including zero) number of negation symbols, the formula is said to be of *positive* polarity; otherwise, it is of *negative* polarity.

Semantics of $\mathsf{K}_{(n)}$ is given, as usual, in terms of a Kripke structure.

Definition 5. *A **Kripke structure** M for n agents over $\mathcal{P}$ is a tuple $M = \langle \mathcal{S}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$, where $\mathcal{S}$ is a set of possible* worlds *(or* states*) with a distinguished world s_0; the function $\pi(s) : \mathcal{P} \to \{true, false\}$, $s \in \mathcal{S}$, is an interpretation that associates with each state in $\mathcal{S}$ a truth assignment to propositions; and $\mathcal{R}_i$ is a binary relation on $\mathcal{S}$.*

The binary relation $\mathcal{R}_i$ is intended to capture the possibility relation according to agent i. So, a pair (s, t) is in $\mathcal{R}_i$ if agent i considers world t possible, given her information in world s. In $\mathsf{K}_{(n)}$, the relations are any subsets of $S \times S$.

Truth is defined in terms of the relation $\models$. We write $(M, s) \models \varphi$ to express that φ is true at world s in the Kripke structure M.

Definition 6. *Truth of a formula is given as follows:*

- $(M, s) \models$ **true**
- $(M, s) \not\models$ **false**
- $(M, s) \models p$ *if, and only if,* $\pi(s)(p) = true$*, where* $p \in \mathcal{P}$
- $(M, s) \models \neg\varphi$ *if, and only if,* $(M, s) \not\models \varphi$
- $(M, s) \models (\varphi \wedge \psi)$ *if, and only if,* $(M, s) \models \varphi$ *and* $(M, s) \models \psi$
- $(M, s) \models (\varphi \vee \psi)$ *if, and only if,* $(M, s) \models \varphi$ *or* $(M, s) \models \psi$
- $(M, s) \models (\varphi \Rightarrow \psi)$ *if, and only if,* $(M, s) \models \neg\varphi$ *or* $(M, s) \models \psi$
- $(M, s) \models \boxed{i}\varphi$ *if, and only if, for all* t*, such that* $(s, t) \in \mathcal{R}_i$*,* $(M, t) \models \varphi$*.*

Formulae are interpreted with respect to the distinguished world s_0. Intuitively, s_0 is the world from which we start reasoning. Let $M = \langle \mathcal{S}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ be a Kripke structure. Thus, a formula φ is said to be *satisfiable in* M if $(M, s_0) \models \varphi$; it is said to be *satisfiable* if there is a model M such that $(M, s_0) \models \varphi$; and it is said to be *valid* if for all models M then $(M, s_0) \models \varphi$.

3 A Normal Form for $\mathsf{K}_{(n)}$

Formulae in the language of $\mathsf{K}_{(n)}$ can be transformed into a normal form called Separated Normal Form for Normal Logics (SNF_K). We introduce a nullary connective **start**, in order to represent the world from which we start reasoning. Formally, we have that $(M, s) \models$ **start** if, and only if, $s = s_0$. A formula in SNF_K is represented by a conjunction of clauses, which are true at all states, that is, they have the general form

$$\Box^* \bigwedge_i A_i$$

where A_i is a clause and $\Box^*$, the universal operator, is introduced. Its semantics is defined as:

$(M, s) \models \Box^*\varphi$ if, and only if, $(M, s) \models \varphi$ and for all s' such that $(s, s') \in \mathcal{R}_i$, for some $i \in \mathcal{A}, (M, s') \models \Box^*\varphi$.

Observe that φ holds at the actual world s and at every world reachable from s, where reachability is defined in the usual way. That is, let M be a model and u and u' be worlds in M. Then u' is reachable from u if, and only if, either (i) $(u, u') \in \mathcal{R}_i$ for some agent $i \in \mathcal{A}$; or (ii) there is a world u'' in M such that u'' is reachable from u and u' is reachable from u''. The universal operator, which surrounds all clauses, ensures that the *translation* of a formula is true at all worlds. Clauses are in one of the following forms:

- Initial clause $\quad \mathbf{start} \Rightarrow \bigvee_{b=1}^{r} l_b$
- Literal clause $\quad \mathbf{true} \Rightarrow \bigvee_{b=1}^{r} l_b$
- $\boxed{i}$-clause $\quad l \Rightarrow m_i$

where l and any l_b are literals and m_i is a modal literal containing a $\boxed{i}$ or a $\neg\boxed{i}$ operator. In general, that is, when we do not need to specify a particular agent, we often say *modal clause* to refer to a $\boxed{i}$-clause.

3.1 Transformation into Normal Form

The translation to SNF_K uses the renaming technique [10], where complex subformulae are replaced by new propositional symbols and the truth of these new symbols is linked to the formulae that they replaced in all states. The translation into SNF_K of a given formula φ of $\mathsf{K}_{(n)}$ is given by the following transformation functions, τ_0 and τ_1, where x is a new propositional symbol:

$$\tau_0(\varphi) = \Box^*(\mathbf{start} \Rightarrow x) \wedge \tau_1(\Box^*(x \Rightarrow \varphi))$$

The function τ_0 is used to anchor the meaning of φ to the initial world, where the formula is evaluated. The function τ_1 proceeds with the translation, removing classical operators, by means of classical rewriting operations, and replacing complex formulae which appear in the scope of the $\boxed{i}$ operator, by means of renaming. The next rewriting rules deal with classical operators (where A and B are formulae, and x is the propositional symbol introduced by τ_0):

$$\begin{aligned}
\tau_1(\Box^*(x \Rightarrow \neg\neg A)) &= \tau_1(\Box^*(x \Rightarrow A)) \\
\tau_1(\Box^*(x \Rightarrow (A \wedge B))) &= \tau_1(\Box^*(x \Rightarrow A)) \wedge \tau_1(\Box^*(x \Rightarrow B)) \\
\tau_1(\Box^*(x \Rightarrow (A \Rightarrow B))) &= \tau_1(\Box^*(x \Rightarrow \neg A \vee B)) \\
\tau_1(\Box^*(x \Rightarrow \neg(A \wedge B))) &= \tau_1(\Box^*(x \Rightarrow \neg A \vee \neg B)) \\
\tau_1(\Box^*(x \Rightarrow \neg(A \Rightarrow B))) &= \tau_1(\Box^*(x \Rightarrow A)) \wedge \tau_1(\Box^*(x \Rightarrow \neg B)) \\
\tau_1(\Box^*(x \Rightarrow \neg(A \vee B))) &= \tau_1(\Box^*(x \Rightarrow \neg A)) \wedge \tau_1(\Box^*(x \Rightarrow \neg B))
\end{aligned}$$

We rename complex subformulae enclosed in a modal operator as follows, where y is a new proposition and A is not a literal.

$$\tau_1(\Box^*(x \Rightarrow \boxed{i}A)) = \tau_1(\Box^*(x \Rightarrow \boxed{i}y)) \wedge \tau_1(\Box^*(y \Rightarrow A))$$
$$\tau_1(\Box^*(x \Rightarrow \neg\boxed{i}A)) = \tau_1(\Box^*(x \Rightarrow \neg\boxed{i}\neg y)) \wedge \tau_1(\Box^*(y \Rightarrow \neg A))$$

Next we use renaming on formulae whose right-hand side has disjunction as its main operator but may not be in the correct form (where y is a new proposition, D is a disjunction of formulae, A is not a literal or an implication, and D' and D'' are formulae):

$$\tau_1(\Box^*(x \Rightarrow D \vee (D' \Rightarrow D''))) = \tau_1(\Box^*(x \Rightarrow D \vee \neg D' \vee D''))$$
$$\tau_1(\Box^*(x \Rightarrow D \vee A)) = \tau_1(\Box^*(x \Rightarrow D \vee y)) \wedge \tau_1(\Box^*(y \Rightarrow A))$$

Finally, we rewrite formulae whose right-hand side is a disjunction of literals into clause form, that is, as an implication. Modal clauses whose right-hand side is a modal literal are already in the normal form, so no further transformation is required. Note that each modal clause contains only one modal literal. So, the different contexts belonging to different agents are already separated at the end of the translation and we do not require further renaming as in [1].

$$\tau_1(\Box^*(x \Rightarrow D)) = \begin{cases} \Box^*(\mathbf{true} \Rightarrow \neg x \vee D) & \text{if } D \text{ is a disjunction of literals} \\ \Box^*(x \Rightarrow D) & \text{if } D \text{ is a modal literal} \end{cases}$$

As an example, the translation of $\boxed{1}(a \Rightarrow b) \Rightarrow (\boxed{1}a \Rightarrow \boxed{1}b)$ is given by:

$$\tau_0(\varphi) = \Box^*(\mathbf{start} \Rightarrow t_1) \wedge \tau_1(\Box^*(t_1 \Rightarrow \boxed{1}(a \Rightarrow b) \Rightarrow (\boxed{1}a \Rightarrow \boxed{1}b)))$$

where

$$\tau_1(\Box^*(t_1 \Rightarrow \boxed{1}(a \Rightarrow b) \Rightarrow (\boxed{1}a \Rightarrow \boxed{1}b))) =$$
$$= \Box^*(\mathbf{true} \Rightarrow \neg t_1 \vee t_2 \vee t_3 \vee t_4) \wedge \Box^*(t_2 \Rightarrow \neg\boxed{1}\neg t_5) \wedge \Box^*(t_3 \Rightarrow \neg\boxed{1}a) \wedge$$
$$\Box^*(t_4 \Rightarrow \boxed{1}b) \wedge \Box^*(\mathbf{true} \Rightarrow \neg t_5 \vee a) \wedge \Box^*(\mathbf{true} \Rightarrow \neg t_5 \vee \neg b)$$

The translation procedure results in 7 clauses: one initial, three literal, and three modal clauses. Note also that the new propositional symbols t_i, $(1 \leq i \leq 5)$, were introduced during renaming of complex formula: either a disjunct which is not a literal or a complex formula inside the scope of a modal operator.

The translation into normal form is satisfiability preserving, that is, we can prove the following:

Theorem 1. *Let φ be a formula in $\mathsf{K}_{(n)}$ and M a model. $M \models \varphi$ if, and only if, there is a model M' such that $M' \models \tau_0(\varphi)$.*

The proof is similar to that of [4] and can be found in [7].

4 Anti-prenexing

Anti-prenexing has been used in first-order theorem proving as a step applied before skolemization, in order to achieve a better set of clauses [2]. Similarly to

first-order, anti-prenexing in the modal context means that all modal operators are moved inwards the formula as far as possible, whilst preserving satisfiability. In the weakest normal logic, $\mathsf{K}_{(n)}$, we can distribute the necessity operator, $\Box$, over conjunctive formulae; and the possibility operator, $\neg\Box\neg$, over disjunctive formulae. The definition of the anti-prenex normal form is given below.

Definition 7. *A **modal term** is a literal or a formula of the form $M_1 \dots M_k l$, where l is a literal and M_i, $1 \leq i \leq k$, is $\boxed{j}$ or $\neg\boxed{j}$ for some $j \in \mathcal{A}$.*

Note that a literal l, which is not preceded by any modal operator, is also a modal term.

Definition 8. *Let φ and ψ be formula in $\mathsf{WFF}_{\mathsf{K}_{(n)}}$. A formula χ is in Anti-Prenex Normal Form (APNF) if, and only if,*

1. *χ is a modal term; or*
2. *χ is of the form $\neg\varphi$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ or $(\varphi \Rightarrow \psi)$, and φ and ψ are in APNF;*
3. *χ is of the form $\boxed{i}\varphi$, φ is not of the form $\boxed{j}\psi$ or of the form $\neg\boxed{j}\psi$, φ is disjunctive, and φ is in APNF;*
4. *χ is of the form $\neg\boxed{i}\varphi$, φ is not of the form $\boxed{j}\psi$ or of the form $\neg\boxed{j}\psi$, φ is conjunctive, and φ is in APNF; or*
5. *χ is of the form $\boxed{i}\varphi$ or $\boxed{i}\varphi$, φ is of the form $\boxed{j}\psi$ or of the form $\neg\boxed{j}\psi$, and φ is in APNF; or*

The following lemma shows that any formula can be transformed into APNF.

Lemma 1. *Let φ be a formula in $\mathsf{K}_{(n)}$ and M a model in $\mathsf{K}_{(n)}$. Then there is a formula φ' in APNF, such that $M \models \varphi$ if, and only if, $M \models \varphi'$.*

Proof. The following schemata are theorems of $\mathsf{K}_{(n)}$:

1. $\boxed{i}(\varphi \wedge \psi) \Leftrightarrow (\boxed{i}\varphi \wedge \boxed{i}\psi)$
2. $\boxed{i}\neg(\varphi \Rightarrow \psi) \Leftrightarrow (\boxed{i}\varphi \wedge \boxed{i}\neg\psi)$
3. $\boxed{i}\neg(\varphi \vee \psi) \Leftrightarrow (\boxed{i}\neg\varphi \wedge \boxed{i}\neg\psi)$
4. $\neg\boxed{i}\neg(\varphi \Rightarrow \psi) \Leftrightarrow (\boxed{i}\varphi \Rightarrow \neg\boxed{i}\neg\psi)$
5. $\neg\boxed{i}\neg(\varphi \vee \psi) \Leftrightarrow (\neg\boxed{i}\neg\varphi \vee \neg\boxed{i}\neg\psi)$
6. $\neg\boxed{i}(\varphi \wedge \psi) \Leftrightarrow (\neg\boxed{i}\varphi \vee \neg\boxed{i}\psi)$ □

The transformation into SNF_K consists of two steps: transforming the formulae into anti-prenex, as defined below, and then applying the transformation function given in Subsection 3.1. Firstly, based on the schemata presented in Lemma 1, we define a function $\alpha(\varphi)$, where φ is a formula, which produces the anti-prenex normal form of φ. The base case occurs when the formula A is already in APNF, that is, A is a modal term. In this case, $\alpha(A) = A$. If the main operator is modal, we only apply the transformation function to formula which satisfies the equivalences in Lemma 1, that is, in the following cases:

$$\begin{aligned}
\alpha(\Box_i(A \wedge B)) &= \alpha(\Box_i A \wedge \Box_i B)\\
\alpha(\Box_i \neg(A \Rightarrow B)) &= \alpha(\Box_i A \wedge \Box_i \neg B)\\
\alpha(\Box_i \neg(A \vee B)) &= \alpha(\Box_i \neg A \wedge \Box_i \neg B)\\
\alpha(\neg\Box_i \neg(A \Rightarrow B)) &= \alpha(\Box_i A \Rightarrow \neg\Box_i \neg B)\\
\alpha(\neg\Box_i \neg(A \vee B)) &= \alpha(\neg\Box_i \neg A \vee \neg\Box_i \neg B)\\
\alpha(\neg\Box_i (A \wedge B)) &= \alpha(\neg\Box_i A \vee \neg\Box_i B)
\end{aligned}$$

If we have two consecutive modal operators, the function is applied recursively, where A is of the form $\Box_j B$ or $\neg\Box_j B$, for any $j \in \mathcal{A}$:

$$\alpha(\Box_i A) = \alpha(\Box_i \alpha(A)) \qquad \alpha(\neg\Box_i A) = \alpha(\neg\Box_i \alpha(A))$$

If the main operator is a modal operator, but the formula inside its scope is not one of the above, we apply the anti-prenexing function to this formula, that is:

$$\alpha(\Box_i A) = \Box_i \alpha(A) \qquad \alpha(\neg\Box_i A) = \neg\Box_i \alpha(A)$$

When the main operator is classical, the transformation function is also applied recursively. Note that when the polarity of a subformula is negative, we rewrite the formula in order to make this explicit.

$$\begin{aligned}
\alpha(\neg\neg A) &= \alpha(A)\\
\alpha(A \Rightarrow B) &= \alpha(\neg A) \vee \alpha(B)\\
\alpha(A \wedge B) &= \alpha(A) \wedge \alpha(B)\\
\alpha(A \vee B) &= \alpha(A) \vee \alpha(B)
\end{aligned}
\qquad
\begin{aligned}
\alpha(\neg(A \Rightarrow B)) &= (\alpha(A) \wedge \alpha(\neg B))\\
\alpha(\neg(A \wedge B)) &= (\alpha(\neg A) \vee \alpha(\neg B))\\
\alpha(\neg(A \vee B)) &= (\alpha(\neg A) \wedge \alpha(\neg B))
\end{aligned}$$

The proof that this transformation is correct and satisfiability preserving can be obtained as in [7] for translation into SNF_K and will not be presented here.

As an example, the APNF of $\Box_i(a \wedge \Box_i(b \wedge \Box_i c))$ is $\Box_i a \wedge \Box_i\Box_i b \wedge \Box_i\Box_i\Box_i c$, whose transformation into normal form is:

$$\Box^*(\mathbf{start} \Rightarrow x) \wedge \Box^*(x \Rightarrow \Box_i a) \wedge \Box^*(x \Rightarrow \Box_i y) \wedge \Box^*(y \Rightarrow \Box_i b) \wedge \Box^*(x \Rightarrow \Box_i z) \wedge \Box^*(z \Rightarrow \Box_i w) \wedge \Box^*(w \Rightarrow \Box_i c).$$

5 Simplification Rules

The anti-prenexing pre-processing of a formula may result in fewer or shorter clauses. For instance, consider the formula $\Box_i(a \wedge b)$. Transformation into SNF_K results in four clauses ($\Box^*(\mathbf{start} \Rightarrow x)$, $\Box^*(x \Rightarrow \Box_i y)$, $\Box^*(\mathbf{true} \Rightarrow \neg y \vee a)$, and $\Box^*(\mathbf{true} \Rightarrow \neg y \vee b)$), whilst transformation into the normal form preceded by anti-prenexing results in three clauses ($\Box^*(\mathbf{start} \Rightarrow x)$, $\Box^*(x \Rightarrow \Box_i a)$, and $\Box^*(x \Rightarrow \Box_i b)$). Also the size of the second transformation is smaller than the size of the first. However, this is not always the case. Depending on the nesting of modal operators in the original formula, the number of clauses as well as the size of the resulting formula, generated after anti-prenexing and translation into SNF_K can be significantly larger than by applying the transformation into SNF_K

alone. The reason is that the modal operator that had appeared only once in the formula now has several copies distributed over subformulae.

However, when applied together with simplification, anti-prenexing may reduce the size of the formula, by collapsing of nested modal operators in the original formula. Obviously, this depends on the particular normal modal logic we are considering. We discuss in this section the simplification rules that could be applied together with anti-prenexing, before transformation into SNF_K, in the case of $\mathsf{KTD45}_{(n)}$ and $\mathsf{KD45}_{(n)}$. The first normal modal logic, also known as $\mathsf{S5}_{(n)}$ – the logic of knowledge for multiple agents – is axiomatisable by the schemata **K**, **T**, **D**, **4**, and **5** and the rules of inference: *modus ponens* (from $\vdash \varphi$ and $\vdash (\varphi \Rightarrow \psi)$ infer $\vdash \psi$) and *modal necessitation* (from $\vdash \varphi$ infer $\vdash \Box_i \varphi$). The logics $\mathsf{KD45}_{(n)}$, known as the logic of belief for multiple agents, is axiomatisable by the schemata **K**, **D**, **4**, and **5** and the rules of inference *modus ponens* and *modal necessitation*. As the schemata $\Box_i \Box_i \varphi \Leftrightarrow \Box_i \varphi$, $\Box_i \neg \Box_i \varphi \Leftrightarrow \neg \Box_i \varphi$, $\neg \Box_i \neg \Box_i \varphi \Leftrightarrow \Box_i \varphi$, $\neg \Box_i \Box_i \varphi \Leftrightarrow \neg \Box_i \varphi$ are valid in $\mathsf{KTD45}_{(n)}$ and in $\mathsf{KD45}_{(n)}$, we extend the anti-prenexing function in the obvious way:

$$\alpha(\Box_i \Box_i \varphi) = \alpha(\Box_i \varphi) \quad \alpha(\neg \Box_i \neg \Box_i \varphi) = \alpha(\Box_i \varphi)$$
$$\alpha(\Box_i \neg \Box_i \varphi) = \alpha(\neg \Box_i \varphi) \quad \alpha(\neg \Box_i \Box_i \varphi) = \alpha(\neg \Box_i \varphi)$$

We note that we only apply simplification when the modal operators have the same index. Other simplification rules could also be introduced, but we chose not to do this and, instead, preserving some of the structure of the formula. Using these simplification rules, the anti-prenex normal form of the previous example, i.e. $\Box_i(a \wedge \Box_i(b \wedge \Box_i c))$, is $\Box_i a \wedge \Box_i b \wedge \Box_i c$, which has the same size as the original formula. Table 1 show the three transformations for comparison.

Table 1. Translation (from left to right) without Anti-Prenexing, after Anti-Prenexing, and after Anti-Prenexing and Simplification

$\Box_i(a \wedge \Box_i(b \wedge \Box_i c))$

SNF_K	AP + SNF_K	AP + SIMP + SNF_K
$\Box_i(a \wedge \Box_i(b \wedge \Box_i c))$	$\Box_i a \wedge \Box_i \Box_i b \wedge \Box_i \Box_i \Box_i c$	$\Box_i a \wedge \Box_i b \wedge \Box_i c$
1. **start** $\Rightarrow x$	1. **start** $\Rightarrow x$	1. **start** $\Rightarrow x$
2. $x \Rightarrow \Box_i y$	2. $x \Rightarrow \Box_i a$	2. $x \Rightarrow \Box_i a$
3. **true** $\Rightarrow \neg y \vee a$	3. $x \Rightarrow \Box_i y$	3. $x \Rightarrow \Box_i b$
4. $y \Rightarrow \Box_i z$	4. $y \Rightarrow \Box_i b$	4. $y \Rightarrow \Box_i c$
5. **true** $\Rightarrow \neg z \vee b$	5. $x \Rightarrow \Box_i z$	
6. $z \Rightarrow \Box_i c$	6. $z \Rightarrow \Box_i w$	
	7. $w \Rightarrow \Box_i c$	

Note that no simplification rule could be applied to the original formula. By moving the modal operators inwards the formula, simplification could be applied, resulting in fewer and shorter clauses.

6 Prenexing

Similarly to first-order logic, prenexing in the modal context means to pull modal operators as far as possible outwards the formula. It is well-known that formulae in $\mathsf{KTD45}_{(1)}$ can be transformed into a formula without nesting of modal operators (see [6], for instance). As we are interested in a more general form of prenexing than that given for $\mathsf{KTD45}_{(1)}$, we say that a formula is in prenex normal form if it corresponds to the inverse of the transformation into anti-prenexing. Thus, the transformation is justified by the same equivalences appearing in Lemma 1. Our definition of the prenexing function is similar to that given in Section 4 and will not be presented here. Instead, in this section, we give the motivation for using both techniques together with simplification for $\mathsf{KTD45}_{(n)}$ and $\mathsf{KD45}_{(n)}$.

When a formula φ is transformed into APNF, the nesting of modal operators is made explicit and can be easily simplified. On the other hand, several copies of a modal operator may now appear in the formula. By performing the transformation into prenex normal form, after anti-prenexing and simplification, we try to remove such copies and make the formula shorter. We note that the order in which the transformations are applied is important. Consider, for instance, the formula given in previous examples, that is, $\boxed{i}(a \wedge \boxed{i}(b \wedge \neg\boxed{i}\neg c))$. This formula is in prenex normal form, as we cannot apply any of the equivalences given in Lemma 1 to move the modal operators outwards the formula. However, if we apply anti-prenexing with simplification, we obtain, as seen before, the formula $\boxed{i}a \wedge \boxed{i}b \wedge \boxed{i}c$. The result of applying the prenex function is $\boxed{i}(a \wedge b \wedge c)$, which is shorter than both the original formula and the one obtained after anti-prenexing with simplification. In this case, however, the transformation into SNF_K results in one more clause. Table 2 gives an example where the number of clauses is smaller than that produced by the other methods without prenexing.

Table 2. Example using Anti-Prenexing, Prenexing, and Simplification

Formula	$\neg\boxed{i}\neg(a \vee \neg\boxed{i}\neg(b \vee \neg\boxed{i}\neg c))$
	$\downarrow$
Anti-Prenexing	$\neg\boxed{i}\neg a \vee \neg\boxed{i}\neg\neg\boxed{i}\neg b \vee \neg\boxed{i}\neg\neg\boxed{i}\neg\neg\boxed{i}\neg c$
	$\downarrow$
Anti-Prenexing + Simplification	$\neg\boxed{i}\neg a \vee \neg\boxed{i}\neg b \vee \neg\boxed{i}\neg c$
	$\downarrow$
Prenex	$\neg\boxed{i}\neg(a \vee b \vee c)$
	$\downarrow$
SNF_K	1. $\Box^*(\mathbf{start} \Rightarrow x)$ 2. $\Box^*(\mathbf{true} \Rightarrow \neg\boxed{i}\neg y)$ 3. $\Box^*(\mathbf{true} \Rightarrow \neg y \vee a \vee b \vee c)$

As a final example, consider the formula $\neg\boxed{i}\neg(\boxed{i}a \wedge \boxed{i}b)$, which is already in APNF, as the modal operator $\neg\boxed{i}\neg$ cannot be distributed over conjunctions. Also, no simplification rule can be applied to the formula. After applying

prenexing, however, we obtain $\neg\Box_i\neg\Box_i(a \wedge b)$, which simplifies to $\Box_i(a \wedge b)$. We do not apply anti-prenexing again, which would result in a shorter translation into SNF_K, as discussed at the beginning of Section 5. Nevertheless, the resulting formula is half the size of the original one and the set of clauses is also smaller.

7 Experimental Results

The examples given here have only the purpose of illustrating the techniques. We cannot prove, in general, that by applying those techniques we will obtain a better set of clauses. In order to have a measure of how anti-prenexing and prenexing behave in comparison to translation to SNF_K alone, we have performed tests using formulae from [5].

The program takes a modal formula and returns its size and number of literals. It also returns the size and number of literals after transforming the formula into SNF_K alone, into SNF_K preceded by anti-prenexing, and into SNF_K preceded by anti-prenexing with simplification. We are currently working on the implementation of prenexing.

Table 3 shows the output from running the program over formulae from the benchmark `s4_45_p.txt`, which contains problems in $\mathsf{KT4}_{(1)}$ which are provable

Table 3. Results for Transformations of Formulae in the Logic Workbench

					SNF After AP				SNF After AP and Simp			
	Initial		SNF Only		After AP		After SNF		After AP		After SNF	
Id	Size	Lits	Size	Lits	Size	Lits	Size	Lits	Size	Lits	Size	Lits
1	119	3	207	27	99	3	189	27	77	3	141	19
2	313	3	531	50	257	3	479	50	213	3	383	44
3	581	3	979	87	473	3	877	87	407	3	733	77
4	923	3	1551	122	747	3	1383	150	659	3	1191	130
5	1339	3	2247	183	1079	3	1997	184	969	3	1757	186
6	1829	3	3067	256	1469	3	2719	296	1337	3	2431	264
7	2393	3	4011	387	1917	3	3549	387	1763	3	3213	349
8	3031	3	5079	489	2423	3	4487	489	2247	3	4103	438
9	3743	3	6271	544	2987	3	5533	604	2789	3	5101	555
10	4529	3	7587	664	3609	3	6687	665	3389	3	6207	667
11	5389	3	9027	797	4289	3	7949	869	4047	3	7421	809
12	6323	3	10591	1020	5027	3	9319	1020	4763	3	8743	954
13	7331	3	12279	1182	5823	3	10797	1182	5537	3	10173	1099
14	8413	3	14091	1265	6677	3	12383	1357	6369	3	11711	1280
15	9569	3	16027	1445	7589	3	14077	1446	7259	3	13357	1448
16	10799	3	18087	1638	8559	3	15879	1742	8207	3	15111	1654
17	12103	3	20271	1953	9587	3	17789	1953	9213	3	16973	1859
18	13481	3	22579	2175	10673	3	19807	2175	10277	3	18943	2060
19	14933	3	25011	2286	11817	3	21933	2410	11399	3	21021	2305
20	16459	3	27567	2526	13019	3	24167	2527	12579	3	23207	2529
21	18059	3	30247	2779	14279	3	26509	2915	13817	3	25501	2799

in both $\mathsf{KT4}_{(1)}$ and $\mathsf{KTD45}_{(1)}$. For each problem, identified in the first column, we present the total size of the formula (*Size*) and the number of different literals (*Lits*). Columns 2 and 3 refer to the original formula. Columns 4 and 5 show the result for transformation into SNF_K alone. Columns 6 to 8 are related to anti-prenexing without simplification, where the first two columns are the size of the problem after transforming into anti-prenexing, and the other two columns are the result of transforming into SNF_K. The last four columns contain the result for anti-prenexing together with simplification: their contents are similar to those for anti-prenexing without simplification. The table shows that transformation into SNF_K preceded by anti-prenexing with simplification performs better than the other two methods. On average, the size of a formula decreases 21% and 24% after anti-prenexing and anti-prenexing with simplification. Also on average, the size of a formula increases 68%, 47%, and 39% after SNF_K only, SNF_K after anti-prenexing, and SNF_K after anti-prenexing with simplification, respectively. Other experimental results can be found in [7].

8 Conclusions

In this paper we have presented an algorithm for transforming any normal modal formula into a normal form. This can be done because the transformation is based on valid schemata of the weakest of the normal modal logics, namely $\mathsf{K}_{(n)}$. Also, we investigate how the use of anti-prenexing and prenexing can help in obtaining a better transformation. Combined with simplification rules, these methods seem to produce smaller clause sets for problems from some normal logics.

There is no way of defining which is the best normal form, in general. Here we focused on the size of the transformed problem as a measure for determining whether the transformation is good. However, we also intend to investigate other parameters, as for instance the number of clauses and their sizes, as well as how efficiently a real theorem-prover responds to those different transformations. As anti-prenexing with simplification moves the modal operators inwards the formula, those operators are usually applied to simpler formulae, indicating that we could have less resolution steps applied to a clause set.

Simplification is an important step in the translation algorithm, but, as discussed before, it cannot be applied to all normal modal logics. We have shown the simplification rules for $\mathsf{KTD45}_{(n)}$ and $\mathsf{KD45}_{(n)}$. The equivalences $\Box_i \neg \Box_i \varphi \Leftrightarrow \neg \Box_i \varphi$ and $\neg \Box_i \neg \Box_i \varphi \Leftrightarrow \Box_i \varphi$ are also valid in $\mathsf{K45}_{(n)}$ and $\mathsf{KT4}_{(n)}$ (also known as $\mathsf{S4}_{(n)}$), so the formula resulting from anti-prenexing can be simplified, but not at the same extent as those of $\mathsf{KTD45}_{(n)}$ and $\mathsf{KD45}_{(n)}$. The other modal logics do not admit simplification rules for collapsing of nested operators. In these cases, as our experimental results show, applying anti-prenexing does not seem to be worthwhile. We are currently working on the complexity of the transformation in order to determine precisely when anti-prenexing and prenexing of a formula would result in a better set of clauses. That is, the techniques shown here could be used *selectively* in a similar way as renaming is used [9].

We are currently working on the implementation of the prenexing algorithm. We believe that anti-prenexing together with prenexing and simplification will give us the best result for formulae in $\mathsf{KTD45}_{(n)}$ and $\mathsf{KD45}_{(n)}$. We hope that the same combination will give us better results for the other logics.

Current work also involves the development of the resolution-based methods for each logic. Our intention is that a uniform approach to deal with those logics – from the designing of the normal forms up to the whole proof-method – will facilitate the task of validity checking for combinations of those logics.

Acknowledgements

The first author was supported by CNPq grants CT-INFO 506598/04-7 and Universal 47171/2004-0.

References

1. C. Dixon and M. Fisher. Resolution-Based Proof for Multi-Modal Temporal Logics of Knowledge. In S. Goodwin and A. Trudel, editors, *Proceedings of the 7th International Workshop on Temporal Representation and reasoning (TIME'00)*, pages 69–78, Cape Breton, Canada, July 2000. IEEE Computer Society Press.
2. U. Egly. On the value of antiprenexing. In F. Pfenning, editor, *Proceedings of the 5th International Conference on Logic Programming and Automated Reasoning*, volume 822 of *LNAI*, pages 69–83, Berlin, July 1994. Springer.
3. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
4. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1), Jan. 2001.
5. G. Jaeger, P. Balsiger, A. Heuerding, S. Schwendimann, M. Bianchi, K. Guggisberg, G. Janssen, W. Heinle, F. Achermann, A. D. Boroumand, P. Brambilla, I. Bucher, and H. Zimmermann. LWB–The Logics Workbench 1.1. http://www.lwb.unibe.ch/. University of Berne, Switzerland.
6. J. J. C. Meyer and W. van der Hoek. *Epistemic Logic for Computer Science and Artificial Intelligence*, volume 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.
7. C. Nalon and C. Dixon. Anti-prenexing and prenexing for modal logics (extended version). Technical Report ULCS-06-003, University of Liverpool, April 2006. Available at http://www.csc.liv.ac.uk/research/techreports/tr2006/ulcs-06-003.pdf.
8. C. Nalon and C. Dixon. Normal modal resolution. Submitted, June 2006.
9. A. Nonnengart and C. Weidenbach. Computing small clause normal forms. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B.V., 2001.
10. D. A. Plaisted and S. A. Greenbaum. A Structure-Preserving Clause Form Translation. *Journal of Logic and Computation*, 2:293–304, 1986.
11. A. S. Rao and M. P. Georgeff. Decision procedures for BDI logics. *Journal of Logic and Computation*, 8(3):293–342, 1998.

A Bottom-Up Method for the Deterministic Horn Fragment of the Description Logic $\mathcal{ALC}$

Linh Anh Nguyen

Institute of Informatics, University of Warsaw
ul. Banacha 2, 02-097 Warsaw, Poland
nguyen@mimuw.edu.pl

Abstract. We study the deterministic Horn fragment of $\mathcal{ALC}$, which restricts the general Horn fragment of $\mathcal{ALC}$ only in that, the constructor $\forall R.C$ is allowed in bodies of program clauses and queries only in the form $\forall\exists R.C$, which is defined as $\forall R.C \sqcap \exists R.C$. We present an algorithm that for a deterministic positive logic program P given as a TBox constructs a finite least pseudo-model $\mathcal{I}$ of P such that for every deterministic positive concept C, $P \models C$ iff $\mathcal{I}$ validates C (and more strongly, iff $\mathcal{I}, \tau \models C$, where τ is the distinguished object of $\mathcal{I}$ and the satisfaction means τ is an instance of C w.r.t. $\mathcal{I}$). Pseudo-interpretations are very similar to (traditional) interpretations, except that they have two interpretation functions for roles, one to deal with the constructor $\exists R.C$ and the other to deal with $\forall R.C$. They are ordered by comparing the sets of validated positive concepts. Our algorithm runs in time $2^{O(n)}$ and returns a pseudo-interpretation of size $2^{O(n)}$. Our method is extendable for instance checking w.r.t. knowledge bases containing also an ABox in more expressive description logics.

1 Introduction

Description logics (DLs) are logics that represent the domain of interest in terms of concepts, objects, and roles. They are useful for modeling and reasoning about structured knowledge. In the recent years, the combination of description logics and Horn logic has been studied by a considerable number of researchers (see, e.g., [2, 8, 6, 5, 7, 3]).

In [8], Levy and Rousset developed the CARIN family of representation languages that combines the expressive power of Horn rules and DLs. CARIN knowledge bases contain a DL terminology and a set of Horn rules defined on the top of the terminology. CARIN combines the two formalisms by allowing the concepts and roles, defined in the terminology, to appear as predicates in the antecedents of the Horn rules. Some works related with this approach are, e.g., [4, 2, 5].

Another approach is to study Horn fragments of DLs [6, 7, 3]. In [6], Grosof et al. introduced the description Horn logic (DHL), which is a restricted fragment of DL, and studied it through a transformation to classical Horn logic. A DHL program consists of Horn clauses defining (relations between) concepts, (relations between) roles, and instances of concepts and roles. Inverse roles and transitive

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 346–358, 2006.

roles are allowed in DHL programs. In order to make the transformation possible, the constructor $\forall R.C$ is disallowed in bodies and the constructor $\exists R.C$ is disallowed in heads of DHL program clauses. In [7], Hustadt et al. introduced a more relaxed Horn version for the expressive description logic $\mathcal{SHIQ}$, which is called Horn-$\mathcal{SHIQ}$. In comparison with DHL, Horn-$\mathcal{SHIQ}$ also disallows the constructor $\forall R.C$ in bodies of program clauses and queries, but it allows the constructor $\exists R.C$ to appear in heads of program clauses. Using a transformation to Datalog and treating the TBox of a knowledge base as the intensional part and the ABox as the extensional part, Hustadt et al. [7] proved that the data complexity of Horn-$\mathcal{SHIQ}$ is complete in PTIME. In [3], Calvanese et al. also studied data complexity of query answering in DLs. To obtain low data complexity they adopted strong restrictions for the form of Horn clauses.

In this paper, we study the deterministic Horn fragment of the description logic $\mathcal{ALC}$. It restricts the general Horn fragment only in that, the constructor $\forall R.C$ is allowed in bodies of program clauses and queries only in the form $\forall\exists R.C$, which is defined as $\forall R.C \sqcap \exists R.C$. That is, in the deterministic Horn fragment of $\mathcal{ALC}$, the constructors $\exists R.C$ and $\forall\exists R.C$ are allowed in bodies of program clauses and queries, and the constructors $\exists R.C$ and $\forall R.C$ are allowed in heads of programs clauses. In the current version, we consider only logic programs which are a TBox and the problem of checking whether $P \models C$ for a program P and a concept C. With the current setting, our programs do not contain certain features allowed in [6, 7]. On the other hand, when restricting only to the problem of checking $P \models C$ for a program P being a TBox in $\mathcal{ALC}$, our deterministic Horn fragment is more general than the Horn fragments studied by Grosof et al. [6] and Hustadt et al. [7]. Recall that in both DHL [6] and Horn-$\mathcal{SHIQ}$ [7], the constructor $\forall R.C$ is disallowed in bodies of program clauses and queries.

The deterministic Horn fragment without ABox of $\mathcal{ALC}$ maybe not useful enough for practical applications. However, it shows how we can extend the results of other authors for more general Horn fragments of DLs. For example, we can extend Horn-$\mathcal{SHIQ}$ to dHorn-$\mathcal{SHIQ}$ by allowing the constructor $\forall\exists R.C$ to appear in bodies of program clauses and queries. Then CARIN-like systems that use dHorn-$\mathcal{SHIQ}$ for the terminology layer are very expressive systems for which we believe that the data complexity is complete in PTIME.

As a computational method for DLs one can apply a transformation to classical logic and use the existing techniques of resolution, logic programming, or deductive databases. However, the mostly studied computational method for DLs is the tableau method (see the overview by Baader and Sattler [1]). The work [8] by Levy and Rousset on CARIN also uses the tableau method. The tableau method is essentially different from the bottom-up method (used in deductive databases) because of the "or" splitting rule.

The motivation of this paper is to develop a bottom-up method for query answering for Horn fragments of DLs, while staying with the syntax of DLs and adopting as less as possible restrictions on the form of Horn programs and queries. In general, the bottom-up method for checking $P \models_L C$ for a program P and a query C in a restricted fragment L' of a logic L is to build a finite

L-model M (or a similar structure) for a given program P of L' such that for every query C of L', $P \models_L C$ iff $M \models C$. The method is especially useful when P is a knowledge base that rarely changes, while C is a query and varies. The problem of constructing such a model M for P is not trivial at all (see, e.g., [9]).

A deterministic positive logic program is a finite set of non-negative concepts of the deterministic Horn fragment. Queries to such a program are deterministic positive concepts, which are positive concepts containing the constructor $\forall R.C$ only in the form $\forall\exists R.C$. To deal with such logic programs and queries, we use pseudo-interpretations, which are very similar to (traditional) interpretations, except that they have two interpretation functions for roles, one to deal with the constructor $\exists R.C$ and the other to deal with $\forall R.C$. They are ordered by comparing the sets of validated positive concepts.

In this paper, we present an algorithm that for a deterministic positive logic program P given as a TBox constructs a finite least pseudo-model $\mathcal{I}$ of P such that for every deterministic positive concept C, $P \models C$ iff $\mathcal{I}$ validates C (and more strongly, iff $\mathcal{I}, \tau \models C$, where τ is the distinguished object of $\mathcal{I}$ and the satisfaction means τ is an instance of C w.r.t. $\mathcal{I}$). Our algorithm runs in time $2^{O(n)}$ and returns a pseudo-interpretation of size $2^{O(n)}$.

This work is related to our previous work [9]. In [9] we gave an algorithm that for a given positive modal logic program P, treated as local assumptions, and a serial monomodal logic $L \in \{KD,\ T,\ KDB,\ B,\ KD4,\ S4,\ KD5,\ KD45,\ S5\}$ constructs a least L-model of P, which is finite if $L \notin \{KDB, B\}$. The monomodal logic K is non-serial and there are positive modal logic programs that do not have any least K-model [9]. $\mathcal{ALC}$ is a syntactic variant of the multimodal version of K and it has a similar problem. From the point of view of [9], the challenge of dealing with $\mathcal{ALC}$ is not the problem of multi-modalities and global assumptions, but is the problem of non-seriality. To overcome this problem for $\mathcal{ALC}$, we restrict to the deterministic Horn fragment and use pseudo-interpretations.

The rest of this paper is structured as follows. In Section 2, we recall the notation and semantics of $\mathcal{ALC}$, define the deterministic Horn fragment of $\mathcal{ALC}$, pseudo-interpretations, the satisfaction relation and an ordering for pseudo-interpretations. In Section 3, we present our algorithm and prove its correctness. In Section 4, we prove some characterizations of least pseudo-models. Section 5 contains concluding remarks. Due to the lack of space, some proofs are presented only in the long version [10] of this paper.

2 Preliminaries

2.1 Notation and Semantics of $\mathcal{ALC}$

Let A denote an atomic concept, C and D arbitrary concepts, and R a role name. Concepts in $\mathcal{ALC}$ are formed with the following syntax:

$$C, D ::= \top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid C \sqsubseteq D \mid C \doteq D \mid \forall R.C \mid \exists R.C$$

An *interpretation* $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$, the *interpretation function* of $\mathcal{I}$, that maps every atomic

concept to a subset of $\Delta^{\mathcal{I}}$ and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to interpret every concept as follows:

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}},\ \bot^{\mathcal{I}} = \emptyset,\ (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}},\ (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}},\ (C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}},$$

$$(C \sqsubseteq D)^{\mathcal{I}} = (\neg C \sqcup D)^{\mathcal{I}}, \quad (C \doteq D)^{\mathcal{I}} = ((C \sqsubseteq D) \sqcap (D \sqsubseteq C))^{\mathcal{I}},$$

$$(\forall R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y.(x, y) \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}, \quad (\exists R.C)^{\mathcal{I}} = (\neg \forall R.\neg C)^{\mathcal{I}}.$$

An interpretation $\mathcal{I}$ *satisfies* C if $C^{\mathcal{I}} \neq \emptyset$, and *validates* C if $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$.

A TBox Γ (of global axioms) is a finite set of concepts.[1] An interpretation $\mathcal{I}$ is a *model* of Γ if $\mathcal{I}$ validates all concepts in Γ. We also use X, Y to denote finite sets of concepts. We say that $\mathcal{I}$ *satisfies* X if there exists $a \in \Delta^{\mathcal{I}}$ such that $a \in C^{\mathcal{I}}$ for all $C \in X$. (Note that satisfaction is defined "locally", and $\mathcal{I}$ satisfies X does not mean that $\mathcal{I}$ is a model of X.)

We say that Γ *entails* C, written $\Gamma \models C$, if every model of Γ validates C.

We say that C is satisfiable w.r.t. Γ if there exists a model of Γ that satisfies C. Similarly, X is satisfiable w.r.t. (a TBox of global axioms) Γ if there exists a model of Γ that satisfies X. Observe that $\Gamma \models C$ iff $\neg C$ is unsatisfiable w.r.t. Γ.

2.2 The Deterministic Horn Fragment of $\mathcal{ALC}$

We extend the primitive language with the concept constructor $\forall\exists$, which creates a concept $\forall\exists R.C$ from a role name R and a concept C. The concept $\forall\exists R.C$ is interpreted as $(\forall R.C) \sqcap (\exists R.C)$.

A *positive concept* is a concept (in the extended language) without the constructors $\bot$, $\neg$, $\sqsubseteq$, $\doteq$. A *(modal-)deterministic positive concept* is a positive concept which does not contain the constructor $\forall$ (but may contain $\exists$ and $\forall\exists$).

A *deterministic program clause* (in $\mathcal{ALC}$) is a concept of one of the forms:

- an atomic concept or $\top$;
- $C \sqsubseteq D$, where C is a deterministic positive concept and D is a deterministic program clause;
- $C \sqcap D$, where C and D are deterministic program clause;
- $\forall R.C$ or $\exists R.C$, where C is a deterministic program clause.

A *deterministic positive logic program* (in $\mathcal{ALC}$) is a finite set of deterministic program clauses.

Example 1. Given the following deterministic positive logic program P :

$$\forall\exists child.(doctor \sqcup lawyer) \sqsubseteq happy\_parent$$
$$\forall child.doctor$$
$$\exists child.\top$$

we can ask, for example, whether $P \models happy\_parent$ and $P \models \forall\exists child.doctor$. The deterministic positive concepts $happy\_parent$ and $\forall\exists child.doctor$ are queries.

[1] Traditionally, a TBox is defined to be a finite set of terminological axioms of the form $C \doteq D$, where C and D are concepts. The two definitions are equivalent.

2.3 Pseudo-interpretations

A *pseudo-interpretation* is a tuple of the form $\langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$, where Δ is the domain, τ is a distinguished element of Δ (like the actual world in a Kripke model), $\mathcal{C}$ is a function that maps every atomic concept to a subset of Δ, and $\mathcal{E}$ and $\mathcal{U}$ are functions that map every role name to a subset of $\Delta \times \Delta$ with the property that $\mathcal{E}(R) \subseteq \mathcal{U}(R)$ for every role name R. The function $\mathcal{E}$ is used to deal with the (**e**xistential) constructor $\exists$, while $\mathcal{U}$ is used to deal with the (**u**niversal) constructor $\forall$.

A pseudo-interpretation $\langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ can be treated as an interpretation if $\mathcal{E} = \mathcal{U}$. Conversely, every interpretation can be treated as a pseudo-interpretation (with τ being some element of the domain).

Given a pseudo-interpretation $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$, an element $x \in \Delta$, and a concept C which is either a positive concept or a deterministic program clause, define $\mathcal{I}, x \models C$ as follows:

$$
\begin{array}{lll}
\mathcal{I}, x \models A & \text{iff} & x \in \mathcal{C}(A) \\
\mathcal{I}, x \models C \sqcap D & \text{iff} & \mathcal{I}, x \models C \text{ and } \mathcal{I}, x \models D \\
\mathcal{I}, x \models C \sqcup D & \text{iff} & \mathcal{I}, x \models C \text{ or } \mathcal{I}, x \models D \\
\mathcal{I}, x \models C \sqsubseteq D & \text{iff} & \mathcal{I}, x \not\models C \text{ or } \mathcal{I}, x \models D \\
\mathcal{I}, x \models \exists R.C & \text{iff} & \exists y.(\mathcal{E}(R)(x, y) \wedge \mathcal{I}, y \models C) \\
\mathcal{I}, x \models \forall R.C & \text{iff} & \forall y.(\mathcal{U}(R)(x, y) \rightarrow \mathcal{I}, y \models C) \\
\mathcal{I}, x \models \forall\exists R.C & \text{iff} & \mathcal{I}, x \models (\forall R.C) \sqcap (\exists R.C)
\end{array}
$$

We say that $\mathcal{I}$ *validates* C if $\mathcal{I}, x \models C$ for every $x \in \Delta$. For Γ being a set of positive concepts or deterministic program clauses, we write $\mathcal{I}, x \models \Gamma$ to denote that $\mathcal{I}, x \models C$ for every $C \in \Gamma$. We say that a pseudo-interpretation $\mathcal{I}$ is a *pseudo-model* of Γ if $\mathcal{I}, x \models \Gamma$ for every $x \in \Delta$.

2.4 Ordering Pseudo-interpretations

In [9] we introduced an ordering between Kripke models. In this subsection, we provide an analogue for ordering pseudo-interpretations in $\mathcal{ALC}$.

A pseudo-interpretation $\mathcal{I}$ is said to be (globally) *less than* or *equal to* a pseudo-interpretation $\mathcal{I}'$, written $\mathcal{I} \leq \mathcal{I}'$, if for every positive concept C, $\mathcal{I}$ validates C implies that $\mathcal{I}'$ also validates C. This differs from [9] at the aspect that, the order $\leq$ given in [9] for comparing Kripke models is "local", as it compares only the contents of the actual worlds.

A pseudo-interpretation $\mathcal{I}$ is called a *least pseudo-model* of a deterministic positive logic program P if it is a pseudo-model of P and is less than or equal to every pseudo-model of P. Note that $\mathcal{I}$ and $\mathcal{I}'$ are least pseudo-models of P does not imply $\mathcal{I} = \mathcal{I}'$, as it only states that, for every positive concept C, $\mathcal{I}$ validates C iff $\mathcal{I}'$ validates C.

Let $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ and $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ be pseudo-interpretations. We say that $\mathcal{I}$ is *locally less than or equal to* $\mathcal{I}'$ *w.r.t. a binary relation* $r \subseteq \Delta \times \Delta'$, and write $\mathcal{I} \leq_r \mathcal{I}'$, if the following conditions hold for every role name R and every atomic concept A:

1. $r(\tau, \tau')$
2. $\forall x, x', y \;\; \mathcal{E}(R)(x, y) \wedge r(x, x') \rightarrow \exists y' \; \mathcal{E}'(R)(x', y') \wedge r(y, y')$
3. $\forall x, x', y' \;\; \mathcal{U}'(R)(x', y') \wedge r(x, x') \rightarrow \exists y \, \mathcal{U}(R)(x, y) \wedge r(y, y')$
4. $\forall x, x' \;\; r(x, x') \rightarrow (x \in \mathcal{C}(A) \rightarrow x' \in \mathcal{C}'(A))$

In the above definition, the first three conditions state that r is a forward-backward bisimulation of the frames of $\mathcal{I}$ and $\mathcal{I}'$, starting from τ and τ'. Intuitively, $r(x, x')$ states that the set of positive concepts containing x is less than or equal to the set of positive concepts containing x'.

Lemma 1. *Let $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ and $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ be pseudo-interpretations. Suppose that $\mathcal{I} \leq_r \mathcal{I}'$. Then for every positive concept C and every $x \in \Delta$ and $x' \in \Delta'$ such that $r(x, x')$ holds, $\mathcal{I}, x \models C$ implies $\mathcal{I}', x' \models C$. In particular, for every positive concept C, $\mathcal{I}, \tau \models C$ implies $\mathcal{I}', \tau' \models C$.*

The proofs of this lemma and the following corollary are presented in [10].

Corollary 1. *Let P be a deterministic positive logic program and $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ be a pseudo-model of P. Suppose that for every pseudo-model $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ of P, there exists $r \subseteq \Delta \times \Delta'$ such that $\mathcal{I} \leq_r \mathcal{I}'$. Then $\mathcal{I}$ is a least pseudo-model of P, and furthermore, for every positive concept C, $\mathcal{I}$ validates C iff $\mathcal{I}, \tau \models C$.*

3 Constructing Finite Least Pseudo-models

In this section, we present an algorithm that, given a deterministic positive logic program P in $\mathcal{ALC}$, constructs a finite least pseudo-model of P. In that algorithm we use the following data structures:

- Δ is a set forming the domain of the constructed pseudo-interpretation.
- $\tau \in \Delta$ is a distinguished element of Δ.
- $\mathbf{C}$ is a map such that for every $x \in \Delta$, $\mathbf{C}(x)$ is a set of concepts. We will treat elements of Δ as possible worlds (as in modal logic), and $\mathbf{C}(x)$ thus denotes the "content" of the possible world x.
- $\mathbf{E}$ is a map such that for $x \in \Delta$ and $\exists R.C \in \mathbf{C}(x)$, $\mathbf{E}(x, \exists R.C) \in \Delta$. The meaning of $\mathbf{E}(x, \exists R.C) = y$ is that $\exists R.C \in \mathbf{C}(x)$, $C \in \mathbf{C}(y)$, and the "requirement" $\exists R.C$ is satisfied at x by going to y via R (treating x and y as possible worlds).
- $\mathbf{U}$ is a map such that for $x \in \Delta$ and a role name R, $\mathbf{U}(x, R) \in \Delta$. Let us give the intuition behind the use of this map. If the content of x contains only $\exists R.C$, then by connecting x to y with $C \in \mathbf{C}(y)$, $\exists R.C$ will be satisfied at x, but $\forall R.C$ will also be satisfied at x, which is unexpected. The solution is that for every $x \in \Delta$ and every role name R, we connect x via R to some y with content forced by the content of x (i.e. $\{D \mid \forall R.D \in \mathbf{C}(x)\}$). However, this has the undesirable side effect that $\exists R.\top$ is satisfied at x. Hence we need to distinguish the edge $R(x, y)$ from the "normal" edges of R and that is why we use both the maps $\mathbf{E}$ and $\mathbf{U}$.

Function $\mathsf{Find}(X)$

1. if there exists $x \in \Delta$ with $\mathbf{C}(x) = X$ then return x,
2. else add a new element x to Δ with $\mathbf{C}(x) := X$ and return x.

Procedure $\mathsf{Simulate\text{-}Changing\text{-}Content}(a, X)$

1. $a_* := \mathsf{Find}(X)$;
2. for every b, R, C, if $\mathbf{E}(b, \exists R.C) = a$ then $\mathbf{E}(b, \exists R.C) := a_*$;
3. for every b and R, if $\mathbf{U}(b, R) = a$ then $\mathbf{U}(b, R) := a_*$;
4. if $\tau = a$ then $\tau := a_*$.

(Note that the above procedure causes a be unreachable from τ unless $a_* = a$.)

Algorithm 1
Input: A deterministic positive logic program P in $\mathcal{ALC}$.
Output: A least pseudo-model $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ of P.

1. $\Delta := \{\tau\}$; $\mathbf{C}(\tau) := P$; let $\mathbf{E}$ and $\mathbf{U}$ be empty;
2. for every $a \in \Delta$ and every $C \in \mathbf{C}(a)$
   (a) case $C = D \sqcap D'$:
   $\mathsf{Simulate\text{-}Changing\text{-}Content}(a, \mathbf{C}(a) \cup \{D, D'\})$;
   (b) case $C = D \sqsubseteq D'$: if $\mathcal{I}, a \models_c D$ then
   $\mathsf{Simulate\text{-}Changing\text{-}Content}(a, \mathbf{C}(a) \cup \{D'\})$;
   (c) case $C = \forall R.D$: for every $b \in \Delta$ such that $\mathcal{U}(R)(a, b)$ holds:
   i. $b_* := \mathsf{Find}(\mathbf{C}(b) \cup \{D\})$;
   ii. for every D', if $\mathbf{E}(a, \exists R.D') = b$ then $\mathbf{E}(a, \exists R.D') := b_*$;
   iii. if $\mathbf{U}(a, R) = b$ then $\mathbf{U}(a, R) := b_*$;
   (d) case $C = \exists R.D$: if $\mathbf{E}(a, \exists R.D)$ is not defined then
   $\mathbf{E}(a, \exists R.D) := \mathsf{Find}(\{D\} \cup P \cup \{D' \mid \forall R.D' \in \mathbf{C}(a)\})$;
3. for every $a \in \Delta$ and every role name R,
   if $\mathbf{U}(a, R)$ is not defined then
   $\mathbf{U}(a, R) := \mathsf{Find}(P \cup \{D' \mid \forall R.D' \in \mathbf{C}(a)\})$;
4. while some change occurred, go to step 2;
5. for every $a \in \Delta$, if a is not reachable from τ (i.e. there does not exist a path $a_0 = \tau$, $a_1, \ldots, a_{k-1}, a_k = a$ with role names $R_1, \ldots, R_k$ such that $\mathcal{U}(R_i)(a_{i-1}, a_i)$ holds for every $1 \leq i \leq k$) then delete a from Δ and delete the elements of $\mathbf{E}$ and $\mathbf{U}$ that are related with a;

Fig. 1. Algorithm for Constructing Least Pseudo-Models

- $\mathcal{C}(A) = \{x \mid A \in \mathbf{C}(x)\}$ for every atomic concept A;
- $\mathcal{E}(R) = \{(x, y) \mid \mathbf{E}(x, \exists R.C) = y \text{ for some } C\}$ for every role name R;
- $\mathcal{U}(R) = \mathcal{E}(R) \cup \{(x, y) \mid \mathbf{U}(x, R) = y\}$ for every role name R;
- $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$, as a pseudo-interpretation.

In Algorithm 1 given in Fig. 1, our construction of a least pseudo-model of a deterministic positive logic program P is done using the technique of building model graphs as in modal logic [9]. The domain Δ of the constructed pseudo-interpretation plays the role of a set of possible worlds. The content $\mathbf{C}(x)$ of

each $x \in \Delta$ is a set of concepts containing P. At the beginning Δ contains only τ with $\mathbf{C}(\tau) = P$. Then for each $a \in \Delta$ and each concept $C \in \mathbf{C}(a)$, we "realize the requirement C at a" as follows:

- Case $C = D \sqcap D'$ (step 2a) : Normally, we would like to add both D and D' to $\mathbf{C}(a)$. But if we do so then there may occur the situation in which $\mathbf{C}(a) = \mathbf{C}(a')$ for some $a' \neq a$. To restrict the size of the constructed pseudo-interpretation, we prevent that situation as follows. We do not change the content of a, but just "simulate the role of a" by a_* with $\mathbf{C}(a_*) = \mathbf{C}(a) \cup \{D, D'\}$. The simulation is done by the procedure Simulate-Changing-Content, which replaces the connections to a by connections to a_* (by modifying the maps $\mathbf{E}$ and $\mathbf{U}$) and sets $\tau := a_*$ if $\tau = a$.
- Case $C = D \sqsubseteq D'$ (step 2b) : If D is "certainly satisfied" at a, denoted by $\mathcal{I}, a \models_c D$, then we simulate the role of a by a_* with $\mathbf{C}(a_*) = \mathbf{C}(a) \cup \{D'\}$ by calling the procedure Simulate-Changing-Content. Let us explain the satisfaction relation $\models_c$. Consider the case when $\mathbf{C}(a) = \{D \sqsubseteq D', \exists R.A\}$ with $D = \forall\exists R.A$, $\mathbf{E}(a, \exists R.A)$ was defined, but $\mathbf{U}(a, R)$ was not. We have that $\mathcal{I}, a \models \forall\exists R.A$, which is undesirable since $\forall\exists R.A$ does not follow from $\exists R.A$. The problem is that $\mathbf{U}(a, R)$ was not yet defined. So, we define the satisfaction relation $\mathcal{I}, a \models_c D$ for a deterministic positive concept D recursively as follows:

$$
\begin{array}{ll}
\mathcal{I}, a \models_c A & \text{iff } a \in \mathcal{C}(A) \\
\mathcal{I}, a \models_c D_1 \sqcap D_2 & \text{iff } \mathcal{I}, a \models_c D_1 \text{ and } \mathcal{I}, a \models_c D_2 \\
\mathcal{I}, a \models_c D_1 \sqcup D_2 & \text{iff } \mathcal{I}, a \models_c D_1 \text{ or } \mathcal{I}, a \models_c D_2 \\
\mathcal{I}, a \models_c \exists R.D & \text{iff } \exists b.(\mathcal{E}(R)(a, b) \wedge \mathcal{I}, b \models_c D) \\
\mathcal{I}, a \models_c \forall\exists R.D & \text{iff } \forall b.(\mathcal{U}(R)(a, b) \rightarrow \mathcal{I}, b \models_c D) \text{ and } \mathcal{I}, a \models_c \exists R.D \\
 & \quad \text{and } \mathbf{U}(a, R) \text{ is defined.}
\end{array}
$$

- Case $C = \forall R.D$ (step 2c) : For every $b \in \Delta$ such that $\mathcal{U}(R)(a, b)$ holds, we would like to add D to $\mathbf{C}(b)$. However, modifying the content of b has two drawbacks: First, other possible worlds connected to b will be affected. For example, if D is added to $\mathbf{C}(b)$ and $\mathcal{E}(R')(c, b)$ holds, then $\exists R'.D$ becomes satisfied at c, while a and c may be independent. Second, modifying $\mathbf{C}(b)$ may cause $\mathbf{C}(b) = \mathbf{C}(b')$ for some $b' \neq b$, which is undesirable. The step 2c contains our solution for these two problems.
- Case $C = \exists R.D$ (step 2d) : We just connect a via R to the possible world with content $\{D\} \cup P \cup \{D' \mid \forall R.D' \in \mathbf{C}(a)\}$) by setting $\mathbf{E}(a, \exists R.D)$ to that world, if it was not done earlier (i.e. if $\mathbf{E}(a, \exists R.D)$ is not defined). Note that P is included because it plays the role of global axioms.

In the step 3 of Algorithm 1, we also guarantee that for every $a \in \Delta$ and every role name R, a is connected via R to the possible world with content $P \cup \{D' \mid \forall R.D' \in \mathbf{C}(a)\}$ by setting $\mathbf{U}(a, R)$ to that world.

When iteration of the steps 2 and 3 does not modify the model graph anymore, we delete all possible worlds that are not reachable from the distinguished world τ (via a path using edges of $\mathcal{U}$). This is necessary because such a possible world

a may contain a concept C which is not satisfied at a (for example, we did not add D and D' to $\mathbf{C}(a)$ for $D \sqcap D' \in \mathbf{C}(a)$, but just simulated the task).

Proposition 1. *Algorithm 1 terminates in $2^{O(n)}$ steps and returns a pseudo-interpretation of size $2^{O(n)}$, where n is the size of the input program P.*

Proof. Before reaching the step 5, no elements of Δ are deleted, and for each $x \in \Delta$, $\mathbf{C}(x)$ never changes. Since $\mathbf{C}(x)$ is a set of sub-concepts of the clauses of P, its size is $O(n)$. Since $\mathbf{C}(x) \neq \mathbf{C}(x')$ for every $x \neq x'$, the size of Δ is $2^{O(n)}$. Hence the sizes of the maps $\mathbf{E}$ and $\mathbf{U}$ are also of rank $2^{O(n)}$, and the number of times executing the steps 2d and 3 is $2^{O(n)}$. For the step 2c, note that $b_* \neq b$ iff $\mathbf{C}(b_*) \supset \mathbf{C}(b)$. Similarly, for the calls of Simulate-Changing-Content, $a_* \neq a$ iff $\mathbf{C}(a_*) \supset \mathbf{C}(a)$. Hence the number of times $\mathbf{E}$ and $\mathbf{U}$ are modified by the steps 2a - 2c is $2^{O(n)}.n = 2^{O(n)}$. The total number of times modifying $\mathbf{E}$ and $\mathbf{U}$ is therefore of rank $2^{O(n)}$. Hence the time complexity of Algorithm 1 is $2^{O(n)}$.

Lemma 2. *Algorithm 1 has the following properties:*

1. *During an execution, for every $x \in \Delta$ and every concept $\exists R.D$, if $\mathbf{E}(x, \exists R.D) = y$ then $\exists R.D \in \mathbf{C}(x)$ and $D \in \mathbf{C}(y)$.*
2. *At the end, $\mathbf{E}(x, \exists R.D)$ is defined for every $x \in \Delta$ and every concept $\exists R.D \in \mathbf{C}(x)$, and $\mathbf{U}(x, R)$ is defined for every $x \in \Delta$ and every role name R.*

Proof. The first assertion clearly holds. For the second assertion, just note that, before executing the step 5, $\mathbf{E}(x, \exists R.D)$ is defined for every $x \in \Delta$ and every $\exists R.D \in \mathbf{C}(x)$, and $\mathbf{U}(x, R)$ is defined for every $x \in \Delta$ and every role name R.

The following lemma states that the pseudo-interpretation $\mathcal{I}$ constructed by Algorithm 1 for P is a pseudo-model of P.

Lemma 3. *Let P be a deterministic positive logic program in $\mathcal{ALC}$ and $\mathcal{I}$ the pseudo-interpretation constructed by Algorithm 1 for P. Let Δ and $\mathbf{C}$ be the data structures used by the algorithm. Then for every $a \in \Delta$ and $C \in \mathbf{C}(a)$, $\mathcal{I}, a \models C$. As a consequence, $\mathcal{I}$ is a pseudo-model of P (since $P \subseteq \mathbf{C}(a)$ for every $a \in \Delta$).*

Proof. By induction on the construction of C.

Consider the case when C is of the form $D \sqsubseteq D'$. Suppose that $\mathcal{I}, a \models D$. Since $\mathbf{U}(x, R)$ is defined for every $x \in \Delta$ and every role name R, it follows that $\mathcal{I}, a \models_c D$ iff $\mathcal{I}, a \models D$. Hence $\mathcal{I}, a \models_c D$. When the step 2b is executed the last time for a and C, because a is reachable from τ via a path using $\mathcal{U}$ (as it remains after executing the step 5) and no changes are made by the step 2b, we have that $a_* = a$ (where a_* is the element simulating the role of a). Since $D \sqsubseteq D' \in \mathbf{C}(a)$ and $\mathcal{I}, a \models_c D$, we have that $D' \in \mathbf{C}(a_*)$, i.e. $D' \in \mathbf{C}(a)$. By the inductive assumption, $\mathcal{I}, a \models D'$. Therefore, $\mathcal{I}, a \models D \sqsubseteq D'$.

The case when C is of the form $D \sqcap D'$ is similar to the above case. The cases when C is of the form A, $\forall R.D$, or $\exists R.D$ are straightforward.

We use the following Lemmas 4 and 5 to show that the pseudo-model $\mathcal{I}$ of P constructed by Algorithm 1 is less than or equal to every pseudo-model of P.

Lemma 4. *Let P be a deterministic positive logic program in $\mathcal{ALC}$ and $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ be an arbitrary pseudo-model of P. Consider a moment after executing a numerated step in an execution of Algorithm 1 for P. Let $r = \{(x, x') \in \Delta \times \Delta' \mid \mathcal{I}', x' \models \mathbf{C}(x)\}$. Then the following conditions hold:*

1. $r(\tau, \tau')$
2. $\forall x, y, x', y', R'', D''$
   $r(x, x') \wedge (\mathbf{E}(x, \exists R''.D'') = y) \wedge \mathcal{E}'(R'')(x', y') \wedge (\mathcal{I}', y' \models D'') \rightarrow r(y, y')$
3. $\forall x, y, x', y', R''$ $r(x, x') \wedge (\mathbf{U}(x, R'') = y) \wedge \mathcal{U}'(R'')(x', y') \rightarrow r(y, y')$

(We use the names R'' and D'' because R, D, and D' occur in Algorithm 1.)

The proof of this lemma is presented in [10].

Lemma 5. *Let P be a deterministic positive logic program in $\mathcal{ALC}$, $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ be the pseudo-interpretation constructed by Algorithm 1 for P, $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ be an arbitrary pseudo-model of P, and $r = \{(x, x') \in \Delta \times \Delta' \mid \mathcal{I}', x' \models \mathbf{C}(x)\}$. Then $\mathcal{I} \leq_r \mathcal{I}'$.*

Proof. By Lemma 4, $r(\tau, \tau')$ holds.

We prove that $\forall\, x, x', y$ $\mathcal{E}(R)(x, y) \wedge r(x, x') \rightarrow \exists y'\, \mathcal{E}'(R)(x', y') \wedge r(y, y')$. Suppose that $\mathcal{E}(R)(x, y)$ and $r(x, x')$ hold. There must exist D such that $\mathbf{E}(x, \exists R.D) = y$. Thus $\exists R.D \in \mathbf{C}(x)$, and hence $\mathcal{I}', x' \models \exists R.D$. Let y' be an element of Δ' such that $\mathcal{E}'(R)(x', y')$ holds and $\mathcal{I}', y' \models D$. By Lemma 4, $r(y, y')$ holds.

We prove that $\forall\, x, x', y'$ $\mathcal{U}'(R)(x', y') \wedge r(x, x') \rightarrow \exists y\, \mathcal{U}(R)(x, y) \wedge r(y, y')$. Suppose that $\mathcal{U}'(R)(x', y')$ and $r(x, x')$ hold. Let $\mathbf{U}(x, R) = y$. By Lemma 4, $r(y, y')$ holds.

By the definition of r, we have that $r(x, x') \rightarrow (x \in \mathcal{C}(A) \rightarrow x' \in \mathcal{C}'(A))$.

Here is the main result of this section:

Theorem 1. *Let P be a deterministic positive logic program in $\mathcal{ALC}$ and $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ be the pseudo-interpretation constructed by Algorithm 1 for P. Then $\mathcal{I}$ is a least pseudo-model of P, and for every positive concept C, $\mathcal{I}$ validates C iff $\mathcal{I}, \tau \models C$.*

Proof. By Lemma 3, $\mathcal{I}$ is a pseudo-model of P. Let $\mathcal{I}' = \langle \Delta', \tau', \mathcal{C}', \mathcal{E}', \mathcal{U}' \rangle$ be an arbitrary pseudo-model of P and let $r = \{(x, x') \in \Delta \times \Delta' \mid \mathcal{I}', x' \models \mathbf{C}(x)\}$. By Lemma 5, $\mathcal{I} \leq_r \mathcal{I}'$. By Corollary 1, it follows that $\mathcal{I}$ is a least pseudo-model of P, and for every positive concept C, $\mathcal{I}$ validates C iff $\mathcal{I}, \tau \models C$.

4 Characterizations of Least Pseudo-models

In this section, we show that every least pseudo-model $\mathcal{I}$ of a deterministic positive logic program P *characterizes* P in the sense that, for every deterministic positive concept C, $P \models C$ iff $\mathcal{I}$ validates C. Moreover, if $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ is the pseudo-model constructed by Algorithm 1 for P then $P \models C$ iff $\mathcal{I}, \tau \models C$.

Given a pseudo-interpretation $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$, let $\mathcal{I}' = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}', \mathcal{U}' \rangle$ be the pseudo-interpretation such that, for every role name R,

$$\mathcal{E}'(R) = \mathcal{E}(R) \cup \{(x,y) \mid \mathcal{U}(R)(x,y) \text{ and } \mathcal{E}(R)(x,y') \text{ hold for some } y'\},$$
$$\mathcal{U}'(R) = \mathcal{U}(R) \setminus \{(x,y) \mid y \in W \text{ and } \mathcal{E}(R)(x,y') \text{ does not hold for any } y'\}.$$

Fix a role name R and $x \in \Delta$. Recall that $\mathcal{E}(R) \subseteq \mathcal{U}(R)$. If $\mathcal{E}(R)(x,y')$ holds for some y' then for every $y \in \Delta$, $\mathcal{E}'(R)(x,y) \equiv \mathcal{U}(R)(x,y) \equiv \mathcal{U}'(R)(x,y)$. If $\mathcal{E}(R)(x,y')$ does not hold for any y' then for every $y \in \Delta$, both $\mathcal{E}'(R)(x,y)$ and $\mathcal{U}'(R)(x,y)$ do not hold. Thus $\mathcal{U}' = \mathcal{E}'$ and $\mathcal{I}'$ can be treated as an interpretation. We call $\mathcal{I}'$ the *interpretation corresponding to* M.

We need the following auxiliary lemma, which is proved in [10].

Lemma 6. *Let P be a deterministic positive logic program in $\mathcal{ALC}$, $\mathcal{I}$ the pseudo-model of P constructed by Algorithm 1, $\mathcal{I}'$ the interpretation corresponding to $\mathcal{I}$, and C a deterministic positive concept. Let $r = \{(x,x') \in \Delta \times \Delta \mid \mathcal{I}, x' \models \mathbf{C}(x)\}$, where Δ and $\mathbf{C}$ are the data structures used by Algorithm 1 for P. Then for every $x, x' \in \Delta$, if $r(x,x')$ holds and $\mathcal{I}', x \models C$ then $\mathcal{I}, x' \models C$. In particular, since r is reflexive (by Lemma 3), for every $x \in \Delta$, $\mathcal{I}', x \models C$ implies $\mathcal{I}, x \models C$. As a consequence, if $\mathcal{I}'$ validates C then $\mathcal{I}$ also validates C.*

Theorem 2. *Let P be a deterministic positive logic program in $\mathcal{ALC}$, $\mathcal{I}$ a least pseudo-model of P, and C a deterministic positive concept. Then $P \models C$ iff $\mathcal{I}$ validates C.*

Proof. Consider the "if" direction. Suppose that $\mathcal{I}$ validates C. Let $\mathcal{I}'$ be an arbitrary model of P. As $\mathcal{I}'$ is also a pseudo-model of P, we have that $\mathcal{I} \leq \mathcal{I}'$. Hence $\mathcal{I}'$ validates C. Therefore $P \models C$.

Now consider the "only if" direction. Suppose that $P \models C$.

Without loss of generality, we can assume that $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ is the pseudo-model of P constructed by Algorithm 1. Let $\mathcal{I}' = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}', \mathcal{U}' \rangle$ be the interpretation corresponding to $\mathcal{I}$. It is sufficient to show that $\mathcal{I}'$ is a model of P, because this implies that $\mathcal{I}'$ validates C, and by Lemma 6, $\mathcal{I}$ validates C.

Let $\mathbf{C}$ be the map used by Algorithm 1 for P. To show that $\mathcal{I}'$ is a model of P, we prove by induction on the construction of D that if $D \in \mathbf{C}(x)$ then $\mathcal{I}', x \models D$. The only non-trivial case is when D is of the form $D_1 \sqsubseteq D_2$. Consider this case and suppose that $D \in \mathbf{C}(x)$ and $\mathcal{I}', x \models D_1$. We need to show that $\mathcal{I}', x \models D_2$. Since $\mathcal{I}', x \models D_1$, by Lemma 6, $\mathcal{I}, x \models D_1$. Since $D \in \mathbf{C}(x)$, by Lemma 3, $\mathcal{I}, x \models D$. Hence $\mathcal{I}, x \models D_2$. This implies that $\mathcal{I}', x \models D_2$ (because $\mathcal{E}(R) \subseteq \mathcal{E}'(R)$ and $\mathcal{U}'(R) \subseteq \mathcal{U}(R)$ for every role name R).

Theorem 3. *Let P be a deterministic positive logic program in $\mathcal{ALC}$, $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ be the pseudo-model of P constructed by Algorithm 1, and $\mathcal{I}'$ be the interpretation corresponding to $\mathcal{I}$. Then:*

1. *For every deterministic positive concept C, the following conditions are equivalent: (a) $P \models C$; (b) $\mathcal{I}$ validates C; (c) $\mathcal{I}, \tau \models C$; (d) $\mathcal{I}'$ validates C; (e) $\mathcal{I}', \tau \models C$.*
2. *For every positive concept C, if $\mathcal{I}, \tau \models C$ then $P \models C$.*

Proof. Consider the first assertion. The equivalence $(a) \Leftrightarrow (b)$ follows from Theorem 2. The equivalence $(b) \Leftrightarrow (c)$ follows from Theorem 1. As shown in the proof of Theorem 2, $\mathcal{I}'$ is a model P. Hence, $(a) : P \models C$ implies that $(d) : \mathcal{I}'$ validates C, which implies $(e) : \mathcal{I}', \tau \models C$. For the implication $(e) \Rightarrow (a)$, suppose that $\mathcal{I}', \tau \models C$. By Lemma 6, $\mathcal{I}, \tau \models C$. By Theorem 1, it follows that $\mathcal{I}$ validates C. Let $\mathcal{I}''$ be an arbitrary model of P. Since $\mathcal{I}$ is a least pseudo-model of P, we have that $\mathcal{I} \leq \mathcal{I}''$, which implies that $\mathcal{I}''$ validates C. Hence $P \models C$.

The previous four sentences also comprise the proof for the second assertion.

5 Further Work and Conclusions

We have given an algorithm that for a deterministic positive logic program P in $\mathcal{ALC}$ treated as a TBox constructs a finite least pseudo-model $\mathcal{I} = \langle \Delta, \tau, \mathcal{C}, \mathcal{E}, \mathcal{U} \rangle$ of P such that for every deterministic positive concept C, $P \models C$ iff $\mathcal{I}$ validates C and iff $\mathcal{I}, \tau \models C$. The interpretation $\mathcal{I}'$ corresponding to $\mathcal{I}$ also satisfies that $P \models C$ iff $\mathcal{I}'$ validates C and iff $\mathcal{I}', \tau \models C$. Thus $\mathcal{I}'$ also characterizes P, but the pseudo-model $\mathcal{I}$ has the additional nice property that for every positive concept C (not necessarily deterministic), if $\mathcal{I}, \tau \models C$ then $P \models C$. Our restriction to the deterministic Horn fragment is to overcome the problem of nondeterminism caused by non-seriality of the relations interpreting role names.

Apart from the idea of using pseudo-interpretations to deal with non-seriality, our algorithm given in this paper for $\mathcal{ALC}$ differs from our algorithm given in [9] for basic serial monomodal logics in the aspect that it uses graphs instead of trees and uses a special caching technique for building model graphs. These techniques are essential for getting the exponential upper bound for the time complexity and the size of the constructed pseudo-interpretation. Our technique of simulating the task of changing contents of nodes is important for the algorithm. Without it we need to modify the contents of nodes, which may cause that merging duplicates is necessary and the old nodes will have to be re-created later, and hence the performance is slowed down and complexity analysis could be difficult.

The exponential time (combined) complexity of our algorithm is not surprising, because $\mathcal{ALC}$ is EXPTIME-complete and in many modal logics, e.g. basic monomodal logics without axiom 5, the restriction to the Horn fragment does not reduce the complexity. We believe that: a) there are deterministic positive logic programs such that their least pseudo-models must have an exponential size; b) the (combined) complexity of the deterministic Horn fragment of $\mathcal{ALC}$ is EXPTIME-complete.

The main contribution of this work is our bottom-up method for query answering for the deterministic Horn fragment of description logics. By using the direct approach instead of transformation to classical Horn logic, we can handle a larger Horn fragment of $\mathcal{ALC}$ for TBoxes than DHL studied by Grosof et al. in [6] and Horn-$\mathcal{SHIQ}$ studied by Hustadt et al. in [7].

Our method is extendable for instance checking (i.e. $P \models C(a)$) w.r.t. logic programs containing also an ABox and relations between roles of the form $Q \subseteq R$ or $R^+ \subseteq R$. The extension looks as follows. We incorporate the relations between

roles by using the corresponding axioms $\forall R.C \sqsubseteq \forall Q.C$ and $\forall R.C \sqsubseteq \forall R.\forall R.C$ as in modal logic. When building a model graph for P, we start with the minimal graph that represents the ABox of P and contains additionally the distinguished node τ (with no edges connecting to or from). Then the algorithm can continue in a similar way as Algorithm 1. Of course, the resulting pseudo-interpretation $\mathcal{I}$ does not anymore satisfy that, for every positive concept C, $\mathcal{I}$ validates C iff $\mathcal{I}, \tau \models C$ (because of the ABox). Further investigation is needed for dealing with inverse roles as in DHL and Horn-$\mathcal{SHIQ}$.

As a further work, we will extend Horn-$\mathcal{SHIQ}$ [7] to dHorn-$\mathcal{SHIQ}$ by allowing the constructor $\forall\exists R.C$ to appear in bodies of program clauses and queries and study the CARIN-like system that uses dHorn-$\mathcal{SHIQ}$ for the terminology layer. We believe that such a system has PTIME data complexity.

Acknowledgements. I would like to thank the reviewers for useful comments.

References

1. F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
2. M. Cadoli, L. Palopoli, and M. Lenzerini. Datalog and description logics: Expressive power. In S. Cluet and R. Hull, editors, *DBPL-6, LNCS 1369*, pages 281–298. Springer, 1998.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Description Logics*, 2005.
4. F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. A hybrid system with Datalog and concept languages. In E. Ardizzone, S. Gaglio, and F. Sorbello, editors, *Trends in Artificial Intelligence, LNAI 549*, pages 88–97. Springer-Verlag, 1991.
5. E. Franconi and S. Tessaris. Rules and queries with ontologies: A unified logical framework. In H.J. Ohlbach and S. Schaffert, editors, *PPSWR 2004, LNCS 3208*, pages 50–60. Springer, 2004.
6. B.N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.
7. U. Hustadt, B. Motik, and U. Sattler. Data complexity of reasoning in very expressive description logics. In L.P. Kaelbling and A. Saffiotti, editors, *IJCAI*, pages 466–471. Professional Book Center, 2005.
8. A.Y. Levy and M.-Ch. Rousset. Combining Horn rules and description logics in carin. *Artificial Intelligence*, 104(1-2):165–209, 1998.
9. L.A. Nguyen. Constructing the least models for positive modal logic programs. *Fundamenta Informaticae*, 42(1):29–60, 2000.
10. L.A. Nguyen. The long version of this paper. Available at `http://www.mimuw.edu.pl/~nguyen/nguyen06jelia.pdf`, 2006.

Fuzzy Answer Set Programming

Davy Van Nieuwenborgh[1,⋆], Martine De Cock[2], and Dirk Vermeir[1]

[1] Vrije Universiteit Brussel, VUB
Dept. of Computer Science
Pleinlaan 2, B-1050 Brussels, Belgium
{dvnieuwe, dvermeir}@vub.ac.be
[2] Universiteit Gent, UGent
Dept. of Applied Mathematics and Computer Science
Krijgslaan 281 (S9), B-9000 Ghent, Belgium
martine.decock@ugent.be

Abstract. In this paper we show how the concepts of answer set programming and fuzzy logic can be succesfully combined into the single framework of fuzzy answer set programming (FASP). The framework offers the best of both worlds: from the answer set semantics, it inherits the truly declarative non-monotonic reasoning capabilities while, on the other hand, the notions from fuzzy logic in the framework allow it to step away from the sharp principles used in classical logic, e.g., that something is either completely true or completely false. As fuzzy logic gives the user great flexibility regarding the choice for the interpretation of the notions of negation, conjunction, disjunction and implication, the FASP framework is highly configurable and can, e.g., be tailored to any specific area of application. Finally, the presented framework turns out to be a proper extension of classical answer set programming, as we show, in contrast to other proposals in the literature, that there are only minor restrictions one has to demand on the fuzzy operations used, in order to be able to retrieve the classical semantics using FASP.

1 Introduction

The answer set programming (ASP) paradigm [15] has gained a lot of popularity in the last years, due to its truly declarative non-monotonic semantics, which has been proven useful in a number of interesting applications, e.g. [21, 12, 19, 16]. The idea behind the answer set semantics, a generalisation of the stable model semantics [14], is both intuitive and elegant. Given a program P and a candidate answer set M, one computes a reduct program P^M of a simpler type for which a semantics $(P^M)^\star$ is known. The reduct P^M is obtained from P by taking into account the consequences of accepting the proposed truth values of the literals in M. The candidate set M is then an answer set just when $(P^M)^\star = M$, i.e. M is "self-producible".

Although ASP provides a powerful solution for knowledge representation and non-monotonic reasoning, it has some drawbacks regarding the configureability of the semantics w.r.t. the type of application under consideration, as witnessed by the large number of extensions, both syntactically and semantically, that have been proposed in

⋆ Supported by the Flemish Fund for Scientific Research (FWO-Vlaanderen).

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 359–372, 2006.

the literature [7, 10, 5, 2]. E.g., most[1] ASP semantics demand that a solution to a program satisfies all the rules. Further, the literals available in the program, i.e. the building blocks of rules, can only be true or false (or unknown when one considers the well-founded semantics [23]), and classical consistency is mandatory, i.e. a and $\neg a$ cannot be true at the same time (or not even "a bit" true at the same time). Also the interpretation of negation as failure, the construct that gives ASP its non-monotonicity, is very sharp: *not* a is true iff a is not true.

Sometimes however, it is impossible to find a solution that fully satisfies all rules of the program. In this case, one might still wish to look for a solution satisfying the program at least to a reasonably high degree. At other times, it may not even be required to obtain a solution that satisfies a program fully. That is, one might be more interested in a solution satisfying the program to a satisfactory high degree, especially if this solution comes at a lower cost. Consider the following problem based on an example from [2].

Example 1. There are four different kinds of sports that we like to practice to some degree. However, only certain combinations of sports lead to a full-body exercise. Furthermore, some of the sports complement each other, i.e. less practice of one automatically leads to more practice of the other (rules $r_1 \ldots r_4$ in the program below).

$$
\begin{aligned}
r_1 &: \quad lift\_weights \leftarrow \mathit{not}\ swim \\
r_2 &: \quad swim \leftarrow \mathit{not}\ lift\_weights \\
r_3 &: \quad run \leftarrow \mathit{not}\ play\_ball \\
r_4 &: \quad play\_ball \leftarrow \mathit{not}\ run \\
r_5 &: full\_body\_exercise \leftarrow lift\_weights, run \\
r_6 &: full\_body\_exercise \leftarrow swim, play\_ball \\
r_7 &: \quad \leftarrow \mathit{not}\ full\_body\_exercise
\end{aligned}
$$

The two classical answer sets of this program are $\{full\_body\_exercise, lift\_weights, run\}$ and $\{play\_ball, full\_body\_exercise, swim\}$. Hence, to achieve a full body exercise, one needs to practice either weight lifting and running, or ball playing and swimming to the highest degree. However, in addition, we might be interested to know which combinations of the four sports we should practice, and to what degree, such that an acceptable degree, e.g. 0.7, of full-body exercise is obtained.

Fuzzy logic is a suitable framework for dealing with degrees of truth and satisfaction [26]. In its most general form, fuzzy logic considers a complete lattice $\mathcal{L}$ of truth values on which it redefines the classical operations of negation, conjunction, disjunction and implication; in such a way that they correspond to the classical ones in the top and bottom elements of the lattice. One of the strengths of fuzzy logic regarding these operations is that a user can freely choose, depending on the type of application under consideration, which specific definition she uses for the operations.

A combination with fuzzy logic increases the flexibility and hence the application potential of ASP. Such flexibility can be introduced at several levels. In the fuzzy answer set programming (FASP) framework introduced in this paper, we consider fuzzy answer sets, which means that literals can belong to an answer set to a certain extent, as opposed

[1] Some semantics that deal with preferences among rules [13, 6, 24] are more flexible.

to either belonging to the answer set or not. In accordance, the literals in a program can be true to a certain degree. We relax the definition of consistency to allow that, if desired, both a and $\neg a$ can be true to a certain degree at the same time without necessarily loosing consistency. Similarly, we allow for a more flexible interpretation of negation as failure. Crucial to our approach is the notion of a satisfaction function, as it enables us to compute the extent to which a rule is satisfied under a given fuzzy interpretation. The satisfaction function is then used to develop the concept of a fuzzy model. As in traditional ASP, in FASP the fuzzy answer sets of simple programs, i.e. programs without negation as failure, coincide with the fuzzy minimal models. For programs containing negation as failure, the idea underlying GL-reduct is extended to a technique that allows to bring to surface whether a fuzzy model is indeed supported by a program, in other words whether it deserves the name of fuzzy answer set.

The rest of the paper is organized as follows. In Section 2 we give some preliminaries on fuzzy logic and answer set programming, while we introduce the combination of both, i.e. fuzzy answer set programming (FASP), in Section 3. Before giving some comparison with related work in Section 5, we show in Section 4 how the classical answer set semantics can be retrieved from FASP. Finally, we conclude and give some directions for future research in Section 6.

2 Preliminaries

2.1 Truth Lattices

In this paper, we consider a complete truth lattice, i.e. a partially ordered set $(\mathcal{L}, \leq_{\mathcal{L}})$ such that every subset of $\mathcal{L}$ has an infimum (greatest lower bound) and a supremum (least upper bound), which we denote by $\inf$ and $\sup$ respectively [4]. Such a lattice is often denoted by $\mathcal{L}$, tacitly assuming the ordering $\leq_{\mathcal{L}}$. Furthermore, we use $0_{\mathcal{L}}$ and $1_{\mathcal{L}}$ to denote respectively the smallest and the greatest element[2] of $\mathcal{L}$.

The traditional logical operations of negation, conjunction, disjunction, and implication can be generalized to logical operators acting on truth values of $\mathcal{L}$ (see e.g. [20]). A *negator* on $\mathcal{L}$ is any decreasing $\mathcal{L} \to \mathcal{L}$ mapping $\mathcal{N}$ satisfying $\mathcal{N}(0_{\mathcal{L}}) = 1_{\mathcal{L}}$ and $\mathcal{N}(1_{\mathcal{L}}) = 0_{\mathcal{L}}$. It is called involutive if $\mathcal{N}(\mathcal{N}(x)) = x$ for all x in $\mathcal{L}$. A *triangular norm* $\mathcal{T}$ on $\mathcal{L}$ is any commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ mapping $\mathcal{T}$ satisfying $\mathcal{T}(1_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{T}$ to be increasing in both of its components. A triangular norm, or t-norm for short, corresponds to conjunction. A *triangular conorm* $\mathcal{S}$ on $\mathcal{L}$ is any increasing, commutative and associative $\mathcal{L}^2 \to \mathcal{L}$ mapping satisfying $\mathcal{S}(0_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{S}$ to be increasing in both of its components. A triangular conorm, t-conorm for short, corresponds to disjunction. An *implicator* $\mathcal{I}$ on $\mathcal{L}$ is any $\mathcal{L}^2 \to \mathcal{L}$–mapping satisfying $\mathcal{I}(0_{\mathcal{L}}, 0_{\mathcal{L}}) = 1_{\mathcal{L}}$, and $\mathcal{I}(1_{\mathcal{L}}, x) = x$, for all x in $\mathcal{L}$. Moreover we require $\mathcal{I}$ to be decreasing in its first, and increasing in its second component.

The dual of a t-norm $\mathcal{T}$ w.r.t. a negator $\mathcal{N}$ is a t-conorm $\mathcal{S}$ defined as $\mathcal{S}(x, y) = \mathcal{N}(\mathcal{T}(\mathcal{N}(x), \mathcal{N}(y)))$ for all x and y in $\mathcal{L}$. The mapping $\mathcal{I}_{\mathcal{S},\mathcal{N}}$ defined by $\mathcal{I}_{\mathcal{S},\mathcal{N}}(x, y) = \mathcal{S}(\mathcal{N}(x), y)$ is an implicator, usually called S-implicator (induced by $\mathcal{S}$ and $\mathcal{N}$). On the

[2] In the literature one will also find the notation $\bot$ and $\top$ to denote $0_{\mathcal{L}}$ and $1_{\mathcal{L}}$ respectively.

other hand, the mapping $\mathcal{I}_\mathcal{T}$ defined by $\mathcal{I}_\mathcal{T}(x,y) = \sup\{\lambda | \lambda \in \mathcal{L} \text{ and } \mathcal{T}(x,\lambda) \leq_\mathcal{L} y\}$ is an implicator, usually called the residual implicator or R-implicator (of $\mathcal{T}$).

While the framework we will introduce to perform fuzzy answer set programming in Section 3 can be used in combination with any complete lattice, we will restrict ourselves for the examples in the current paper to the complete lattice $([0,1], \leq)$. The following example presents some fuzzy logical operators on this lattice.

Example 2. The mapping $\mathcal{N}_s$ defined as $\mathcal{N}_s(x) = 1 - x$ for all x in $[0,1]$ is called the standard negator. The t-norms $\mathcal{T}_\mathrm{M}$, $\mathcal{T}_\mathrm{P}$, and $\mathcal{T}_\mathrm{W}$ and their dual t-conorms $\mathcal{S}_\mathrm{M}$, $\mathcal{S}_\mathrm{P}$, and $\mathcal{S}_\mathrm{W}$ w.r.t. the standard negator, are defined as

$$\begin{array}{ll} \mathcal{T}_\mathrm{M}(x,y) = \min(x,y) & \mathcal{S}_\mathrm{M}(x,y) = \max(x,y) \\ \mathcal{T}_\mathrm{P}(x,y) = x \cdot y & \mathcal{S}_\mathrm{P}(x,y) = x + y - x \cdot y \\ \mathcal{T}_\mathrm{W}(x,y) = \max(x+y-1,0) & \mathcal{S}_\mathrm{W}(x,y) = \min(x+y,1) \end{array}$$

for all x and y in $[0,1]$. They induce the following implicators (the mappings on the right are R-implicators while those on the left are S-implicators; for ease of notation the inducing negator $\mathcal{N}_s$ has been omitted):

$$\begin{array}{ll} \mathcal{I}_{\mathcal{S}_\mathrm{M}}(x,y) = \max(1-x,y) & \mathcal{I}_{\mathcal{T}_\mathrm{M}}(x,y) = \begin{cases} 1, \text{ if } x \leq y \\ y, \text{ else} \end{cases} \\ \mathcal{I}_{\mathcal{S}_\mathrm{P}}(x,y) = 1 - x + x \cdot y & \mathcal{I}_{\mathcal{T}_\mathrm{P}}(x,y) = \begin{cases} 1, \text{ if } x \leq y \\ \frac{y}{x}, \text{ else} \end{cases} \\ \mathcal{I}_{\mathcal{S}_\mathrm{W}}(x,y) = \min(1-x+y,1) & \mathcal{I}_{\mathcal{T}_\mathrm{W}}(x,y) = \min(1-x+y,1) \end{array}$$

Every implicator induces a negator by defining $\mathcal{N}(x) = \mathcal{I}(x, 0_\mathcal{L})$. The above mentioned S-implicators induce the standard negator $\mathcal{N}_s$, while $\mathcal{I}_{\mathcal{T}_\mathrm{M}}$ and $\mathcal{I}_{\mathcal{T}_\mathrm{P}}$ induce the Gödel negator[3] $\mathcal{N}_g$ defined by $\mathcal{N}_g(x) = 1$ if $x = 0$ and $\mathcal{N}_g(x) = 0$ otherwise.

A fuzzy set in U is a $U \mapsto \mathcal{L}$ mapping. For fuzzy sets A and B in U, A is said to be included in B, denoted by $A \preceq_\mathcal{L} B$, iff $A(u) \leq_\mathcal{L} B(u)$ for all u in U. As usual, we have $A \prec_\mathcal{L} B$ iff $A \preceq_\mathcal{L} B$ and not $B \preceq_\mathcal{L} A$.

2.2 Answer Set Programming

We give some preliminaries concerning the answer set semantics for logic programs [3]. A *literal* is an atom a or a negated atom $\neg a$. For a set of literals X, we use $\neg X$ to denote $\{\neg l \mid l \in X\}$ where $\neg\neg a = a$. When $X \cap \neg X = \emptyset$ we say that X is *consistent*. An *extended literal* is a literal or a *naf-literal* of the form *not* l where l is a literal. The latter form denotes negation as failure. For a set of extended literals Y, we use Y^- to denote the set of ordinary literals underlying the naf-literals in Y, i.e. $Y^- = \{l \mid \mathit{not}\ l \in Y\}$. Further, we use *not* X to denote the set $\{\mathit{not}\ l \mid l \in X\}$. An extended literal l is true w.r.t. X, denoted $X \models l$ if $l \in X$ in case l is ordinary, or $a \notin X$ if $l = \mathit{not}\ a$ for some ordinary literal a. As usual, $X \models Y$ iff $\forall l \in Y \cdot X \models l$.

A *rule* is of the form $\alpha \leftarrow \beta$ where[4] $\alpha \cup \beta$ is a finite set of extended literals and $|\alpha| \leq 1$. Thus the *head* of a rule is either an extended literal or empty. A finite set of

[3] This negator is also known in the literature as the Heyting negator.

[4] For simplicity, we assume that programs have already been grounded.

rules is called a *(logic) program*. The *Herbrand base* $\mathcal{B}_P$ of a program P contains all atoms appearing in P. The set of all literals that can be formed with the atoms in P, denoted by Lit_P, is defined by $Lit_P = \mathcal{B}_P \cup \neg\mathcal{B}_P$. Similarly, we define the set of all extended literals that can be formed with the atoms in P as $Elit_P = Lit_P \cup not\ Lit_P$. Any consistent subset $I \subseteq Lit_P$ is called an *interpretation* of P.

A rule $r = \alpha \leftarrow \beta$ is *satisfied* by an interpretation I, denoted $I \models r$, if $I \models \alpha$ and $\alpha \neq \emptyset$, whenever $I \models \beta$, i.e. if r is *applicable* ($I \models \beta$), then it must be *applied* ($I \models \alpha \cup \beta$ and $\alpha \neq \emptyset$). Note that this implies that a *constraint*, i.e. a rule with empty head ($\alpha = \emptyset$), can only be satisfied if it is not applicable ($I \not\models \beta$). For a program P, an interpretation I is called a *model* of P if $\forall r \in P \cdot I \models r$, i.e. I satisfies all rules in P. It is a minimal model of P if there is no model J of P such that $J \subset I$.

A *simple program* is a program without negation as failure. For simple programs P, we define an *answer set* of P as a minimal model of P. On the other hand, for a program P, i.e. a program containing negation as failure, we define the *GL-reduct* [14] for P w.r.t. I, denoted P^I, as the program consisting of those rules[5] $(\alpha \backslash not\ \alpha^-) \leftarrow (\beta \backslash not\ \beta^-)$ where $\alpha \leftarrow \beta$ is in P, $I \models not\ \beta^-$ and $I \models \alpha^-$. Note that all rules in P^I are free from negation as failure, i.e. P^I is a simple program. An interpretation I is then an *answer set* of P iff I is a minimal model of the GL-reduct P^I.

Example 3. Consider the program

$$r_1 : a \leftarrow not\ b \qquad r_2 : b \leftarrow not\ a$$

Clearly, both $\{a\}$ and $\{b\}$ are answer sets of this program as the GL-reducts $P^{\{a\}} = \{a \leftarrow\}$ and $P^{\{b\}} = \{b \leftarrow\}$ have $\{a\}$ and $\{b\}$ respectively as their minimal model. On the other hand, $\emptyset$ and $\{a, b\}$ are not answer sets. For the former interpretation, the reduct $P^{\emptyset} = \{a \leftarrow\ ;\ b \leftarrow\}$ has $\{a, b\}$ as its minimal model which differs from $\emptyset$, while the latter has an empty reduct, thus an empty minimal model, which differs from $\{a, b\}$.

3 Fuzzy Answer Set Programming

Classical ASP, as defined in the previous subsection, is in some ways a very strict framework in its semantics. In particular, an answer set is required to satisfy all rules of the program fully. In a more flexible setting, we wish to be able to deal with interpretations that satisfy rules possibly only to a certain extent. To this end, we allow literals to be true to a degree, as opposed to either being true or not true. As such, interpretations, and hence also answer sets, become fuzzy sets in Lit_P.

As the high configurability of fuzzy logic can be seen as one of its main strengths, we will adopt this behavior to the FASP framework presented in this section. Therefore, we allow a user to choose, in function of the application at hand, how the different classical operations need to be interpreted. More specifically, a user has to fix a complete lattice $\mathcal{L}$ first. Then, she has to choose two negators $\mathcal{N}_c$ and $\mathcal{N}_n$, which will be used to define consistency and the semantics of negation as failure respectively. Further, two t-norms

[5] As usual, $\backslash$ denotes set difference.

$\mathcal{T}_c$ and $\mathcal{T}_a$ need to be fixed, respectively used for defining consistency and applicability of rules. Also an implicator $\mathcal{I}$ is needed to obtain the degree of satisfaction of a rule. Finally, an aggregator $\mathcal{A}$ is needed that combines all the degrees of satisfaction of rules into a single truth value denoting the degree in which a fuzzy interpretation is a fuzzy model. For the rest of this paper, we assume, without loss of generality, that the above choices have been made, and we will not repeat them everytime in the definitions, but just use them.

The first classical notions that need to be tackled are containment and consistency. In ASP a literal l is either true or false; and thus it is either contained in an interpretation or not. When both l and $\neg l$ are contained in an interpretation, it is said to be inconsistent. In a fuzzy context, a literal l can be a bit true, and both l and $\neg l$ can be a bit true in a consistent way, making a modified notion of consistency necessary.

Definition 1. *Let P be a program. A **fuzzy interpretation** I for P is a fuzzy set in Lit_P, i.e. a $I : Lit_P \mapsto \mathcal{L}$ mapping. I is called **x-consistent**, $x \in \mathcal{L}$, iff*

$$\mathcal{N}_c(\sup_{a \in \mathcal{B}_P} \mathcal{T}_c(I(a), I(\neg a))) \geq_{\mathcal{L}} x \ .$$

Intuitively, the definition of x-consistency allows a user to choose the point where the degree of containment of both l and $\neg l$ in a fuzzy interpretation I, makes that interpretation inconsistent. The classical notion of an interpretation emerges from the above definition for the lattice $\mathcal{L} = \{0_{\mathcal{L}}, 1_{\mathcal{L}}\}$. In this particular case, an interpretation I is called $1_{\mathcal{L}}$-consistent iff there does not exist an a in $\mathcal{B}_P$ such that both $I(a) = 1_{\mathcal{L}}$ and $I(\neg a) = 1_{\mathcal{L}}$.

As fuzzy interpretations only assign truth values to ordinary literals explicitly, we need a mechanism to retrieve truth values for naf-literals. While complementary literals l and $\neg l$ are only weakly related to each other using $\mathcal{N}_c$, $\mathcal{T}_c$, and a certain x-consistency boundary, naf-literals l and *not* l need a tighter connection since, intuitively, a naf-literal *not* l can only be true to the degree that the underlying ordinary literal l is false, and vice versa. Hence, we use $\mathcal{N}_n$ to extend a fuzzy interpretation I to cover naf-literals by defining $I(\mathit{not}\ l) = \mathcal{N}_n(I(l))$ for each $l \in Lit_P$.

Having fuzzy interpretations and x-consistency, we need to redefine the satisfaction of rules. While a rule in ASP is either satisfied or not, in a more flexible setting we should allow a rule to be partially (to a certain degree) satisfied. Further, each rule does not have to be satisfied to the same degree, which is, e.g., useful in applications having preferences among rules. To obtain these degrees, we use $\mathcal{T}_a$ and $\mathcal{I}$ to induce, for a fuzzy interpretation I, a satisfaction function $I_{\models}$ that assigns a truth value to the bodies of rules and to the rules themselves. Later on, this satisfaction function will be used, in combination with the aggregator $\mathcal{A}$, to obtain the degree in which a fuzzy interpretation is a model of a program.

Definition 2. *Let P be a program and let I be a fuzzy interpretation. The induced satisfaction function $I_{\models} : 2^{Elit_P} \cup P \mapsto \mathcal{L}$ is defined by*

$$\begin{aligned} I_{\models}(\emptyset) &= 1_{\mathcal{L}} \\ I_{\models}(\{l\} \cup \beta) &= \mathcal{T}_a(I(l), I_{\models}(\beta)) \\ I_{\models}(\ \leftarrow \beta) &= \mathcal{I}(I_{\models}(\beta), 0_{\mathcal{L}}) \\ I_{\models}(l \leftarrow \beta) &= \mathcal{I}(I_{\models}(\beta), I(l)) \end{aligned}$$

Note that $I_{\models}(\{l\}) = I(l)$ and $I_{\models}(\mathit{not}\ l) = \mathcal{N}_n(I(l))$. Intuitively, $I_{\models}(s)$, with $s \in P$, defines to which degree a rule s is satisfied taking into account the truth assignments of the head and body of s in I. To define a fuzzy model, the different $I_{\models}(s)$, with $s \in P$, need to be accumulated in some way. The user defined aggregator $\mathcal{A}$, which takes as input a program and a satisfaction function, will accomplish this job and result in a truth value denoting the degree in which the fuzzy interpretation I is a model of P. However, we demand that an aggregator is increasing whenever the degrees of satisfaction of the rules increase.

Definition 3. *Let P be a program and let I be an x-consistent fuzzy interpretation. Then, I is an x-consistent **fuzzy y-model** of P, $y \in \mathcal{L}$, iff $\mathcal{A}(P, I_{\models}) \geq y$.*

Example 4. Consider the lattice $\mathcal{L} = [0, 1]$ and the program

$$r_1 : a \leftarrow \mathit{not}\ b \qquad r_2 : b \leftarrow \mathit{not}\ a \qquad r_3 : c \leftarrow a$$

and consider the fuzzy interpretations[6] $K = \{(a, 0.9), (b, 0.3), (c, 0.2)\}$ and $L = \{(a, 0.4), (b, 0.7), (c, 0.8)\}$. Both of these fuzzy interpretations are 1-consistent, independently of the choices for $\mathcal{N}_c$ and $\mathcal{T}_c$. For negation as failure, we use the negator $\mathcal{N}_s$. To compute the satisfaction of the rules, we use the implicator $\mathcal{I}_{\mathcal{S}_M}$. Finally, as an aggregator we use $\mathcal{A}(P, I_{\models}) = \inf\{I_{\models}(s) \mid s \in P\}$, i.e. the weakest rule dominates the solution.

We have $K_{\models}(r_1) = max(1 - K(\mathit{not}\ b), K(a)) = max(1 - \mathcal{N}_s(K(b)), K(a)) = max(1 - (1 - 0.3), 0.9) = 0.9$. Similarly, $K_{\models}(r_2) = max(1 - (1 - 0.9), 0.3) = 0.9$ and $K_{\models}(r_3)$ = $max(1 - 0.9, 0.2)$ = 0.2. As a result, K is a 1-consistent 0.2-model of P. On the other hand, one can verify that $L_{\models}(r_1) = L_{\models}(r_2) = 0.7$ and $L_{\models}(r_3) = 0.8$, yielding that L is a 1-consistent 0.7-model of P.

The above definitions are conservative extensions of classical principles, i.e. the classical definitions are special cases of the ones presented here. Hence it is not surprising that the extensions suffer the same difficulties when used to define a fuzzy answer set semantics. For instance, both $I = \{(a, 0_{\mathcal{L}}), (b, 0_{\mathcal{L}})\}$ and $J = \{(a, 1_{\mathcal{L}}), (b, 1_{\mathcal{L}})\}$ are "perfect" fuzzy interpretations of the program $\{a \leftarrow b\ ;\ b \leftarrow a\}$ as they both satisfy all rules to a maximal degree $1_{\mathcal{L}}$. In traditional ASP, the set $\{a, b\}$ is called "unfounded"[23] and answer sets should be free of such sets. This is achieved by imposing a minimality requirement.

Definition 4. *Let P be a program. An x-consistent y-model M is an x-consistent **minimal fuzzy y-model** iff M is $\prec_{\mathcal{L}}$ minimal among all x-consistent fuzzy y-models of P.*

Applied to the examples I and J above, this results in $I \prec_{\mathcal{L}} J$, yielding that I is the single $1_{\mathcal{L}}$-consistent minimal fuzzy $1_{\mathcal{L}}$-model of the two rules.

Example 5. Reconsider the program and the choices for logical operators from Example 4. One can check that the fuzzy interpretations $M = \{(a, 0.9), (b, 0.8), (c, 1)\}$

[6] As usual, a fuzzy set I in Lit_P is denoted as $\{(l, x) \mid I(l) = x \wedge l \in Lit_P\}$, omitting the literals $(l, 0_{\mathcal{L}})$.

and $N = \{(a, 0.9), (b, 0.2), (c, 1)\}$ are both 1-consistent 0.9-models of P. However, one can verify that $N \prec_{\mathcal{L}} M$, which fits our intuition as the degree in which b is assumed true is overestimated in M. Still, N is not minimal, as one can verify that $S = \{(a, 0.9), (c, 0.9)\}$ is $\prec_{\mathcal{L}}$-minimal[7], i.e. a 1-consistent minimal fuzzy 0.9-model of P.

While the above minimization process is necessary, it does not yet suffice to prevent unwanted models, as witnessed by the following example.

Example 6. Consider the program

$$r_1 : a \leftarrow \qquad\qquad r_2 : b \leftarrow a, \mathit{not}\ c$$

and the fuzzy interpretations $K = \{(a, 0.9), (b, 0.9)\}$ and $L = \{(a, 0.9), (c, 0.9)\}$. We make the same choice for the logical operators as in Example 4. To evaluate the body of r_2 we use $\mathcal{T}_a = \mathcal{T}_{\mathrm{M}}$. One can verify that K and L are both minimal fuzzy 0.9-models. However, intuitively L is not acceptable as a good solution as there is no support for accepting c at degree 0.9, i.e. there is no applicable rule with c in the head.

In traditional ASP, the above problem is solved by taking the GL-reduct which will remove, for $I = \{a, c\}$, the rule r_2 from the reduct P^I, because r_2 is not applicable due to the *not* c literal in its body. Now, the minimal model of this reduct does not equal I, hence, it is rejected as an answer set. Note that the removal of a rule does not mean that this rule does not have to be satisfied anymore. On the contrary, in the example above, rule r_2 is removed because it is not applicable under interpretation I, hence it is satisfied by default, independently of the truth value of b.

Note that in traditional ASP, there are two possible scenarios for a model I to satisfy a rule of the form $l \leftarrow \beta$. Either it is applicable ($I \models \beta$), hence l must assume the truth value $1_{\mathcal{L}}$ to satisfy the rule, or it is unapplicable ($I \not\models \beta$), hence the rule is satisfied by default and l can assume any truth value from the lattice $\mathcal{L} = \{0_{\mathcal{L}}, 1_{\mathcal{L}}\}$. In the first case, the truth value of l is fully determined by the rule, while in the latter case, the rule does not impose any restrictions on the truth value of l, hence taking it into account does not influence the result and we can remove the rule.

In FASP, such a removal strategy for naf-literals is not feasible as such literals may be true only to a certain degree, making the bodies of some rules applicable to a certain degree, which requires that they also need to be applied to a certain degree. Hence, as opposed to either fully determining the truth value of the head of a rule (full information), or leaving it completely arbitrary (no information), in FASP a rule may also carry *some* information that delimits the set of possible truth values that can be assumed by the head.

Thus, we define for each rule in the program a subset $Y \subseteq \mathcal{L}$ such that none of the values in Y lowers the degree of satisfaction of the rule. Next, for a literal $l \in Lit_P$, we consider these sets Y for each rule of the form $l \leftarrow \beta$. By taking the intersection of these sets, we obtain a range of truth values. Choosing an alternative truth value for l within this range does not lower the degrees of satisfaction of the rules with l in the head. However, interpretations that choose the lower values in the range are called better supported.

[7] Note that when another implicator is chosen, S not necessarily remains a minimal fuzzy 0.9-model of P. E.g., using $\mathcal{I}_{\mathcal{T}_M}$ would make S only a fuzzy 0-model.

Definition 5. *Let P be a program and let I be a fuzzy interpretation. The* ***supportedness function*** *I_s associated with I is defined by*

$$I_s(l) = \bigcap_{\{l\} \leftarrow \beta \in P} \{y \in \mathcal{L} \mid \mathcal{I}(I_{\models}(\beta), y) \geq_{\mathcal{L}} I_{\models}(\{l\} \leftarrow \beta)\} \ ,$$

for each $l \in Lit_P$, where, by definition, $\bigcap \emptyset = \{0_{\mathcal{L}}\}$. A minimal x-consistent fuzzy y-model of P is called an x-consistent ***fuzzy y-answer set*** *iff we have for each $l \in Lit_P$ that $I(l) = \inf(I_s(l))$.*

Example 7. Reconsider Example 6. Clearly, $L_s(c) = \{0\}$, yielding that L is not a fuzzy 0.9-answer set of P. On the other hand, one can verify that $K_s(a) = K_s(b) = [0.9, 1]$ and $K_s(c) = \{0\}$, implying that K is a fuzzy 0.9-answer set of P.

Proposition 1. *For a simple program P, I is a minimal x-consistent fuzzy y-model of P iff I is an x-consistent fuzzy y-answer set of P.*

Example 8. Reconsider Example 1 from the introduction. We are interested to know to what degrees we have to practice the various sports such that an acceptable degree, e.g. 0.7, of full-body exercise is obtained. Since our main concern is a satisfactory degree of full-body exercise, we will use an aggregator that gives more importance to the constraint rule r_7. An appropriate choice could be an aggregator that only takes r_7 into account. In this case a fuzzy interpretation is a model to the degree to which it satisfies r_7. Of course we also require the model to be minimal and supported, which is where other rules come into play.

Further, we will also use $\mathcal{T}_a = \mathcal{T}_{\mathrm{M}}$ to evaluate the body of rules, and the implicator $\mathcal{I} = \mathcal{I}_{\mathcal{T}_W}$ to evaluate the satisfaction of the rules. A fuzzy 0.7-answer set K for the above program must at least satisfy $K_{\models}(r_7) = 0.7$. This yields that

$$min(1 - K(\textit{not full\_body\_exercise}) + 0, 1) = 0.7 \ ,$$

which implies that $K(\textit{not full\_body\_exercise}) = 0.3$, and thus, using $\mathcal{N}_s$ for negation as failure, that $K(\textit{full\_body\_exercise}) = 0.7$. To have support for the literal, i.e. $\inf(K_s(\textit{full\_body\_exercise})) = 0.7$, one of the two rules r_5 or r_6 have to be made applicable to a certain degree, in turn implying that some of the four sports will have to be exercised in a higher degree than others to achieve that sufficient degree of applicability of r_5 or r_6[8]. One can verify that

$$K = \{(\textit{lift\_weights}, 0.8), (\textit{swim}, 0.2), (\textit{run}, 0.7), (\textit{play\_ball}, 0.3),$$
$$(\textit{full\_body\_exercise}, 0.7)\} \ ,$$

is a fuzzy 0.7-answer set of the above program.

Intuitively, this solution is acceptable as it describes a configuration where two sports, which are together in rule r_5, are assigned a higher degree than there complementary

[8] Note that this example also illustrates how the proposed framework can be used to do fuzzy diagnostic reasoning. The constraint r_7 can be seen as an encoding of the observations, r_5 and r_6 represent the system description, while r_1, r_2, r_3 and r_4 provide the explanations.

variants, and due to this choice we have support for full-body exercise up to a degree of 0.7.

On the other hand, one can check that for the fuzzy interpretation

$$L = \{(\mathit{lift\_weights}, 0.8), (\mathit{swim}, 0.2), (\mathit{run}, 0.3), (\mathit{play\_ball}, 0.7),$$

$$(\mathit{full\_body\_exercise}, 0.7)\} \ ,$$

it turns out that $L_s(\mathit{full\_body\_exercise}) = [0.3, 1] \cap [0.2, 1] = [0.3, 1]$. Hence, L is not a fuzzy 0.7-answer set of the program, fitting our intuition.

4 Retrieving Classical Answer Sets

The FASP framework presented in the previous section turns out to be a proper generalisation of the classical answer set programming paradigm with the notions of fuzzy logic. First of all, ASP can be retrieved as a special case of FASP by choosing the truth lattice $\mathcal{L} = \{0_\mathcal{L}, 1_\mathcal{L}\}$.

Proposition 2. *Consider a program P and let $\mathcal{L}$ be the lattice $\{0_\mathcal{L}, 1_\mathcal{L}\}$. Furthermore let the aggregator $\mathcal{A}$ be such that $\mathcal{A}(P, I_\models) = 1_\mathcal{L}$ iff $I_\models(s) = 1_\mathcal{L}$ for every rule $s \in P$. An interpretation M is an answer set of P iff the fuzzy interpretation f_M, with $f_M(l) = 1_\mathcal{L}$ if $l \in M$ and $f_M(l) = 0_\mathcal{L}$ otherwise, is a $1_\mathcal{L}$-consistent fuzzy $1_\mathcal{L}$-answer set of P.*

Note that $1_\mathcal{L}$-consistency is needed to forbid (classical) contradictions, and the restriction to fuzzy $1_\mathcal{L}$-answer sets is mandated by the need to classically satisfy all rules and have the foundedness property of answer sets.

Example 9. Reconsider the program from Example 3. The empty set is not a $1_\mathcal{L}$-model of this program as it satisfies neither of the rules to degree $1_\mathcal{L}$. $K = \{(a, 1_\mathcal{L})\}$, $L = \{(b, 1_\mathcal{L})\}$, and $M = \{(a, 1_\mathcal{L}), (b, 1_\mathcal{L})\}$ are $1_\mathcal{L}$-models, but the latter is obviously not minimal. One can verify that $K_s(a) = \{1_\mathcal{L}\}$ and $K_s(b) = \{0_\mathcal{L}, 1_\mathcal{L}\}$, and similarly that $L_s(a) = \{0_\mathcal{L}, 1_\mathcal{L}\}$ and $L_s(b) = \{1_\mathcal{L}\}$, in other words both K and L are 1-consistent fuzzy 1-answer sets.

In the proposition above, no choice for $\mathcal{N}_c$, $\mathcal{N}_n$, $\mathcal{T}_c$, $\mathcal{T}_a$, and $\mathcal{I}$ is specified as all negators, t-norms and implicators on $\{0_\mathcal{L}, 1_\mathcal{L}\}$ coincide. However, when we allow for intermediate truth values, a choice for logical operators opens up. Below we argue that certain choices are more "answer set behaved" than others.

Classical answer sets cannot contain both a and $\neg a$. If one wants to preserve this behaviour for fuzzy answer sets, i.e. such that a $1_\mathcal{L}$-consistent fuzzy answer set can not contain a and $\neg a$ simultaneously, not even to some degree, $\mathcal{T}_c$ should be chosen with care. E.g., on $\mathcal{L} = [0, 1]$, take $\mathcal{T}_c = \mathcal{T}_\mathrm{W}$ and consider the fuzzy interpretation $I = \{(a, 0.4), (\neg a, 0.4)\}$. Then, $\mathcal{T}_c(I(a), I(\neg a)) = \mathit{max}(0.4 + 0.4 - 1, 0) = 0$. For this t-norm it holds, in general, that $\mathcal{T}_c(I(a), I(\neg a)) = 0$ iff $I(a) + I(\neg a) \leq 1$, which certainly does not correspond to a classical answer set semantics. However, there exist some stronger versions for $\mathcal{T}_c$ that do not suffer from this problem, i.e. for

which $\mathcal{T}_c(I(a), I(\neg a)) = 0$ iff $I(a) = 0$ or $I(\neg a) = 0$. Both $\mathcal{T}_M$ and $\mathcal{T}_P$ are such t-norms, and can be used to retrieve fuzzy answer sets with a classical ASP consistency notion.

Next, we consider the possible choices for the implicator. By definition, an implicator satisfies $\mathcal{I}(0_\mathcal{L}, 0_\mathcal{L}) = \mathcal{I}(0_\mathcal{L}, 1_\mathcal{L}) = \mathcal{I}(1_\mathcal{L}, 1_\mathcal{L}) = 1_\mathcal{L}$ and $\mathcal{I}(1_\mathcal{L}, 0_\mathcal{L}) = 0_\mathcal{L}$, which implies that any choice for $\mathcal{I}$ is sufficient to retrieve classical answer sets from the $1_\mathcal{L}$-consistent fuzzy $1_\mathcal{L}$-answer sets. However, when intermediate truth values are considered, certain choices for $\mathcal{I}$ are more answer set alike, as witnessed by the following example.

Example 10. Reconsider the program from Example 3 and let $J = \{(a, 0.6), (b, 0.4)\}$ be a fuzzy interpretation. When using $\mathcal{I}_{S_M}$, we get $J_\models(r_1) = max(1 - 0.6, 0.6) = 0.6$ and $J_\models(r_2) = max(1 - 0.4, 0.4) = 0.6$, yielding that J will be at most a 0.6-answer set. However, in a classical answer set context this looks a bit unintuitive as the heads of both rules are satisfied to exactly the same degrees as to which their bodies are applicable, and thus intuitively the rules should be totally satisfied. Applying $\mathcal{I}_{T_M}$ on the program yields $J_\models(r_1) = J_\models(r_2) = 1$, which fits that intuition.

The implicator $\mathcal{I}_{T_M}$ belongs to the class of R-implicators, for which, in general, it holds that $\mathcal{I}(x, y) = 1_\mathcal{L}$ whenever $x \leq_\mathcal{L} y$. All R-implicators in Example 2 satisfy the residuation principle or adjoint condition, i.e. $\mathcal{T}(x, y) \leq_\mathcal{L} z$ iff $x \leq \mathcal{I}_\mathcal{T}(y, z)$ for all x, y, and z in $\mathcal{L}$. Note that for such implicators the degree of satisfaction of a rule $l \leftarrow \beta$ does not go down as long as the truth value of the head is greater than or equal to $\mathcal{T}(I_\models(l \leftarrow \beta), I_\models(\beta))$. In other words, in this case we obtain a more direct expression for y to be used in Definition 5.

Finally, to preserve classical answer set semantics, an interpretation should be said to satisfy a program to degree $1_\mathcal{L}$ iff it satisfies all rules of the program to degree $1_\mathcal{L}$. The aggregator $\mathcal{A}(P, I_\models) = \inf\{I_\models(s) \mid s \in P\}$ we introduced before satisfies this condition and can be used to retrieve classical answer sets.

5 Related Work

Logic programming in the presence of uncertainty or imprecision has received a considerable amount of attention (see e.g. [1, 9] for overviews). It is however interesting to observe that the well known existing frameworks, including those that consider fuzzy interpretations, hold on to two valued concepts of rule satisfaction, model, etc. in the sense that a fuzzy interpretation satisfies a rule or not, it is a model or not, etc. This clearly sets them apart from the approach introduced in this paper.

The enrichment of ASP with concepts from fuzzy logic as well as from the closely related possibilistic logic [11] has been studied from various angles already. In annotated answer set programming [22], a rule is of the form $l\{f(z_1, z_2, \ldots, z_n)\} \leftarrow l_1\{z_1\}, l_2\{z_2\}, \ldots, l_n\{z_n\}$ where $l, l_1, l_2, \ldots, l_n$ denote literals and $z_1, z_2, \ldots, z_n$ are annotation terms that can be understood as truth degrees. Such a rule asserts that l is true at least to degree $f(z_1, z_2, \ldots, z_n)$ whenever l_i is true at least to degree z_i (for $i = 1 \ldots n$). Because of this early revertment to the two valued case, no fuzzy logical operators are needed in this approach.

The approach in [17] adheres closer to ours. A rule of the form $\alpha \overset{z}{\leftarrow} \beta$ is said to be satisfied by the interpretation iff (in our notation) $I_{\models}(\alpha) \geq_{\mathcal{L}} \mathcal{T}(I_{\models}(\beta), z)$. The residuation principle reveals a clear connection with our approach when committing to an R-implicator $\mathcal{I}_{\mathcal{T}}$: namely that I satisfies the rule $\alpha \overset{z}{\leftarrow} \beta$ according to [17] iff I satisfies this rule at least to degree z in our approach. This is also in accordance with [8] where the use of adjoint pairs $(\mathcal{T}, \mathcal{I}_{\mathcal{T}})$ is strongly advocated to preserve important theoretical results. Being able to impose specific satisfaction requirements for individual rules is in general an interesting feature, e.g., when rules and facts originate from different knowledge bases that are not all equally trusted. Note that this can be easily incorporated in our approach by choosing a suitable aggregator $\mathcal{A}$.

In a similar way, [25] can be seen as a special case of the FASP framework presented in this paper as [25] commits itself, with limited motivation, to very specific choices for the user-selectable operators on the lattice $[0, 1]$. Some of these choices are at least questionable. E.g., using the Gödel negator $\mathcal{N}_g$ for interpreting naf yields that a rule $a \leftarrow \mathit{not}\ b$ will not be applied in any way although b is only true to a small degree, e.g. 0.1. Using the standard negator $\mathcal{N}_s$, as we do in our examples, this would yield a rule that is applicable to a degree 0.9, and, if a rule satisfaction of at least 0.8 is wanted with e.g. $\mathcal{I}_{S_M}$, we have $max(0.1, y) = 0.8$, which implies that a will be derived at degree 0.8 in a fuzzy answer set.

A possibilistic definite logic program [18] consists of rules annotated with certainty degrees. These degrees are used to establish a possibility distribution on the universe of atom sets, from which a possibilistic model is derived. The authors choose implicitly for the Gödel negator, as they first compute the classical answer sets, and afterwards compute, for an answer set S, the possibility to which each literal l is contained in S.

6 Conclusions and Future Research

There are many ways to increase the expressive power of answer set programming (ASP) by enriching it with mechanisms to deal with imprecision and uncertainty. In this paper we presented a general and elegant fuzzification of ASP, called fuzzy answer set programming (FASP). The generality is reflected in a high configurability by the user, which allows the system to be tailored to the application at hand. Among other things, the ability to choose an aggregator allows for future extensions of the semantics, e.g. incorporating rule preferences on fuzzy programs. The elegance is due to a close adherence to both the fuzzy logic and the answer set programming paradigm: as opposed to other approaches, FASP does not revert soon to the two valued case but instead allows to compute the actual degree to which a fuzzy interpretation is an answer set. Furthermore we have shown that FASP extends the traditional answer set semantics.

Clearly, there are a lot of topics that still need to be investigated, e.g. a fixpoint characterization, the complexity of the semantics, the use of disjunction etc., all parametrized by the choice of the (lattice) operations. In addition, we intend to explore natural FASP applications areas such as “web of trust”, diagnosis, and decisision support.

References

[1] T. Alsinet, L. Godo, and S. Sandri. Two formalisms of extended possibilistic logic programming with context-dependent fuzzy unification: a comparative description. *Electronic Notes in Theoretical Computer Science*, 66(5), 2002.

[2] M. Balduccini and M. Gelfond. Logic programs with consistency-restoring rules. In *Proceedings of the International Symposium on Logical Formalization of Commonsense Reasoning*, AAAI 2003 Spring Symposium Series, 2003.

[3] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[4] G. Birkhoff. Lattice theory. *American Mathematical Society Colloquium Publications*, 25(3), 1967.

[5] G. Brewka. Logic programming with ordered disjunction. In *Proceedings of the 18th National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 100–105. AAAI Press, July 2002.

[6] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, April 1999.

[7] F. Buccafurri, N. Leone, and P. Rullo. Strong and weak constraints in disjunctive datalog. In *Proc. of the 4th Intl. Conf. on Logic Programming (LPNMR '97)*, pages 2–17, 1997.

[8] C. Damasio, J. Medina, and M. Ojeda-Aciego. Sorted multi-adjoint logic programs: termination results and applications. *Journal of Applied Logic*, page To appear, 2006.

[9] C. V. Damasio and L. M. Pereira. Sorted monotonic logic programs and their embedding. In *Proc. of the 10th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-04)*, pages 807–814, 2004.

[10] M. De Vos and D. Vermeir. On the Role of Negation in Choice Logic Programs. In *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *LNAI*, pages 236–246. Springer, 1999.

[11] D. Dubois and H. Prade. Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144(1):3–23, 2004.

[12] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. The diagnosis frontend of the dlv system. *AI Communications*, 12(1-2):99–111, 1999.

[13] D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217. Morgan Kaufmann, 1991.

[14] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, Seattle, Washington, August 1988. The MIT Press.

[15] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.

[16] D. N. Juergen Dix, Ugur Kuter. Planning in answer set programming using ordered task decomposition. In *Proc. of the 27th German Annual Conf. on Artificial Intelligence (KI '03)*, volume 2821 of *LNAI*, pages 490–504. Springer, 2003.

[17] C. Mateis. Extending disjunctive logic programming by t-norms. In *Proc. of the 5th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR99)*, volume 1730 of *LNAI*, pages 290–304. Springer, 1999.

[18] P. Nicolas, L. Garcia, and I. Stéphan. Possibilistic stable models. In *Proc. of the 19th Intl. Joint Conf. on Artificial Intelligence*, pages 248–253, 2005.

[19] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry. An a-prolog decision support system for the space shuttle. In *Third International Symposium on Practical Aspects of Declarative Languages*, volume 1990 of *LNCS*, pages 169–183. Springer, 2001.

[20] V. Novák, I. Perfilieva, and J. Močkoř. *Mathematical Principles of Fuzzy Logic*. Kluwer Academic Publishers, 1999.

[21] T. Soininen and I. Niemelä. Developing a declarative rule language for applications in product configuration. In *Proc. of the 1st Intl. Workshop on Practical Aspects of Declarative Languages (PADL '99)*, volume 1551 of *LNCS*, pages 305–319. Springer, 1999.

[22] U. Straccia. Annotated answer set programming. In *Proc. of the 11th Intl. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU-06)*, 2006.

[23] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, 38(3):620–650, 1991.

[24] D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. *Theory and Practice of Logic Programming*, 6(1-2):107–167, 2006.

[25] G. Wagner. A logical reconstruction of fuzzy inference in databases and logic programs. In *Proceedings of the International Fuzzy Set Association World Congress (IFSA'97)*, 1997.

[26] L. Zadeh. Fuzzy logic and approximate reasoning. *Synthese 30*, pages 407–428, 1975.

Reasoning About an Agent Based on Its Revision History with Missing Inputs

Alexander Nittka

Universität Leipzig, Institut für Informatik, Augustusplatz 10-11,
04109 Leipzig, Germany
nittka@informatik.uni-leipzig.de

Abstract. In this paper, we extend on work presented in [1] where we proposed a method for reconstructing an agent's initial epistemic state from an observation on its belief revision behaviour. There, we assumed that the observation is complete in the sense that *all* revision inputs during the time of observation were known to us. Here, we drop this assumption and investigate the case where there are intermediate inputs we have no information about. The focus will be on determining the core belief of the agent — a belief the agent commits to at all times.

1 Introduction

The problem of belief revision, i.e., of how an agent should modify *its* beliefs about the world given some new information which possibly contradicts its current beliefs, is by now a well-established research area in AI [2]. Reasoning about *other* agents is another important capability necessary for successful agents. This requires investigating them from a third person perspective, rather than a first person one as is done the traditional work in belief revision. In [1], we proposed one possible approach to the area.

The general setting is that we are given observations of another agent's belief revision behaviour, containing information about what the agent received as revision inputs, what it believed and did not believe after receiving them. We are interested in constructing a model of the observed agent in order to be able to draw conclusions about what it believed before receiving the observed inputs, what it might believe after receiving a further input and what it believed apart from what the observation tells us.

In [1], we assumed to be given an observation that is complete in the sense that all revision inputs received by the agent are known — a very strong assumption. In the present paper we weaken it to a certain extent by allowing intermediate inputs. In other words, the given observation is incomplete in the sense that the agent may have received further revision inputs but we have no information about them, neither what the input was nor what was and was not believed upon receiving it. The focus of the paper is on results concerning the core belief of the agent, the belief it commits to at all times.

We work in a propositional setting. L will always be some finitely generated propositional language based on variables from $\{a, b, p, q, \ldots\}$, the constants $\bot$

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 373–385, 2006.

and $\top$ for contradiction and tautology, and the usual connectives. $\blacktriangle, \alpha, \beta, \varphi, \theta$, etc. stand for arbitrary elements of such a language, that is, propositional formulae, D for a (finite) set of formulae, and σ and ρ for sequences of formulae. $\sigma_1 \cdot \sigma_2$ and $\sigma \cdot \varphi$ denote the concatenation of two sequences and the concatenation of a sequence with a single formula. $\vdash$ denotes the classical entailment relation, Cn the closure under classical consequence, and $m \models \varphi$ that an assignment m satisfies a formula φ.

o will be used to denote observations made on the belief revision behaviour of an agent $\mathcal{A}$. Formally, an observation $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$ is a (possibly empty) sequence of triples $(\varphi_i, \theta_i, D_i)$ with the following interpretation. After $\mathcal{A}$ received the revision inputs φ_1 up to φ_i (in that order), starting in its initial epistemic state, it believed at least θ_i but did not believe any element of D_i. We remark that this notation of an observation deviates from that in [1]. Extended to the case allowing negative information (what the agent did not believe) which was not treated there, an observation would have been a triple of sequences. $o_1 \cdot o_2$ denotes the concatenation of two observations.

We will first recall the agent model we use as well as main results from [1] that extend to the case of negative information, although this has only been hinted at in [3]. We will then turn to intermediate inputs, give a more formal definition for them and illustrate their impact on explanations of an observation. In order to provide more specific results, we will then assume that the number and positions of the intermediate inputs in the observation are given and fixed.

1.1 The Agent Model

We assume a particular belief revision framework for iterated non-prioritised revision that has been studied in [4]. According to this framework an agent's epistemic state is defined by two components: *(i)* a sequence ρ of sentences representing the sequence of revision inputs the agent has received thus far, and *(ii)* a single sentence $\blacktriangle$ standing for the agent's set of *core* beliefs, which intuitively are those beliefs of the agent it considers "untouchable". We denote the agent's epistemic state by $[\rho, \blacktriangle]$. The definitions of the revision operator and the belief set in our agent model are as follows.

Definition 1. *Given an epistemic state* $[\rho, \blacktriangle]$ *and a formula* λ*, the revision operator* $*$ *is defined by*

$$[\rho, \blacktriangle] * \lambda = [\rho \cdot \lambda, \blacktriangle]$$

Definition 2. *The set of beliefs* $Bel([\rho, \blacktriangle])$ *in the epistemic state* $[\rho, \blacktriangle]$ *is* $Bel([\rho, \blacktriangle]) = Cn(f(\rho \cdot \blacktriangle))$*, where*

$$f(\beta_k, \ldots, \beta_1) = \begin{cases} \beta_1 & \text{if } k = 1 \\ \beta_k \wedge f(\beta_{k-1}, \ldots, \beta_1) & \text{if } k > 1 \text{ and } \beta_k \wedge f(\beta_{k-1}, \ldots, \beta_1) \nvdash \bot \\ f(\beta_{k-1}, \ldots, \beta_1) & \text{otherwise} \end{cases}$$

The epistemic state of $\mathcal{A}$ is revised by simply appending the new formula to the sequence of formulae received so far. $f(\sigma)$ for a sequence of formulae σ is almost

exactly the "linear base-revision operation" of [5], the only difference being that here the first formula is accepted even if it is inconsistent. This ensures the correct treatment of our notion of core belief — a belief an agent always commits to. $\mathcal{A}$'s belief set is calculated by starting with the core belief $\blacktriangle$ and then consistently adding the received formulae in reversed order, simply leaving out those which cannot be added consistently closing the result under consequence.

1.2 Problem Definition

The intuitive interpretation of an observation $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$ on the belief revision behaviour of an agent $\mathcal{A}$ whose initial epistemic state is $[\rho, \blacktriangle]$ can be formalised by:

$$\begin{array}{l} \text{for all } i \text{ such that } 1 \leq i \leq n: \\ f(\rho \cdot (\varphi_1, \ldots, \varphi_i) \cdot \blacktriangle) \vdash \theta_i \quad \text{and} \\ \forall \delta \in D_i : f(\rho \cdot (\varphi_1, \ldots, \varphi_i) \cdot \blacktriangle) \nvdash \delta \end{array} \tag{1}$$

Definition 3. *Let* $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$. *Then* $[\rho, \blacktriangle]$ explains o *(or* is an explanation for o*) iff* $\blacktriangle \nvdash \bot$ *and (1) above holds. We say* $\blacktriangle$ *is an* o-acceptable core *iff* $[\rho, \blacktriangle]$ *explains* o *for* some ρ.

Here we slightly deviate from the original definition in [1] where an inconsistent core might have been o-acceptable, as well. We still allow an agent to have an inconsistent core belief, but we do not call such a core an explanation. Note that $\blacktriangle = \bot$ satisfies (1) iff $D_i = \emptyset$ for all i, i.e., whenever there is no information about what the agent did not believe after any revision step. Eliminating this possibility makes the results more coherent.

For a given observation o the task is to find (a best) initial epistemic state $[\rho, \blacktriangle]$ satisfying (1), i.e., explaining o. Having this initial state of the agent will allow us to take justified guesses at what the agent believed at various points in time and might believe after receiving further revision inputs.

2 The Rational Explanation

In [1], we defined the *rational explanation* — an algorithm that calculates a best epistemic state $[\rho_R(o, \blacktriangle_\vee(o)), \blacktriangle_\vee(o)]$ for a given observation o — for the case where there is no negative information, i.e., $D_i = \emptyset$ for all i. This algorithm employs a method known as *rational closure* [6, 7]. $\rho_R(o, \blacktriangle)$ denotes its result, a sequence of formulae calculated from conditional beliefs — in our case constructed from the given observation o and a core belief $\blacktriangle$. In [3] we hinted that the results from [1] carry over to the general case, where we allow nonempty D_i. In order to achieve this, we use a generalisation of the original rational closure [8] that also handles information about which formulae should not be entailed. In addition, Algorithm 1 has been adapted to our notion that inconsistent cores are not o-acceptable.

The generalisation of a second important result from [1] is Proposition 1. It expresses that we can weaken two o-acceptable cores and can be sure that the

Algorithm 1. calculation of the rational explanation

Require: observation $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$
Output: the rational explanation for o
  $\blacktriangle \Leftarrow \top$
  repeat
    $\rho \Leftarrow \rho_R(o, \blacktriangle)$ $\quad \{\rho = (\alpha_m, \ldots, \alpha_0)\}$
    $\blacktriangle \Leftarrow \blacktriangle \wedge \alpha_m$
  until $\alpha_m \equiv \top$
Return $[\rho, \blacktriangle]$ if $\blacktriangle \not\equiv \bot$, "no explanation" otherwise

result will still be o-acceptable. This implies that there is a unique logically weakest o-acceptable core $\blacktriangle_\vee$ that is entailed by *every* o-acceptable core $\blacktriangle$. The rational explanation algorithm, which always terminates, has been shown to calculate $\blacktriangle_\vee$ and hence yield the desired result.

Proposition 1. *Let* $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$. *If* $\blacktriangle_1$ *and* $\blacktriangle_2$ *are* o*-acceptable then so is* $\blacktriangle_1 \vee \blacktriangle_2$.

The rational explanation assumes that the given observation is complete in the sense that all revision inputs are given, i.e., that there is no further input between φ_i and φ_{i+1}. In the current paper we want to investigate what happens if this assumption is dropped, i.e., if we allow for the possibility that $\mathcal{A}$ has received intermediate inputs.

3 Intermediate Inputs

3.1 Introductory Notes

An intermediate input is a revision input about which we have no further information. In terms of observations it can and will be represented by $\check{\phi} = \langle(\phi, \top, \emptyset)\rangle$ telling us that a revision input was received, but nothing about what is and is not believed afterwards. It makes a difference whether we use $\langle(\phi, \top, \emptyset)\rangle$ or $\langle(\phi, \phi, \emptyset)\rangle$ as the latter would express that the intermediate input was indeed accepted — but we cannot be sure of that. Note that ϕ is just a placeholder for some formula, as we do not know what the intermediate input actually is. Most of our propositions rely only on the *existence* of some intermediate inputs ϕ_i. In the most general setting we are given an observation $o = \langle(\varphi_1, \theta_1, D_1), \ldots, (\varphi_n, \theta_n, D_n)\rangle$ and as there might have been intermediate inputs, the complete observation would be $o' = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot o^m \cdot \check{\phi}_m \cdot o^{m+1}$, with $m + 1$ observations o^i such that $o^1 \cdot o^2 \cdot \ldots \cdot o^{m+1} = o$ and m unknown intermediate inputs ϕ_i. Note that an o^i might be empty, that is, successive intermediate inputs are allowed, and that the o^i are not fixed but only constrained. It should be clear that once the intermediate inputs and their positions are known, i.e., all ϕ_i are instantiated and the o^i fixed, we can simply use the rational explanation to calculate a possible initial epistemic state which then allows us to answer the questions we had in the original case where we did not consider intermediate inputs.

One might wonder whether intermediate inputs and core beliefs are interchangeable concepts, that we could replace one with the other while being able to explain the same observations. However, this is not the case, as the following examples will illustrate.

Example 1. (i) To explain $\langle(p, \neg p, \emptyset)\rangle$ core beliefs are needed, as otherwise p will be introduced into the belief set. (ii) To explain $\langle(p, q, \{\neg q\}), (p, \neg q, \{q\})\rangle$ we need an intermediate input, e.g., $p \rightarrow \neg q$, as otherwise $\mathcal{A}$'s change in mind concerning q cannot be accounted for. (iii) $\langle(p, \neg p, \{q\}), (\neg p, q, \emptyset)\rangle$ can only be explained with both an intermediate input and a core belief. (iv) $\langle(p, \top, \{p, \neg p\})\rangle$ cannot be explained at all with the current concepts, but this is beyond the scope of this paper.

We further note that the number and positions of intermediate inputs matter. The more intermediate inputs we allow, the weaker the necessary core belief may be. This is no surprise. The core belief does not only allow inputs to be blocked directly as is necessary for explaining observations like $\langle(p, \neg p, \emptyset)\rangle$, where the input p must be blocked so that $\neg p$ can be consistently believed. It also handles interaction between revision inputs. To explain $\langle(a, a, \emptyset), (b, \neg a, \emptyset)\rangle$ the core belief needs to be such that in the calculation of the belief set a is blocked after b has been received. A core belief entailing $b \rightarrow \neg a$ does that. However, the same effect can be achieved by an intermediate input $b \rightarrow \neg a$ between a and b and thus allows for a weaker core. And the more such interactions are dealt with by intermediate inputs, the weaker the core may be.

Example 2. Consider $o = \langle(a, a, \emptyset), (b, \neg a, \emptyset), (c, c, \emptyset), (d, \neg c, \emptyset)\rangle$. a and c are accepted by the agent, but after receiving the next input they are discarded. The weakest core explaining the observation o is $\blacktriangle_\vee(o) = (b \rightarrow \neg a) \wedge (d \rightarrow \neg c)$. Note that $\blacktriangle_\vee(\langle(a, a, \emptyset), (b, \neg a, \emptyset)\rangle) = b \rightarrow \neg a$ and $\blacktriangle_\vee(\langle(c, c, \emptyset), (d, \neg c, \emptyset)\rangle) = d \rightarrow \neg c$.

There are three possible positions for a single intermediate input. Between a and b, between b and c, and between c and d. If we assume the intermediate input $b \rightarrow \neg a$ between a and b, we get the minimal core $d \rightarrow \neg c$ for the modified observation. If we assume the intermediate input $d \rightarrow \neg c$ between c and d, we get the minimal core $b \rightarrow \neg a$. However, we will not find a single intermediate input such that $(b \rightarrow \neg a) \vee (d \rightarrow \neg c)$ let alone $\top$ will be an acceptable core. This follows from the monotonicity properties of the minimal core belief [1].

If we allow for two intermediate inputs we might assume $b \rightarrow \neg a$ between a and b and $d \rightarrow \neg c$ between c and d. In this case the core belief $\top$ is indeed an explanation.

Also, successive intermediate inputs cannot be joined into a single one. We postpone the proof of this claim, as it is easier with the results of the following section. In a sense, the rational explanation also calculates intermediate inputs — usually a non-trivial sequence of unobserved inputs the agent may have received before the observation started. Intuitively, if joining them was possible then a sequence containing a single sentence would always exist, which is not the case.

3.2 Intermediate Inputs at Fixed Positions

We have seen that varying the number and positions of the intermediate inputs has a great effect on the (minimal) core belief we can assign to the observed agent. From now on, we will fix the number and positions of the intermediate inputs in the observation — in order to give more specific results than just saying there is an effect. We assume we are given an observation $o = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot o^n \cdot \check{\phi}_n \cdot o^{n+1}$, with $n+1$ specific observations o^i of arbitrary but finite length. Note that o fixes the positions of n unknown intermediate inputs ϕ_i.

A first question that arises is whether an explanation for the observation o exists at all: Are there instantiations of ϕ_i such that there is an o-acceptable core? The naive way to approach this question is to go through all possible combinations of intermediate inputs and calculate the rational explanation. But can we do better? Indeed, we can give a necessary condition for the existence of an explanation that employs a more systematic and efficient method.

The idea is to use as intermediate inputs new propositional variables that do not occur in the known part of the observation. We will first show that this works for the case of a single intermediate input, i.e., $n = 1$, and then extend this result to an arbitrary number.

Proposition 2. *If* $[\rho, \blacktriangle]$ *explains* $o = o^1 \cdot (\phi, \top, \emptyset) \cdot o^2$ *and* x *is a propositional variable not appearing in* $\blacktriangle$, ϕ, ρ *or any formula in* o^1 *or* o^2 *then* $[\rho, \blacktriangle \wedge (x \to \phi)]$ *explains* $o' = o^1 \cdot (x, \top, \emptyset) \cdot o^2$.

In other words, if there is an intermediate input such that o has an explanation, then we can assume the intermediate input to be x and the observation so obtained still has an explanation. The contraposition yields that if there is no consistent explaining core when using a new variable as the intermediate input, then there cannot be *any* intermediate input that yields a consistent core.

The result is proved using an induction over the calculation of the belief set in a particular epistemic state of the agent. This has to be done for all epistemic states covered by the given observation. The basic idea is that the intermediate input x together with the additional core belief $x \to \phi$ behaves exactly like the intermediate input ϕ — the same revision inputs and elements from ρ are collected into the belief set. The only difference is that x or $\neg x$ might be believed in the case of the modified observation. But this cannot contradict the original observation as x does not appear there.

The result also implies that when assuming a *single* intermediate input, an extension of the language of the known part of the observation by *one* variable is enough for finding an explanation. The core belief given in the proposition does not satisfy that property, of course — there we assume the core to be $\blacktriangle \wedge (x \to \phi)$ and $\blacktriangle$ and ϕ might indeed contain further variables not appearing in the known part of o itself. However, the proposition just showed that an explanation exists which implies that the rational explanation will find one, as well. As the rational explanation uses only variables contained in the observation, i.e., x and the variables appearing in o^1 and o^2, one additional variable is indeed enough.

Now we can easily generalise this result to n intermediate inputs. Recall that $\check{\phi}$ denotes $\langle(\phi, \top, \emptyset)\rangle$.

Proposition 3. *If* $[\rho, \blacktriangle]$ *explains* $o = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot o^n \cdot \check{\phi}_n \cdot o^{n+1}$ *and* $x_1, \ldots, x_n$ *are* n *different propositional variables not appearing in* $\blacktriangle$, ρ, *any formula in* o^i, *or any* ϕ_i, *then* $\left[\rho, \blacktriangle \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i)\right]$ *is an explanation for the observation* $o' = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot o^n \cdot \check{x}_n \cdot o^{n+1}$.

Proof. We apply Proposition 2 n times, each time replacing one ϕ_i *by* x_i. □

Proposition 3 provides a powerful tool. We can apply the rational explanation construction to $o' = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot o^n \cdot \check{x}_n \cdot o^{n+1}$. This will give us the best explanation for o' including the weakest possible o'-acceptable core belief. If the rational explanation construction tells us that there is no explanation, we know that there are *no* intermediate inputs such that o can be explained. Note that the rational explanation will generally not return $\blacktriangle \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i)$ but some other (logically weaker) core belief. The proposition did not claim to give the weakest possible core but only some o'-acceptable one.

The additional variables might still be contained in the core calculated for o'. We will proceed to prove that all x_i can be eliminated from the core by modifying the intermediate inputs. This means there is a core belief whose language is restricted to that of the known part of the observation, i.e., variables appearing in some o^i. We will further show that there is a single weakest such core belief. In other words, analogous to the result from [1], there is a core belief $\blacktriangle_\vee$ that explains the observation for a certain instantiation of the intermediate inputs such that for *any* core $\blacktriangle'$ explaining the observation for some (other) intermediate inputs we have $\blacktriangle' \vdash \blacktriangle_\vee$.

The following proposition plays an essential role in eliminating the additional variables from the core. It is more general but in its application in the current context it formalises that it suffices to keep the part of the core that exclusively talks about the known part of the observation. The rest can be transferred to the intermediate inputs without effect on the belief set. Think of σ as the sequence of revision inputs from the observation *without* ρ, i.e., without the sequence from the agent's initial epistemic state. σ' corresponds to σ where the additional variables that we put instead of the unknown intermediate inputs are replaced. In fact, they are strengthened by the part of the original core that also talks about them. $\blacktriangle$ is such that it already entails all formulae from the language of the known part of the observation that the actual core entails. The proposition then implies that in the process of calculating the beliefs of the agent, before processing ρ an equivalent formula has been constructed. This means that after processing ρ the modified core and intermediate inputs will still yield the same result.

Proposition 4. *Let* L *be a finitely generated propositional language.*
Let $x_1, \ldots x_n \notin L$ *be additional propositional variables.*
Let $\sigma = (\alpha_m, \ldots, \alpha_1)$ *be a sequence of formulae such that for all* i *either* $\alpha_i \in L$ *or* $\alpha_i = x_j$ *for some* $j \in \{1, \ldots n\}$.

Let $\blacktriangle \wedge \psi$ be a formula such that $\blacktriangle \in L$ and $Cn(\blacktriangle) = Cn(\blacktriangle \wedge \psi) \cap L$.
Let $\sigma' = (\alpha'_m, \dots, \alpha'_1)$ such that $\alpha'_i = \alpha_i$ if $\alpha_i \in L$ and $\alpha'_i = x_j \wedge \psi$ if $\alpha_i = x_j$.
Then $f(\sigma \cdot \blacktriangle \wedge \psi) \equiv f(\psi \cdot \sigma' \cdot \blacktriangle)$.

Proof. *The proof shows inductively that both calculations collect the corresponding elements of σ and σ', i.e., $f(\sigma \cdot \blacktriangle \wedge \psi)$ accepts α_i if and only if $f(\psi \cdot \sigma' \cdot \blacktriangle)$ accepts α'_i. A consequence of this is that the two are indeed equivalent.*

The proposition holds trivially for inconsistent $\blacktriangle \wedge \psi$, so now we have to show that it also holds in case $\blacktriangle \wedge \psi$ is consistent. As L is finitely generated $\blacktriangle$ always has a finite representation.

$\blacktriangle \wedge \psi \nvdash \bot$ implies $\blacktriangle \nvdash \bot$. Assume that no $x_j/x_j \wedge \psi$ has been accepted so far, by inductive hypothesis both have collected the same sentences from the first i elements of σ/σ' so far, let their conjunction be denoted by χ. We want to compare $f((\alpha_m, \dots, \alpha_{i+1}) \cdot \blacktriangle \wedge \psi \wedge \chi)$ and $f\big(\psi \cdot (\alpha'_m, \dots, \alpha'_{i+1}) \cdot \blacktriangle \wedge \chi\big)$

- *Consider the case that $\alpha_{i+1} = x_j$, i.e., $\alpha'_{i+1} = x_j \wedge \psi$. If x_j is rejected this means that $(\blacktriangle \wedge \psi \wedge \chi) \wedge x_j$ is inconsistent and hence $(\blacktriangle \wedge \chi) \wedge (x_j \wedge \psi)$ is inconsistent, so $x_j \wedge \psi$ is also rejected. In this case, we can proceed with the next inductive step as χ has not changed.*
  If x_j is accepted we know $(\blacktriangle \wedge \psi \wedge \chi) \wedge x_j$ is consistent and consequently $(\blacktriangle \wedge \chi) \wedge (x_j \wedge \psi)$ is consistent, so $x_j \wedge \psi$ also accepted. As a consequence we have $f((\alpha_m, \dots, \alpha_{i+1}) \cdot \blacktriangle \wedge \psi \wedge \chi) = f((\alpha_m, \dots, \alpha_{i+2}) \cdot \blacktriangle \wedge \psi \wedge \chi \wedge x_j)$ and $f\big(\psi \cdot (\alpha'_m, \dots, \alpha'_{i+1}) \cdot \blacktriangle \wedge \chi\big) = f\big(\psi \cdot (\alpha'_m, \dots, \alpha'_{i+2}) \cdot \blacktriangle \wedge \chi \wedge x_j \wedge \psi\big)$. Note that the last element is equivalent, i.e., the two will accept exactly the same elements of the respective sequences. The difference between x_k and $x_k \wedge \psi$ as well as the ψ appended to the front will be irrelevant as ψ is already entailed. Hence, in case $x_j/x_j \wedge \psi$ is accepted the two clearly will be equivalent. Note that this fully covers all cases where at least one $x_j/x_j \wedge \psi$ has been accepted. If in the end all $x_j/x_j \wedge \psi$ were rejected, we have $f(\sigma \cdot \blacktriangle \wedge \psi) = \blacktriangle \wedge \psi \wedge \chi$ and by our induction $f(\psi \cdot \sigma' \cdot \blacktriangle) = f(\psi \cdot \blacktriangle \wedge \chi) = \blacktriangle \wedge \chi \wedge \psi$ as that is obviously consistent. Hence the two are equivalent as claimed.
- *We still have to show the case where $\alpha_i \in L$ implying $\alpha'_{i+1} = \alpha_{i+1}$. As noted above the only interesting case is when no $x_j/x_j \wedge \psi$ has been accepted so far. If $f((\alpha_m, \dots, \alpha_{i+1}) \cdot \blacktriangle \wedge \psi \wedge \chi)$ accepts α_{i+1} we know that $\blacktriangle \wedge \psi \wedge \chi \wedge \alpha_{i+1}$ is consistent, so $\blacktriangle \wedge \chi \wedge \alpha_{i+1}$ is consistent and consequently $f\big(\psi \cdot (\alpha'_m, \dots, \alpha'_{i+1}) \cdot \blacktriangle \wedge \chi\big)$ accepts α'_{i+1} as claimed.*
  If $f((\alpha_m, \dots, \alpha_{i+1}) \cdot \blacktriangle \wedge \psi \wedge \chi)$ rejects α_{i+1} we know that $\blacktriangle \wedge \psi \wedge \chi \wedge \alpha_{i+1}$ is inconsistent. This means $\blacktriangle \wedge \psi \wedge \chi \vdash \neg\alpha_{i+1}$ and hence $\blacktriangle \wedge \psi \vdash \chi \to \neg\alpha_{i+1}$. We know that no $x_j/x_j \wedge \psi$ has been accepted so far, which implies that $\chi \in L$ and as $\alpha_{i+1} \in L$ we also have $\chi \to \neg\alpha_{i+1} \in L$.
  $\blacktriangle \wedge \psi \vdash \chi \to \neg\alpha_{i+1}$ together with the condition on $\blacktriangle$ that any consequence of the conjunction that is an element of L must already be entailed by $\blacktriangle$ now yields $\blacktriangle \vdash \chi \to \neg\alpha_{i+1}$. This implies $\blacktriangle \wedge \chi \vdash \neg\alpha_{i+1}$ and hence $f\big(\psi \cdot (\alpha'_m, \dots, \alpha'_{i+1}) \cdot \blacktriangle \wedge \chi\big)$ rejects α'_{i+1} as claimed. □

Proposition 5. *If $[\rho, \blacktriangle]$ is an explanation for $o = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ then there is a $\blacktriangle'$ and a ψ such that $\blacktriangle \vdash \blacktriangle'$ and $\blacktriangle'$ does not contain any*

x_j and $[\rho \cdot \psi, \blacktriangle']$ is an explanation for $o' = o^1 \cdot \check{o}^1 \cdot o^2 \cdot \ldots \cdot \check{o}^n \cdot o^{n+1}$ where $\check{o}^i = \langle (x_i \wedge \psi, \top, \emptyset) \rangle$.

Proof. This follows almost immediately from Proposition 4. The revision inputs of o and o' fit the requirements for σ, σ' respectively. It is possible to construct $\blacktriangle'$ and ψ such that $\blacktriangle \equiv \blacktriangle' \wedge \psi$ and $\blacktriangle'$ has the properties needed for the application of Proposition 4. If we represent $\blacktriangle$ in clausal form, construct the set of all its resolvents, we can construct $\blacktriangle'$ from those clauses not containing an additional variable x_i and ψ from the remaining ones.

Proposition 4 tells us that $f(\cdot)$ will be equivalent in both cases before processing ρ, so they remain equivalent after then processing ρ. Hence, if $[\rho, \blacktriangle]$ is an explanation for o then $[\rho \cdot \psi, \blacktriangle']$ is an explanation for o'. □

Proposition 5 formalises that we can indeed rid the core of the additional variables by modifying the intermediate inputs. If we do so for the weakest core there is for $o = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ we might have found the weakest core there is for *any* instantiation of the intermediate inputs. That this is indeed the case follows from the next two propositions.

Proposition 6. *Let $[\rho, \blacktriangle]$ be the rational explanation of $o{=}o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ and $\blacktriangle'$ be an o'-acceptable core with $o' = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot \check{\phi}_n \cdot o^{n+1}$. Further let $\blacktriangle''$ such that $Cn(\blacktriangle'') = Cn(\blacktriangle) \cap L$ where L is the language of $o^1 \cdot o^2 \cdot \ldots \cdot o^{n+1}$.*

Then $\blacktriangle' \vdash \blacktriangle''$

Proof. Note that it does not matter which additional variables we use, so we can ensure that no x_j appears in o' or $\blacktriangle'$ without semantically changing the outcome of the rational explanation. Then $\blacktriangle''$ is uniquely determined (semantically). Proposition 3 tells us that $\blacktriangle' \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i)$ is an explanation for $o = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$. By Proposition 1 $\blacktriangle' \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i) \vdash \blacktriangle$ as $\blacktriangle$ is the weakest explaining core for that observation. Hence $\blacktriangle' \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i) \vdash \blacktriangle''$.

Now assume $\blacktriangle' \not\vdash \blacktriangle''$, i.e., there is an assignment m such that $m \models \blacktriangle'$ but $m \not\models \blacktriangle''$. No x_j appears in $\blacktriangle'$ or $\blacktriangle''$, so we can construct an assignment m' that is equivalent to m except for falsifying all x_j. Hence $m' \models \blacktriangle'$, $m' \not\models \blacktriangle''$ but also $m' \models \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i)$. But this contradicts $\blacktriangle' \wedge \bigwedge_{1 \leq i \leq n}(x_i \rightarrow \phi_i) \vdash \blacktriangle''$. Hence our assumption was wrong and indeed $\blacktriangle' \vdash \blacktriangle''$. □

Consider $\blacktriangle'$ which is an explanation for some observation o with intermediate inputs. We replace those inputs with new variables and apply the rational explanation, $[\rho, \blacktriangle]$ being the result. Now, for any φ from the language of the known part of the observation such that $\blacktriangle \vdash \varphi$ we know $\blacktriangle' \vdash \varphi$. As a consequence, we can calculate a formula any agent must at least commit to, no matter what the intermediate inputs were. This gives us a lower bound to the core belief an agent must have according to the observation. That there is also an upper bound (which coincides with the lower bound we gave) is formalised in the following result.

Proposition 7. *Let $[\rho, \blacktriangle]$ be the rational explanation of $o{=}o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ and $\blacktriangle'$ such that $Cn(\blacktriangle') = Cn(\blacktriangle) \cap L$ where L is the language of $o^1 \cdot o^2 \cdot \ldots \cdot o^{n+1}$.*

Then $\blacktriangle'$ is the weakest explaining core for any set of intermediate inputs at the given positions.

Proof. Proposition 6 yields that any explaining core will entail $\blacktriangle'$ *and (the proof of) Proposition 5 tells us that* $\blacktriangle'$ *is indeed an explaining core.* □

We have thus shown that there is a unique weakest core belief explaining an observation, given that the number and position of the intermediate inputs is fixed. Analogous to Proposition 1, we can also show that given two specific cores $\blacktriangle_1$ and $\blacktriangle_2$ that explain an observation using different intermediate inputs, we can find intermediate inputs such that $\blacktriangle_1 \vee \blacktriangle_2$ explain the observation thus obtained.

Proposition 8. *If* $\blacktriangle_1$ *is an explanation for* $o_1 = o^1 \cdot \check{\phi}_{11} \cdot o^2 \cdot \ldots \cdot \check{\phi}_{1n} \cdot o^{n+1}$ *and* $\blacktriangle_2$ *an explanation for* $o_2 = o^1 \cdot \check{\phi}_{21} \cdot o^2 \cdot \ldots \cdot \check{\phi}_{2n} \cdot o^{n+1}$ *then there are intermediate inputs* $\phi_1, \ldots, \phi_n$ *such that* $\blacktriangle_1 \vee \blacktriangle_2$ *is an explanation for* $o = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot \check{\phi}_n \cdot o^{n+1}$.

Proof. Applying Proposition 3 for o_1 *and* o_2 *we know that* $\blacktriangle_1 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{1i})$ *and* $\blacktriangle_2 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{2i})$ *are explanations for* $o' = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ *where all* x_i *are new propositional variables not appearing in any* o^j, ϕ_{kl}, *or* $\blacktriangle_k$.

Proposition 1 yields that $(\blacktriangle_1 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{1i})) \vee (\blacktriangle_2 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{2i}))$ *is an explanation for* $o' = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$.

Note that $(\blacktriangle_1 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{1i})) \vee (\blacktriangle_2 \wedge \bigwedge_{1\leq i\leq n}(x_i \to \phi_{2i}))$ *can equivalently be written as* $(\blacktriangle_1 \vee \blacktriangle_2) \wedge \bigwedge_{1\leq i\leq n}(x_i \to \psi_i)$ *for some* ψ_i. *We abbreviate* $\blacktriangle_1 \vee \blacktriangle_2$ *with* $\blacktriangle$ *and* $\bigwedge_{1\leq i\leq n}(x_i \to \psi_i)$ *with* ψ. *Hence* $[\rho, \blacktriangle \wedge \psi]$ *is an explanation for* $o' = o^1 \cdot \check{x}_1 \cdot o^2 \cdot \ldots \cdot \check{x}_n \cdot o^{n+1}$ *for some sequence* ρ.

Proposition 4 can be applied to show that $[\rho \cdot \psi, \blacktriangle]$ *is an explanation for* $o'' = o^1 \cdot \check{\phi}_1 \cdot o^2 \cdot \ldots \cdot \check{\phi}_n \cdot o^{n+1}$ *where* $\check{\phi}_i = \langle (x_i \wedge \psi, \top, \emptyset) \rangle$, *i.e., we replaced all intermediate inputs* x_i *with* $x_i \wedge \psi$.

To see that Proposition 4 is applicable, we have to show that the conditions are satisfied. The language L *of* o *(neglecting the intermediate inputs) is finitely generated by definition and it does not contain the additional variables* x_i. *We apply the proposition to the revision inputs of any prefix of* o' *which indeed has the property that any formula contained is either an element of* L *or a variable* x_i. o'' *is defined as the proposition requires for* σ'. *We still need to show that* $Cn(\blacktriangle) = Cn(\blacktriangle \wedge \psi) \cap L$. $Cn(\blacktriangle) \subseteq Cn(\blacktriangle \wedge \psi) \cap L$ *is obvious. So let* $\varphi \in L$ *with* $\varphi \in Cn(\blacktriangle \wedge \psi)$. *Assume* $\blacktriangle \nvdash \varphi$, *which implies the existence of an assignment* m *with* $m \models \blacktriangle$ *but* $m \not\models \varphi$. *As both* $\blacktriangle$ *and* φ *are elements of* L *they contain no* x_j, *hence we can construct an assignment equivalent to* m *except for falsifying all* x_j. $m' \models \blacktriangle$ *and* $m' \models \psi$ $(\psi = \bigwedge_{1\leq i\leq n}(x_i \to \psi_i))$. *Hence* $m' \models \blacktriangle \wedge \psi$ *but* $m' \not\models \varphi$ — *contradiction. So* $\blacktriangle \vdash \varphi$ *showing that Proposition 4 is applicable.* □

3.3 Some Effects of Extending the Language

Before we discuss the use of additional variables, we want to return to the claim that two consecutive intermediate inputs generally cannot be replaced by a single one. The following example shows that the number of intermediate inputs received in succession cannot be disregarded as having no impact. Even extending the language cannot compensate for trying to reduce their number by one!

Example 3. Consider an observation with two known parts $o^1{=}\langle(a,a,\emptyset),(c,c,\emptyset)\rangle$ and $o^2 = \langle(b,\neg a,\emptyset),(d,\neg c,\emptyset),(a\wedge b\wedge p,\neg c,\emptyset),(c\wedge d\wedge\neg p,\neg a,\emptyset)\rangle$. Using the intermediate inputs $\phi_1 = a\rightarrow\neg b$ and $\phi_2 = c\rightarrow\neg d$ it is quite easy to see that $[(),\top]$ is an explanation for $o = o^1\cdot\check{\phi}_1\cdot\check{\phi}_2\cdot o^2$.

Proposition 7 allows us to calculate the weakest core any single intermediate input will yield. Assume we could combine the two intermediate inputs into a single one, we would have to get $\top$ as there are clearly two intermediate inputs that yield that explanation. However, the rational explanation calculates the following core belief for $o' = o^1\cdot\check{x}\cdot o^2$:
$\blacktriangle = (\neg a\vee\neg b\vee\neg c\vee\neg d\vee p)\wedge(\neg x\vee\neg b\vee\neg c\vee\neg d)\wedge(\neg x\vee\neg a\vee\neg b\vee\neg c)$. Hence any core with a single intermediate input must entail $\neg a\vee\neg b\vee\neg c\vee\neg d\vee p$, as that formula follows from $\blacktriangle$ and belongs to the language of o^1 and o^2. So $\top$ will never work.

The message of the previous section was that there is a unique weakest core belief for a fixed number of intermediate input at fixed positions. We can calculate this core belief (and intermediate inputs that will yield it), assuming each intermediate input to be a new variable, employing the rational explanation construction and then eliminating the new variables from the core. This result comes at a price. We assumed, that the intermediate inputs may contain propositional variables that are not present in the known part of the observation. It is indeed the case that some observations can only be explained when allowing additional variables.

Example 4. Consider $o^1 = \langle(b,b,\emptyset),(d,c,\emptyset),(a,a\wedge b\wedge c\wedge d,\emptyset),(\neg b,\neg b,\emptyset)\rangle$, $o^2 = \langle(d,c,\emptyset),(a,a\wedge b\wedge\neg c\wedge d,\emptyset),((a\wedge c\wedge d)\vee(\neg b\wedge\neg c\wedge d),b,\emptyset)\rangle$, and $o^3 = \langle(\varphi_1,\varphi_1,\emptyset),\ldots(\varphi_n,\varphi_n,\emptyset)\rangle$, where φ_i varies over all (finitely many) semantically different formulae containing the variables a,b,c,d.

The observation $o = o^1\cdot o^2\cdot o^3$ does not have an explanation when not allowing intermediate inputs. In fact, there is no explanation for a prefix of o, already. $\langle(b,b,\emptyset),(d,c,\emptyset),(a,a{\wedge}b{\wedge}c{\wedge}d,\emptyset),(\neg b,\neg b,\emptyset),(d,c,\emptyset),(a,a{\wedge}b{\wedge}\neg c{\wedge}d,\emptyset)\rangle$ does not have an acceptable core. Intuitively, the reason is as follows. What is supposed to be believed after a is received the first and the second time? In both cases $a\wedge b\wedge d$ is to be believed, but once c and once $\neg c$. We can conclude that when receiving the second a the prior input $\neg b$ needs to be blocked. But no other input must be blocked as all of the others are to be believed. However, this means there is no way to distinguish whether the agent has received the first or the second a. Technically, this observation gives rise to two contradictory conditional beliefs whose conflict cannot be resolved by modifying the core belief, as except for $\neg b$ no inputs must be blocked.

An intermediate input is exactly what does the trick — allowing a distinction between having received the first or the second a. For $o' = o^1{\cdot}\check{x}{\cdot}o^2{\cdot}o^3$ the rational explanation gives $\blacktriangle = \neg a\vee b\vee\neg d\vee\neg x$ as the weakest core. Intuitively, when the first a is seen, the agent believes $\neg x$, when the second a has been received, it believes x. And this difference can explain different attitudes towards c.

But why does no intermediate input ϕ just containing a,b,c and d work? We note that o^3 forces any formula made up of these variables to be accepted upon receiving it. Hence, the core belief of the agent must be a tautology. We then can

infer the following restrictions on the intermediate input ϕ. Firstly, $\phi \vdash a \wedge d \rightarrow b$ is required, as otherwise $\neg b$ would not be blocked when considering the second a. Secondly, ϕ must be consistent with $((a \wedge c \wedge d) \vee (\neg b \wedge \neg c \wedge d)) \wedge a \wedge d \equiv (a \wedge c \wedge d) \vee (a \wedge \neg b \wedge \neg c \wedge d)$. This is because $\neg b$ must still be blocked when the revision input $(a \wedge c \wedge d) \vee (\neg b \wedge \neg c \wedge d)$ is considered. $\blacktriangle = \top$ so both a and d are accepted which means that ϕ must be consistent with the conjunction of these formulae. Thirdly, as $a \wedge d \rightarrow b$ is inconsistent with $(a \wedge \neg b \wedge \neg c \wedge d)$, ϕ must in fact be consistent with $a \wedge c \wedge d$. So in particular, ϕ must *not* entail $a \wedge d \rightarrow \neg c$. But now we can apply a similar argument as for the observation without intermediate input. The above mentioned restrictions do not allow to construct an intermediate input that can distinguish between having received the first and the second a.

This means that the proposed method, which assumes the intermediate inputs were outside the language of the known part of the observation, might say there is an explanation, while there is none when restricting the language of the intermediate inputs. So, how is the use of additional variables to be evaluated? Doubtlessly, it is useful for efficiently determining whether an explanation exists at all, and what a potential core belief must entail. It is not absurd to assume that in its epistemic evolution the agent has encountered more variables than are on record in the observation. Even when only considering the observation, not all variables appear at each point of time and new ones may occur or vanish later in the observation. So why should a new variable not also appear with an intermediate input? On the other hand, nothing in the observation tells us that an input containing a new variable has been received as intermediate input — and even if there has, we cannot know *which* variable it is.

We have shown that the additional variables can be eliminated from the core belief assigned to the agent. In some cases, it is also possible to eliminate them from the intermediate inputs (and thereby from the explanation in total, as the rational explanation does not invent variables), but currently we cannot specify the conditions under which this is possible. That it is not always possible has been illustrated by the last example.

4 Conclusion and Open Problems

In the present paper, we have generalised work started in [1, 3] for reconstructing an agent's epistemic state from an observation on its belief revision behaviour. We have done so by dropping the assumption that the observation provided is complete in the sense that all revision inputs received were recorded. By allowing intermediate inputs to have occurred, more observations can be explained. We showed that the number and positions of the intermediate inputs have an effect on the explanation of the observation.

In order to give specific results, we have fixed the number and position of the intermediate inputs in the observation. We were then able to show that there is a unique weakest core belief that explains the observation for some intermediate inputs. This was achieved by assuming the intermediate inputs to

be propositional variables not contained in the known part of the observation, applying the rational explanation construction and then eliminating the new variables from the core calculated.

The conditions under which it is possible to eliminate the additional variables from the intermediate inputs and thereby from the explanation are not known. This is an important point for further work. A sufficient condition under which an explanation from the language of the known part of the observation exists is also yet to be found. We have focussed on the core belief, i.e., finding *some* intermediate inputs yielding the best core there is. Finding the *best* intermediate inputs achieving that is an open problem.

Some general questions also remain to be addressed. How to trade off between a weak core and few intermediate inputs if their number is not known? How to compare explanations where number and position of the intermediate inputs do not coincide — can we define a sensible preference relation?

References

1. Booth, R., Nittka, A.: Reconstructing an agent's epistemic state from observations. In Kaelbling, L.P., Saffiotti, A., eds.: IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005, Professional Book Center (2005) 394–399
2. Gärdenfors, P.: Knowledge in Flux. MIT Press (1988)
3. Booth, R., Nittka, A.: Beyond the rational explanation. In Delgrande, J., Lang, J., Rott, H., Tallon, J.M., eds.: Belief Change in Rational Agents: Perspectives from Artificial Intelligence, Philosophy, and Economics. Number 05321 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany (2005)
4. Booth, R.: On the logic of iterated non-prioritised revision. In: Conditionals, Information and Inference – Selected papers from the Workshop on Conditionals, Information and Inference, 2002, Springer's LNAI 3301 (2005) 86–107
5. Nebel, B.: Base revision operations and schemes: Semantics, representation and complexity. In: Proceedings of ECAI'94. (1994) 342–345
6. Lehmann, D., Magidor, M.: What does a conditional knowledge base entail? Artificial Intelligence **55**(1) (1992) 1–60
7. Freund, M.: On the revision of preferences and rational inference processes. Artificial Intelligence **152**(1) (2004) 105–137
8. Booth, R., Paris, J.B.: A note on the rational closure of knowledge bases with both positive and negative knowledge. Journal of Logic, Language and Information **7**(2) (1998) 165–190

Knowledge Base Revision in Description Logics

Guilin Qi, Weiru Liu, and David A. Bell

School of Electronics, Electrical Engineering and Computer Science
Queen's University Belfast, UK
{G.Qi, W.Liu, DA.Bell}@qub.ac.uk

Abstract. Ontology evolution is an important problem in the Semantic Web research. Recently, Alchourrón, Gärdenfors and Markinson's (AGM) theory on belief change has been applied to deal with this problem. However, most of current work only focuses on the feasibility of the application of AGM postulates on *contraction* to description logics (DLs), a family of ontology languages. So the explicit construction of a revision operator is ignored. In this paper, we first generalize the AGM postulates on revision to DLs. We then define two revision operators in DLs. One is the weakening-based revision operator which is defined by weakening of statements in a DL knowledge base and the other is its refinement. We show that both operators capture some notions of minimal change and satisfy the generalized AGM postulates for revision.

1 Introduction

Ontologies play a crucial role for the success of the Semantic Web [6]. One of the challenging problems for the development of ontology is ontology evolution, which is defined as the timely adaptation of an ontology to the arisen changes and the consistent management of these changes [10]. Ontology evolution is a very complex process, i.e. it consists of six phases [27]. In this paper, we consider an important phase called *semantics of change phase*, which prevents inconsistencies by computing additional changes that guarantee the transition of the ontology into a consistent state [27]. A center problem in this phase is inconsistency handling. There are various forms of inconsistencies, such as structural inconsistency, logical inconsistency and user-defined inconsistency. Among them, logical inconsistency in ontology evolution has attached lots of attention in recent years, where ontologies are represented by logical theories, such as description logics [21, 1, 8, 11, 10, 14, 19, 25].

AGM's theory of belief change [9] has been widely used to deal with logical inconsistency resulting from revising a knowledge base by newly received information. There are three types of belief change, i.e. *expansion*, *contraction* and *revision*. Expansion is simply to add a sentence to a knowledge base; contraction requires to consistently remove a sentence from a knowledge base and revision is the problem of accommodating a new sentence to a knowledge base consistently. Alchourrón, Gardenfors and Markinson proposed a set of postulates to characterize each belief change operator. The application of AGM' theory to description

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 386–398, 2006.

logics is not trivial because it is based on the assumptions that generally fail for DLs [7]. For example, a DL is not necessarily closed under the usual operators such as $\neg$ and $\wedge$ [8]. In [7, 8], the basic AGM postulates for contraction were generalized to DLs and the feasibility of applying the generalized AGM theory of contraction to DLs and OWL was studied. However, no explicit belief change operators were proposed in their papers. Furthermore, they did not consider the application of AGM postulates for revision in DLs.

In this paper, we first generalize the AGM postulates for revision to DLs. Instead of discussing the feasibility of applying the postulates, we propose two revision operators in DLs. One is the weakening-based revision operator which is defined by weakening of statements in a DL knowledge base. Since the weakening-based revision operator may result in counterintuitive results in some cases, we propose an operator to refine it. We show that both operators capture some notions of minimal change and satisfy the generalized AGM postulates on revision.

This paper is organized as follows. Section 2 gives a brief review of description logics. In Section 3, we generalize the Gärdenfors postulates on revision to DLs. We then propose two revision operators and discuss their logical properties in Section 4. In Section 5, we have a brief discussion on related work. Finally, we conclude the paper in Section 6 and give some further work.

2 Description Logics

In this section, we will introduce some basic notions of Description Logics (DLs), a family of well-known knowledge representation formalisms [3]. To make our approach applicable to a family of interesting DLs, we consider the well-known DL $\mathcal{ALC}$ [26], which is a simple yet relatively expressive DL. Let N_C and N_R be pairwise disjoint and countably infinite sets of *concept names* and *role names* respectively. We use the letters A and B for concept names, the letter R for role names, and the letters C and D for concepts. $\top$ and $\bot$ denote the universal concept and the bottom concept respectively. The set of $\mathcal{ALC}$ concepts is the smallest set such that: (1) every concept name is a concept; (2) if C and D are concepts, R is a role name, then the following expressions are also concepts: $\neg C$ (full negation), $C \sqcap D$ (concept conjunction), $C \sqcup D$ (concept disjunction), $\forall R.C$ (value restriction on role names) and $\exists R.C$ (existential restriction on role names).

An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$, called the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every concept C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and every role R to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for all concepts C, D, role R, the following properties are satisfied:

(1) $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$, $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
(2) $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$, $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$,
(3) $(\exists R.C)^{\mathcal{I}} = \{x | \exists\ y\ s.t. (x, y) \in R^{\mathcal{I}}$ and $y \in C^{\mathcal{I}}\}$,
(4) $(\forall R.C)^{\mathcal{I}} = \{x | \forall y (x, y) \in R^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}\}$.

A DL knowledge base consists of two components, the *terminological box (TBox)* and the *assertional box (ABox)*. A TBox is a finite set of terminological

axioms of the form $C \sqsubseteq D$ (general concept inclusion or GCI for short) or $C \equiv D$ (equalities), where C and D are two (possibly complex) $\mathcal{ALC}$ concepts. An interpretation $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies an equality $C \equiv D$ iff $C^{\mathcal{I}} = D^{\mathcal{I}}$. It is clear that $C \equiv D$ can be seen as an abbreviation for the two GCIs $C \sqsubseteq D$ and $D \sqsubseteq C$. Therefore, we take a TBox to contain only GCIs. We can also formulate statements about individuals. We denote individual names as $a,\ b,\ c$. A *concept* (*role*) assertion axiom has the form $C(a)$ ($R(a, b)$), where C is a concept description, R is a role name, and $a,\ b$ are *individual names*. To give a semantics to ABoxs, we need to extend interpretations to individual names. For each individual name a, $\cdot^{\mathcal{I}}$ maps it to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The mapping $\cdot^{\mathcal{I}}$ should satisfy the *unique name assumption* (UNA), that is, if a and b are distinct names, then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a concept axiom $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, it satisfies a role axiom $R(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An *ABox* contains a finite set of concept and role axioms. A DL knowledge base K consists of a TBox and an ABox, i.e. it is a set of GCIs and assertion axioms. An interpretation $\mathcal{I}$ is a *model* of a DL (TBox or ABox) axiom iff it satisfies this axiom, and it is a model of a DL knowledge base K if it satisfies every axiom in K. In the following, we use $M(\phi)$ (or $M(K)$) to denote the set of models of an axiom ϕ (or DL knowledge base K). K is consistent iff $M(K) \neq \emptyset$. Two DL knowledge bases K_1 and K_2 are said to be element-equivalent iff there is a bijectin f from K_1 to K_2 such that for every ϕ in K_1, $M(f(\phi)) = M(\phi)$. Let K be an inconsistent DL knowledge base. A set $K' \subseteq K$ is a *conflict* of K if K' is inconsistent, and any sub-knowledge base $K'' \subset K'$ is consistent. Given a DL knowledge base K and a DL axiom ϕ, we say K *entails* ϕ, denoted as $K \models \phi$, iff $M(K) \subseteq M(\phi)$. We use $\mathcal{KB}$ to denote the set of all possible DL knowledge bases.

3 Generalizing the AGM Postulates for Revision to DLs

Let L be a propositional language constructed from a finite alphabet P of propositional symbols using the usual operators $\neg$ (not), $\vee$ (or) and $\wedge$ (and). An interpretation is a mapping from P to $\{true, false\}$. A *model* of a formula ϕ is an interpretation that makes ϕ true in the usual sense. $M(\phi)$ denotes the set of all the models of ϕ. A formula ϕ is satisfiable if $M(\phi) \neq \emptyset$. We denote the classical consequence relation by $\vdash$. Two formulas ϕ and ψ are equivalent, denoted as $\phi \equiv \psi$ iff $M(\phi) = M(\psi)$. In [17], AGM postulates for revision are rephrased as follows, where $\circ$ is a revision operator which is a function from a pair of formulas ψ and μ to a new formula denoted by $\psi \circ \mu$.

(R1) $\psi \circ \mu \vdash \mu$
(R2) If $\psi \wedge \mu$ is satisfiable then $\psi \circ \mu \equiv \psi \wedge \mu$
(R3) If μ is satisfiable then $\psi \circ \mu$ is also satisfiable
(R4) If $\psi_1 \equiv \psi_2$ and $\mu_1 \equiv \mu_2$ then $\psi_1 \circ \mu_1 \equiv \psi_2 \circ \mu_2$
(R5) $(\psi \circ \mu) \wedge \phi$ implies $\psi \circ (\mu \wedge \phi)$
(R6) If $(\psi \circ \mu) \wedge \phi$ is satisfiable then $\psi \circ (\mu \wedge \phi)$ implies $(\psi \circ \mu) \wedge \phi$

We first define a revision operator in DLs. Before that, we need to introduce the notion of a *disjunctive DL knowledge base* (or DKB) in [19], which is defined

as a set of DL knowledge bases. In the following, a DL knowledge base is viewed as a disjunctive DL knowledge base which contains a single DL knowledge base. In propositional logic, disjunction $\vee$ is a very important connective used to define revision operators. For example, the result of Dalal's revision operator is (syntactically) in disjunction form [5]. However, DL languages do not allow disjunctions of TBox statements with ABox statements. The semantics of DKBs is defined as follows [19]:

Definition 1. *A DKB $\mathcal{K}$ is satisfied by an interpretation $\mathcal{I}$ (or $\mathcal{I}$ is a model of $\mathcal{K}$) iff $\exists K \in \mathcal{K}$ such that $\mathcal{I} \models K$. $\mathcal{K}$ entails ϕ, denoted $\mathcal{K} \models \phi$, iff every model of $\mathcal{K}$ is a model of ϕ.*

Let $\mathcal{DKB}$ denote a set of (disjunctive) DL knowledge bases. A revision operator in DLs can be defined as follows.

Definition 2. *A knowledge base revision operator (or revision operator for short) in DLs is a function $\circ : \mathcal{DKB} \times \mathcal{KB} \rightarrow \mathcal{DKB}$ which satisfies the following condition: $\mathcal{K} \circ K' \models \phi$, for all $\phi \in K'$.*

That is, both the original knowledge base and the resulting knowledge base can be a DKB, Whist the newly received knowledge base must be an ordinary DL knowledge base (i.e. it is not a DKB).

We next generalize postulates (R1)-(R6) to DLs. The generalization is not as trivial as we have thought. The problem is that both the original knowledge base the result of revision may be a disjunctive DL knowledge base. To generalize (R1)-(R6), we need to define the conjunction of a disjunctive DL knowledge base and an ordinary DL knowledge base. A more simple way to generalize AGM postulates is to define them in a model-theoretic way as follows.

It is clear that (R1)-(R3) can be generalized in the following way. Let K be a (disjunctive) DL knowledge base and K' be a DL knowledge base, we have

(G1) $K \circ K' \models \phi$ for all $\phi \in K'$

(G2) If $M(K) \cap M(K') \neq \emptyset$, then $M(K \circ K') = M(K) \cap M(K')$

(G3) If K' is consistent, then $M(K \circ K') \neq \emptyset$

(G1) guarantees that the new information is inferred from the revised knowledge base. (G2) requires that when there is no conflict between K and K', the result of revision be equivalent to the "union" of K and K', i.e. the set of its models are $M(K) \cap M(K')$. (G3) is a condition preventing a revision from introducing unwarranted inconsistency.

The postulate (R4) is the principle of irrelevance of syntax. Its generalization has the following form:

(G4) If $M(K) = M(K_1)$ and $M(K') = M(K_2)$, then $M(K \circ K') = M(K_1 \circ K_2)$.

(G4) requires that the revised knowledge base be independent of the syntax of both original knowledge bases and new information. The rule (R4) (and its generalization (G4)) is (are) very strong condition(s) because many syntax-based revision operators in propositional logic do not satisfy it. It is interesting to consider a weakened version of (G4) as follows.

(G4)′ If K_1 and K_2 are element-equivalent and $M(K_1') = M(K_2')$, then $M(K_1 \circ K_1') = M(K_2 \circ K_2')$.

Finally, (R5) and (R6) are generalized as follows.

(G5) $M(K \circ K') \cap M(K'') \subseteq M(K \circ (K' \cup K''))$

(G6) If $M(K \circ K') \cap M(K'')$ is not empty, then $M(K \circ (K' \cup K'')) \subseteq M(K \circ K') \cap M(K'')$

We have the following definition.

Definition 3. *A revision operator $\circ$ is said to be AGM compliant if it satisfies (G1-G6). It is quasi-AGM compliant if it satisfies (G1)-(G3), (G4)$'$, (G5-G6).*

4 Revision Operators for DLs

4.1 Definition

In this subsection, we propose a revision operator for DLs and provide a semantic explanation of it.

In this paper, we only consider inconsistencies arising due to objects being explicitly introduced in the ABox. That is, suppose K and K' are the original knowledge base and the newly received knowledge base respectively, then for each conflict K_c of $K \cup K'$, K_c must contain an ABox statement. For example, we exclude the following case: $\top \sqsubseteq \exists R.C \in K$ and $\top \sqsubseteq \forall R.\neg C \in K'$. The handling of conflicting axioms in the TBox has been discussed recently in [25, 22]. In this paper, we discuss the resolution of conflicting information which contains assertional axioms in the context of knowledge revision.

In order to define our approach, we need to extend $\mathcal{ALC}$ with nominals $\mathcal{O}$ (also called *individual names* [24]). A nominal has the form $\{a\}$, where a is an individual name. It can be viewed as a powerful generalization of DL ABox individuals. The semantics of $\{a\}$ is defined by $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$ for an interpretation $\mathcal{I}$. Nominals are very important expressions and they are included in many important DLs, such as $\mathcal{SHOQ}$ [13].

We give a method to weaken a GCI first.

Definition 4. *Let $C \sqsubseteq D$ be a GCI. A weakened GCI $(C \sqsubseteq D)_{weak}$ of $C \sqsubseteq D$ has the form $(C \sqcap \neg\{a_1\} \sqcap ... \sqcap \neg\{a_n\}) \sqsubseteq D$, where n is the number of individuals to be removed from C. We use $d((C \sqsubseteq D)_{weak}) = n$ to denote the degree of $(C \sqsubseteq D)_{weak}$.*

It is clear that when $d((C \sqsubseteq D)_{weak}) = 0$, $(C \sqsubseteq D)_{weak} = C \sqsubseteq D$. The idea of weakening a GCI is similar to weaken an uncertain rule in [4]. That is, when a GCI is involved in conflict, instead of dropping it completely, we remove those individuals which cause the conflict.

The weakening of an assertion is simpler than that of a GCI. The weakened assertion ϕ_{weak} of an ABox assertion $\phi = C(a)$ is of the form $\phi_{weak} = \top(a)$ or $\phi_{weak} = \phi$. When $\phi_{weak} = \top(a)$, we have $\mathcal{I} \models \phi_{weak}$ for all $\mathcal{I}$. Therefore, when $\phi_{weak} = \top(a)$, we simply delete ϕ. Indeed, we denote ϕ_{weak} by $\top(a)$ when ϕ is to be deleted for convenience of theoretical analysis. The degree of ϕ_{weak}, denoted as $d(\phi_{weak})$, is defined as $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top(a)$ and 0 otherwise.

Definition 5. *Let K and K' be two DL knowledge bases. Suppose K' is consistent and $K \cup K'$ is inconsistent. A DL knowledge base $K_{weak,K'}$ is a weakened knowledge base of K w.r.t K' if it satisfies:*

- $K_{weak,K'} \cup K'$ *is consistent, and*
- *There is a bijection* f *from* K *to* $K_{weak,K'}$ *such that for each* $\phi \in K$, $f(\phi)$ *is a weakening of* ϕ.

The set of all weakened base of K *w.r.t* K' *is denoted by* $Weak_{K'}(K)$.

Example 1. Let $K = \{bird(tweety), bird \sqsubseteq flies\}$ and $K' = \{\neg flies(tweety)\}$, where $bird$ and $flies$ are two concepts and $tweety$ is an individual name. It is easy to check that $K \cup K'$ is inconsistent. Let $K_1 = \{\top(tweety), bird \sqsubseteq flies\}$, $K_2 = \{bird(tweety), bird \sqcap \neg\{tweety\} \sqsubseteq flies\}$, then both K_1 and K_2 are weakened bases of K *w.r.t* K'.

The degree of a weakened base is defined as follows.

Definition 6. *Let* $K_{weak,K'}$ *be a weakened base of a DL knowledge base* K *w.r.t* K'. *The degree of* $K_{weak,K'}$ *is defined as*

$$d(K_{weak,K'}) = \Sigma_{\phi \in K_{weak,K'}} d(\phi)$$

In Example 1, we have $d(K_1) = d(K_2) = 1$.

We now define a revision operator.

Definition 7. *Let* $\mathcal{K}$ *be a (disjunctive) DL knowledge base, and* K' *be a newly received DL knowledge base. The result of weakening-based revision of* $\mathcal{K}$ *w.r.t* K', *denoted as* $\mathcal{K} \circ_w K'$, *is defined as follows: If* K' *is inconsistent, then* $\mathcal{K} \circ_w K' = \{K \cup K' : K \in \mathcal{K}\}$; *Otherwise,*

$$\mathcal{K} \circ_w K' = \bigcup_{K \in \mathcal{K}} \{K' \cup K_{weak,K'} : \ K_{weak,K'} \in Weak_{K'}(K), \ and \ \nexists K_i \in Weak_{K'}(K), d(K_i) < d(K_{weak,K'})\}.$$

If K' is inconsistent, the result of revision is an inconsistent disjunctive DL knowledge base. When K' is consistent, the result of revision of $\mathcal{K}$ by K' is a disjunctive DL knowledge base consisting of DL knowledge bases which are unions of K' and a weakened base of a DL knowledge base K in $\mathcal{K}$ with the minimal degree. In the following, we assume that the original knowledge bases are ordinary DL knowledge base. This assumption is used to simply our discussions.

We next consider the semantic aspect of our revision operator.

Definition 8. *Let* $\mathcal{W}$ *be a non-empty set of interpretations and* $\mathcal{I} \in \mathcal{W}$, ϕ *a DL axiom, and* K *a DL knowledge base. If* ϕ *is an assertion, the number of* ϕ*-exceptions* $e^{\phi}(\mathcal{I})$ *is 0 if* $\mathcal{I}$ *satisfies* ϕ *and 1 otherwise. If* ϕ *is a GCI of the form* $C \sqsubseteq D$, *the number of* ϕ*-exceptions for* $\mathcal{I}$ *is:*

$$e^{\phi}(\mathcal{I}) = \begin{cases} |C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}})| & \text{if } C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}}) \text{ is finite} \\ \infty & \text{otherwise.} \end{cases} \quad (1)$$

The number of K*-exceptions for* $\mathcal{I}$ *is* $e^K(\mathcal{I}) = \Sigma_{\phi \in K} e^{\phi}(\mathcal{I})$. *The ordering* $\preceq_K$ *on* $\mathcal{W}$ *is:* $\mathcal{I} \preceq_K \mathcal{I}'$ *iff* $e^K(\mathcal{I}) \leq e^K(\mathcal{I}')$, *for* $\mathcal{I}' \in \mathcal{W}$.

The definition of ϕ-exception originates from Definition 6 in [19]. However, in [19], it is used to define an ordering $\preceq_K^{\pi}$ on a set of interpretations with the same *pre-interpretation* $\pi = (\Delta^{\pi}, d^{\pi})$, where Δ^{π} is a domain and d^{π} is a *denotation function* which maps every individual name a to a different element in Δ^{π}.

We give a proposition to give a semantic explanation of our weakening-based revision operator.

Proposition 1. *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. $\circ_w$ is the weakening-based revision operator. We then have*

$$M(K \circ_w K') = min(M(K'), \preceq_K).$$

Proposition 1 says that the models of the resulting knowledge base of our revision operator are models of K' which are minimal $w.r.t$ the ordering $\preceq_K$ induced by K. So it captures some kind of minimal change. All proofs of this paper can be found in [23].

Example 2. Let $K = \{\forall hasChild.RichHuman(Bob), hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom)\}$. Suppose we now receive new information $K' = \{hasChild\ (Bob, John), \neg RichHuman(John)\}$. It is clear that $K \cup K'$ is inconsistent. Since $\forall hasChild.\ RichHuman(Bob)$ is the only assertion axiom involved in conflict with K', we only need to delete it to restore consistency, that is, $K \circ_w K' = \{\top(Bob), hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom), hasChild(Bob, John), \neg RichHuman\ (John)\}$.

We have the following proposition.

Proposition 2. *Given two DL knowledge bases K and K'. The weakening-based revision operator is not AGM-compliant but it is quasi-AGM compliant, that is, it satisfies postulates (G1), (G2), (G3), (G4′), (G5) and (G6).*

4.2 Refined Weakening-Based Revision

In the weakening-based revision, to weaken a conflicting assertion axiom, we simply delete it. The problem for this method of weakening is that it does not take the constructors of description languages, such as conjunction ($\sqcap$) and value restriction ($\forall R.C$), into account. This may result in counterintuitive conclusions. In Example 2, after revising K by K' using the weakening-based operator, we cannot infer that $RichHuman(Tom)$ because $\forall hasChild.RichHuman(Bob)$ is discarded, which is counterintuitive. From $hasChild(Bob, Tom)$ and $\forall hasChild.\ RichHuman(Bob)$ we should have known that $RichHuman(Tom)$ and this assertion is not in any conflict of $K \cup K'$. The solution for this problem is to treat *John* as an *exception* and that all children of *Bob* other than *John* are rich humans.

For an ABox assertion of the form $\forall R.C(a)$, it is weakened by dropping some individuals which are related to the individual a by the relation R, i.e. its weakening has the form $\forall R.(C \sqcup \{b_1, ..., b_n\})(a)$, where b_i $(i = 1, n)$ are individuals.

We give another example to illustrate the problem of the weakening method.

Example 3. Let $K = \{bird \sqcap flies(tweety), bird(chirpy)\}$ and $K' = \{\neg flies(tweety)\}$. Clearly, $bird \sqcap flies(tweety)$ is in conflict with $\neg flies(tweety)$ in K'. Let $\phi = bird \sqcap flies(tweety)$. The weakening of ϕ is $\phi_{weak} = \top(tweety)$.

In Example 3, to weaken ϕ, we simply delete it. However, $bird(tweety)$, which can be inferred from K, is not responsible for any conflict of $K \cup K'$. Therefore, it is counterintuitive to delete it. This intuition is based on the assumption of the independence of concept names. That is, we take concept names as the "basic unit of change".

Before defining the new weakening method, we need to define an atomic concept.

Definition 9. *A concept is an atomic concept iff it is either a concept name or is of one of the forms $\{a\}$, $\forall R.C$ or $\exists R.C$, where a is an individual name and C is a (complex) concept.*

We assume that each concept C occurring in the original DL knowledge base K is in conjunctive normal form, i.e., $C = C_1 \sqcap ... \sqcap C_n$ such that $C_i = C_{i1} \sqcup ... \sqcup C_{im}$, where C_{ij} is either an atomic concept or the negation of a concept name. Conjunctive normal forms can be generated by the following steps. First, we transform the concept C into its negation normal form by the following equalities: $\neg\neg C_i \equiv C_i$, $\neg(C_i \sqcap D_i) \equiv \neg C_i \sqcup \neg D_i$, $\neg(C_i \sqcup D_i) \equiv \neg C_i \sqcap \neg D_i$, $\neg(\exists R.C_i) \equiv \forall R.\neg C_i$, $\neg(\forall R.C_i) \equiv \exists R.\neg C_i$. Second, we move disjunction inward and conjunction outward according to De Morgan's law: $C_1 \sqcup (C_2 \sqcap C_3) \equiv (C_1 \sqcup C_2) \sqcap (C_1 \sqcup C_3)$. Suppose $C(a) \in K$, where C is a concept in conjunctive normal form, we assume that each concept assertion $C(a)$ is decomposed into ϕ_1, ..., ϕ_n such that $\phi_i = (C_{i1} \sqcup ... \sqcup C_{im})(a)$. Note that a cannot be moved inside the disjunction constructor because disjunction of ABox assertions is not allowed in DLs.

We now define a new weakening method. The idea is that we weaken a concept assertion by weakening its atomic concepts. That is, we have the following definition.

Definition 10. *Let $\phi = R(a, b)$ be a role assertion. A weakened relation assertion ϕ_{weak} of ϕ is defined as $\phi_{weak} = \top_R(a, b)$ or $\phi_{weak} = \phi$, where $\top_R$ is interpreted as $\top_R^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for each interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. Let $\phi = C(a)$ be a concept assertion. A weakened concept assertion ϕ_{weak} of ϕ is defined recursively as follows:*

1) if $C = A$ or $\neg A$ for a concept name A, then $\phi_{weak} = \top(a)$ or $\phi_{weak} = \phi$,
2) if $C = \exists R.D$, then $\phi_{weak} = \top(a)$ or $\phi_{weak} = \phi$,
3) if $C = \forall R.D$, then $\phi_{weak} = \forall R.(D \sqcup \{b_1, ..., b_n\})(a)$ or $\top(a)$,
4) if $C = \{b\}$, where b is an individual name, then $\phi_{weak} = \top(a)$ or $\phi_{weak} = \phi$,
5) if $C = C_{i1} \sqcup ... \sqcup C_{im}$, where C_{ij} is either an atomic concept or the negation of an atomic concept, then $\phi_{weak} = ((C_{i1})_{weak} \sqcup ... \sqcup (C_{im})_{weak})(a)$[1] if $(C_{ij})_{weak} \not\equiv \top$ for all j and $\phi_{weak} = \top(a)$ otherwise,

[1] According to 1), 2), 3), and 4), we have $(C_{ij})_{weak} = \top$ or C_{ij} if C_{ij} is either a concept name or the negation of a concept name or of the form $\exists R.D$ or $\{b\}$, and $(C_{ij})_{weak} = \forall R.(D \sqcup \{b_1, ..., b_n\})$ if C_{ij} is of the form $\forall R.D$.

Let us explain the part 5) of Definition 10. Since the concept of ϕ is in disjunctive form, if there exists a C_{ij} such that $(C_{ij})_{weak} \equiv \top$, then $C \equiv \top$. That is, the weakening of a disjunct concept of ϕ may influence the weakening of other disjuncts. When weakening a role assertion, we introduce the top role. However, in implementation, the top role does not exist in the resulting knowledge base because the role assertion is simply deleted if the role name is weakened into the top role. In this paper, we only consider the refinement of the weakening of ABox assertions. Similarly, we can also refine the weakening of TBox axioms.

We next define the degree of a weakened assertion.

Definition 11. *Let $\phi = R(a,b)$, then $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top_R(a,b)$ and 0 otherwise. Let $\phi = C(a)$, then $d(\phi)$ is defined recursively as follows:*
1) if $C = A$ or $\neg A$ for a concept name A, then $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top(a)$ and 0 otherwise,
2) if $C = \exists R.C$, then $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top(a)$ and 0 otherwise,
3) if $C = \forall R.D$, then $d(\phi_{weak}) = n$ if $\phi_{weak} = \forall R.(D \sqcup \{b_1, ..., b_n\})(a)$ and $+\infty$ otherwise,
4) if $C = \{b\}$, where b is an individual name, then $d(\phi_{weak}) = 1$ if $\phi_{weak} = \top(a)$ and 0 otherwise,
5) if $C = C_{i1} \sqcup ... \sqcup C_{im}$, where C_{ij} is either an atomic concept or the negation of an atomic concept, then $d(\phi_{weak}) = max\{d(((C_{ij})_{weak})(a)) : j = 1, ..., m\}$,

In part 5) of Definition 11, we use *max* (instead of sum) to determine the degree of an assertion in "disjunction" form. This definition agrees with the semantic interpretations of disjunction in many logics such as fuzzy logic and possibilistic logic.

We call the weakened base obtained by applying weakening of GCIs in Definition 4 and weakening of assertions in Definition 10 as a refined weakened base. We then replace the weakened base by the refined weakened base in Definition 7 and get a new revision operator, which we call a refined weakening-based revision operator which is denote by $\circ_{rw}$. Let us go back to Example 2 again. According to our discussion before, $\forall hasChild.Rich\ Human(Bob)$ is the only assertion axiom involved in the conflict in K and John is the only *exception* which makes $\forall hasChild.RichHuman(Bob)$ in conflict with K', so $K \circ_{rw} K' = \{\forall hasChild.(RichHuman \sqcup \{John\})(Bob), hasChild(Bob, Mary), RichHuman(Mary), hasChild(Bob, Tom), hasChild(Bob, John), \neg RichHuman(John)\}$. We can then infer that RichHuman(Tom) from $K \circ_{rw} K'$.

We consider another example. Let $K = \{((\forall R.C) \sqcup D)(a), R(a,b)\}$ and $K' = \{\neg D(a), \neg C(b)\}$, where C and D are concept names. Clearly, $K \cup K'$ is inconsistent. We can either weaken $((\forall R.C) \sqcup D)(a)$ or $R(a,b)$ to restore consistency. To weaken $R(a,b)$, we can simply delete it, i.e. its weakening has the form $\top_R(a,b)$. We have $d(\top_R(a,b)) = 1$. For $\phi = ((\forall R.C) \sqcup D)(a)$, we should weaken $\forall R.C$ instead of D. This is because if we weaken D to $\top$ then $(\forall R.C) \sqcup D$ also needs to be weakened to $\top$. In this case, we have $d(\phi_{weak}) = +\infty$. In contrast, if we weaken $(\forall R.C) \sqcup D$ to $(\forall R.(C \sqcup \{b\})) \sqcup D$, then D does not need to be weakened. In this case, we have $d((\forall R.(C \sqcup \{b\})) \sqcup D)(a)) = 1$ and $d(\phi_{weak}) = 1$. Therefore, there

are two weakened bases of K *w.r.t* K', i.e. $K_1 = \{((\forall R.(C \sqcup \{b\})) \sqcup D)(a), R(a,b)\}$ and $K_2 = \{((\forall R.C) \sqcup D)(a)\}$.

To give a semantic explanation of the refined weakening-based revision operator, we need to define a new ordering between interpretations.

Definition 12. *Let $\mathcal{W}$ be a non-empty set of interpretations and $\mathcal{I} \in \mathcal{W}$, ϕ a DL axiom, and K a DL knowledge base. If ϕ is a concept assertion, then the number of ϕ-exceptions for $\mathcal{I}$ is defined recursively as follows:*

1) if $\phi = A(a)$ or $\neg A(a)$ for a concept name A, then $e_r^{\phi}(\mathcal{I}) = 0$ if $\mathcal{I} \models \phi$ and 1 otherwise,

2) if $\phi = \exists R.C(a)$, then $e_r^{\phi}(\mathcal{I}) = 0$ if $\mathcal{I} \models \phi$ and 1 otherwise,

3) If ϕ is an assertion of the form $\forall R.C(a)$, the number of ϕ-exceptions for $\mathcal{I}$ is:

$$e_r^{\phi}(\mathcal{I}) = \begin{cases} |R^{\mathcal{I}}(a^{\mathcal{I}}) \cap (\neg C^{\mathcal{I}})| & \text{if } R^{\mathcal{I}}(a^{\mathcal{I}}) \cap (\neg C^{\mathcal{I}}) \text{ is finite} \\ \infty & \text{otherwise,} \end{cases} \tag{2}$$

where $R^{\mathcal{I}}(a^{\mathcal{I}}) = \{b \in \Delta^{\mathcal{I}} : (a^{\mathcal{I}}, b) \in R^{\mathcal{I}}\}$.

4) If $\phi = \{b\}(a)$, where b is an individual name, then $e_r^{\phi}(\mathcal{I}) = 0$ if $\mathcal{I} \models \phi$ and 1 otherwise,

5) $\phi = (C_{i1} \sqcup ... \sqcup C_{im})(a)$, where C_{ij} is either an atomic concept or the negation of an atomic concept, then $e_r^{\phi}(\mathcal{I}) = max\{e_r^{C_{ij}(a)}(\mathcal{I}) : j = 1, ..., m\}$.

If ϕ is a role assertion, then $e_r^{\phi}(\mathcal{I}) = 0$ if $\mathcal{I} \models \phi$ and 1 otherwise.

If ϕ is a GCI of the form $C \sqsubseteq D$, the number of ϕ-exceptions for $\mathcal{I}$ is:

$$e_r^{\phi}(\mathcal{I}) = \begin{cases} |C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}})| & \text{if } C^{\mathcal{I}} \cap (\neg D^{\mathcal{I}}) \text{ is finite} \\ \infty & \text{otherwise.} \end{cases} \tag{3}$$

The number of K-exceptions for $\mathcal{I}$ is $e_r^K(\mathcal{I}) = \Sigma_{\phi \in K} e_r^{\phi}(\mathcal{I})$. The refined ordering $\preceq_{r,K}$ on $\mathcal{W}$ is: $\mathcal{I} \preceq_{r,K} \mathcal{I}'$ iff $e_r^K(\mathcal{I}) \leq e_r^K(\mathcal{I}')$, for $\mathcal{I}' \in \mathcal{W}$.

The following proposition gives the semantic interpretation of the refined weakening-based revision operator.

Proposition 3. *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. $\circ_{rw}$ is the refined weakening-based revision operator. We then have*

$$M(K \circ_{rw} K') = min(M(K'), \preceq_{r,K}).$$

Proposition 3 says that the refined weakening-based operator can be accomplished with minimal change. The proof is similar to that of Proposition 1.

Proposition 4. *Let K be a consistent DL knowledge base. K' is a newly received DL knowledge base. We then have*

$$M(K \circ_{rw} K') \subseteq M(K \circ_w K').$$

By Example 3, $K \circ_{rw} K'$ and $K \circ_w K'$ are not equivalent. Thus, we have shown that the resulting knowledge base of the refined weakening-based revision contains more information than that of the weakening-based revision. However, the

refined weakening-based revision need to convert every ABox assertion to its conjunctive normal form. In some cases this conversion can lead to an exponential explosion of the size of the ABox assertion. So the sizes of the revised DL knowledge bases of the refined weakening-based operator are exponentially larger than those of the weakening-based operator in the worst case.

The refined weakening-based revision operator is still not AGM compliant.

Proposition 5. *Given two DL knowledge bases K and K'. The refined weakening-based revision operator is not AGM-compliant but it is quasi-AGM compliant.*

5 Related Work

The importance of applying AGM theory on belief change to terminological systems has not been fully recognized until recent years. In his book [20], Nebel considered the revision problem in terminological logics in 1990. He proposed some revision operators based on several existing approaches on modification of a terminological knowledge base. When defining his revision operator, he presumed that the terminological knowledge is more relevant than the assertional knowledge. Recently, some work has been done to analyze the feasibility of applying AGM theory on belief change to DLs [16, 7, 8]. However, none of them considers the explicit construction of a revision operator. Furthermore, they did not consider the application of AGM postulates for revision in DLs where knowledge bases instead of knowledge sets are considered. The work in [16, 8] is based on the *coherence model*, i.e. both the original and the revised knowledge bases should be knowledge sets which are knowledge bases closed under logical consequence. In [7], Fuhrmann's postulates for knowledge base contraction is generalized to DLs. One may wonder if we can establish the relationship between revisions and contractions via the Levi and Harper identities. However, the problem is that Levi and Harper identities are not applicable in DLs [8]. In [19], some revision operators were proposed for revising a *stratified* DL knowledge base. The semantic aspects of these revision operators are also considered. To define their operators, an extra expression in DLs, called *cardinality restrictions* on concepts, is necessary. In contrast, our operators are based on nominals. Since cardinality restrictions can be encoded as nominals, our revision operators can be seen as a refinement of the revision operators in [19]. In [14], a general framework for reasoning with inconsistent ontologies was given based on *concept relevance*. A problem with their framework is that they do not consider the structure of DL language. For example, when a GCI is in conflict in a DL knowledge base, it is deleted to restore consistency. Our work is also related to the work in [1], where Reiter's default logic is embedded into terminological representation formalisms. In their paper, conflicting information is also treated as *exceptions*. To deal with conflicting default rules, they instantiated each rule using individuals appearing in the ABox and applied two existing default reasoning methods to compute all extensions. This instantiation step is not necessary for our revision operators. Furthermore, in [1], the resolution of conflicting ABox assertions was not considered. This work is also related to the work on updating DL ABoxes in [15]. They

showed that in any standard DL in which nominals and the "@" constructor are not expressible, updated ABoxes cannot be expressed. They only consider a simple form of ABox update where the update information contains possibly negated ABox assertions that involve only atomic concepts and roles.

6 Conclusions and Further Work

In this paper, we have discussed the problem of applying AGM theory of belief revision to DLs. We first generalized the reformulated AGM postulates for revision to DLs. Then two revision operators were proposed by weakening assertion axioms and GCIs. We showed that both revision operators satisfy the generalized postulates and capture some notions of minimal change.

Several problems are left as further work. First, none of our revision operators is AGM compliant, that is, they do not satisfy (G4). We are looking for a revision operator satisfying all the AGM postulates. Second, to implement our revision operators, an important problem is to detect GCIs and and assertions which are responsible for the conflict. Some existing techniques on debugging of unsatisfiable classes (such as [25, 22]) can be adopted or generalized to deal with this problem. We will develop tableaux-based algorithms for implementing our revision operators. Based on the results in [25], it is expected that the computational complexity of our operators may not increase the complexity of consistency checking in the DL under consideration.

Acknowledgement

We would like to thank the anonymous reviewers for their useful comments which have helped us to improve the quality of this paper.

References

1. F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms, *Journal of Automated Reasoning*, 14(1):149-180, 1995.
2. F. Baader, M. Buchheit, and B. Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88:195-213, 1996.
3. F. Baader, D.L. McGuiness, D. Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, implementation and application*, Cambridge University Press, 2003.
4. S. Benferhat, and R.E. Baida. A stratified first order logic approach for access control. *International Journal of Intelligent Systems*, 19:817-836, 2004.
5. M. Dalal. Investigations into a theory of knowledge base revision: Preliminary report, *Proc. of AAAI'88*, 3-7, 1988.
6. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web, *Scientific American*, 284(5):3443, 2001.
7. G. Flouris, D. Plexousakis and G. Antoniou. Generalizing the AGM postulates: preliminary results and applications. In *Proc. of NMR'04*, 171-179, 2004.

8. G. Flouris, D. Plexousakis and G. Antoniou. On applying the AGM theory to DLs and OWL, In *Proc. of ISWC'05*, 216-231, 2005.
9. P. Gärdenfors, *Knowledge in Flux-Modeling the Dynamic of Epistemic States*, The MIT Press, Cambridge, Mass, 1988.
10. P. Haase and L. Stojanovic. Consistent evolution of OWL ontologies. In *Proc. of 2nd European Semantic Web Conference (ESWC'05)*, 182-197, 2005.
11. P. Haase, F. van Harmelen, Z. Huang, H. Stuckenschmidt, and Y. Sure. A framework for handling inconsistency in changing ontologies, In *Proc. of ISWC'05, LNCA3729*, 353-367, 2005.
12. S.O. Hansson. In denfence of base contraction. *Synthese*, 91: 239-245, 1992.
13. I. Horrocks, and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic, In *Proc. of IJCAI'01*, 199-204, 2001.
14. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies, In *Proc. of IJCAI'05*, 254-259, 2005.
15. H. Liu, C. Lutz, M. Miličić, and F. Wolter. Updating description logic ABoxes. In *Proc. of KR'06*, 2006.
16. S.H. Kang and S.K. Lau. Ontology revision using the concept of belief revision. In *Proc. of 8th International Conference on Knowledge-base Intelligent Information and Engineering Systems (KES'04)*, 261-267, 2004.
17. H. Katsuno and A.O. Mendelzon. Propositional Knowledge Base Revision and Minimal Change, *Artificial Intelligence*, 52(3): 263-294, 1992.
18. C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains, *Journal of Artificial Intelligence Research*, 23:667-726, 2005.
19. T. Meyer, K. Lee, and R. Booth. Knowledge integration for description logics, In *Proc. of AAAI'05*, 645-650, 2005.
20. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, LNAI 422, Springer Verlag, Berlin, Heidelberg, New York, 1990.
21. B. Nebel. *What is Hybrid in Hybrid Representation and Reasoning Systems?*, In F. Gardin and G. Mauri and M. G. Filippini, editors, *Computational Intelligence II: Proc. of the International Symposium Computational Intelligence 1989*, North-Holland, Amsterdam, 217-228, 1990.
22. B. Parsia, E. Sirin and A. Kalyanpur. Debugging OWL ontologies, In *Proc. of WWW'05*, 633-640, 2005.
23. G. Qi, W. Liu and D.A. Bell. Knowledge base revision in description logics. Available at http://www.cs.qub.ac.uk/ G.Qi/qlb06e.pdf
24. A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141-176, 1994.
25. S. Schlobach, and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies, In *Proc. of IJCAI'2003*, 355-360, 2003.
26. M. Schmidt-Schauß, and G. Smolka. Attributive Concept descriptions with complements, *Artificial Intelligence*, 48:1-26, 1991.
27. L. Stojanovic. *Methods and Tools for Ontology Evolution*, PhD thesis, University of Karlsruhe, 2004.

Incomplete Knowledge in Hybrid Probabilistic Logic Programs

Emad Saad

College of Computer Science and Information Technology
Abu Dhabi University
Abu Dhabi, UAE
emad.saad@adu.ac.ae

Abstract. Although negative conclusions are presented implicitly in Normal Hybrid Probabilistic Programs (NHPP) [26] through the closed world assumption, representing and reasoning with explicit negation is needed in NHPP to allow the ability to reason with incomplete knowledge. In this paper we extend the language of NHPP to explicitly encode classical negation in addition to non-monotonic negation. The semantics of the extended language is based on the answer set semantics of traditional logic programming [9]. We show that the proposed semantics is a natural extension to the answer set semantics of traditional logic programming [9]. In addition, the proposed semantics is reduced to stable probabilistic model semantics of NHPP [26]. The importance of that is computational methods developed for NHPP can be applied to the proposed language. Furthermore, we show that some commonsense probabilistic knowledge can be easily represented in the proposed language.

1 Introduction

Hybrid Probabilistic Programs (HPP) [25] is a probabilistic logic programming framework that modifies the original Hybrid Probabilistic Programming framework of [5], and generalizes and modifies the *probabilistic annotated logic programming framework*, originally proposed in [17] and further extended in [18]. HPP [25] enables the user to *explicitly* encode his/her knowledge about the type of dependencies existing between the probabilistic events being described by the programs. In addition, it allows the ability to encode the user's knowledge about how to combine the probabilities of the same event derived from different rules. The semantics of HPP [25], intuitively, captures the probabilistic reasoning according to how likely are the various events to occur. It was shown that the HPP [25] framework is more suitable for reasoning and decision making tasks. In addition, it subsumes Lakshmanan and Sadri's [12] probabilistic implication-based framework, as well as, it is a natural extension of traditional logic programming. As a step towards enhancing its reasoning capabilities, the framework of HPP was extended to cope with non-monotonic negation [26] by introducing the notion of Normal Hybrid Probabilistic Programs (NHPP) and providing two different semantics namely; stable probabilistic model semantics and well-founded probabilistic model semantics. It was shown in [26] that the relationship

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 399–412, 2006.

between the stable probabilistic model semantics and the well-founded probabilistic model semantics *preserves* the relationship between the stable model semantics and the well-founded semantics for normal logic programs [7]. More importantly, the stable probabilistic models semantics *naturally extends* the stable model semantics [8] of normal logic programs as well as the well-founded probabilistic model semantics *naturally extends* the well-founded semantics [7] of normal logic programs. A consequence of that is efficient algorithms and implementations for computing those semantics can be developed by extending the existing efficient algorithms and implementations for computing the stable model semantics and the well-found semantics for normal logic programs, e.g., SMODELS [21].

An important limitation of the language of NHPP compared to traditional logic programming [9] is its inability to represent and reason directly in the presence of classical negation to cope with incomplete knowledge. This is because HPP [25] and NHPP [26] allow the *closed world assumption* in defining their semantics. Therefore, any event represented by a program in either HPP or NHPP has an associated probability interval (probability interval represents the bounds on the degree of belief a rational agent has about the truth of an event.) This means that events that cannot be derived from the facts and rules in a program are assigned the probability interval $[0, 0]$, by default, which represent the negative conclusions. Events that can be derived from the program are assigned probability intervals other than $[0, 0]$, which represent the positive conclusions. However, a third possibility, which is *unknown* or *undecidable*, is possible which represents information incompleteness. The reason for that is assuming that non-derivable events have the probability interval $[0, 0]$ could lead to serious implications.

Consider a medical doctor who treats his/her patient from a certain disease (di) by taking specific medication (med) for that disease. The doctor knows that if the patient took this medication he will be recovered. But the doctor also knows that the patient is suffering from a heart disease and taking that medication could affect the function of his heart and lead to death, although the medication is very effective. So that the doctor can give the medication to the patient with probability $[0.87, 0.95]$ if there are no side effects of the medication on the heart with probability $[0.85, 0.85]$. This situation can be represented as an NHPP program as follows:

$$give(di, med) : [0.87, 0.95] \leftarrow not\ (effect(med, heart) : [0.15, 0.15]).$$

If our knowledge regarding the side effects of the medication on the heart is incomplete because they might have not been yet clinically proven, then the medication should not be given to this specific patient, otherwise, he would probably die. The current semantics of NHPP allows us to assume that probability interval of the side effects of the medication on the heart is $[0, 0]$, which is strictly less than $[0.15, 0.15]$, and hence, medication is given to the patient, although, the program has no enough knowledge to assume the contrary.

We propose to overcome this limitation by extending the language of NHPP to explicitly allow classical negation (or explicit negation) as well as non-monotonic

negation (negation-as-failure) *not*, by introducing the notion of *Extended Hybrid Probabilistic Programs (EHPP)*. The semantics of EHPP is based on the answer set semantics of traditional logic programming [9] and employs the *Open World Assumption*. We show that some commonsense probabilistic knowledge can be easily represented in the proposed language. We show that the proposed semantics is a natural extension to the answer set semantics [9]. Moreover, we show that the proposed semantics is reduced to the stable probabilistic model semantics of NHPP [26]. The importance of that is computational methods developed for NHPP can be applied to the language of EHPP. This paper is organized as follows. Sections 2 and 3 describe the syntax and declarative semantics of EHPP respectively. The probabilistic answer set semantics for EHPP is presented in section 4. Finally, related work and conclusions with some perspectives are given in sections 5 and 6 respectively.

2 Syntax

In this section we define the syntax of *Extended Hybrid Probabilistic Programs (EHPP)*, which are hybrid probabilistic logic programs with classical and non-monotonic negation. In the following two subsections we review the basic notions associated with EHPP [5, 26].

2.1 Probabilistic Strategies

Let $C[0,1]$ denotes the set of all closed intervals in $[0,1]$. In the context of EHPP, probabilities are assigned to primitive events (atoms) and compound events (conjunctions or disjunctions of atoms) as intervals in $C[0,1]$. Let $[\alpha_1,\beta_1],[\alpha_2,\beta_2] \in C[0,1]$. Then the *truth order* asserts that $[\alpha_1,\beta_1] \leq_t [\alpha_2,\beta_2]$ iff $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$. The set $C[0,1]$ and the relation $\leq_t$ form a complete lattice. In particular, the join ($\oplus_t$) operation is defined as $[\alpha_1,\beta_1] \oplus_t [\alpha_2,\beta_2] = [max(\alpha_1,\alpha_2), max(\beta_1,\beta_2)]$ and the meet ($\otimes_t$) is defined as $[\alpha_1,\beta_1] \otimes_t [\alpha_2,\beta_2] = [min(\alpha_1,\alpha_2), min(\beta_1,\beta_2)]$ w.r.t. $\leq_t$. The type of dependency among the primitive events within a compound event is described by *probabilistic strategies* [5], which are explicitly selected by the user. We call ρ, a pair of functions $\langle c, md\rangle$, a probabilistic strategy (p-strategy), where $c : C[0,1] \times C[0,1] \rightarrow C[0,1]$, the *probabilistic composition function*, which is *commutative*, *associative*, *monotonic* w.r.t. $\leq_t$, and meets the following *separation* criteria: there are two functions c_1, c_2 such that $c([\alpha_1,\beta_1],[\alpha_2,\beta_2]) = [c_1(\alpha_1,\alpha_2), c_2(\beta_1,\beta_2)]$. Whereas, $md : C[0,1] \rightarrow C[0,1]$ is the *maximal interval function*. The maximal interval function md of a certain p-strategy returns an estimate of the probability range of a primitive event, e, from the probability range of a compound event that contains e. The composition function c returns the probability range of a conjunction (disjunction) of two events given the ranges of its constituents. For convenience, given a multiset of probability intervals $M = \{\!\{[\alpha_1,\beta_1],\ldots,[\alpha_n,\beta_n]\}\!\}$, we use cM to denote $c([\alpha_1,\beta_1], c([\alpha_2,\beta_2],\ldots,c([\alpha_{n-1},\beta_{n-1}],[\alpha_n,\beta_n]))\ldots)$. According to the type of combination among events, p-strategies are classified into *conjunctive*

p-strategies and *disjunctive* p-strategies. Conjunctive (disjunctive) p-strategies are employed to compose events belonging to a conjunctive (disjunctive) formula (please see [5, 25] for the formal definitions).

2.2 Language Syntax

In this subsection, we describe the syntax of EHPP. Let L be an arbitrary first-order language with finitely many predicate symbols, constants, and infinitely many variables. Function symbols are disallowed. In addition, let $S = S_{conj} \cup S_{disj}$ be an arbitrary set of p-strategies, where S_{conj} (S_{disj}) is the set of all conjunctive (disjunctive) p-strategies in S. The Herbrand base of L is denoted by B_L. A literal is either an atom a or the negation of an atom $\neg a$, where $\neg$ is the classical negation. We denote the set of all literals in L by Lit. More formally, $Lit = \{a | a \in B_L\} \cup \{\neg a | a \in B_L\}$. An *annotation* denotes a probability interval and it is represented by $[\alpha_1, \alpha_2]$, where α_1, α_2 are called annotation items. An *annotation item* is either a constant in $[0, 1]$, a variable (*annotation variable*) ranging over $[0, 1]$, or $f(\alpha_1, \ldots, \alpha_n)$ (called *annotation function*) where f is a representation of a computable total function $f : ([0, 1])^n \rightarrow [0, 1]$ and $\alpha_1, \ldots, \alpha_n$ are annotation items. The building blocks of the language of EHPP are *hybrid basic formulae*. Let us consider a set of literals $l_1, \ldots, l_n$ and the p-strategies ρ and ρ'. Then $l_1 \wedge_\rho \ldots \wedge_\rho l_n$ and $l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n$ are called *hybrid basic formulae*. A *hybrid literal* is a hybrid basic formula $l_1 \wedge_\rho \ldots \wedge_\rho l_n (l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n)$ or the negation of hybrid basic formula $\neg(l_1 \wedge_\rho \ldots \wedge_\rho l_n)(\neg(l_1 \vee_{\rho'} \ldots \vee_{\rho'} l_n))$. $bf_S(Lit)$ is the set of all ground hybrid literals formed using distinct literals from Lit and p-strategies from S. Note that any hybrid basic formula F can be represented in terms of another hybrid basic formula G such that $F = \neg G$, since $\neg\neg a = a$, $(a_1 \wedge_\rho a_2) = \neg(\neg a_1 \vee_\rho \neg a_2)$ and $(a_1 \vee_{\rho'} a_2) = \neg(\neg a_1 \wedge_{\rho'} \neg a_2)$ and $\wedge_\rho, \vee_\rho, \vee_{\rho'}$, and $\wedge_{\rho'}$ are associative and commutative. An annotated hybrid basic formula is an expression of the form $F : \mu$ where F is a hybrid basic formula and μ is an annotation. An *annotated hybrid literal* is an annotated *positive* hybrid basic formula $F : \mu$ or an annotated *negative* hybrid basic formula $(\neg F) : \mu$.

Definition 1 (E-rules). *An extended hybrid probabilistic rule (E-rule) is an expression of the form*

$$l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n)$$

where l is a literal, L_i $(1 \leq i \leq n)$ are hybrid literals, and μ, μ_i $(1 \leq i \leq n)$ are annotations.

The intuitive meaning of an E-rule is that, if for each $L_i : \mu_i$ $(1 \leq i \leq m)$, L_i is true with probability interval at least μ_i and for each *not* $(L_j : \mu_j)$ $(m + 1 \leq j \leq n)$, it is not *known* that L_j is true with probability interval at least μ_j, then l is true with probability interval at least μ.

Definition 2 (E-programs). *An extended hybrid probabilistic program over S (*E-program*) is a pair $P = \langle R, \tau \rangle$, where R is a finite set of E-rules with p-strategies from S, and τ is a mapping $\tau : Lit \rightarrow S_{disj}$.*

The mapping τ in the above definition associates to each literal l a disjunctive p-strategy that will be employed to combine the probability intervals obtained from different E-rules having l in their heads. An E-program is ground if no variables appear in any of its rules.

3 Satisfaction and Models

In this section, we define the declarative semantics of EHPP. We define the notions of interpretations, models, and satisfaction of E-programs. The notion of a probabilistic model (p-model) is based on *hybrid formula function*.

Definition 3. *A hybrid formula function is a mapping* $h : bf_S(Lit) \rightarrow C[0,1]$ *that satisfies the following conditions:*

- *Commutativity:* $h(L_1 *_\rho L_2) = h(L_2 *_\rho L_1)$, $* \in \{\wedge, \vee\}$, $\rho \in S$
- *Composition:* $c_\rho(h(L_1), h(L_2)) \leq_t h(L_1 *_\rho L_2)$, $* \in \{\wedge, \vee\}$, $\rho \in S$
- *Decomposition. For any hybrid basic formula* L, $* \in \{\wedge, \vee\}, \rho \in S$, *and* $M \in bf_S(Lit)$: $md_\rho(h(L *_\rho M)) \leq_t h(L)$.

If the probability of an event e is $pr(e)$, then the probability of $\neg e$ is $pr(\neg e) = 1 - pr(e)$. This can be generalized to probability intervals as follows. Given $pr(e) = [\alpha_1, \alpha_2]$ is the probability interval of an event e then the probability interval of the event $\neg e$ is given by $pr(\neg e) = [1,1] - pr(e) = [1 - \alpha_2, 1 - \alpha_1]$. Note that Definition 3 does not restrict the assignment of probability intervals to formulae in hybrid formula functions. However, since we allow both an event and its negation to be defined in hybrid formula functions, more conditions need to be imposed on hybrid formula functions to ensure their consistency. This can be characterized by the following definition.

Definition 4. *A total (partial) hybrid formula function* h *is inconsistent if there exists* $F, \neg F \in bf_S(Lit)$ $(F, \neg F \in (dom(h))$ *such that* $h(\neg F) \neq [1,1] - h(F)$.

Definition 4 states that a hybrid formula function h is consistent if for any $F, \neg F \in dom(h)$ we have $h(\neg F) = [1,1] - h(F)$.

Definition 5. *We say a set* C, *a subset of* Lit, *is a set of consistent literals if there is no pair of complementary literals* a *and* $\neg a$ *belonging to* C. *Similarly, a consistent set of hybrid literals* C^* *is a subset of* $bf_S(Lit)$ *such that there is no pair of complementary hybrid literals* F *and* $\neg F$ *belonging to* C^*.

Definition 6. *A consistent hybrid formula function* h *is either not inconsistent or maps a consistent set of hybrid literals* C^* *to* $C[0,1]$.

A consistent hybrid formula function is a partial or total hybrid formula function. The notion of truth order can be employed to hybrid formula functions (partial or total). Given hybrid formula functions h_1 and h_2, we say

$$(h_1 \leq_o h_2) \Longrightarrow (dom(h_1) \subseteq dom(h_2) \text{ and } \forall L \in dom(h_1)\ h_1(L) \leq_t h_2(L)).$$

The set of all hybrid formula functions, HFF, and the order $\leq_o$ form a complete lattice. The meet $\otimes_o$ and the join $\oplus_o$ operations are defined respectively as follows.

Definition 7. *Let h_1 and h_2 be two hybrid formula functions. The meet $\otimes_o$ and join $\oplus_o$ operations corresponding to the partial order $\leq_o$ are defined respectively as:*

- $(h_1 \otimes_o h_2)(F) = h_1(F) \otimes_t h_2(F)$ *for all* $F \in (dom(h_1) \cap dom(h_2))$, *otherwise, undefined.*
- $(h_1 \oplus_o h_2)(F) = h_1(F) \oplus_t h_2(F)$ *for all* $F \in (dom(h_1) \cap dom(h_2))$,
  $(h_1 \oplus_o h_2)(F) = h_1(F)$ *for all* $F \in (dom(h_1) \setminus dom(h_2))$, *and*
  $(h_1 \oplus_o h_2)(F) = h_2(F)$ *for all* $F \in (dom(h_2) \setminus dom(h_1))$, *otherwise, undefined.*

Definition 8. *A probabilistic interpretation (p-interpretation) of an E-program P is a (partial or total) hybrid formula function.*

The satisfiability of an E-program is based on the satisfaction of its E-rules.

Definition 9 (Probabilistic Satisfaction). *Let $P = \langle R, \tau \rangle$ be a ground E-program, h be a p-interpretation, and*
$r \equiv l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n) \in R$. *Then*

- *h satisfies $L_i : \mu_i$ (denoted by $h \models L_i : \mu_i$) iff $L_i \in dom(h)$ and $\mu_i \leq_t h(L_i)$.*
- *h satisfies $not\ (L_j : \mu_j)$ (denoted by $h \models not\ (L_j : \mu_j)$) iff $L_j \in dom(h)$ and $\mu_j \not\leq_t h(L_j)$ or $L_j \notin dom(h)$.*
- *h satisfies $Body \equiv L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n)$ (denoted by $h \models Body$) iff $\forall (1 \leq i \leq m), h \models L_i : \mu_i$ and $\forall (m+1 \leq j \leq n), h \models not\ (L_j : \mu_j)$.*
- *h satisfies $l : \mu \leftarrow Body$ iff $h \models l : \mu$ or h does not satisfy $Body$.*
- *h satisfies P iff h satisfies every E-rule in R and for every literal $l \in dom(h)$, $c_{\tau(l)}\{\!\{\mu | l : \mu \leftarrow Body \in R \text{ and } h \models Body\}\!\} \leq_t h(l)$.*

Definition 10 (Models). *Let P be an E-program. A probabilistic model (*p-model*) of P is a p-interpretation h of P that satisfies P.*

We say that h is a minimal p-model of P if there is no p-model h' of P such that $h' <_o h$. An E-program without non-monotonic negation is simpler and has exactly one minimal p-model. The following results allow us to characterize the minimal (least) p-model (we call this least p-model *probabilistic answer set*) of an E-program without non-monotonic negation.

Proposition 1. *Let $P = \langle R, \tau \rangle$ be a ground E-program without non-monotonic negation, i.e. $n = m$ for each E-rule $r \in R$, and h_1, h_2 be two p-models of P. Then $h_1 \otimes_o h_2$ is also a p-model of P.*

Corollary 1. *Let P be a ground E-program without non-monotonic negation and let H_P be the set of all p-models of P. Then, $h_P = \otimes_o \{h | h \in H_P\}$ is the probabilistic answer set of P.*

However, it is possible to get the probabilistic answer set of an E-program P without non-monotonic negation and this probabilistic answer set is inconsistent. If this is the case, we say P is inconsistent. In other words, P is inconsistent if it has inconsistent probabilistic answer set. If P is inconsistent then LIT, where $LIT : bf_S(Lit) \rightarrow [1,1]$, is the probabilistic answer set of P. In this case every hybrid literal with probability interval $[1,1]$ follows from P. We adopt this view from the answer set semantics of traditional logic programming [9].

Example 1. Consider the following E-program $P = \langle R, \tau \rangle$ without non-monotonic negation, where R is

$$\begin{array}{ll} c : [0.35, 0.91] & \leftarrow a : [0, 0.11], b : [0.8, 0.99] \\ \neg c : [0, 0.21] & \leftarrow a : [0.1, 0.13], \neg b : [0.05, 0.08] \\ d : [0.12, 0.18] & \leftarrow c : [0.35, 0.65] \\ \neg d : [0.45, 0.55] & \leftarrow a : [0, 0.15], \neg b : [0.02, 0.22], \neg c : [0, 0.1] \\ \neg b : [0.15, 0.3] & \leftarrow \\ a : [0.1, 0.2] & \leftarrow \end{array}$$

and τ is any arbitrary assignment of disjunctive p-strategies. It is easy to verify that P has *unique* probabilistic answer set h where $h(a) = [0.1, 0.2], h(\neg b) = [0.15, 0.3], h(\neg c) = [0, 0.21], h(\neg d) = [0.45, 0.55]$.

Example 2. Consider the following E-program $P = \langle R, \tau \rangle$ where R is

$$\begin{array}{ll} b : [0.3, 0.4] & \leftarrow \neg a : [0.7, 0.8] \\ a : [0.1, 0.22] & \leftarrow b : [0.55, 0.7] \\ a : [0.2, 0.3] & \leftarrow \\ b : [0.4, 0.5] & \leftarrow \end{array}$$

and $\tau(a) = \tau(b) = pcd$ and $\tau(\neg b) = \tau(\neg a) = \pi$ where π is any arbitrary disjunctive p-strategy. The *pcd* denotes the disjunctive positive correlation p-strategy whose composition function is defined as: $c_{pcd}([\alpha_1, \beta_1], [\alpha_2, \beta_2]) = [\max(\alpha_1, \alpha_2), \max(\beta_1, \beta_2)]$. Then h where $h(a) = [0.2, 0.3], h(b) = [0.4, 0.5]$ is the probabilistic answer set of P.

Proposition 2. *Every E-program P without non-monotonic negation has unique probabilistic answer set h_P.*

Associated with each E-program P without non-monotonic negation, is an operator, T_P, called the *fixpoint operator*, which maps a p-interpretation to a p-interpretation.

Definition 11. *Let $P = \langle R, \tau \rangle$ be a ground E-program without non-monotonic negation, h be a p-interpretation, and HFF be the set of all hybrid formula functions. The fixpoint operator T_P is a mapping $T_P : HFF \rightarrow HFF$ which is defined as follows:*

1. *if l is a literal, $T_P(h)(l) = c_{\tau(l)}\ M_l$ where $M_l = \{\!\{\mu | l : \mu \leftarrow Body \in R$ such that $h \models Body\}\!\}$.*

2. $T_P(h)(L_1 \wedge_\rho L_2) = c_\rho(T_P(h)(L_1), T_P(h)(L_2))$ *where* $(L_1 \wedge_\rho L_2) \in bf_S(Lit)$ *and* $L_1, L_2, \in dom(T_P(h))$
3. $T_P(h)(L_1 \vee_{\rho'} L_2) = c_{\rho'}(T_P(h)(L_1), T_P(h)(L_2))$ *where* $(L_1 \vee_{\rho'} L_2) \in bf_S(Lit)$ *and* $L_1, L_2, \in dom(T_P(h))$.

If M_l is empty—i.e., there are no E-rules in P whose heads contain l such that their bodies are satisfied by h—then no probability interval is assigned to l. This means the probability interval of l is *unknown* with respect to h. Let us now proceed in the construction of the probabilistic answer set as repeated iteration of the fixpoint operator T_P.

Definition 12. *Let P be a ground E-program without non-monotonic negation. Then*

- $T_P \uparrow 0 = \emptyset$ *where $\emptyset$ is the empty set.*
- $T_P \uparrow \alpha = T_P(T_P \uparrow (\alpha - 1))$ *where α is a successor ordinal.*
- $T_P \uparrow \lambda = \oplus_o \{T_P \uparrow \alpha | \alpha < \lambda\}$ *where λ is a limit ordinal.*

Lemma 1. *The T_P operator is monotonic.*

The properties of the T_P operator guarantee the existence of a least fixpoint and its correspondence to the probabilistic answer set of E-programs without non-monotonic negation.

Proposition 3. *Let P be an E-program without non-monotonic negation and h be a p-interpretation. Then h is a p-model of P iff $T_P(h) \leq_o h$.*

Theorem 1. *Let P be an E-program without non-monotonic negation. Then, $h_P = lfp(T_P)$.*

Example 3. Let us reconsider the E-program P, without non-monotonic negation, described in Example 1. It is easy to see that the $lfp(T_P)$ assigns $[0.1, 0.2]$ to a, $[0.15, 0.3]$ to $\neg b$, $[0, 0.21]$ to $\neg c$, and $[0.45, 0.55]$ to $\neg d$.

4 Probabilistic Answer Set Semantics for E-Programs

In this section we define the *probabilistic answer sets* of E-programs (with non-monotonic negation), which extend the notion of answer sets for traditional logic programming [9]. The semantics is defined in two steps. First, we guess a probabilistic answer set h for a certain E-program P, then we define the notion of the probabilistic reduct of P with respect to h. The probabilistic reduct is an E-program without non-monotonic negation which has a unique probabilistic answer set. Second, we determine whether h is a probabilistic answer set for P. This is verified by determining whether h is the probabilistic answer set of the probabilistic reduct of P w.r.t. h.

Definition 13 (Probabilistic Reduct). *Let $P = \langle R, \tau \rangle$ be a ground E-program and h be a p-interpretation. The probabilistic reduct P^h of P w.r.t. h is $P^h =$*

$\langle R^h, \tau \rangle$ *where:*

$$R^h = \left\{ l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m \;\middle|\; \begin{array}{l} l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, \\ \quad not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n) \in R \text{ and} \\ \forall (m+1 \leq j \leq n),\ \mu_j \not\leq_t h(L_j) \text{ or } L_j \notin dom(h) \end{array} \right\}$$

The probabilistic reduct P^h is an E-program without non-monotonic negation. Therefore, its probabilistic answer set is well-defined. For any *not* $(L_j : \mu_j)$ in the body of $r \in R$ with $\mu_j \not\leq_t h(L_j)$ means that it is not known that the probability interval of L_j is at least μ_j given the available knowledge, and *not* $(L_j : \mu_j)$ is removed from the body of r. In addition, if $L_j \notin dom(h)$, i.e., L_j is undefined in h, then it is completely *not known (undecidable)* that the probability interval of L_j is at least μ_j. In this case, *not* $(L_j : \mu_j)$ is also removed from the body of r. Here we distinguish between the case where it is not known the probability of L_j is at least μ_j, because we have some but incomplete knowledge about the probability of L_j (by $\mu_j \not\leq_t h(L_j)$), and the case where we have entirely no knowledge about the probability interval of L_j (by $L_j \notin dom(h)$). If $\mu_j \leq_t h(L_j)$ then we know that the probability interval of L_j is at least μ_j and the body of r is not satisfied and r is trivially ignored.

Definition 14. *A p-interpretation h is a probabilistic answer set of an E-program P if h is the probabilistic answer set of* P^h.

The domain of a probabilistic answer set of an E-program represents an agent set of beliefs based on the knowledge represented by the E-program. However, the probability intervals associated to these beliefs represent the agent belief degrees on these beliefs. Intuitively, the probabilistic answer sets of an E-program are the possible sets of beliefs with associated beliefs degrees an agent might have. Note that E-programs without classical negation (normal hybrid probabilistic programs [26]), i.e., E-programs that contain no negative literals either in head or in the body of E-rules, have probabilistic answer sets with hybrid literals consisting of only atoms. In other words, the domain of probabilistic answer set in this case consists of positive hybrid basic formulae. Moreover, the definition of probabilistic answer sets coincides with the definition of stable probabilistic models defined in [26]. This implies that the probabilistic answer sets for a normal hybrid probabilistic program are equivalent to its stable probabilistic models. This means that the application of probabilistic answer set semantics to normal hybrid probabilistic programs is reduced to the stable probabilistic model semantics for normal hybrid probabilistic programs. However, there are a couple of main differences between the two semantics. A probabilistic answer set may be a partial p-interpretation, however, a stable probabilistic model is a total p-interpretation. In addition, each hybrid basic formula F with probability interval [0,0]— i.e. there is no proof that F has probability interval different from $[0, 0]$ or F is false by default— in a stable probabilistic model of a normal hybrid probabilistic program corresponds to the fact that the probability interval of F is unknown, and hence undefined, in its equivalent probabilistic answer set.

Proposition 4. *Let P be an E-program without classical negation. Then h is a probabilistic answer set for P iff h' is a stable probabilistic model of P, where $h(F) = h'(F)$ for $h'(F) \neq [0,0]$ and $h(F)$ is undefined for $h'(F) = [0,0]$.*

Proposition 4 shows that there is a simple reduction from E-programs to normal hybrid probabilistic programs. The importance of this is that, under the consistency condition, computational methods developed for normal hybrid probabilistic programs can be applied to extended hybrid probabilistic programs.

Example 4. In addition to the intuitive representation, the undesirable consequences due to the use of non-monotonic negation represented by $give(di, med) : [0.87, 0.95] \leftarrow not\ (effect(med, heart) : [0.15, 0.15])$, described in the introduction, can be eliminated by using classical negation instead. Therefore, by using the classical negation we get

$$give(di, med) : [0.87, 0.95] \leftarrow \neg\, effect(med, heart) : [0.85, 0.85].$$

Then, $give(di, med)$ can be concluded with probability interval $[0.87, 0.95]$ if no side effects of the medication on the heart ($\neg\, effect(med, heart)$) is concluded with probability interval at least $[0.85, 0.85]$.

Example 5. Suppose that we know a bird can fly with probability interval at least the probability range between 70% and 85% as long as it is not known that it is incapable of flying with probability interval at least the probability range from 30% to 35%. However, a bird is incapable of flying with probability interval at least the probability range from 48% to 65% if it is wounded with probability interval at least the probability range from 50% to 68%. Nevertheless, certainly, a bird is incapable of flying if it is a penguin. In addition, we also know that Tweety and Rocky are birds. Rocky is penguin, and there is a 70% to 100% chance that Tweety is injured. This can be represented by the following E-program $P = \langle R, \tau \rangle$ where R is

$$\begin{array}{ll}
fly(X) : [0.7, 0.85] & \leftarrow bird(X) : [1,1], not(\neg fly(X) : [0.3, 0.35]) \\
\neg fly(X) : [0.48, 0.65] & \leftarrow wounded(X) : [0.5, 0.68] \\
\neg fly(X) : [1,1] & \leftarrow penguin(X) : [1,1] \\
bird(tweety) : [1,1] & \leftarrow \\
wounded(tweety) : [0.7,1] & \leftarrow \\
bird(rocky) : [1,1] & \leftarrow \\
penguin(rocky) : [1,1] & \leftarrow
\end{array}$$

and τ is any arbitrary assignment of disjunctive p-strategies. P has only one probabilistic answer set h where $h(bird(tweety)) = [1,1], h(bird(rocky)) = [1,1], h(wounded(tweety)) = [0.7,1], h(penguin(rocky)) = [1,1], h(\neg fly(tweety)) = [0.48, 0.65], h(\neg fly(rocky)) = [1,1]$.

In the following we define the *immediate consequence operator* of E-programs and study its relationship to the probabilistic answer sets.

Definition 15. *Let $P = \langle R, \tau \rangle$ be a ground E-program and $h \in HFF$. The* immediate consequence operator T'_P *is a mapping $T'_P : HFF \rightarrow HFF$ defined as follows:*

1. $T'_P(h)(l) = c_{\tau(l)}\ M'_l$ *where*

$$M'_l = \left\{\!\!\left\{ \mu \;\middle|\; \begin{array}{c} l : \mu \leftarrow L_1 : \mu_1, \ldots, L_m : \mu_m, not\ (L_{m+1} : \mu_{m+1}), \ldots, not\ (L_n : \mu_n) \in R\ and \\ \forall (1 \leq i \leq m), h \models L_i : \mu_i\ and\ \forall (m+1 \leq j \leq n), h \models not\ (L_j : \mu_j) \end{array} \right\}\!\!\right\}$$

2. $T'_P(h)(L_1 \wedge_\rho L_2) = c_\rho(T'_P(h)(L_1), T'_P(h)(L_2))$ *where* $(L_1 \wedge_\rho L_2) \in bf_S(Lit)$ *and* $L_1, L_2 \in dom(T'_P(h))$.
3. $T'_P(h)(L_1 \vee_{\rho'} L_2) = c_{\rho'}(T'_P(h)(L_1), T'_P(h)(L_2))$ *where* $(L_1 \vee_{\rho'} L_2) \in bf_S(Lit)$ *and* $L_1, L_2 \in dom(T'_P(h))$.

It is easy to see that T'_P extends T_P to handle E-rules with non-monotonic negation, and hence, $T'_P = T_P$ for any E-program P without non-monotonic negation.

Theorem 2. *Let $P = \langle R, \tau \rangle$ be an E-program such that for every E-rule in R, $n = m$. Then $T'_P = T_P$.*

The operator T'_P is not monotonic w.r.t. $\leq_o$. This can be seen by the following result.

Proposition 5. *T'_P is not monotonic w.r.t. $\leq_o$.*

Example 6. Consider the E-program: $a : [0.2, 0.3] \leftarrow not\ (b : [0.6, 0.8])$. Let $h_1 = \emptyset$ be a p-interpretation. In addition, let h_2 be a p-interpretation that assigns $[0.65, 0.9]$ to b. Hence, $h_1 \leq_o h_2$. But $T'_P(h_1)$ assigns $[0.2, 0.3]$ to a and $T'_P(h_2) = \emptyset$. Thus, $T'_P(h_1) \not\leq_o T'_P(h_2)$

The following results establish the relationship between the T'_P operator and the probabilistic answer set semantics.

Lemma 2. *Let P be an E-program and h be a probabilistic answer set of P. Then $T'_P(h) = h$, i.e., h is a fixpoint of T'_P.*

Theorem 3. *Let P be an E-program and h be a probabilistic answer set of P. Then h is a minimal fixpoint of T'_P.*

It is worth noting that not every minimal fixpoint of T'_P is a probabilistic answer set for P. Consider the following E-program P.

Example 7. Let $P = \langle R, \tau \rangle$ where τ is arbitrary and R contains

$$a : [0.1, 0.33] \leftarrow not\ (a : [0.1, 0.33])$$
$$a : [0.1, 0.33] \leftarrow b : [1, 1]$$

It is easy to verify that the p-interpretation $h(a) = [0.1, 0.33]$ and $h(b) = [1, 1]$ is a minimal fixpoint of T'_P. However, P^h consists of $a : [0.1, 0.33] \leftarrow b : [1, 1]$ where $lfp(T_{P^h}) = \emptyset$. Hence, h is not a probabilistic answer set for P.

Let us show that the probabilistic answer set semantics generalizes the answer set semantics of extended logic programs in traditional logic programming [9]. An extended logic program P can be represented as an E-program $P' = \langle R, \tau \rangle$ where each extended rule

$$l \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \in P$$

can be encoded, in R, as an E-rule of the form

$$l : [1,1] \leftarrow l_1 : [1,1], \ldots, l_m : [1,1], not\ (l_{m+1} : [1,1]), \ldots, not\ (l_n : [1,1]) \in R$$

where $l, l_1, \ldots, l_m, l_{m+1}, \ldots, l_n$ are literals and $[1,1]$ represents the truth value *true*. τ is any arbitrary assignment of disjunctive p-strategies. We call the class of E-programs that consists only of E-rules of the above form as $EHPP_1$. The following result shows that extended logic programs [9] are subsumed by EHPP.

Proposition 6. *Let P be an extended logic program. Then S' is an answer set of P iff h is a probabilistic answer of $P' \in EHPP_1$ that corresponds to P where $h(l) = [1,1]$ iff $l \in S'$ and $h(l')$ is undefined iff $l' \notin S'$.*

5 Related Work

The problem of extending uncertain logic programming in general and probabilistic logic programming in particular with non-monotonic negation (negation-as-failure or default negation) has been extensively studied in the literature. A survey on these various approaches can be found in [26]. However, the main difference in this work is that we allow classical negation as well as non-monotonic negation to reason with incomplete knowledge, given the underlying semantics is the answer set semantics for traditional logic programming [9], which has not been addressed by the current work in probabilistic logic programming. The closest to our work is the work presented in [2]. In [2], an elegant way has been presented to reason with causal Bayesian nets by considering a body of logical knowledge, by using the answer set semantics of traditional logic programming [9]. Answer set semantics [9] has been used in [2] to emulate the possible world semantics. Probabilistic logic programs of [2] is expressive and straightforward and relaxed some restrictions on the logical knowledge representation part existed in similar approaches to Bayesian reasoning, e.g., [10, 16, 23, 24, 27]. Since [17, 18, 19, 5] provided a different semantical characterization to probabilistic logic programming, it was not clear that how these proposals relate to [2]. However, the work presented in this paper and [26], which are modification and generalization of the work presented in [17, 18, 19, 5], are closely related to [2]. The work presented in this paper strictly syntactically and semantically subsumes probabilistic logic programs of [2]. This can be easily argued by the fact that EHPP naturally extends traditional logic programming with answer set semantics [9], and probabilistic logic programs of [2] mainly rely on traditional logic programming with answer set semantics [9] as a knowledge representation

and inference mechanism for reasoning with causal Bayesian nets. This is true although EHPP does not allow disjunctions in the head of rules since it is easy to transform an extended disjunctive logic program into an equivalent extended logic program via a simple transformation [1]. In this sense, the comparisons established between [2] and the existing probabilistic logic programming approaches such as [10, 16, 23, 24, 27, 17, 18, 19, 5, 15, 4] also carry over to EHPP and these approaches. In addition, unlike [2], EHPP does not put any restriction on the type of dependency existing among events.

6 Conclusions and Future Work

We presented an extension to the language of normal hybrid probabilistic programs [26], called extended hybrid probabilistic programs, to allow classical negation, in addition to, non-monotonic negation. The extension is important to provide the capability of reasoning with incomplete knowledge. We developed a semantical characterization of the extended language, which relies on a probabilistic generalization of the answer set semantics, originally developed for extended logic programs [9]. We showed that the probabilistic answer set semantics naturally generalizes the answer set semantics for extended logic programs [9]. Furthermore, we showed that the proposed semantics is reduced to stable probabilistic model semantics of NHPP proposed in [26]. The importance of that is computational methods developed for NHPP can be applied to the language of EHPP. Moreover, we showed that some commonsense probabilistic knowledge can be easily represented in the proposed language.

A topic of future research is to extend the language of extended hybrid probabilistic programs to allow disjunctions of annotated literals in the heads of rules. In addition, we intend to investigate the computational aspects of the probabilistic answer set semantics—by developing algorithms and implementations for computing the proposed semantics. The algorithms and implementations we will develop will be based on appropriate extensions of the existing technologies for computing the answer semantics for extended logic programs.

References

1. C. Baral. *Knowledge representation, reasoning, and declarative problem solving.* Cambridge University Press, 2003.
2. C. Baral et al. Probabilistic reasoning with answer sets. *In 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, Springer Verlag, 2004.
3. C.V. Damasio et al. Coherent well-founded annotated logic programs. *LPNMR*, Springer, 1999.
4. A. Dekhtyar and I. Dekhtyar. Possible worlds semantics for probabilistic logic programs. *International Conference on Logic Programming*, 137-148, 2004.
5. A. Dekhtyar and V.S. Subrahmanian. Hybrid probabilistic program. *Journal of Logic Programming*, 43(3): 187-250, 2000.

6. M. Dekhtyar, A. Dekhtyar, and V. S. Subrahmanian. Hybrid Probabilistic Programs: Algorithms and Complexity. *In Proc. of UAI Conference*, pages 160-169, 1999.
7. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620-650, 1991.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. *ICSLP*, 1988, MIT Press.
9. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):363-385, 1991.
10. K. Kersting and L. De Raedt. Bayesian Logic Programs. In *Inductive LP*, 2000.
11. M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367, 1992.
12. L.V.S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Journal of Theory and Practice of Logic Programming*, 1(1):5-42, January 2001.
13. Y. Loyer and U. Straccia. The well-founded semantics in normal logic programs with uncertainty. *FLOPS*, 2002, Springer Verlag.
14. Y. Loyer and U. Straccia. The approximate well-founded semantics for logic programs with uncertainty. *In 28th International Symposium on Mathematical Foundations of Computer Science*, 2003.
15. T. Lukasiewicz. Probabilistic logic programming. *In 13th European Conference on Artificial Intelligence*, 388–392, 1998.
16. S. Muggleton. Stochastic logic programming. *In Proceedings of the 5th International Workshop on Inductive Logic Programming*, 1995.
17. R.T. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information & Computation*, 101(2), 1992.
18. R.T. Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probabilities in deductive databases. *ARJ*, 10(2), 1993.
19. R.T. Ng and V.S. Subrahmanian. Stable semantics for probabilistic deductive databases. *Information & Computation*, 110(1), 1994.
20. P. Nicolas, L. Garcia , and I. Stphan. Possibilistic stable models. *In International Joint Conference on Artificial Intelligence*, 2005.
21. I. Niemela and P. Simons. Efficient implementation of the well-founded and stable model semantics. *In Joint International Conference and Symposium on Logic Programming*, 289-303, 1996.
22. J. Pearl. *Causality.* Cambridge University Press, 2000.
23. D. Poole. The Independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2), 7-56, 1997.
24. D. Poole. Abducing through negation as failure: stable models within the independent choice logic. *Journal of Logic Programming*, Vol 44, 5-35, 2000.
25. E. Saad and E. Pontelli. Towards a more practical hybrid probabilistic logic programming framework.In *Practical Aspects of Declarative Languages*, 2005.
26. E. Saad and E. Pontelli. Hybrid probabilistic logic programs with non-monotonic negation. In *International Conference on Logic Programming.* Springer Verlag, 2005.
27. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. *International Conference on Logic Programming*, 431-445, 2004.

A Formal Analysis of KGP Agents

F. Sadri and F. Toni

Department of Computing, Imperial College London
{fs, ft}@doc.ic.ac.uk

Abstract. This paper contributes to the identification, formalisation and analysis of desirable properties of agent models in general and of the *KGP* model in particular. This model is specified in computational logic, and consequently lends itself well to formal analysis. We formalise three notions of welfare, in terms of goal achievement, progress, and reactive awareness, and we prove results related to these notions for KGP agents. These results broadly demonstrate the coherence of some of the design decisions in the KGP model, the need for some of its components for effectiveness in goal achievement, the extent to which the welfare of KGP agents can be shown to improve during their life-time, and the awareness of the agents of their reactions to changes in the environment.

1 Introduction

The use of logic to formalise and prove formal properties of agent models has been advocated by several researchers in the filed of agents. The KGP model of agency [9, 7] was designed with these aims in mind, as well as allowing agents with proactive and reactive behaviour in a dynamic environment. The KGP model is modular and allows for design of heterogeneous agents, each equipped with its own profile. Agents in the KGP model are equipped with knowledge bases, capabilities and transition rules that allow them to plan for their goals, make observations in the environment in which they are situated, update their beliefs, react to changes in their environment, communicate with other KGP agents, revise their states, and dynamically change their goals. The model has been described in detail in [9, 7] and compared with other models of agency, for example IMPACT [1], BDI [16], 3APL [6], AgentSpeak [15] and MINERVA [11]. All the components of the KGP model have been specified using computational logic. This was done to facilitate formalisation and verification of formal properties in addition to enabling a verifiable implementation of the model [4, 3]. In this paper we focus on the former, and propose three notions of agent welfare: *goal achievement*, referring to the achievement of goals held by an agent; *progress*, referring to how close an agent may be to achieving its goals; *reactive awareness*, referring to how aware an agent is of reactions that are necessary to its circumstances and environment.

Our notions of welfare amount to assessing how effective an agent is in achieving its goals, or at least in working towards achieving them, and in reacting to its decisions and environment. This work, therefore, is significant in allowing us

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 413–425, 2006.

to analyse the effectiveness of the KGP model. Also, a study of the environmental or internal conditions that would help, guarantee or hinder improvement of welfare, could help give guidelines to designers of agents.

In our analysis we mostly adopt a *subjective* approach to the notion of welfare, whereby, for example, achievement of a goal is assessed wrt the agent's beliefs (knowledge base). We briefly discuss, in sections 4 and 5, *objective* notions of welfare, wrt the agent's actual environment, rather than its perception of it.

The paper is structured as follows. In Section 2 we give the abstract agent model and its instantiation as the KGP model. In Section 3 we give preliminary definitions and results, used in the rest of the paper. In Sections 4, 5 and 6 we study the concepts of goal achievement, progress and reactive awareness. In Section 7 we conclude with some related work.

2 Agent Model

We will assume an agent model, generalising the KGP model, whereby agents are equipped with the following components:

- an *internal (or mental) state*, consisting of the agent's *beliefs*, *goals* and *plans*; goals and plan components have associated times and temporal constraints, inducing a partial order;
- a set of *reasoning capabilities*, reasoning with the information available in the agent state;
- a *Sensing capability*, allowing agents to observe their environment and actions by other agents;
- a set of *transition rules*, changing the agent's state; the transition rules are defined in terms of the capabilities, and their effect is dependent on the concrete time of their application;
- a set of *selection functions* to select inputs to transitions from the state;
- a *control*, for deciding which enabled transition should be next, based on the selection functions, the current time, and the previous transition.

The application of a transition T at time τ, mapping state S onto S' given (a possibly empty) input X, will be represented as $T(S, X, S', \tau)$. We will assume for simplicity that the application of transitions is instantaneous, namely τ is also the time when S' is generated.

The control of the agent is responsible for its behaviour, in that it induces an *operational trace*, namely a (typically infinite) sequence of transitions

$$T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), T_{i+1}(S_i, X_{i+1}, S_{i+1}, \tau_{i+1}), \ldots$$

such that S_0 is the given initial state, and for each $i \geq 1$, τ_i is given by the clock of the system and $\tau_i < \tau_{i+i}$, (namely time increases).

State in KGP agents: $\langle KB, Goals, Plan, TCS \rangle$. The KB component holds the agent's *beliefs*. It includes a dynamic part KB_0, updated when the agent observes the environment (via its Sensing capability) and executes actions in plans. In

particular, the fact that action $a[t]$ has been executed at time τ is recorded by means of $executed(a[t], \tau)$. KB also includes knowledge bases to support the various reasoning capabilities, as we will discuss later.

The $Plan$ consists of *plan-trees* whose roots are goals in $Goals$ or simply actions [1], and whose non-root nodes are actions or sub-goals. In a plan-tree, the set of all children of a node form a (partial) plan for the node, the set of leaves of any sub-tree form a (partial) plan for the root of the sub-tree, and actions are leaves. Each goal in $Goals$ is the root of at most one plan-tree.

The TCS component is a set of (temporal) constraints, built from operators including $<, \leq, =, \neq$. Goals, actions and sub-goals have time parameters constrained by TCS. These temporal constraints induce a partial order on actions and sub-goals in plans.

(Sub-)Goals are *timed fluent literals* of the form $l[t]$, where l is a fluent literal (property) of the form p or $\neg p$ and t is its associated time, and actions are *timed action literals* of the form $a[t]$, where a is an action operator and t is its associated time. Implicitly, all time variables are existentially quantified within the agent's state. Actions may be "physical", communicative or sensing, and fluents may be "mental" (to be brought about by plans) or sensing (to be observed).

Reasoning capabilities in KGP agents. In KGP agents, the main reasoning capabilities support Planning, Temporal Reasoning, Reactivity, Goal Decision, and Temporal Constraint Satisfiability. The Planning, Reactivity and Goal Decision capabilities all need to incorporate Temporal Constraint Satisfiability within them. This is to ensure that the agent state is always "consistent" as it is updated by means of goals, sub-goals and actions generated by these capabilities.

We will indicate with $\models_{tcs} C$ that the set of temporal constraints C is satisfiable by means of the Temporal Constraint Satisfiability capability. Intuitively, satisfiability amounts to the existence of a concrete instantiation of the variables in C that render C true wrt the underlying domain for the evaluation of the constraint operators. Thus, e.g. for the integers, $\models_{tcs} 3 \leq t$ but $\not\models_{tcs} 3 \leq t \wedge t < 2$.

Given a state $S = \langle KB, Goals, Plan, TCS \rangle$ and a concrete time point τ, we will use the notation $S, Y \models^{\tau}_{cap} Z$ to indicate, intuitively, that Z is "generated" as the result of the application of capability cap at time τ in state S, where cap is one amongst $plan$ (for Planning), tr (for Temporal Reasoning), $react$ (for Reactivity) and gd (for Goal Decision). Formally, $S, Y \models^{\tau}_{cap} Z$ stands for $KB_{cap} \cup KB_0 \cup X, Y \models^{\tau}_{cap} Z$, where:

- for $cap = plan$, X is $Plan$, Y is a set of (sub-)goals in $Plan$ to be planned for, together with TCS, and Z is either a plan for Y (consisting of a set of actions/sub-goals and a new set of temporal constraints), or $\perp$, representing that no such plan exists;
- for $cap = tr$, X and Y are empty and Z is a timed fluent literal together with some temporal constraints including TCS; intuitively, this capability is

[1] If an action is the root of a plan-tree, that action is necessarily an outcome of reacting, using the knowledge base that supports the Reactivity capability.

used to verify whether or not the literal in Z holds, at a time point satisfying the temporal constraints in Z;
- for $cap = react$, X is $Plan$ and TCS, Y is empty, and Z is either a set of actions/goals and a new set of temporal constraints or $\bot$, representing that no such reaction exists;
- for $cap = gd$, X and Y are empty and Z is a set of timed fluent literals (representing new goals) and a new set of temporal constraints.

In the sequel, when Y is empty we will simply drop it.

The outcome of the capabilities is affected by the current time τ, e.g., in the case of Planning and Reactivity, because the generated actions need to be executable in the future (after τ), or, in the case of Goal Decision, because the generated goals need to be achievable in the future.

For $cap = plan$, if Z consists solely of actions, Z is called a *full plan* for Y. In every state the KB of the agent, in addition to KB_0, includes a modular collection of specialised knowledge bases. These are KB_{plan}, KB_{gd}, KB_{tr}, KB_{react}, used, respectively, by the Planning, Goal Decision, Temporal Reasoning and Reactivity capabilities. KB_{plan}, for example, may contain a plan library or a theory of action and causality such as the event calculus [10]. Independently of the concrete realisation choices for these modules and the corresponding capabilities, we will assume that KB_{plan} and KB_{tr} in KB in any $S = \langle KB, Goals, Plan, TCS\rangle$ are related in such a way that, informally, if $S, g[t] \models^{\tau}_{plan} P$, for $P \neq \bot$ and a full plan, then $S' \models^{\tau'}_{tr} g[t]$, where S' is the state S after having executed all actions in P at times satisfying TCS and all constraints in P and τ' is after τ.

Transition rules in KGP agents. Transitions affect the agent's state and are defined in terms of the capabilities, as follows:

- *Goal Introduction (GI)*, changing the $Goals$ and TCS, using Goal Decision, and changing $Plan$, by adding one plan-tree (consisting solely of the root) for each new goal in $Goals$;
- *Plan Introduction (PI)*, changing the $Plan$, by adding children to (sub-)goals, and changing TCS, and using Planning;
- *Reactivity (RE)*, changing $Goals$, by adding any new "reactive goals", $Plan$, by adding one plan-tree (consisting solely of the root) for each new goal in $Goals$ and any new "reactive actions", and TCS; the new goals, actions and temporal constraints are obtained by using Reactivity;
- *Sensing Introduction (SI)*, changing $Plan$ and TCS, by introducing new (temporally constrained) *sensing actions*, e.g. for checking the preconditions of actions already in $Plan$, and using Sensing;
- *Passive Observation Introduction (POI)*, changing KB_0 by introducing information coming from the environment without being actively sought by the agent, and using Sensing;
- *Active Observation Introduction (AOI)*, changing KB_0 by introducing actively sought information from the environment, and using Sensing; this information might be needed, for example, to confirm that actions have been successfully executed;
- *Action Execution (AE)*, executing actions, and changing KB_0;

– *State Revision (SR)*, revising *Plan*, and using Temporal Reasoning and Temporal Constraint Satisfiability. In particular, SR deletes from *Plan* all achieved or timed-out (sub-)goals, as well as all their descendents in the plan-trees in *Plan*, and all executed or timed-out actions. It also deletes all descendents of (sub-)goals with one or more timed-out children, thus eliminating plans which have no chance of success.

The effect of transitions is dependent on the concrete time of their application, taken into account by the capabilities called therein.

Selection functions in KGP agents. These include c_{GS} and c_{AS}, to select goals and sub-goals to be planned for and actions to be executed, respectively, and c_{FS} and c_{PS}, to select fluents to be sensed immediately, by AOI, and fluents to be sensed eventually, by SI, respectively. These functions provide appropriate inputs to (some of) the transitions and enable them, as discussed below.

Control in KGP agents. The operational traces are not fixed a priori, as in conventional agent architectures, but are determined dynamically by reasoning with declarative cycle theories, giving a form of flexible control. In this paper, we do not provide details of these cycle theories (see [9, 7, 8, 18]). Here, it suffices to say that the cycle theory induces an operational trace
$T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), T_{i+1}(S_i, X_{i+1}, S_{i+1}, \tau_{i+1}), \ldots$
such that T_i is one of the KGP transitions, and X_i is an input as follows

– if T_i is one of GI, RE, POI, SR, then X_i is empty;
– if T_i is PI, then X_i is non-empty and it is the set of all (sub-)goals determined by c_{GS}; these are (sub-)goals to be planned for;
– if T_i is AE, then X_i is non-empty and it is the set of all actions determined by c_{AS}; these are actions to be executed;
– if T_i is SI, then X_i is non-empty and it is the set of all fluents determined by c_{PS}; these are fluents to be actively sensed in the future, e.g. preconditions of actions;
– if T_i is AOI, the X_i is non-empty and it is the set of all fluents determined by c_{FS}; these are fluents to be actively sensed immediately.

We say that the control is *fair* iff no transition is ever postponed indefinitely.

3 Preliminaries

In this section we show how certain design choices for KGP agents allow to prove some basic, desirable properties of KGP agents needed to prove later results.

Definition 1. *Given a state* $S{=}\langle KB, Goals, Plan, TCS\rangle$ *and a time point* τ*:*

– *a goal, action or sub-goal* $z[t]$ *is* timed-out *at* τ *iff* $\not\models_{tcs} t \geq \tau \wedge TCS$*;*
– *an action or sub-goal* $z[t]$ belongs to a plan *for a goal or sub-goal* $g[t']$ *iff* $z[t]$ *is a descendent of* $g[t']$ *in a plan-tree in Plan;*
– *a goal or sub-goal* $g[t]$ *is* believed to be achieved *in* S *at* τ *iff* $S \models^{\tau}_{tr} g[t] \wedge TCS \wedge t \leq \tau$*;*

- *two actions* $a_1[t_1]$ *and* $a_2[t_2]$ *in any plan-trees in Plan are said to be* incompatible *at* τ *iff* $\not\models_{tcs} TCS \wedge t_1 = \tau \wedge t_2 = \tau$*;*
- *a timed (fluent literal or action) literal* $x[t]$ matches *a timed (fluent literal or action) literal* $x[t']$ *in* S *iff* $\models_{tcs} t = t' \wedge TCS$*;*
- *a timed action literal* $a[t]$ is executed in S *iff* $executed(a[t'], \tau) \in KB_0$ *and* $a[t]$ *matches* $a[t']$ *in* S*.*

The following is a property of KGP agents that they do not attempt to execute actions that they believe are infeasible or unnecessary, and do not attempt to plan for goals if a plan is not needed or if it is too late to plan for them.

Theorem 1

- *KGP agents never attempt to execute actions that*
  - *are timed-out, or*
  - *have an ancestor or the child of an ancestor that is timed-out, or*
  - *belong to a sub-tree for a goal that they believe is achieved, or*
  - *have an ancestor that they believe is achieved, or*
  - *have a precondition whose complement they believe is achieved;*
- *KGP agents never attempt to plan for a goal or a sub-goal that*
  - *already has children in a plan-tree, or*
  - *is timed-out or that they believe is already achieved, or*
  - *belongs to a plan-tree for a goal that is timed-out or that they believe is achieved, or*
  - *has an ancestor that is timed-out or that they believe is achieved.*

This result follows from the definition of action selection and goal selection functions. The following result is another consequence of the definition of action selection function.

Theorem 2. *Incompatible actions are never executed concurrently.*

4 Goal Achievement

In this and section 5, to simplify the presentation, we ignore the RE transition. This section builds upon, in part, some of the work reported in [14].
Given a (possibly infinite) operational trace for an agent:

$T_1(S_0, X_1, S_1, \tau_1), \ldots, T_j(S_{j-1}, X_j, S_j, \tau_j), \ldots, T_l(S_{l-1}, X_l, S_l, \tau_l), \ldots$

with $0 \leq j < l$, we refer to the (possibly infinite) sequence of states:

$S_0, S_1, \ldots, S_{j-1}, S_j, \ldots, S_{l-1}, S_l, \ldots$

as the *state-sequence* (of the trace), and to any (possibly infinite) sub-sequence

$S_{j-1}, S_j, \ldots, S_{l-1}, S_l, \ldots$

of a state-sequence as a *portion* (of the state-sequence). We also refer to the time τ_j at which a state S_j is generated in a trace $\ldots, T_j(S_{j-1}, X_j, S_j, \tau_j), \ldots$ giving a state sequence $SS = \ldots S_{j-1}, S_j \ldots$ as $time(S_j, SS)$.

The following definition gives the criterion according to which we judge a state-sequence or portion as providing successive improvements over states. It is parametric wrt a notion of preference $\ll$ between states. Note that this definition is somewhat naive, as will be explained later.

Definition 2. *Let $\ll$ be any notion of preference between states.*

- *An* infinite *state-sequence or portion $S_0, S_1, \ldots, S_n, \ldots$* improves welfare wrt $\ll$ *iff for each $j \geq 0$, there exists $l > j$ such that $S_j \ll S_l$.*
- *A* finite *state-sequence or portion $S_0, S_1, \ldots, S_n$* improves welfare wrt $\ll$ *iff for each $j \geq 0$, $j < n$, there exists $l > j$, $l \leq n$ such that $S_j \ll S_l$.*
- *An agent is* $\ll$-improving wrt some initial state *iff the state-sequence corresponding to any operational trace induced by its control, from the given initial state, improves welfare wrt $\ll$.*

Note that this definition does not impose any condition on intermediate states between S_j and S_l, and in particular any such state might actually bring the agent into a worse state than S_j, wrt $\ll$. Stronger notions could be adopted, for example that for each $j \geq 0$, for each $l > j$, $S_j \ll S_l$. However, we believe that such stronger notions would be too limiting in practice. Note also that we could define a much weaker notion of $\ll$-improvement for an agent, requiring only for it to be $\ll$-improving wrt *some* given class of initial states.

For every concrete notion of preference between states we obtain a concrete notion of improvement in definition 2. One such notion of preference, that we will refer to as $\ll_1$, sanctions, informally, that $S \ll_1 S'$ iff in S' at least as many goals have been achieved as in S. Another such notion, that we will refer to as $\ll_2$, sanctions that $S \ll_2 S'$ iff in S' strictly more goals have been achieved than in S. There are clear connections between these notions of achievement and modelling the preferences of agents using utility functions, as the number of achieved goals gives a very simple kind of utility function. Formally:

Definition 3. *Given a state $S = \langle KB, Goals, Plan, TCS \rangle$ and a time τ, we define the* number of achieved goals in S *at τ as*
$A^+(S, \tau) = \#\{l[t] \mid l[t] \in Goals$ *and* $l[t]$ *is believed to be achieved in S at $\tau\}$.*
Then, given a (portion of a) state-sequence SS and states S, S' in SS with $\tau = time(S, SS)$ and $\tau' = time(S', SS)$:

- $S \ll_1 S'$ *iff* $A^+(S, \tau) \leq A^+(S', \tau')$;
- $S \ll_2 S'$ *iff* $A^+(S, \tau) < A^+(S', \tau')$.

Intuitively, (the designer of) an agent adopting $\ll_1$ ($\ll_2$) believes that its welfare can be improved by never decreasing (always increasing) the number of achieved goals. Note that we take a subjective view of achievement: goals are achieved if they can be proven subjectively by the agent via its Temporal Reasoning capability. There are alternative notions of achievement that we could have adopted, e.g. a stronger subjective notion, whereby only immediately after SR the agent can evaluate its achievement, or a fully objective notion, whereby it is some "external observer", knowing exactly what holds and does not hold in the environment, who evaluates whether goals are achieved and when via its own "temporal reasoning capability" wrt its complete knowledge of the environment. Finally, note that other choices of $\ll$ would have been possible, e.g. by considering the number of unachievable goals or *ratio* between achieved and unachievable goals.

Example 1. Assume g_1, g_2, g_3 are timed fluent literals. The following is a possible (finite portion of a) state-sequence starting with S_0 with $KB_0 = Goals = Plan = \{\}$, with the associated values of A^+ (but ignoring the time of states). Here, we indicate with "_" components that we ignore for simplicity as irrelevant.

$S_0 = \langle\{\}, \{\}, \{\}, \{\}\rangle$ $\quad A^+(S_0) = 0$
T_1 is GI:
$S_1 = \langle\{\}, \{g_1, g_2, g_3\}, \{g_1, g_2, g_3\}, \_\rangle$ $\quad A^+(S_1) = 0$
T_2 is POI, leading to g_1 holding:
$S_2 = \langle\_, \{g_1, g_2, g_3\}, \{g_1, g_2, g_3\}, \_\rangle$ $\quad A^+(S_2) = 1$
T_3 is PI for g_3, introducing a full plan:
$S_3 = \langle\_, \{g_1, g_2, g_3\}, \_, \_\rangle$ $\quad A^+(S_3) = 1$
T_4 is AE, executing all actions for g_3 in $Plan$:
$S_4 = \langle\{\}, \{g_1, g_2, g_3\}, \_, \_\rangle$ $\quad A^+(S_4) = 2$
T_5 is SR (g_1, g_3 achieved and g_2 timed-out):
$S_5 = \langle\_, \{g_1, g_2, g_3\}, \{\}, \_\rangle$ $\quad A^+(S_5) = 2$

Here, every state is better that any earlier state wrt $\ll_1$, thus this trace is $\ll_1$-improving. However, it is not $\ll_2$-improving (e.g. S_4 cannot be improved upon).

So far we have assumed that any two states in a state-sequence or portion of it can be compared using $\ll$. This is inappropriate when the GI transition modifies the *Goals* in a state and after all (sub-)goals have been achieved or become timed-out, as illustrated by the next example wrt the concrete notion of $\ll_2$.

Example 2. Assume the following state-sequence (with associated A^+):

$S_0 = \langle\_, \{g_1\}, \_, \_\rangle$ $\quad A^+(S_0) = 0$ (g_1 not achieved yet)
$S_1 = \langle\_, \{g_1\}, \_, \_\rangle$ $\quad A^+(S_1) = 0$ (g_1 not achieved yet)
$S_2 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S_2) = 0$ (g_1 dropped, g_3, g_4 introduced by GI and not achieved yet)
$S_3 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S_3) = 1$ (g_1 achieved)

Here, g_1 may be (believed to be) achieved because of a POI. According to definition 2, $S_0, \ldots, S_3$ is $\ll_2$-improving, which is counter-intuitive, since the agent should not be better off at achieving goals that it has dropped in favour of newly adopted goals. Thus, $A^+(S_3)$ should be the number of goals in $\{g_3, g_4\}$ that are believed to be achieved. Consider now the state-sequence S_0, S_1, S_2 followed by

$S'_3 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S'_3) = 1$ (g_3 achieved)
$S_4 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S_4) = 1$ (g_3 achieved, g_4 timed-out)
$S_5 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S_4) = 1$ (g_3 achieved, g_4 timed-out)
$S_6 = \langle\_, \{g_3, g_4\}, \_, \_\rangle$ $\quad A^+(S_4) = 1$ (g_3, g_4 dropped by SR)

S_5 might be the outcome of a POI. According to definition 2, $S_0, \ldots, S_6$ is not $\ll_2$-improving, which is counter-intuitive, since the agent has done its best to achieve as many goals as possible and reach state S_4. This is the last state that should "count" as far as $\ll$ is concerned.

The notion of $\ll$-improvement can be easily modified to look at portions related to the same *Goals* and ignoring states following other states with all goals in *Goals* either achieved or timed-out. We omit this definition here for lack of space.

Theorem 3. *Any KGP agent is $\ll_1$-improving.*

This result holds because of the features of the KGP model, according to which goals, once achieved, can never become "unachieved". This is due to the fact that goals are existentially quantified in the model, and that the agents do not observe information about the past. If at τ an agent believes that a goal holds at some time t, then there is an instance τ' of t that satisfies the temporal constraints, τ' is before τ and the agent believes the goal held at τ'. Then, in all future states the agent will continue to believe that the goal held at τ'.

The analogous result for $\ll_2$ does not hold, e.g. see example 1. However, we can prove the following results regarding $\ll_2$, given a state sequence SS and states $S = \langle KB, Goals, Plan, TCS\rangle$ and $S' = \langle KB', Goals', Plan', TCS'\rangle$ in SS playing the role of (and satisfying the conditions for) S_j and S_l in definition 2.

Theorem 4. *If $S \ll_2 S'$, then $KB \subset KB'$ and, in particular, $KB_0 \subset KB'_0$.*

This theorem shows the importance of sensing the environment and executing actions to improve welfare according to $\ll_2$. This is made more explicit by theorem 6. Note that KB_0, holding the outcome of the agent's sensing of the environment and the recording of any action executed by the agent, is the only part of the agent KB that is dynamically modified.

Theorem 5. *If $S \ll_2 S'$, then there exists a state S'' such that either S'' is in between (but different from) S and S' in SS or $S'' = S'$ and there exists a (sub-)goal $g[t]$ in S'' such that $g[t]$ is believed to be achieved in S'', but not in S.*

As a consequence of theorem 5 we can show:

Theorem 6. *If $S \ll_2 S'$, there is a state S'' in between (but different from) S and S' in SS such that in S'' one of POI or AE or PI has been performed.*

This theorem shows the importance of the three transitions, POI, AE and PI, in improving welfare in terms of goal achievement. However, because of time criticality and the dynamism of the environment, these transitions cannot guarantee achievement of all goals. But they help *make progress* towards achievement of goals in a sense that will be formalised in section 5.

In general, we cannot guarantee that all achievable goals will eventually be achieved, namely that the maximum element of either of the $\ll_1$ and $\ll_2$ orderings can be found. There are three main reasons for this: (i) goal are temporally constrained and it may not be possible to achieve them by their deadlines, (ii) the environment can be unpredictable, (iii) the agent may not know of compatible plans for the goals given what it believes about the environment. Goals can be *guaranteed* to be achieved under some restrictive conditions, omitted here for lack of space.

5 Progress

In this section we define a new ordering between states, based on a notion of "progress" and relate this ordering to the one induced by A^+. We also show how some of the transitions in the KGP model affect progress. For simplicity, here we will assume that the Planning capability always produces full plans.

Definition 4. *A state* $S{=}\langle KB, Goals, Plan, TCS\rangle$ *at time* τ potentially achieves a goal $g[t] \in Goals$ with a set of timed actions $A = \{a_1[t_1], \ldots, a_n[t_n]\}$ *iff*

- $g[t]$ *is not already believed to be achieved in* S *at* τ*, and*
- A *is a set of actions in* $Plan$*, and*
- *no action in* A *has already been executed in* S*, and*
- *there exist concrete times* $\tau', \tau_1, \ldots, \tau_n$ *such that*
  - $\models_{tcs} \tau < \tau_1 \leq \tau' \wedge \ldots \wedge \tau < \tau_n \leq \tau'$*, and*
  - $\models_{tcs} t_1 = \tau_1 \wedge \ldots \wedge t_n = \tau_n$*, and*
  - $KB \cup \{executed(a_1[t_1], \tau_1), \ldots, executed(a_n[t_n], \tau_n)\} \models^{\tau}_{tr} g[\tau']$.

Definition 5. *Given states* S *and* S' *and times* τ, τ' *and a goal* $g[t]$*, we say that* $S \prec_{g[t]} S'$ *iff*

- S *potentially achieves* $g[t]$ *with some* A *and*
- S' *potentially achieves* $g[t]$ *with some* A' *and*
- $A' \subset A$ *and*
- S *does not potentially achieve* $g[t]$ *with* A'.

Intuitively, if $S \prec_{g[t]} S'$ then S' is a more progressed state than S as far as the achievement of $g[t]$ is concerned, since there are fewer actions still to be executed before $g[t]$ is actually achieved (if all goes according to the $Plan$).

In a state with maximal goal achievement, either all goals are achieved or no more progress is possible, namely:

Theorem 7. *Given a state sequence* SS*, for any* S *in* SS *if there is no state* S' *in* SS *such that* $S \ll_2 S'$*, then*
either all the goals in S *are believed to be achieved at* $time(S, SS)$
or for no goal $g[t]$ *in* S *there exists a state* S'' *such that* $S \prec_{g[t]} S''$.

Note that S'' does not need to be in SS.

The following theorem sanctions that AE improves progress towards achievement. It follows from theorems 1 and 2.

Theorem 8. *Let* $T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), \ldots$ *be any trace. If* T_i *is AE then*
either all the goals in S_i *are believed to be achieved at* τ_i
or there exists $g[t]$ *such that* $S_{i-1} \prec_{g[t]} S_i$.

The following theorem sanctions that PI is needed in order to pave the ground for progress:

Theorem 9. *Let* SS *be a state sequence and* S, S' *be two states in* SS *such that* $time(S, SS) < time(S', SS)$*. Then, if there exists* $g[t]$ *such that* $S \prec_{g[t]} S'$*, then there exists* S'' *resulting from a PI such that* $time(S'', SS) < time(S', SS)$.

6 Reactive Awareness

Reactivity is a major feature of the KGP model. It allows condition-action type of behaviour and some element of dynamic plan repair. It also allows to generate dynamically obligations and prohibitions [17]. Next we define R^+, which intuitively gives a measure of how "aware" the agent is of its reactive necessities.

Definition 6. *Given a state* $S = \langle KB, Goals, Plan, TCS \rangle$ *and a time* τ*, let*

- $Set_1(S, \tau) = \{x[t] | S \models^{\tau}_{react} x[t]\}$
- $Set_2(S, \tau) = \{x[t] | x[t] \in Set_1(S, \tau)$ *and*
  $x[t]$ *matches a (fluent or action) literal in* $Plan$ *or*
  $x[t]$ *is believed to be achieved in* S *at* τ *or*
  $x[t]$ *is executed in* $S\}$.

Then, $R^+(S, \tau) = \dfrac{\#Set_2(S, \tau)}{\#Set_1(S, \tau)}$.

Intuitively, $Set_1(S, \tau)$ represents all the actions that have to be executed and all the goals that have to be achieved in reaction to the circumstances the agent finds itself in state S at time τ (according to its KB_{react}). $Set_2(S, \tau)$ represents the subset of $Set_1(S, \tau)$ that the agent is "explicitly aware" of, namely "reactive actions" that it has already executed or at least included in its $Plan$ to execute, and "reactive goals" that it has already achieved or included in its $Plan$ to achieve. Then the ratio R^+ gives a measure of reactive awareness.

The Reactivity capability in the KGP model is designed so that, if it is possible to have mutually consistent reactions, it produces all the necessary reactions (i.e. R^+=1). This is captured by the following theorem.

Theorem 10. *Let* $T_1(S_0, X_1, S_1, \tau_1), \ldots, T_i(S_{i-1}, X_i, S_i, \tau_i), \ldots$ *be any trace. If* T_i *is RE,* $S_{i-1} \models^{\tau_i}_{react} Z$ *and* $Z \neq \bot$*, then* $R^+(S_i, \tau_i) = 1$.

Alternative design choices might be suitable, in the case of an inconsistent set of reactions, to guarantee as high a value of R^+ as possible, e.g. by returning a maximally consistent subset of the set of inconsistent reactions, or by imposing preferences over reactive rules to eliminate inconsistencies.

The result in theorem 10 cannot be guaranteed after all the transitions, and in particular, after POI, AOI, GI, PI and SI. Indeed, POI and AOI might introduce new observations into the (KB_0 part of the) state, PI and SI will typically produce new actions and sub-goals and GI might introduce new goals in the state, and the Reactivity capability might require that some new reactions are introduced to take into account these extensions to the state. However, it is natural in general that R^+ fluctuates, because it depends heavily on the dynamics of the environment and the agent.

An agent equipped with a fair control will always reach, at some point, a state wrt maximal R^+, if one such state exists (i.e. in the absence of inconsistencies).

Theorem 11. *If the agent control is fair, then, for every operational trace, for every state S in the trace, there exists a later state S' in the trace such that*
either $R^+(S', time(S'))$ is not defined
or $R^+(S', time(S')) = 1$.

Note that an agent that is maximally achieved wrt A^+ can still work towards reaching an ideal R^+. Note also that the notion of R^+ above is *subjective*, in that it only considers reactions to what the agent believes about its environment (as well as its $Plan$ and $Goals$). We can also define an objective notion of R^+:

Definition 7. *Let E represent* complete information *about the environment. Then, given a time τ and a state $S = \langle KB, Goals, Plan, TCS\rangle$, let S_E be $\langle KB - KB_0 \cup E, Goals, Plan, TCS\rangle$ and let*

- $Set_1(S, E, \tau) = \{x[t] | S_E \models^{\tau}_{react} x[t]\}$
- $Set_2(S, E, \tau) = \{x[t] | x[t] \in Set_1(S, E, \tau)$ *and*
  $x[t]$ *matches a fluent or action literal in S or*
  $x[t]$ *is believed to be achieved in S at τ*
  $x[t]$ *is executed in S}.

Then, $R^+(S, E, \tau) = \dfrac{\#Set_2(S, E, \tau)}{\#Set_1(S, E, \tau)}$.

It is possible to design the control of the agent in order to guarantee maximal values of this new notion of R^+, mirroring theorems 10 and 11 above. This control needs to ensure that any RE transition is immediately preceded by an AOI transition, sensing fluents that are triggers in reactive rules in KB_{react}. In a reactive rule $l[t] \Rightarrow a[t'] \wedge t' < t + 10$ in KB_{react} $l[t]$ is one such trigger if l is a sensing fluent.

7 Conclusion

We believe our work on the specific properties addressed in this paper, in particular that related to improving goal achievement and reactive-awareness, is novel. However, our work complements the work of others in the field of formal analysis of agent systems. Amongst these are the following. Kacprzak et al [13] who have explored the use of unbounded model checking for verification of temporal epistemic properties. Lomuscio and Raimondi [12] have proposed a model checker called MCMAS that extends verification from temporal modalities to other modalities, such as correctness and cooperation, relevant to agents. Bordini et al [2] have used model checking for verification of properties of BDI agents expressed as AgentSpeak programs. Wooldridge et al [19] have presented a language called MABLE for multi-agent systems, allowing BDI-like agents and supporting automatic verification of properties via model checking.

We believe that the properties we have identified and discussed in this paper are interesting in general for all agent frameworks. Part of our future work is to study them in the context of other agent models.

Dunne et al [5] give computational complexity results for achievement and maintenance tasks of agents for a variety of environmental properties, for ex-

ample whether or not the environment is deterministic, history dependent or bounded. It would be interesting to see whether some of our results could be strengthened for specific kinds of environments.

References

1. K. A. Arisha, F. Ozcan, R. Ross, V. S. Subrahmanian, T. Eiter, and S. Kraus. IMPACT: a platform for collaborating agents. *IEEE Intell. Systems*, 14(2), 1999.
2. R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Model checking rational agents. *IEEE Intell. Systems*, 19(5), 2004.
3. A. Bracciali, N. Demetriou, U. Endriss, A. Kakas, W. Lu, and K. Stathis. Crafting the mind of prosocs agents. *Applied Artificial Intelligence*, 20(2–4), 2006.
4. A. Bracciali, N. Demetriou, U. Endriss, A.C. Kakas, W. Lu, P. Mancarella, F. Sadri, K. Stathis, G. Terreni, and F. Toni. The KGP model of agency for global computing: Computational model and prototype implementation. In *Global Computing 2004 Workshop*, volume 3267 of *LNCS*. Springer Verlag, 2005.
5. P. E. Dunne, M. Laurence, and M. Woolridge. Complexity results for agent design problems. *Annals of Math., Computing and Teleinformatics*, 1(1), 2003.
6. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J. Ch. Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4), 1999.
7. A. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Deliverable d4: A logic-based approach to model computees. Technical report, SOCS Consortium, 2003. Avaialble from `http://lia.deis.unibo.it/Research/Projects/SOCS/`.
8. A. C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. Declarative agent control. In *Post-proc. CLIMA V*, volume 3487 of *LNAI*. Springer Verlag, 2005.
9. A.C. Kakas, P. Mancarella, F. Sadri, K. Stathis, and F. Toni. The KGP model of agency. In *Proc. ECAI-2004*, 2004.
10. R. A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1), 1986.
11. J. A. Leite, J. J. Alferes, and L. M. Pereira. $\mathcal{MINERVA}$: A dynamic logic programming agent architecture. In *Proc. ATAL01*, volume 2333 of *LNAI*. Springer Verlag, 2002.
12. A. Lomuscio and F. Raimondi. MCMAS: a tool for verifying multi-agent systems. In *Proc. TACAS 2006*, volume 3920 of *LNCS*. Springer Verlag, 2006.
13. W. Penczek M. Kacprzak, A. Lomuscio. Verification of multiagent systems via unbounded model checking. In *Proc. AAMAS04*, 2004.
14. P. Mancarella, F. Sadri, K. Stathis, F. Toni, and A. Bracciali. Computees and welfare. Technical report, SOCS Consortium, 2005.
15. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In *Proc. MAAMAW96*, volume 1038 of *LNCS*. Springer Verlag, 1996.
16. A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Readings in Agents*. Morgan Kaufmann Publishers, 1997.
17. F. Sadri, K. Stathis, and F. Toni. Normative KGP agents. *Computational and Mathematical Organization Theory*, 2006. To appear.
18. F. Sadri and F. Toni. Variety of behaviours through profiles in logic-based agents. In *Post-proc. CLIMA VI*, volume 3900 of *LNAI*. Springer Verlag, 2006.
19. M. Wooldridge, M.-P. Huget, M. Fisher, and S. Parsons. Model checking multi-agent systems: the mable language and its applications. *International Journal on Artificial Intelligence Tools*, 15(2), 2006.

Irrelevant Updates and Nonmonotonic Assumptions

Ján Šefránek

Comenius University, Bratislava, Slovakia
sefranek@fmph.uniba.sk

Abstract. The second postulate of Katsuno and Mendelzon characterizes irrelevant updates. We show that the postulate has to be modified, if nonmonotonic assumptions are considered. Our characterization of irrelevant updates is based on a dependency framework, which provides an alternative semantics of multidimensional dynamic logic programming.

Keywords: foundations of logic-based AI systems, nonmonotonic knowledge bases, updates, logic programming, nonmonotonic reasoning.

1 Introduction

Background. Nonmonotonic knowledge bases (NMKB) represent an important topic for a logic-based research in artificial intelligence. There are essentially two sources of nonmonotony in knowledge bases – an evolution of incomplete knowledge and a use of assumptions which can be overridden (falsified).

We discuss in this paper the second postulate for updates by Katsuno and Mendelzon, 2[KM] hereafter: if an update follows from a knowledge base (KB), then the updated KB and the original KB should be equivalent according to the postulate [8]. In other words: if an update follows from a KB, then the update is irrelevant.

Tautological and cyclic updates represent an important problem also for multidimensional dynamic logic programming (MDyLP) research [2, 9, 10, 3]). MDyLP contributed to logic-based knowledge representation research by focusing on dynamic aspects of knowledge; it can be considered as a formal model of NMKB with preferences (also along dimensions different from time). The role of both sources of nonmonotony, as mentioned above, is taken into account in MDyLP.

Problem. Unwanted generation of new models caused by cyclic or tautological updates (they should be irrelevant) has been a serious problem of MDyLP for a time. Recently, the problem has been solved for dynamic logic programs in [3] and for general MDyLP in [4]. The solution of [3] is based on a principle, called refined extension principle and its ambition is to express fundamental features enabling to distinguish the "right" semantics of logic program updates. Unfortunately, the principle has not been extended to the general case of MDyLP and, moreover, the trivial semantics assigning empty set of models to each dynamic logic program satisfies the principle, see [4]. Finally, the principle is expressed in terms too close to the specific conceptual apparatus of MDyLP. Semantics of MDyLP are based on rejection of rules. They satisfy the causal rejection principle (CRP): if there is a conflict between heads of rules, the less preferred rule is

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 426–438, 2006.

rejected. CRP has some drawbacks,[1] see [12], which challenges an effort to find a more general platform for expressing the notion of irrelevant updates.

On the other hand, 2[KM] is not an appropriate one, if (nonmonotonic) assumptions are considered (see Example 25).

Goal and proposed solution. The main task of this paper is an analysis of the problem of irrelevant updates. Cyclic and tautological updates do not represent the only source of unwanted generation of models after an update. We are aiming to extend the horizon behind CRP and behind cyclic and tautological updates.

We represent an NMKB in terms of a dependency framework, where nonmonotonic assumptions and dependencies on assumptions are "first class citizens". Motivations for the dependency framework and an alternative semantics of MDyLP were presented in [12]. Each knowledge base consisting of rules (hence, also a logic program and a MDyLP) can be mapped into the dependency framework. Nonmonotony of a knowledge base is manifested in the framework by (nonmonotonic) assumptions which can be falsified and by solving of conflicts. We emphasize the role of nonmonotonic assumptions for the theory of updates of NMKB. The notion of irrelevant updates proposed in this paper modifies 2[KM] by focusing on nonmonotonic assumptions.

For a short version of this paper see [13],

Main contributions. of the paper are as follows: analysis and definition of irrelevant updates, an adaptation of 2[KM] for updates of NMKB, we show that irrelevant updates do not generate new models and that models of the original program are preserved after an irrelevant update (Theorem 33 and Consequence 34).

The rest. of the paper is organized as follows: First, an informal explanation of the notion of nonmonotonic knowledge base is presented. A brief view on semantics of MDyLP based on CRP is provided in Section 3. The dependency framework is introduced in Section 4. Irrelevant updates are discussed in Section 5 and defined in Section 6.

2 Nonmonotonic Knowledge Base

In this section we provide a schematic view on a NMKB. We assume an unspecified language $\mathcal{L}$ for knowledge representation. (An example of such language is presented in Section 3.) Pairs of conflicting expressions are specified for $\mathcal{L}$. A notion of rule is defined in $\mathcal{L}$. A knowledge base is a set of rules. We assume that a semantics of $\mathcal{L}$ is defined as a mapping from $\mathcal{L}$ to a set of models. (An example of such semantics, stable model semantics, is presented in Section 3.)

We are proceeding to the basic features of $\mathcal{L}$ and of its semantics which enable to speak about nonmonotony of a knowledge base.

- Some expressions of $\mathcal{L}$ are considered as assumptions. To each assumptions is assigned a conflicting expression. (Negative literals are assumptions of the language introduced in Section 3.)

[1] There are conflicts which cannot be recognized according to CRP. On the other hand, some conflicts recognized as conflicts between heads of rules should be considered as irrelevant.

- The semantics of assumptions satisfies the condition as follows: an assumption is true unless it is known that the conflicting expression is true.
- Some principles for solving conflicts in sets of expressions of $\mathcal{L}$ are specified.
- A nonmonotonic consequence operator is specified for $\mathcal{L}$.

MDyLP represent a well understood formalization of NMKBs, see Section 3. Also sets of formulas in other formalisms can be considered as idealizations of NMKBs. We are interested in mappings from sets of rules to sets of dependencies (where dependencies on nonmonotonic assumptions are of crucial importance). Comparison of various formalisms in terms of the dependency framework is a goal of our future research.

MDyLP as a language for NMKBs representation (and its mapping into the dependency framework) is considered in this paper. However, the dependency framework is independent on MDyLP and our results can be generalized into a form independent on MDyLP.

3 Multidimensional Dynamic Logic Programs

The language of MDyLP. i.e. a propositional language with default negations also in heads, is introduced in this section.

Let $\mathcal{A}$ be a set of atoms. The set of *literals* is defined as $Lit = \mathcal{A} \cup \{not\ A : A \in \mathcal{A}\}$. Literals of the form $not\ A$, where $A \in \mathcal{A}$ are called *negative*. Notation: $Ngt = \{not\ A \mid A \in \mathcal{A}\}$. A convention: $not\ not\ A = A$. If X is a set of literals then $not\ X = \{not\ L \mid L \in X\}$.

A *rule* is each expression of the form $L \leftarrow L_1, \ldots, L_k$, where $k \geq 0$, L, L_i are literals. If r is a rule of the form as above, then L is denoted by $head(r)$ and $\{L_1, \ldots, L_k\}$ by $body(r)$. A finite set of rules is called *generalized logic program* (program hereafter).

The set of *conflicting literals* is defined as $CON = \{(L_1, L_2) \mid L_1 = not\ L_2\}$. Two rules r_1, r_2 are called conflicting, if $head(r_1)$ and $head(r_2)$ are conflicting literals. Notation: $r_1 \bowtie r_2$. A set of literals S is *consistent* if it does not contain a pair of conflicting literals. An *interpretation* is a consistent set of literals. A *total* interpretation is an interpretation I such that for each atom A either $A \in I$ or $not\ A \in I$. Let I be an interpretation. Then $I^- = I \cap Ngt$. A literal is *satisfied* in an interpretation I iff $L \in I$. A set of literals S is satisfied in I iff $S \subseteq I$. Stable model of generalized logic programs is defined in [2] as follows.

Definition 1 ([2]). *A total interpretation S is a* stable model *of a program P iff*

$$S = least(P \cup S^-),$$

where $P \cup S^-$ is considered as a Horn theory and $least(P \cup S^-)$ is the least model of the theory. □

The set of all stable models of a program P is denoted by $SM(P)$. A program is called *coherent*[2] iff it has a stable model.

[2] We prefer this term over 'consistent", see also [5].

A literal L (set of literals X) *SM-follows* from a program P iff it is satisfied in each $S \in SM(P)$ (notation: $P \models_{SM} L$, $P \models_{SM} X$). A rule r SM-follows from P iff for each $S \in SM(P)$ holds that $head(r)$ is satisfied in S whenever $body(r)$ is satisfied in S ($P \models_{SM} r$). If U is a set of rules then $P \models_{SM} U$ iff $\forall r \in U\ P \models_{SM} r$.

A *multidimensional dynamic logic program* (also *multiprogram* hereafter) is a set of programs with a preference relation on programs. The relation is specified by an acyclic directed graph. If the preference relation collapsed to a sequence (to a strict linear order), the corresponding multiprogram is called *dynamic logic program*.

There are various different semantics of multiprograms, based on rejection of rules. For a comparison see [10, 11, 7]. They all can be viewed as instances of the abstract schema below.

Let a semantics be a mapping Σ from multiprograms to sets of total interpretations. If $\mathcal{P}$ is a multiprogram and M is a total interpretation then $M \in \Sigma(\mathcal{P})$ holds iff

$$M = least((\mathcal{P} \setminus Rejected(\mathcal{P}, M)) \cup Assumptions(\mathcal{P}, M). \quad (1)$$

Various semantics differ in definitions of predicates $Rejected$ and $Assumptions$. Intuitively, $Rejected(\mathcal{P}, M)$ represents the set of all rules rejected from $\mathcal{P}$ w.r.t. M and $Assumptions(\mathcal{P}, M)$ represents the set of all accepted default negations.

In this paper we consider — because of limited space – only the simplest multiprograms of the form $\langle P, U\rangle$, where U is more preferred than P, notation $P \prec U$. U is called an update of P. However, it is straightforward to generalize the analysis, notions and results to the case of arbitrary multiprograms.

We will now define the instance of $Assumptions(\langle P, U\rangle, M)$ used in this paper as

$$\{not\ A \mid \neg\exists r \in P \cup U\ (A = head(r), M \models body(r))\}$$

Two instances of $Rejected$ – Rej and Rej^R – are defined as follows. *Dynamic stable model* semantics [2] of a multiprogram $\langle P, U\rangle$ w.r.t. interpretation M, notation $DSM(\langle P, U\rangle, M)$, uses Rej:

$$Rej(\langle P, U\rangle, M) = \{r \in P \mid \exists r' \in U\ (r \bowtie r', M \models body(r'))\}$$

Refined dynamic stable model semantics ($RDSM(\langle P, U\rangle, M)$) enables also mutual rejection of rules in one program, [3]. $Rej^R(\mathcal{P}, M)$ is defined as the set

$$\{r \in P \mid \exists r' \in P \cup U\ (r \bowtie r', M \models body(r'))\} \cup \{r \in U \mid \exists r' \in U\ (r \bowtie r', M \models body(r'))\}$$

Dynamic stable model semantics of multiprograms suffers from tautological and cyclic updates.

Example 2 ([3]).

$$P = \{d \leftarrow not\ n \qquad U = \{s \leftarrow v$$
$$n \leftarrow not\ d \qquad v \leftarrow s\}$$
$$s \leftarrow n, not\ c$$
$$not\ s \leftarrow$$

There are two dynamic stable models of $\langle P, U\rangle$*:* $M_1 = \{d, not\ n, not\ c, not\ s, not\ v\}$ *and* $M_2 = \{s, v, n, not\ d, not\ c\}$*. However,* M_2 *is a counterintuitive model – truth of* s *and* v *is not supported and rejection of the fact* $not\ s \leftarrow$ *by a cyclic dependence of* s *on* s *is not a reasonable one.* □

Refined semantics solves the problem. It obeys the refined extension principle (REP), introduced in [3]. However, the principle is expressed in terms too close to the specific conceptual apparatus of MDyLP. Two (not very intuitive) sequences of logic programs are considered in the definition of REP and the definition uses predicates $Assumptions$ and $Rejected$.

Moreover, refined semantics for the general case of multiprograms is not known. The well supported semantics for general multiprograms is defined in [4] and it solves the problem of cyclic updates. The well supported semantics for MDyLP coincides with the refined one on dynamic logic programs. We focus on the simplest dynamic logic programs of the form $\langle P, U\rangle$ in this paper, therefore the refined semantics is discussed in examples. However, our arguments are relevant w.r.t. any semantics based on rejection of rules and satisfying CRP.

The main goal of [3] is to explore the conditions guaranteeing that the addition of a set of rules to a dynamic logic program does not generate new models. The authors of REP observed in [4] that REP is too weak. For example, the trivial semantics that assigns to each dynamic logic program the empty set of models satisfies the principle. It is noted in [4] that stronger criteria and techniques are needed. We believe that the dependency framework of [12] enables to create such criteria and techniques by providing a finer analysis of unwanted models of updated logic programs.

4 Dependency Framework

We now introduce the dependency framework of [12] in order to be able to analyze the problem of irrelevant updates.

Definition 3 (Dependency relation). *A* dependency relation *is a set of pairs* $\{(L, W) \mid L \in Lit, W \subseteq Lit, L \notin W\}$. □

The notion of dependency relation is rather a general one and it is not connected to a special logical form (of a knowledge base or logic program). Each knowledge base consisting of a set of rules (with one literal in the head) can be mapped into a dependency relation. We define now a mapping for the language introduced in Section 3.

Definition 4 ($\ll_P$). *A literal* L depends *on a set of literals* W*,* $L \notin W$*,* with respect to a program P *(*$L \ll_P W$*) iff there is a sequence of rules* $\langle r_1, \ldots, r_k\rangle$*,* $k \geq 1$*,* $r_i \in P$ *and*

- $head(r_k) = L$*,*
- $W \models body(r_1)$*,*
- *for each* i*,* $1 \leq i < k$*,* $W \cup \{head(r_1), \ldots, head(r_i)\} \models body(r_{i+1})$*.*

It is said that the dependency relation $\ll_P$ *is* generated *by the program* P. □

Notice that a literal cannot depend on itself (also in a context of other literals).

Example 5. *Let P be $\{a \leftarrow not\ b; c \leftarrow a\}$. It holds that $a \ll_P \{not\ b\}$, $c \ll_P \{a\}$, $c \ll_P \{not\ b\}$. We can see that some dependencies of L on W are of crucial interest, namely those, where $W \subseteq Ngt$ and W generates (or contributes to a generation) of a stable model.* □

Note that $\ll_P$ does not coincide with the derivability from P.

Definition 6 (Closure property). *A* closure operator *Cl assigns the set of all pairs*

$$\{(L, W) \mid L \ll W \vee (\exists U\ (L \ll U \wedge \forall L' \in U \setminus W\ (L' \ll W)))\}$$

to a dependency relation $\ll$.

A dependency relation $\ll$ has the closure property iff $\mathrm{Cl}(\ll) = \ll$. □

Proposition 7. *Let P be a program. Then* $\mathrm{Cl}(\ll_P) = \ll_P$.

We have seen in Example 5 that dependencies on negative literals are crucial from the viewpoint of stable model semantics. Therefore the role of (default) assumptions is emphasized.

Definition 8 (SSOA, TSSOA). *$Xs \subseteq Ngt$ is called a* sound set of assumptions *(SSOA) with respect to the dependency relation $\ll$ iff the set*

$$Cn_{\ll}(Xs) = \{L \in Lit \mid L \ll Xs\} \cup Xs$$

is non-empty and consistent.

It is said that Xs, a SSOA, is total *(TSSOA) iff for each $A \in \mathcal{A}$ holds either $A \in Cn_{\ll}(Xs)$ or $not\ A \in Cn_{\ll}(Xs)$.*

The set of all (T)SSOAs w.r.t. $\ll$ is denoted by $(T)SSOA(\ll)$. □

Example 9. *Let P be $\{a \leftarrow not\ b; b \leftarrow not\ a\}$. There are two TSSOAs w.r.t. $\ll_P$: $Xs_1 = \{not\ b\}$ and $Xs_2 = \{not\ a\}$. $Cn_{\ll_P}(Xs_1) = \{not\ b, a\}$ and $Cn_{\ll_P}(Xs_2) = \{not\ a, b\}$. Notice that both TSSOAs generate (all) stable models of P.* □

Also an empty set of literals may be a (T)SSOA w.r.t. some $\ll_P$.

Theorem 10. *X is a TSSOA w.r.t. $\ll_P$ iff $Cn_{\ll_P}(X)$ is a stable model of P.*

Let S be a stable model of P. Then there is $X \subseteq Ngt$, a TSSOA w.r.t. $\ll_P$ s.t. $S = Cn_{\ll_P}(X)$.[3] □

Semantics based on assumptions and dependencies. Consider two mappings Σ, Σ'. Let Σ assign to each program P the set of all its stable models. Let Σ' assign to each program P the set of all TSSOAs w.r.t. $\ll_P$. We have seen in Theorem 10 that (the semantics characterized by) Σ is equivalent to (the semantics characterized by) Σ'. So, we can speak about a semantics based on assumptions and dependencies.

Dependencies in a multiprogram. We intend to use our framework for handling conflicting dependencies in a multiprogram. Note that dependencies in a multiprogram are well defined.

[3] Proofs or proof sketches of propositions/theorems can be found in [14].

Proposition 11. *Let $\langle P, U\rangle$ be a multiprogram. Then $\ll_{P\cup U}$ is well defined. It holds*

$$(\ll_P \cup \ll_U) \subseteq \ll_{P\cup U},$$

but the converse inclusion does not hold.

Example 12. *Consider a multiprogram $\langle P, U\rangle$, where P is as in Example 9 and U is as follows: $\{c \leftarrow a; b \leftarrow c; not\ b \leftarrow not\ a\}$.*

$P \cup U$ is a program and it generates a dependency relation. Observe that $c \ll_{P\cup U} \{not\ b\}$, but $(c, \{not\ b\}) \notin (\ll_P \cup \ll_U)$. □

In general, dependencies on assumptions in a multiprogram can be conflicting. There are essentially two possible sources of incoherence/inconsistency:[4]

- two conflicting literals depend on a set of literals,
- or literal L_1 depends on a set of literals W, $(L_1, L_2) \in CON$ and $L_2 \in W$.

Definition 13. *It is said that $\ll$* contains a conflict *C (where $C \subseteq \ll$) iff for some $A \in \mathcal{A}$ is $C = \{(A, Y), (not\ A, Y) \mid Y \subseteq Ngt\}$ or $C = \{(A, Y) \mid Y \subseteq Ngt, not\ A \in Y\}$.* □

Definition 13 does not contain a reference w.r..t. a (multi)program. Moreover, it is possible to replace pairs $A, not\ A$ by a more general objects – pairs of conflicting literals containing a (general) nonmonotonic assumption.

Example 14. *Dependency relation $\ll_{P\cup U}$ from Example 12 contains conflicts $C_1 = \{(b, \{not\ a\}), (not\ b, \{not\ a\})\}$ and $C_2 = \{(b, \{not\ b\})\}$.* □

It is assumed that the preference relation $P \prec U$ is preserved also for dependency relations assigned to the programs: $\ll_P \prec \ll_U$. It enables us to prefer some dependencies according to the preference relation defined for programs when solving conflicts.

We propose to solve a conflict by ignoring some dependencies (taking away from given dependency relation). Good solutions of a conflict are sets of dependencies, which are minimal (w.r.t. the set inclusion) and minimally preferred (w.r.t. the given preference relation).

Definition 15. *It is said that a set of dependencies D is a* solution *of the conflict C iff each $d \in D$ is of the form $L \ll_P W$ or of the form $L \ll_U W$ and $C \not\subseteq Cl((\ll_P \cup \ll_U) \setminus D)$. D is called* minimal *iff there is no proper subset of D which is a solution of C.*

Let D and D' be minimal solutions of C. It is said that D is more suitable *than D' iff there is an injection $\kappa : D \longrightarrow D'$ such that $\forall d \in D\ (d \in \ll_P \wedge\ \kappa(d) \in \ll_P \cup \ll_U)$. If the cardinality of D and D' is the same then for at least one $d \in D$ holds $d \in \ll_P \wedge\ \kappa(d) \in \ll_U$. A minimal solution D of a conflict C is called* good solution *iff there is no more suitable solution of C.* □

Example 16. *Consider conflicts from Example 14. $D = \{(not\ b, \{not\ a\})\}$ is a minimal solution of $C_1 = \{(b, \{not\ a\}), (not\ b, \{not\ a\})\}$. However, $D' = \{(b, \{not\ a\})$ is a more suitable solution of C_1 than D and it is also the good solution of C_1* □

[4] (In)coherent dependency relation is defined in Definition 18 and we use (in)coherence as a technical term in the paper.

Definition 17. *An assumption* $not\ A$*, where* $A \in \mathcal{A}$*, is* falsified *in a dependency relation* $\ll$ *iff* $A \ll \emptyset$*,* $not\ A \not\ll \emptyset$ *and* $\emptyset$ *is a SSOA w.r.t.* $\ll$*.*

A set of assumptions $Xs \subseteq Ngt$ *is falsified in* $\ll$ *iff it contains a literal falsified in* $\ll$*.* □

The notion of falsified assumption is illustrated in Example 21.

Definition 18 (Coherent dependency relation). *A dependency relation* $\ll$ *is called* coherent *iff there is a TSSOA w.r.t.* $\ll$*. A dependency relation is called* incoherent *iff it is not a coherent one.* □

The approach based on the dependency framework is focused on looking for assumptions which can serve as TSSOA w.r.t. a subset of given dependency relation $\ll_{P \cup U}$. The goal is to construct all possible dependency (sub)relations which are coherent (w.r.t. a TSSOA).

Definition 19 (Semantics of multiprograms). *Semantics of a multiprogram* $\langle P, U\rangle$ *is a mapping* Σ *which assigns to* $\langle P, U\rangle$ *a set of pairs of the form* $(Z, View)$*, where* $View$ *is a coherent subset of* $\ll_{P \cup U}$ *and* Z *is a TSSOA w.r.t.* $View$*.*

Example 20. *Recall example 14.* $\Sigma(\langle P, U\rangle) = \{(\{not\ \ a, not\ \ b, not\ \ c\}, View)\}$*, where* $View = \ll_U$*.*

5 Irrelevant Updates – Intuitions

We motivate the notion of irrelevant updates by an analysis of a set of examples in this section.[5] Afterward a definition of irrelevant updates is given in next section.

Example 21 ([6]).

$$P = cloudy \leftarrow raining$$
$$raining \leftarrow$$
$$U = not\ raining \leftarrow not\ cloudy$$

$RDSM(\langle P, U\rangle) = \{\{not\ raining, not\ cloudy\}, \{raining, cloudy\}\}$*. The assumption* $not\ it\_is\_cloudy$ *is falsified in* $\ll_{P \cup U}$ *because of* $it\_is\_cloudy \ll_{P \cup U} \emptyset$*. Information given by* U *does not override the information of* P *(which is based on the empty set of assumptions). The only TSSOA w.r.t.* $\ll_{P \cup U}$ *is* $\emptyset$*.* □

In general, troubles with all semantics based on rejection of rules are caused also by a too free choice of an interpretation involved in the fixpoint condition (1). We mean an interpretation containing assumptions (default negations) which are falsified by the multiprogram. Interpretations generated by falsified assumptions do not provide an appropriate candidate for a semantic characterization of a multiprogram (according to our

[5] Multiprograms of the form $\langle P, U\rangle$ are used in the examples. However, the dependency framework is used in the analysis and our intuitions apply to an arbitrary NMKB which can be mapped to the dependency framework.

view). A remark is in place: conflicts involving assumptions did not attract an adequate attention until now.

If an update U has only such TSSOAs w.r.t. $\ll_U$ which are falsified in $\ll_{P \cup U}$, we consider it as irrelevant. However, some further criteria of irrelevant updates are needed. The first, rather naive, idea how to understand the principle of minimal change for the case $\langle P, U \rangle$ is as follows: if $P \cup U$ is a coherent program, then an update of P specified by U is irrelevant and the meaning of $P \cup U$ is retained by inertia.[6]

Next example shows that that idea is not an appropriate one.

Example 22. *Let P be $\{not\ a \leftarrow not\ b\}$ and U be $\{a \leftarrow not\ b; b \leftarrow not\ a\}$.*

$P \cup U$ has only one stable model $S = \{not\ a, b\}$. However, if we respect the preference of U over P then we have to ignore the information of P. The dependence of $not\ a$ on $not\ b$ should be ignored. Hence, also $\{not\ b\}$ is a TSSOA w.r.t. the modified dependency relation and interpretation $S' = \{not\ b, a\}$ represents an intended meaning of $\langle P, U \rangle$, too.

We emphasize the role of (new) assumptions. Acceptance of new assumptions can provide a basis for a generation of some alternative belief sets. In general, this observation may be relevant for investigation of hypothetical reasoning. □

Next step when looking for a formalization of irrelevant updates may be 2[KM]. It can be expressed in terms of logic program updates as follows: if $P \models_{SM} U$ then update of P by U is equivalent to P. It means, a (stable-models-like) semantic characterization of $\langle P, U \rangle$ should coincide with stable models of P. It is straightforward to show that a weaker condition than coherence of $P \cup U$ is supposed in 2[KM], if P is coherent.

Proposition 23. *Let P be coherent. If $P \models_{SM} U$ then $P \cup U$ is coherent (but not vice versa).*

Note that condition $P \models_{SM} U$ is an important one. If we add seemingly irrelevant (cyclic) update U to a program P and $P \not\models_{SM} U$, then this may lead to cutting off some models.

Example 24 ([3]).

$$P = \{friends \leftarrow not\ alone \quad alone \leftarrow not\ friends \quad happy \leftarrow not\ depressed \quad depressed \leftarrow not\ happy\}$$

$$U = \{depressed \leftarrow alone \quad alone \leftarrow depressed\}$$

P has four models (only first letters are used):$\{f, d, not\ a, not\ h\}, \{f, h, not\ a, not\ d\}$, $\{a, h, not\ d, not\ f\}, \{a, d, not\ h, not\ f\}$. Notice that $P \not\models_{SM} U$. It is natural to reject the models of P which do not satisfy the more preferred program U. U eliminates two of the models of P: $\{f, d, not\ a, not\ h\}$ and $\{a, h, not\ d, not\ f\}$ (if alone is true, depressed is forced to be true and vice versa). □

[6] Consider Example 21. The only stable model of $P \cup U$, $\{it\_is\_raining, it\_is\_cloudy\}$, provides a reasonable semantic characterization of $\langle P, U \rangle$.

It seems that 2[KM] could be a criterion of irrelevant updates of logic programs. Unfortunately, 2[KM] does not work as the criterion.

Example 25.

$$P = \{a_1 \leftarrow not\ b_1 \qquad U = \{b_2 \leftarrow not\ a_2\}$$
$$b_1 \leftarrow not\ a_1$$
$$a_2 \leftarrow not\ b_2$$
$$not\ b_2 \leftarrow not\ a_2\}$$

There are two stable models of P: $S_1 = \{not\ b_1, a_1, not\ b_2, a_2\}$ and $S_2 = \{not\ a_1, b_1, not\ b_2, a_2\}$. U is satisfied both in S_1 and in S_2. Note that assumption $not\ b_2$ holds in both models.

However, there is no reason to reject an alternative assumption $not\ a_2$ (which is false in both stable models of P). The set of assumptions $\{not\ a_2\}$ is a SSOA w.r.t. $\ll_U$ and $Cn_{\ll_P}(\{not\ a\}) \cup Cn_{\ll_U}(\{not\ a\})$ is inconsistent. The inconsistency can be overridden if we prefer $b_2 \ll_U \{not\ a_2\}$ over $not\ b_2 \ll_P \{not\ a_2\}$.

Hence, it is reasonable to accept also interpretations $S_3 = \{not\ b_1, a_1, not\ a_2, b_2\}$ and $S_4 = \{not\ a_1, b_1, not\ a_2, b_2\}$ as intended meanings of the updated program. U is really a relevant update: it provides a reasonable alternative assumption $not\ a_2$. By "reasonable" we mean that unwanted dependencies on $\{not\ a_2\}$ are overridden because of the preference relation. □

If a set of axioms is extended in a monotonic logic then the set of models is reduced or the same. However, in NMKBs (and in stable model semantics, too) it is not true:

Example 26 ([1]). *Let be $P = \{a \leftarrow not\ b; b \leftarrow not\ a; c \leftarrow not\ a; c \leftarrow not\ c\}, U = \{c \leftarrow\}$.*

While P has the only stable model $S = \{not\ a, b, c\}$, $P \cup U$ has two stable models – besides S also $S' = \{a, not\ b, c\}$. Observe that the only model of P encodes in a way that the truth of c is dependent on the assumption $not\ a$. Hence, the dependence of beliefs on assumptions is implicit also in the stable model semantics.

Note that $P \models_{SM} U$, but $Cn_{\ll_U}(\emptyset) \setminus Cn_{\ll_P}(\emptyset) \neq \emptyset$. Some literals depend on $\emptyset$ w.r.t. $\ll_U$, but they do not depend on $\emptyset$ w.r.t. $\ll_P$. This could be generalized to a criterion of a relevant update. □

Example 27.

$$P = \{a \leftarrow not\ b\} \qquad U = \{b \leftarrow not\ a\}$$

$P \models_{SM} U$, but U introduces a new assumption, which is false in all stable models of P and generates a new stable model of $P \cup U$. □

In order to summarize: If $P \not\models_{SM} U$, then U is a relevant update of P. Otherwise, if $P \models_{SM} U$, then U is a relevant update of P in two (classes of) cases. First, U introduces assumptions, which contribute to a new TSSOA w.r.t. $\ll_{P \cup U}$ (see Example 26 or Example 27). Second, U introduces a set of assumptions which is inconsistent

with P, but a coherent view on $\ll_{P\cup U}$ is possible thanks to the preference relation (see Example 25).

Let $P \models_{SM} U$. A set of assumptions $Xs \subseteq Ngt$ of a relevant update U satisfies the conditions as follows. The conditions 1 - 3 are common to both classes of cases mentioned above. The condition 4 should be satisfied by the first case (contribution to a new TSSOA w.r.t. $\ll_{P\cup U}$). The conditions 5 and 6 should be satisfied by the second case.

1. Xs is not falsified in $\ll_{P\cup U}$,
2. Xs is false in each stable model of P,
3. $Xs \in SSOA(\ll_U)$ and $Cn_{\ll_U}(Xs) \setminus Xs \neq \emptyset$,
4. there is $Bs \subseteq Ngt$ s.t. $Xs \subseteq Bs$ and $Bs \in TSSOA(\ll_{P\cup U})$,
5. $Cn_{\ll_P}(Xs) \cup Cn_{\ll_U}(Xs)$ is inconsistent,
6. there is $View \subset \ll_{P\cup U}$ and $Bs \subseteq Ngt$ s.t. $Xs \subseteq Bs$ and $Bs \in TSSOA(View)$

Note that the condition 3 qualifies cyclic updates as irrelevant. Recall Example 2, it holds that $Cn_{\ll_U}(Xs) \setminus Xs = \emptyset$ for each set $Xs \subseteq Ngt$.

6 Irrelevant Updates – Formal Elaboration

Irrelevant updates are defined in this section for programs of the form $\langle P, U\rangle$ and for corresponding dependency relations $\ll_{P\cup U}$. However, we can abstract from programs – a generalization to arbitrary dependency relation (on which a preference is defined) is straightforward.

Convention 28. *Let $\langle P, U\rangle$ be a multiprogram. Then U is an irrelevant update of P iff $TSSOA(P \cup U) = TSSOA(P)$.*

We are going to specify criteria for $TSSOA(P\cup U) = TSSOA(P)$ in terms of assumptions and dependencies. First some trivial implications.

Proposition 29. *If $P \not\models_{SM} U$, then $TSSOA(P) \not\subseteq TSSOA(P \cup U)$*

Proposition 30. *If $TSSOA(P \cup U) = TSSOA(P)$, then $P \models_{SM} U$*

Proposition 31. *If $P \models_{SM} U$, then $TSSOA(P) \subseteq TSSOA(P \cup U)$.*

Example 27 shows that $TSSOA(P \cup U) \subseteq TSSOA(P)$ does not hold in general, if $P \models_{SM} U$.

The condition $P \models_{SM} U$ is a necessary, but not sufficient condition for irrelevancy of an update U of P, see Examples 25 and 27.

2[KM] is a very intuitive postulate for updates (an intuition of updates is also in the background of dynamic logic programming according to [10]). However, 2[KM] cannot be accepted literally for updates of NMKB. Defeasible (nonmonotonic) assumptions are not considered by Katsuno and Mendelzon in [8]. A careful treatment of assumptions and dependencies on assumptions is required for an appropriate understanding of updates of NMKB.

We can define now irrelevant update of a program.

Definition 32 (Irrelevant update). *Let $\langle P, U\rangle$ be a multiprogram, P be coherent and $P \models_{SM} U$.*

It is said that U is an irrelevant update of P iff there is no $Xs \subseteq Ngt$ s.t.

- $Xs \in SSOA(\ll_U)$ *and* $Cn_{\ll_U}(Xs) \setminus Xs \neq \emptyset$,
- Xs *is not falsified in* $\ll_{P\cup U}$,
- Xs *is false in each stable model of* P,
- *there is* $Bs \subseteq Ngt$ *s.t.* $Xs \subseteq Bs$ *and* $Bs \in TSSOA(\ll_{P\cup U})$ *or*
  - $Cn_{\ll_P}(Xs) \cup Cn_{\ll_U}(Xs)$ *is inconsistent,*
  - *there is* $View \subseteq \ll_{P\cup U}$ *and* $Bs \subseteq Ngt$ *s.t.* $Xs \subseteq Bs$ *and* $Bs \in TSSOA(View)$

A nondeterministic algorithm for computation of TSSOAs is presented in [12].

Theorem 33. *If U is an irrelevant update of P then $TSSOA(\ll_{P\cup U}) = TSSOA(\ll_P)$.* □

Note that the converse implication does not hold – see Example 25. The following trivial consequence states that the semantics of an original program P is preserved after an irrelevant update.

Consequence 34. *Let Xs be a TSSOA w.r.t. $\ll_P$ and U be an irrelevant update of P. Then Xs is a TSSOA w.r.t. $\ll_{P\cup U}$.*

7 Conclusions

A notion of irrelevant updates based on a dependency framework is introduced in the paper. The dependency framework provides a general base for discussing updates of NMKBs. The role of nonmonotonic assumptions in updates (and also in hypothetical, nonmonotonic reasoning) has been emphasized.

It has been shown that irrelevant updates do not generate new TSSOAs (sets of assumptions, which generate stable models, Theorem 33) and that TSSOAs corresponding to the original program generate also all stable models of updated program (Consequence 34). The dependency framework solves also troubles caused by tautological and cyclic updates, [12].

As regards our research concerning conditions of updates of NMKBs, a relevancy postulate can be added to postulates from [12].

Attention has been frequently focused on MDyLP in this paper. The reason is that MDyLP is a well understood idealization of NMKB and also because of importance of the problem of irrelevant updates in MDyLP. Finally, our approach to irrelevant updates is based on the dependency framework, which provides an alternative semantics of MDyLP [12]. However, our notion of irrelevant updates can be expressed independently on MDyLP and it is among our future goals.

A short comment concerning related work: a modified version of 2[KM], [8], which reflects the role of nonmonotonic assumptions, is presented. A more general view on irrelevant updates is given as in [3]. The notion of falsified assumptions enables to cover a more broad range of irrelevant updates. The trivial semantics (which satisfies REP) is not a problem for our approach.

References

1. Alferes, J.J., Pereira, L.M.: Reasoning with logic programming. LNAI, Springer 1996
2. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In: Procs. of KR'98. (1998) 98–109
3. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica **1** (2005)
4. Banti, F., Alferes, J.J., Brogi, A., Hitzler, P.: The well supported semantics for multidimensional dynamic logic programs. LPNMR 2005, LNCS 3662, Springer, 356-368
5. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press, 2003.
6. Eiter, T., Sabbatini, G., Fink, M., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming (2002) 711–767
7. Homola, M.: Dynamic Logic Programming: Various Semantics Are Equal on Acyclic Programs. CLIMA V 2004: 78-95
8. Katsuno, H., Mendelzon, A.O.: On the difference between updating a knowledge base and revising it. Proc. of KR'91
9. Leite, J.A., Alferes, J.J., Pereira, L.M.: Multi-dimensional dynamic logic programming. In: Procs. of CLIMA'00. (2000) 17–26
10. Leite, J.A.: Evolving Knowledge Bases: Specification and Semantics. IOS Press (2003)
11. Leite, J. On Some Differences Between Semantics of Logic Program Updates. IBERAMIA 2004: 375-385
12. Šefránek, J.: Rethinking semantics of dynamic logic programming. Accepted for NMR 2006.
13. Šefránek, J.: Irrelevant updates of nonmonotonic knowledge bases. Accepted as a poster for ECAI 2006.
14. Šefránek, J.: Irrelevant updates and nonmonotonic assumptions. http://www.ii.fmph.uniba.sk/ sefranek/online/jelia06Full.ps (or pdf)

Towards Top-k Query Answering in Description Logics: The Case of DL-Lite

Umberto Straccia

ISTI - CNR, Pisa Italy
straccia@isti.cnr.it

Abstract. We address the problem of evaluating ranked top-k queries in description logics. The problem occurs whenever we allow queries such as "find cheap hotels close to the conference location" in which fuzzy predicates like cheap and close occur. We show how to efficiently compute the top-k answers of conjunctive queries with fuzzy predicates over DL-LITE like knowledge bases.

1 Introduction

Description Logics (DLs) [2] provide popular features for the representation of structured knowledge. Nowadays, DLs have gained even more popularity due to their application in the context of the *Semantic Web*. DLs play a particular role as they are essentially the theoretical counterpart of state of the art languages to specify ontologies, such as *OWL DL* [12]. It becomes also apparent that in these contexts, data are typically very large and dominate the intentional level of the ontologies. Hence, while in the above mentioned contexts one could still accept reasoning that is exponential on the intentional part, it is mandatory that reasoning is polynomial in the data size, i.e. in *data complexity* [21]. Only recently efficient management of large amounts of data and its computational complexity analysis has become a primary concern of research in DLs and in ontology reasoning systems [1, 4, 5, 7, 11, 13].

In this paper we start addressing a novel issue for DLs with huge data repositories, namely the problem of *evaluating ranked top-k queries*. So far, an answer to a query is a set of tuples that satisfy a query. Each tuple does or does not satisfy the predicates in the query. However, very often the information need of a user involves so-called *fuzzy predicates* [22]. For instance, a user may need:"Find *cheap* hotels *near* to the conference location". Here, *cheap* and *near* are fuzzy predicates. Unlike the classical case, tuples satisfy now these predicates to a score (usually in $[0, 1]$). In the former case the score may depend, e.g., on the price, while in the latter case it may depend e.g. on the distance between the hotel location and the conference location. Therefore, a major problem we have to face with in such cases is that now an answer is a set of tuples *ranked* according to their *score*. This poses a new challenge in case we have to deal with a huge amount of instances. Indeed, virtually every tuple may satisfy a query with a non-zero score and, thus, has to be ranked. Computing all these scores, ranking them and then selecting the top-k ones is not feasible in practice, as we may deal with millions of tuples.

Our purpose is to address this problem for a DL-Lite like [4] description logic. DL-Lite has been specifically tailored to capture some basic ontology language features, while keeping a low complexity of reasoning. Reasoning means not only computing

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 439–451, 2006.

the subsumption relationships between concepts, and checking satisfiability, but also answering complex queries (i.e. conjunctive queries) over a huge set of instances.

We extend DL-Lite by allowing fuzzy predicates to appear in the queries (we call the language DL-Lite) and propose methods to efficiently compute the top-k ranked answers. Similarly to DL-Lite, we may rely on existing techniques for top-k query answering developed in the context relational databases [6, 8, 16]. Furthermore, as for DL-Lite, query answering in DL-Lite is (sub) linear in data complexity, which makes the language appealing for real world scenarios.

We proceed as follows. In the next section we present DL-Lite. Then, we show how to efficiently compute the top-k answer set of conjunctive queries.

2 DL-Lite

As usual in DLs, *DL-Lite* allows for representing the domain of interest in terms of concepts, denoting sets of objects, and roles, denoting binary relations between objects. Similarly to DL-Lite [1], in DL-Lite concepts and roles are defined as follows:

$$\begin{array}{l} B \rightarrow A \mid \exists R \mid B_1 \sqcap B_2 \mid B_1 \sqcup B_2 \\ C \rightarrow A \mid \bot \mid \exists R \mid C_1 \sqcap C_2 \\ R \rightarrow P \mid P^- \end{array}$$

where A denotes an atomic concept and P denotes an atomic role. A role R can be either an atomic role P or its *inverse* P^-. B denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists R$, i.e. the standard DL construct of unqualified existential quantification. C denotes a *general concept*. A DL-Lite *knowledge base* is pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ and $\mathcal{A}$ are finite sets of DL-Lite axioms and assertions. $\mathcal{T}$ is the TBox and is used to represent intentional knowledge, while $\mathcal{A}$ is the ABox and is used to represent extensional knowledge. An *axiom* is of the form $B \sqsubseteq C$ (inclusion axiom) and $\mathtt{fun}(R)$ (functionality axiom). A functionality axiom expresses the functionality of a role. *Assertions* are of the form $B(a)$ (concept assertion) and $P(a, b)$ (role assertion). Assertions state the membership of an individual (resp. pair of individuals) to a basic concept (resp. role).

DL-Lite allows for querying the extensional knowledge of a KB by means of conjunctive queries of arbitrary complexity, where fuzzy predicates may appear. Furthermore, we allow disjunctive queries as well. A *conjunctive query* q over a knowledge base $\mathcal{K}$ is an expression of the form

$$q(\mathbf{x}, s) \leftarrow \exists \mathbf{y}.conj(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$$

where

1. $\mathbf{x}$ are the *distinguished variables*;
2. s is the *score variable*, taking values in $[0, 1]$;
3. $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables*;
4. $conj(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms of the form $A(z)$, or $P(z, z')$, where A and P are respectively an atomic concept and a role (but, not inverse role) in $\mathcal{K}$. z, z' are constants in $\mathcal{K}$ or variables in $\mathbf{x}$ or $\mathbf{y}$;
5. $\mathbf{z_i}$ are tuples of constants in $\mathcal{K}$ or variables in $\mathbf{x}$ or $\mathbf{y}$;

[1] See [5] for extensions of DL-Lite with their computational complexity analysis.

6. p_i is an n_i-ary *fuzzy predicate* assigning to each n_i-ary tuple $\mathbf{c}_i$ as *score* $p_i(\mathbf{c}_i) \in [0,1]$. We require that an n-ary fuzzy predicate p is *safe*, i.e. there is not an m-ary fuzzy predicate p' such that $m < n$ and $p = p'$. Informally, all parameters are needed in the definition of p;
7. f is a *scoring* function $f\colon [0,1]^n \to [0,1]$, which combines the scores of the n fuzzy predicates $p_i(\mathbf{c}_i)$ into an overall *query score* to be assigned to the score variable s. We assume that f is *monotone*, i.e., for each $\mathbf{v}, \mathbf{v}' \in [0,1]^n$ such that $\mathbf{v} \leq \mathbf{v}'$, $f(\mathbf{v}) \leq f(\mathbf{v}')$ holds, where $(v_1, \ldots, v_n) \leq (v'_1, \ldots, v'_n)$ iff $v_i \leq v'_i$ for all i. [2]

A *disjunctive query* $\mathbf{q}$ is a finite set of conjunctive queries in which all the rules have the same head.

Example 1. Suppose we have information about hotels and conferences. Assume we have a fuzzy predicate `close` measuring the closeness degree between hotels and conference locations, depending on the distance, and a fuzzy predicate `cheap`, which given the price determines how "cheap" a hotel is. We may ask to find cheap hotels close to a conference location, i.e. rank the hotels according to their degree of closeness and cheapness. We may represent the scenario in DL-Lite as follows.

$$\begin{aligned} \texttt{Hotel} &\sqsubseteq \exists \texttt{HasHLoc.Location} \\ \texttt{Hotel} &\sqsubseteq \exists \texttt{HasHPrice.Price} \\ \texttt{Conference} &\sqsubseteq \exists \texttt{HasCLoc.Location} \\ \texttt{Hotel} \sqcap \texttt{Conference} &\sqsubseteq \bot \end{aligned}$$

HasHLoc	
HotelID	HasLoc
h1	hl1
h2	hl2
⋮	⋮

HasCLoc	
ConfID	HasLoc
c1	cl1
c2	cl2
⋮	⋮

HasHPrice	
HotelID	Price
h1	150
h2	200
⋮	⋮

Then we may express our information need using the conjunctive query (`c1` is our conference location)

$$\begin{aligned} q(h,s) \leftarrow & \texttt{HasHLoc}(h, hl) \wedge \texttt{HasHPrice}(h, p) \wedge \\ & \texttt{HasCLoc}(\texttt{c1}, cl) \wedge s = \texttt{cheap}(p) \cdot \texttt{close}(hl, cl)\ . \end{aligned}$$

where the fuzzy predicates `cheap` and `close` are defined as

$$\begin{aligned} \texttt{close}(\texttt{hl}, \texttt{cl}) &= \max(0, 1 - \tfrac{\texttt{distance(hl,cl)}}{2000}) \\ \texttt{cheap}(\texttt{price}) &= \max(0, 1 - \tfrac{\texttt{price}}{300}) \end{aligned}$$

The `distance` function returns the distance between hotels and conferences, obtained from an external source, e.g. database relation or web page. Note that the scoring function is the product $f(\texttt{cheap}(p), \texttt{close}(hl, cl)) = \texttt{cheap}(p) \cdot \texttt{close}(hl, cl)$, which is monotone in its arguments `cheap` and `close`.

We want to retrieve the top-k answers according to the score s. Please note that it is not feasible to compute all scores first and then rank them (there may be a huge amount of hotels and conference locations).

[2] We assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant.

Note also that if we would like to find hotels, which are either cheap or close to the conference location, then we may use the disjunctive query:

$$q(h,s) \leftarrow \texttt{HasHPrice}(h,p) \wedge s = \texttt{cheap}(p)$$
$$q(h,s) \leftarrow \texttt{HasHLoc}(h,hl) \wedge \texttt{HasCLoc}(\texttt{c1},cl) \wedge s = \texttt{close}(hl,cl)$$

We point out that the above disjunctive query is different from the conjunctive query

$$q(h,s) \leftarrow \texttt{HasHLoc}(h,hl) \wedge \texttt{HasHPrice}(h,p) \wedge \texttt{HasCLoc}(\texttt{c1},cl) \wedge s = \max(\texttt{cheap}(p), \texttt{close}(hl,cl))$$

as in the former we may find hotels, which are close to the conference location, though the price is unknown. □

Form a semantics point of view, it is similar to the usual semantics for DLs. The major difference is that we consider a *fixed infinite domain* Δ. [3] We assume to have one object for each constant, denoting exactly that object. In other words, we have standard names [15], and we will not distinguish between the alphabet of constants and Δ. So, an *interpretation* is a first-order structure $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ and consists of a fixed infinite domain Δ with an interpretation function $\cdot^{\mathcal{I}}$ such that:

$$A^{\mathcal{I}} \subseteq \Delta \qquad P^{\mathcal{I}} \subseteq \Delta \times \Delta \qquad \bot^{\mathcal{I}} = \emptyset$$
$$(C_1 \sqcap C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}} \quad (P^-)^{\mathcal{I}} = \{\langle c,c'\rangle \mid \langle c',c\rangle \in P^{\mathcal{I}}\}$$
$$(C_1 \sqcup C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cup {C_2}^{\mathcal{I}} \quad \exists R^{\mathcal{I}} = \{c \mid \exists c'.\langle c,c'\rangle \in R^{\mathcal{I}}\}$$

An interpretation $\mathcal{I}$ is a *model* of (i) an inclusion axiom $B \sqsubseteq C$ iff $B^{\mathcal{I}} \subseteq C^{\mathcal{I}}$; (ii) a functionality axiom $\texttt{fun}(R)$ iff $\langle c,c'\rangle \in R^{\mathcal{I}} \wedge \langle c,c''\rangle \in R^{\mathcal{I}} \Rightarrow c' = c''$; $(iiii)$ an assertion $B(a)$ (resp. $R(a,b)$) iff $a \in C^{\mathcal{I}}$ (resp. $(a,b) \in P^{\mathcal{I}}$); and (iv) a KB $\mathcal{K}$ iff $\mathcal{I}$ is a model of each axiom and assertion occurring in $\mathcal{K}$. A KB is *satisfiable* if it has a model. A KB $\mathcal{K}$ *entails* an assertion (resp. inclusion axiom) iff each model of the KB is also a model of the assertion (resp. inclusion axiom).

We recall that despite the simplicity of its language, the DL is able to capture the main notions (though not all, obviously) to represent structured knowledge. In particular, the axioms allow us to specify that concept A_1 is subsumed by concept A_2, using $A_1 \sqsubseteq A_2$; *disjointness*, e.g., between concepts A_1 and A_2, using $A_1 \sqcap A_2 \sqsubseteq \bot$; *role-typing*, using $\exists P \sqsubseteq A_1$ and $\exists P^- \sqsubseteq A_2$; *participation constraints*, using $A \sqsubseteq \exists P$ and $A \sqsubseteq \exists P^-$; *non-participation constraints*, using $A \sqcap \exists P \sqsubseteq \bot$ and $A \sqcap \exists P^- \sqsubseteq \bot$. Additionally, observe that we allow cyclic axioms. Notice that DL-Lite is a strict subset of OWL Lite and, thus of OWL DL [12], which presents some constructs (e.g., some kinds of role restrictions) that are non expressible in DL-Lite(and that make reasoning in OWL Lite non-tractable in general).

Concerning queries, informally a conjunctive query $q(\mathbf{x},s) \leftarrow \exists \mathbf{y}.conj(\mathbf{x},\mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1),\ldots,p_n(\mathbf{z}_n))$ is interpreted in an interpretation $\mathcal{I}$ as the set $q^{\mathcal{I}}$ of tuples $\langle \mathbf{c}, v\rangle$, such that when we substitute the variables $\mathbf{x}$ and s with the constants $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and the real value $v \in [0,1]$, the formula $\exists \mathbf{y}.conj(\mathbf{x},\mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1),\ldots,p_n(\mathbf{z}_n))$ evaluates to true in $\mathcal{I}$. But, due to the existential quantification $\exists \mathbf{y}$, for a fixed $\mathbf{c}$, there may be many substitutions $\mathbf{c}'$ for $\mathbf{y}$ and, thus, we may have many possible scores for the tuple $\mathbf{c}$. Among all these scores for $\mathbf{c}$, we select the highest one. Furthermore, if the

[3] The domain is infinite as pointed out in [4].

query is a disjunctive query $\mathbf{q}$, for each tuple $\mathbf{c}$ there may be a score v_i computed by each conjunctive query $q_i \in \mathbf{q}$. In that case, the overall score for $\mathbf{c}$ is the maximum among the scores v_i. Specifically, we assume that the score combination function f and the fuzzy predicates p_i have a given *fixed interpretation*. Now, let $\theta = \{\mathbf{x}/\mathbf{c}, \mathbf{y}/\mathbf{c}', s/v\}$ be a substitution of the variables $\mathbf{x}, \mathbf{y}$ and s with the tuples $\mathbf{c}, \mathbf{c}'$ and score value $v \in [0,1]$. Let $\psi(\mathbf{x}, \mathbf{y}, s)$ be $conj(\mathbf{x}, \mathbf{y}) \wedge s = f(p_1(\mathbf{z}_1), \ldots, p_n(\mathbf{z}_n))$. With $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ we denote the ground formula obtained by applying to $\psi(\mathbf{x}, \mathbf{y}, s)$ the substitution θ. We say that an interpretation $\mathcal{I}$ is a *model* of $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ iff $\psi(\mathbf{x}, \mathbf{y}, s)\theta$ evaluates to true in $\mathcal{I}$. We will write $\mathcal{I} \models \psi(\mathbf{x}, \mathbf{y}, s)\theta$ in this case. Then, the interpretation $\mathbf{q}^{\mathcal{I}}$ of a disjunctive query $\mathbf{q} = \{q_1, \ldots, q_m\}$ in $\mathcal{I}$ is

$$\mathbf{q}^{\mathcal{I}} = \{\langle \mathbf{c}, v\rangle \in \Delta \times \ldots \times \Delta \times [0,1] \mid v = \max(v_1, \ldots, v_m), \\ v_i = \sup_{\mathbf{c}' \in \Delta \times \ldots \times \Delta} \{v' \mid \mathcal{I} \models \psi_i(\mathbf{x}, \mathbf{y}, s)\theta'\}\},$$

where θ' is as θ, except that $\mathbf{y}$ is substituted with $\mathbf{c}'$ and s is substituted with v', each conjunctive query $q_i \in \mathbf{q}$ is of the form $q(\mathbf{x}, s) \leftarrow \exists \mathbf{y}.\psi_i(\mathbf{x}, \mathbf{y}, s)$, $\sup \emptyset$ is undefined, and $\max(v_1, \ldots, v_n)$ is undefined iff all its arguments are undefined. Therefore, some tuples $\mathbf{c}$ may not have a score in $\mathcal{I}$ and, thus, $\langle \mathbf{c}, v\rangle \notin \mathbf{q}^{\mathcal{I}}$ for no $v \in [0,1]$. Alternatively we may define $\sup \emptyset = 0$ and, thus, all tuples $\mathbf{c}$ have a score in $\mathcal{I}$, i.e. $\langle \mathbf{c}, v\rangle \in \mathbf{q}^{\mathcal{I}}$ for some $v \in [0,1]$. We use the former formulation to distinguish the case where a tuple $\mathbf{c}$ is retrieved, though the score is 0, from the tuples which do not satisfy the query and, thus, are not retrieved. Finally, for all $\mathbf{c} \in \Delta \times \ldots \times \Delta$ and for all $v \in [0,1]$, we say that $\mathcal{I}$ is a *model* of $q(\mathbf{c}, v)$ iff $\langle \mathbf{c}, v\rangle \in \mathbf{q}^{\mathcal{I}}$. Also, we say that a satisfiable KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ *entails* $q(\mathbf{c}, v)$, written $\mathcal{K} \models q(\mathbf{c}, v)$ iff any model $\mathcal{I}$ of $\mathcal{K}$ is also model of $q(\mathbf{c}, v)$ (note that $\mathcal{K}$ is required to be satisfiable).

The basic reasoning services that mainly concerns us is the knowledge base satisfiability problem and the top-k retrieval problem, where this latter is defined as:

Top-k retrieval: Given a satisfiable KB $\mathcal{K}$, retrieve the top-k ranked tuples $\langle \mathbf{c}, v\rangle$ that instantiate the disjunctive query $\mathbf{q}$ and rank them in decreasing order w.r.t. the score, i.e. find the top-k ranked tuples of the answer set of $\mathbf{q}$, denoted $ans_k(\mathcal{K}, \mathbf{q}) = \mathtt{Top}_k\{\langle \mathbf{c}, v\rangle \mid \mathcal{K} \models q(\mathbf{c}, v)\}$.

Some comments are in order on the form of the queries. Overall, our language extension to classical DLs, such as DL-Lite, concerns only the query language part and not the data representation language (which remains a classical DLs). This is exactly as it happens in top-k retrieval in the context of relational databases [6, 8, 16]: the data is represented as usual in relational tables and the SQL query language is extended to allow to express a scoring function as well, which may use the values occurring in the retrieved records, to compute the score of the record. By referring to Example 1, one may naturally ask why we do not allow to represent a fuzzy concept such as "cheap hotel" in the language and associate to each instance of it a score, as it happens usually in fuzzy DLs [17, 19]. Besides a semantical shift from a classical semantics to a fuzzy one (and, thus, likely changing the kind of inferences allowed), we assume here that queries are not defined once for ever, but may be issued by users to the systems. This means that it is not feasible to compute all scores in advance, as the queries are not known a priory. Furthermore, even the data may be available on query time only, e.g. if it has to be gathered from the Web ("find a flat with a big living room"). It is thus not surprising that most work on top-k retrieval in relation databases focusses on minimizing the

number of score function evaluations. What we will show here is that we can enhance query answering for classical DL-Lite with almost no additional effort.

In the following, for the sake of illustrative purposes, we consider the following abstract example.

Example 2. Suppose the set of inclusion axioms is $\mathcal{T} = \{\exists P_2^- \sqsubseteq A, A \sqsubseteq \exists P_1, B \sqsubseteq \exists P_2\}$. We also assume that the set of assertions $\mathcal{A}$ is stored in the three sets below (P_2 is a role, while B and C are basic concepts):

$$\begin{aligned} P_2 &= \{\langle 0, \mathtt{s}\rangle, \langle 3, \mathtt{t}\rangle, \langle 4, \mathtt{q}\rangle, \langle 6, \mathtt{q}\rangle\} \\ B &= \{\langle 1\rangle, \langle 2\rangle, \langle 5\rangle, \langle 7\rangle\} \\ C &= \{\langle 5\rangle, \langle 3\rangle, \langle 2\rangle, \langle 4\rangle\} \end{aligned}$$

Assume our disjunctive query is $\mathbf{q} = \{q', q''\}$ where q' is $q(x,s) \leftarrow \exists y \exists z.P_2(x,y) \wedge P_1(y,z) \wedge s = f(\mathtt{p}(x))$, q'' is $q(x,s) \leftarrow C(x) \wedge s = f(\mathtt{r}(x))$, the scoring function f is the identity $f(z) = z$ (f is monotone, of course), the fuzzy predicate $\mathtt{p}$ is $\mathtt{p}(x) = \max(0, 1 - x/10)$, and the fuzzy predicate $\mathtt{r}$ is $\mathtt{r}(x) = \max(0, 1 - (x/5)^2)$. Therefore, we can rewrite the query $\mathbf{q}$ as

$$\begin{aligned} q(x,s) \leftarrow & \exists y \exists z.P_2(x,y) \wedge P_1(y,z) \wedge s = \max(0, 1 - x/10) \\ q(x,s) \leftarrow & C(x) \wedge s = \max(0, 1 - (x/5)^2) \ . \end{aligned}$$

Now, it can be verified that $\mathcal{K} \models q(3, 0.7)$, $\mathcal{K} \models q(2, 0.84)$ and for any $v \in [0,1]$, $\mathcal{K} \not\models q(9, v)$. In the former case, any model $\mathcal{I}$ of $\mathcal{K}$ satisfies $P_2(3, \mathtt{t})$. But, $\mathcal{I}$ satisfies $\mathcal{T}$, so $\mathcal{I}$ satisfies $\exists P_2^- \sqsubseteq \exists P_1$. As $\mathcal{I}$ satisfies $P_2(3, \mathtt{t})$, $\mathcal{I}$ satisfies $(\exists P_2^-)(\mathtt{t})$ and, thus, $(\exists P_1)(\mathtt{t})$. As $0.7 = \max(0, 1 - 3/10)$, it follows that $\langle 3, 0.7\rangle$ evaluates the body of q' true in $\mathcal{I}$. On the other hand, $\langle 3, 0.64\rangle$ evaluates the body of q'' true in $\mathcal{I}$. Hence, the maximal score for 3 is 0.7, i.e., $\mathcal{I}$ is a model of $q(3, 0.7)$. The other cases can be shown similarly. In summary, it can be shown that the top-4 answer set of $\mathbf{q}$ is $ans_4(\mathcal{K}, \mathbf{q}) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$. □

3 Top-k Query Answering

We discuss now how to determine the top-k answers of a disjunctive query over a DL-Lite knowledge base $\mathcal{K}$. To this end:

1. We have to check if $\mathcal{K}$ is satisfiable, as querying a non-satisfiable KB is undefined in our case.
2. By considering $\mathcal{T}$ only, the user query $\mathbf{q}$ is *reformulated* into a set of conjunctive queries $r(\mathbf{q}, \mathcal{T})$. Informally, the basic idea is that the reformulation procedure closely resembles a top-down resolution procedure for logic programming, where each inclusion axiom $B_1 \sqsubseteq B_2$ is seen as a logic programming rule of the form $B_2(x) \leftarrow B_1(x)$. For instance, given the query $q(x,s) \leftarrow A(x) \wedge s = f(\ldots)$ and suppose that $\mathcal{T}$ contains the inclusion axioms $B_1 \sqsubseteq A$ and $B_2 \sqsubseteq A$, then we can reformulate the query into two queries $q(x,s) \leftarrow B_1(x) \wedge s = f(\ldots)$ and $q(x,s) \leftarrow B_2(x) \wedge s = f(\ldots)$, exactly as it happens for top-down resolution methods in logic programming.
3. The reformulated queries in $r(\mathbf{q}, \mathcal{T})$ are *evaluated* over $\mathcal{A}$ only (which is stored in a database), producing the requested top-k answer set $ans_k(\mathcal{K}, \mathbf{q})$. For instance, for the previous query, the answers will be the top-k answers of the union of the answers produced by all three queries.

A feature of DL-Lite is that satisfiability checking and query reformulation is as for DL-Lite [4] [4], while the top-k evaluation step is novel. For the sake of completeness of the paper, we start with step 1 and step 2 above. So, we start by preparing our knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ for effective management. That means, we first normalize it into a suitable form and then store the data in $\mathcal{A}$ into a relational database.

KB normalization. The *normalization* of $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is obtained by transforming $\mathcal{K}$ as follows. $\mathcal{A}$ is expanded by adding to $\mathcal{A}$ the assertions $(\exists P)(a)$ and $(\exists P^-)(b)$ for each $P(a, b) \in \mathcal{A}$. Concerning $\mathcal{T}$, we split concept conjunctions using the rule: if $\mathcal{T}$ contains $B \sqsubseteq C_1 \sqcap C_2$, then replace it with the two axioms $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$. Similarly, we split concept disjunctions: if $\mathcal{T}$ contains $B_1 \sqcup B_2 \sqsubseteq C$, then replace it with the two axioms $B_1 \sqsubseteq C$ and $B_2 \sqsubseteq C$. Furthermore, let $\sqsubseteq^*$ be the reflexive and transitive closure of the $\sqsubseteq$ relation over the roles inclusion axioms in $\mathcal{T}$.

Now $\mathcal{T}$ contains role inclusion axioms, functionality axioms and concept inclusion axioms of the form $B \sqsubseteq C$, according to the syntax rules

$$B \rightarrow A \mid \exists R \mid B_1 \sqcap B_2$$
$$C \rightarrow A \mid \bot \mid \exists R\,.$$

Then $\mathcal{T}$ is expanded by closing it with respect to the following inference rule: if $B_1 \sqcap C \sqsubseteq \bot$ occurs in $\mathcal{T}$ and $B_2 \sqsubseteq C$ occurs in $\mathcal{T}$, then add $B_1 \sqcap B_2 \sqsubseteq \bot$ to $\mathcal{T}$ (the rule applies also if B_1 is omitted).

It is easy to show that the normalization process transforms $\mathcal{K}$ into a model preserving form. In the following, without loss of generality we assume that every concept name or role name occurring in $\mathcal{A}$ also occurs in $\mathcal{T}$. Now we store $\mathcal{A}$ in a relational database. That is, (i) for each basic concept B occurring in $\mathcal{A}$, we define a relational table tab_B of arity 1, such that $\langle a \rangle \in tab_B$ iff $B(a) \in \mathcal{A}$; and (ii) for each role P occurring in $\mathcal{A}$, we define a relational table tab_P of arity 2, such that $\langle a, b \rangle \in tab_P$ iff $P(a, b) \in \mathcal{A}$. We denote with $\mathtt{DB}(\mathcal{A})$ the relational database thus constructed.

KB satisfiability. To check the satisfiability of a normalized KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, we verify the following conditions, which can easily be derived from [4]: (i) there exists $B \sqsubseteq \bot \in \mathcal{T}$ and a constant a such $B(a) \in \mathcal{A}$; (ii) there exists $B_1 \sqcap B_2 \sqsubseteq \bot \in \mathcal{T}$ and a constant a such $\{B_1(a), B_2(a)\} \subseteq \mathcal{A}$; (iii) there exists an axiom $\mathtt{fun}(P)$ (respectively, $\mathtt{fun}(P^-)$) in $\mathcal{T}$ and three constants a, b, c such that both $P(a, b)$ and $P(a, c)$ (resp., $P(b, a)$ and $P(c, a)$) belong to $\mathcal{A}$; If one of the conditions above holds, then $\mathcal{K}$ is not satisfiable. Otherwise, $\mathcal{K}$ is satisfiable. Note that the algorithm can verify such conditions by posing to $\mathtt{DB}(\mathcal{A})$ simple SQL queries.

Query reformulation. The query reformulation step is adapted from [4] to our case and is as follows. We say that a variable in a conjunctive query is *bound* if it corresponds to either a distinguished variable or a shared variable, i.e., a variable occurring at least twice in the query body (inclusive the scoring function), or a constant, while we say that a variable is *unbound* if it corresponds to a non-distinguished non-shared variable (as usual, we use the symbol "_" to represent non-distinguished non-shared variables). Note that an atom of the form $(\exists P)(x)$ (resp. $(\exists P^-)(x)$) has the same meaning as $P(x, \_)$ (resp. $P(\_, x)$). For ease of exposition, in the following we will use the latter form only. An axiom τ is *applicable* to an atom $B(x)$, if τ has B in its right-hand side, and τ is

[4] Strictly speaking, DL-Lite does not support disjunctive queries, though it can easily be extended to that case.

applicable to an atom $P(x_1, x_2)$, if either (i) $x_2 = \_$ and the right-hand side of τ is $\exists P$, or (ii) $x_1 = \_$ and the right-hand side of τ is $\exists P^-$. We indicate with $gr(g; \tau)$ the atom obtained from the atom g by applying the inclusion axiom τ. Specifically, if $g = B_1(x)$ (resp., $g = P_1(x, \_)$ or $g = P_1(\_, x)$) and $\tau = B_2 \sqsubseteq B_1$ (resp., $\tau = B_2 \sqsubseteq \exists P_1$ or $\tau = B_2 \sqsubseteq \exists P_1^-$), we have:

- $gr(g, \tau) = A(x)$, if $B_2 = A$, where A is an atomic concept;
- $gr(g, \tau) = P_2(x, \_)$, if $B_2 = \exists P_2$;
- $gr(g, \tau) = P_2(\_, x)$, if $B_2 = \exists P_2^-$;
- $gr(g, \tau) = B_3(x) \wedge B_4(x)$, if $B_2 = B_3 \sqcap B_4$.

We are now ready to present the query reformulation algorithm. Given a disjunctive query $\mathbf{q}$ and a set of axioms $\mathcal{T}$, the algorithm reformulates $\mathbf{q}$ in terms of a set of conjunctive queries $r(\mathbf{q}, \mathcal{T})$, which then can be evaluated over $\mathtt{DB}(\mathcal{A})$. In the algo-

Algorithm 1. $\mathtt{QueryRef}(\mathbf{q}, \mathcal{T})$

Input: Disjunctive query $\mathbf{q}$, DL-Lite axioms $\mathcal{T}$.
Output: Set of reformulated conjunctive queries $r(\mathbf{q}, \mathcal{T})$.

1: $r(\mathbf{q}, \mathcal{T}) := \mathbf{q}$
2: **repeat**
3: $S = r(\mathbf{q}, \mathcal{T})$
4: **for all** $q \in S$ **do**
5: **for all** $g \in q$ **do**
6: **if** $\tau \in \mathcal{T}$ is applicable to g **then**
7: $r(\mathbf{q}, \mathcal{T}) := r(\mathbf{q}, \mathcal{T}) \cup \{q[g/gr(g, \tau)]\}$
8: **for all** $g_1, g_2 \in q$ **do**
9: **if** g_1 and g_2 unify **then**
10: $r(\mathbf{q}, \mathcal{T}) := r(\mathbf{q}, \mathcal{T}) \cup \{\kappa(reduce(q, g_1, g_2))\}$
11: **until** $S = r(\mathbf{q}, \mathcal{T})$
12: $r(\mathbf{q}, \mathcal{T}) := removeSubs(r(\mathbf{q}, \mathcal{T}))$
13: **return** $r(\mathbf{q}, \mathcal{T})$

rithm, $q[g/g']$ denotes the query obtained from q by replacing the atom g with a new atom g'. At step 8, for each pair of atoms g_1, g_2 that unify, the algorithm computes the query $q' = reduce(q, g_1, g_2)$, by applying to q the most general unifier between g_1 and g_2 [5]. Due to the unification, variables that were bound in q may become unbound in q'. Hence, inclusion axioms that were not applicable to atoms of q, may become applicable to atoms of q' (in the next executions of step (5)). Function κ applied to q' replaces with $\_$ each unbound variable in q'. Finally, in step 12 we remove from the set of queries $r(\mathbf{q}, \mathcal{T})$, those which are already subsumed in $r(\mathbf{q}, \mathcal{T})$. The notion of query subsumption is similar as for the classical database theory [20]. Given two queries q_i $(i = 1, 2)$ with same head $q(\mathbf{x}, s)$ and $q_1 \neq q_2$, we say that q_1 *is subsumed by* q_2, denoted $q_1 \sqsubseteq q_2$, iff for any interpretation $\mathcal{I}$, ${q_1}^{\mathcal{I}} \preceq {q_2}^{\mathcal{I}}$, where this latter is defined as: ${q_1}^{\mathcal{I}} \preceq {q_2}^{\mathcal{I}}$ iff for all $\langle \mathbf{c}, v_1 \rangle \in {q_1}^{\mathcal{I}}$ there is $\langle \mathbf{c}, v_2 \rangle \in {q_2}^{\mathcal{I}}$ such that $v_1 \leq v_2$. Essentially, if $q_1 \sqsubseteq q_2$ and both q_1 and q_2 belong to $r(\mathbf{q}, \mathcal{T})$ then we can remove q_1 from $r(\mathbf{q}, \mathcal{T})$ as q_1 produces a lower ranked result than q_2 with respect to the same tuple $\mathbf{c}$. In order to decide query subsumption, we can take advantage of the results in [14], related to the query containment part. A condition for query subsumption is the following. Assume that q_1 and q_2 do not share any variable. This can be accomplished by renaming all variables in e.g. q_1. Then it can be shown that

[5] We say that two atoms $g_1 = r(x_1, \ldots, x_n)$ and $g_2 = r(y_1, \ldots, y_n)$ *unify*, if for all i, either $x_i = y_i$ or $x_i = \_$ or $y_i = \_$. If g_1 and g_2 unify, then the unification of g_1 and g_2 is the atom $r(z_1, \ldots, z_n)$, where $z_i = x_i$ if $x_i = y_i$ or $y_i = \_$, otherwise $z_i = y_i$ [3].

Proposition 1. *If q_1 and q_2 share the same score combination function, then $q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each predicate $P(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a predicate $P(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $P(\mathbf{z}_2) = P(\mathbf{z}_1)\theta$.*

More complicated are cases in which q_1 and q_2 do not share the same score combination function. For instance, given

$$\begin{aligned} q_1 &:= q(x_1, s_1) \leftarrow P(x_1, y_1) \wedge s_1 = y_1 \\ q_2 &:= q(x_2, s_2) \leftarrow P(x_2, y_2) \wedge s_2 = \min(1, (x_2 + y_2)/2) \\ q_3 &:= q(x, s) \quad \leftarrow P(x, y) \wedge s = \min(1, x + y) \end{aligned}$$

It can be shown that $q_2 \sqsubseteq q_3$ is the only subsumption relation among the queries above. Note that there is a variable substitution $\theta_{23} = \{x_2/x, y_2/y, s_2/s\}$ such that $P(x, y) = P(x_2, y_2)\theta_{23}$ and $\min(1, (x_2 + y_2)/2)\theta_{23} \leq \min(1, x + y)$, for all x, y. On the other hand, $q_1 \not\sqsubseteq q_2$. Note that we can find $\theta_{12} = \{x_1/x_2, y_1/y_2, s_1/s_2\}$ such that $P(x_2, y_2) = P(x_1, y_1)\theta_{12}$. However, $y_1\theta_{12} = y_2 \not\leq \min(1, (x_2 + y_2)/2)$, for all x_2, y_2. Similarly, $q_2 \not\sqsubseteq q_1$ and we can find $\theta_{21} = \{x_2/x_1, y_2/y_1, s_2/s_1\}$ such that $P(x_1, y_1) = P(x_2, y_2)\theta_{21}$ with $\min(1, (x_2 + y_2)/2)\theta_{21} = \min(1, (x_1 + y_1)/2) \not\leq y_1$, for all x_1, y_1. Hence, we can extend the query subsumption condition in Proposition 1 in the following way. Let q_1 and q_2 be two queries with same head and let σ_1 and σ_2 be the scoring component of q_1 and q_2, respectively. Then it can be shown that

Proposition 2. *$q_1 \sqsubseteq q_2$ iff there is a variable substitution θ such that for each predicate $P(\mathbf{z}_2)$ occurring in the rule body of q_2 there is a predicate $P(\mathbf{z}_1)$ occurring in the rule body of q_1 such that $P(\mathbf{z}_2) = P(\mathbf{z}_1)\theta$, and $\sigma_1\theta \leq \sigma_2$ for all variables occurring in $\sigma_1\theta$ and σ_2.*

Of course, the complexity of checking a condition such as $\sigma_1\theta \leq \sigma_2$ depends on the scoring functions and fuzzy predicates involved, and may be computationally expensive. We will not analyze this issue further in this paper and, thus, we assume that procedure $removeSubs$ removes subsumed queries according to Proposition 1, which is easy to check and not time consuming (we also could be more specific in the query subsumption definition, by restricting the interpretations to the models of the knowledge base, but this may lead to a query containment checking algorithm requiring a non-negligible amount of time). This concludes the query reformulation step.

Example 3. Consider Example 2. At step 1 $r(\mathbf{q}, \mathcal{T})$ is initialized with $\{q', q''\}$. It is easily verified that both conditions in step 6 and step 9 fail for q''. So we proceed with q'. Let σ be $s = \max(0, 1 - x/10)$. Then at the first execution of step 7, the algorithm inserts query q_1, $q(x, s) \leftarrow P_2(x, y) \wedge A(y) \wedge \sigma$ into $r(\mathbf{q}, \mathcal{T})$ using the axiom $A \sqsubseteq \exists P_1$. At the second execution of step 7, the algorithm inserts query q_2, $q(x, s) \leftarrow P_2(x, y) \wedge P_2(\_, y) \wedge \sigma$ using the axiom $\exists P_2^- \sqsubseteq A$. Since the two atoms of the second query unify, $reduce(q, g_1, g_2)$ returns $q(x, s) \leftarrow P_2(x, y) \wedge \sigma$ and since now y is unbound (y does not occur in σ), after application of κ, step 10 inserts the query q_3, $q(x, s) \leftarrow P_2(x, \_) \wedge \sigma$. At the third execution of step 7, the algorithm inserts query q_4, $q(x, s) \leftarrow B(x) \wedge \sigma$ using the axiom $B \sqsubseteq \exists P_2$ and stops.

Note that we need not to evaluate all queries q_i. Indeed, it can easily be verified that for each query q_i and all constants c, the scores of q_3 and q_4 are not lower than all the

other queries q_i and q'. That is, we can restrict the evaluation of the set of reformulated queries to $r(\mathbf{q}, \mathcal{T}) = \{q'', q_3, q_4\}$ only. As a consequence, the top-4 answers to the original query are $[\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle, \langle 3, 0.7\rangle]$, which are the top-4 ranked tuples of the union of the answer sets of q'', q_3 and q_4. □

Computing top-k answers. The main property of the query reformulation algorithm is as follows. It can be shown that

$$ans_k(\mathcal{K}, \mathbf{q}) = \mathtt{Top}_k\{\langle \mathbf{c}, v\rangle \mid q_i \in r(\mathbf{q}, \mathcal{T}), \mathcal{A} \models q_i(\mathbf{c}, v)\} .$$

The above property dictates that the set of reformulated queries $q_i \in r(\mathbf{q}, \mathcal{T})$ can be used to find the top-k answers, by evaluating them over the set of instances $\mathcal{A}$ only, without referring to the ontology $\mathcal{T}$ anymore. In the following, we show how to find the top-k answers of the union of the answer sets of conjunctive queries $q_i \in r(\mathbf{q}, \mathcal{T})$.

A naive solution to the top-k retrieval problem is as follows: we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$ the whole answer set $ans(q_i, \mathcal{A}) = \{\langle \mathbf{c}, v\rangle \mid \mathcal{A} \models q_i(\mathbf{c}, v)\}$, then we compute the union, $\bigcup_{q_i \in r(\mathbf{q}, \mathcal{T})} ans(q_i, \mathcal{A})$, of these answer sets, order it in descending order of the scores and then we take the top-k tuples. We note that each conjunctive query $q_i \in r(\mathbf{q}, \mathcal{T})$ can easily be transformed into an SQL query expressed over $\mathtt{DB}(\mathcal{A})$. The transformation is conceptually simple. The only non-trivial case concerns binary atoms with unbound terms: for any atom in a query $q_i \in r(\mathbf{q}, \mathcal{T})$ of the form $P(\_, x)$, we introduce a view predicate that represents the union of $tab_P[2]$ with $tab_{\exists P^-}$, where $tab_P[2]$ is the projection of tab_P on its second column (the case $P(x, \_)$ is similar). A major drawback of this solution is the fact that there might be too many tuples with non-zero score and hence for any query $q_i \in r(\mathbf{q}, \mathcal{T})$, all these scores should be computed and the tuples should be retrieved. This is *not feasible* in practice, as a there may be millions of tuples in the knowledge base.

A more effective solution consists in relying on existing top-k query answering algorithms for relational databases (see, e.g. [6, 8, 16]), which support efficient evaluations of ranking top-k queries in relational database systems. Though there is no work supporting top-k query answering for disjunctive queries, we can still profitably use top-k query answering methods for relational databases. Indeed, an immediate and much more efficient method to compute $ans_k(\mathcal{K}, \mathbf{q})$ is: we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$, the top-$k$ answers $ans_k(\mathcal{A}, q_i)$, using e.g. the system RankSQL [16] [6]. If both k and the number, $n_q = |r(\mathbf{q}, \mathcal{T})|$, of reformulated queries is reasonable, then we may compute the union, $U(q, \mathcal{K}) = \bigcup_{q_i \in r(\mathbf{q}, \mathcal{T})} ans_k(\mathcal{A}, q_i)$, of these top-$k$ answer sets, order it in descending order w.r.t. score and then we take the top-k tuples.

As an alternative, we can avoid to compute the whole union $U(q, \mathcal{K})$, so further improving the answering procedure, by relying on a *disjunctive* variant of the so-called *Threshold Algorithm* (TA) [9], which we call *Disjunctive TA* (DTA). We recall that the TA has been developed to compute the top-k answers of a conjunctive query with monotone score combination function. In the following we show that we can use the same principles of the TA to compute the top-k answers of the union of conjunctive queries, i.e. a disjunctive query.

1. First, we compute for all $q_i \in r(\mathbf{q}, \mathcal{T})$, the top-$k$ answers $ans_k(\mathcal{A}, q_i)$, using top-k rank-based relational database engine. Now, let us assume that the tuples in the top-k answer set $ans_k(\mathcal{A}, q_i)$ are sorted in decreasing order with respect to the score.

[6] RankSQL will be available in the middle of this year. Personal communication.

2. Then we process each top-k answer set $ans_k(\mathcal{A}, q_i)$ ($q_i \in r(\mathbf{q}, \mathcal{T})$) in parallel or alternating fashion, and top-down (i.e. the higher scored tuples in $ans_k(\mathcal{A}, q_i)$ are processed before the lower scored tuples in $ans_k(\mathcal{A}, q_i)$).
   (a) For each tuple $\mathbf{c}$ seen, if its score is one of the k highest we have seen, then remember tuple $\mathbf{c}$ and its score $s(\mathbf{c})$ (ties are broken arbitrarily, so that only k tuples and their scores need to be remembered at any time).
   (b) For each answer set $ans_k(\mathcal{A}, q_i)$, let s_i be the score of the last tuple seen in this set. Define the threshold value θ to be $\max(s_1, ..., s_{n_q})$. As soon as at least k tuples have been seen whose score is at least equal to θ, then halt (indeed, any successive retrieved tuple will have score $\leq \theta$).
   (c) Let Y be the set containing the k tuples that have been seen with the highest scores. The output is then the set $\{\langle \mathbf{c}, s(\mathbf{c})\rangle \mid \mathbf{c} \in Y\}$. This set is $ans_k(\mathcal{K}, \mathbf{q})$.

The following example illustrates the DTA.

Example 4. Consider Example 3. Suppose we are interested in retrieving the top-3 answers of the disjunctive query $\mathbf{q} = \{q', q''\}$. We have seen that it suffices to find the top-3 answers of the union of the answers to q_3, q_4 and to q''. Let us show how the DTA works. First, we submit q_3, q_4 and q'' to a rank-based relational database engine, to compute the top-3 answers. It can be verified that

$$\begin{aligned} ans_3(\mathcal{A}, q_3) &= [\langle 0, 1.0\rangle, \langle 3, 0.7\rangle, \langle 4, 0.6\rangle] \\ ans_3(\mathcal{A}, q_4) &= [\langle 1, 0.9\rangle, \langle 2, 0.8\rangle, \langle 5, 0.5\rangle] \\ ans_3(\mathcal{A}, q'') &= [\langle 2, 0.84\rangle, \langle 3, 0.64\rangle, \langle 4, 0.36\rangle] \ . \end{aligned}$$

The lists are in descending order w.r.t. the score from left to right. Now we process alternatively $ans_k(\mathcal{A}, q_3)$, then $ans_k(\mathcal{A}, q_4)$ and then $ans_k(\mathcal{A}, q'')$ in decreasing order of the score. The table below summaries the execution of our DTA algorithm. The ranked list column contains the list of tuples processed.

Step	Tuple	s_{q_3}	s_{q_4}	$s_{q''}$	θ	ranked list
1	$\langle 0, 1.0\rangle$	1.0	-	-	1.0	$\langle 0, 1.0\rangle$
2	$\langle 1, 0.9\rangle$	1.0	0.9	-	1.0	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle$
3	$\langle 2, 0.84\rangle$	1.0	0.9	0.84	1.0	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle$
4	$\langle 3, 0.7\rangle$	0.7	0.9	0.84	0.9	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle\langle 3, 0.7\rangle$
5	$\langle 2, 0.8\rangle$	0.7	0.8	0.84	0.84	$\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle\langle 3, 0.7\rangle$

At step 5 we stop as the ranked list already contains three tuples above the threshold $\theta = 0.84$. So, the final output is

$$ans_k(\mathcal{A}, q_3) = [\langle 0, 1.0\rangle, \langle 1, 0.9\rangle, \langle 2, 0.84\rangle] \ .$$

Note that not all tuples have been processed. □

As computing the top-k answers of each query $q_i \in r(\mathbf{q}, \mathcal{T})$ requires (sub) linear time w.r.t. the database size (using, e.g. [6]), it is easily verified that the disjunctive TA algorithm is linear in data complexity.

Proposition 3. *Given a DL-Lite KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ and a disjunctive query $\mathbf{q}$ then the DTA computes $ans_k(\mathcal{K}, \mathbf{q})$ in (sub) linear time w.r.t. the size of $\mathcal{A}$.*

Furthermore, the above method has the non-negligible advantage to be based on existing technology for answering top-k queries over relational databases, improves significantly the naive solution to the top-k retrieval problem, and is rather easy to implement.

4 Conclusions

DLs have been proposed as a mean to describe structured knowledge and find a natural application in the context of the Semantic Web. We have presented DL-Lite in which fuzzy predicates are allowed to appear in conjunctive queries. Thus, we may express queries such as "find *cheap* hotels". Such queries are already very common on the Web. To the best of our knowledge, this is the first time this problem has been addressed for classical DLs. A major distinction of DL-Lite is that an answer to a query is a set of tuples *ranked* according to their score. As a consequence, whenever we deal with a huge amount of tuples, the ranking of the answer set becomes the major problem that has to be addressed.

We have shown how to answer disjunctive queries efficiently over a huge set of instances. The main ingredients of our solution is a simple and effective query reformulation procedure, the use of existing top-k query answering technology over relational databases and the DTA algorithm. Indeed, a user query is reformulated into a set of conjunctive queries using the inclusion axioms only and, then, the reformulated queries can be submitted to the top-k query answering engine over a relational database where the tuples have been stored. Finally, the DTA algorithm performs the final computation to retrieve the actual top-k results. We point out that, due to the results described in [5], it is difficult to extend the language proposed here with additional constructs. For instance, it is shown that adding qualified role restrictions $\exists R.C$ would lead to a NLOGSPACE data complexity, which rules out the possibility of using current top-k relational database technology. Furthermore, the complexity result shows that it is the same as for DL-Lite and, thus, whenever DL-Lite can be considered as useful, so is DL-Lite as well.

We note that in [18] we considered the case of top-k query answering within fuzzy DL-Lite. [18] and this work are orthogonal in the sense that in [18] tuples may have a score, but no score combination function is allowed in the query language, while here we consider a classical semantics with score combination function in the query language. The combination of both features is an open direction for further research. Additionally, other topics for future research may be: (i) to verify the applicability of our method to other tractable DLs such as [1]; (ii) to address the problem of top-k query answering to more expressive DLs than DL-Lite; (iii) to improve the DTA by using more sophisticated, but better performing TA-based algorithms such as [10]; and (iv) to improve the core top-k conjunctive query answering technology towards the management of the disjunctive queries.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, pages 364-369, 2005.
2. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
3. Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 16–21, 2003.

4. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, 2005.
5. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. In *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, 2005.
6. Kevin Chen-Chuan Chang and Seung won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD Conference*, 2002.
7. CuiMing Chen, Volker Haarslev, and JiaoYue Wang. Las: Extending racer by a large abox store. In Ian Horrocks, Ulrike Sattler, and Frank Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL-05)*, 2005.
8. Ronald Fagin. Combining fuzzy information: an overview. *SIGMOD Rec.*, 31(2):109–118, 2002.
9. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Symposium on Principles of Database Systems*, 2001.
10. Ulrich Güntzer, Wolf-Tilo Balke, and Werner Kiesling. Optimizing multi-feature queries for image databases. In *The VLDB Journal*, pages 419–428, 2000.
11. Ian Horrocks, Lei Li, Daniele Turi, and Sean Bechhofer. The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40, 2004.
12. Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.
13. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Data complexity of reasoning in very expressive description logics. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 466–471, 2005.
14. Laks V.S. Lakshmanan and Nematollaah Shiri. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
15. Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases*. MIT Press, 2001.
16. Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. RankSQL: query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, pages 131–142, 2005.
17. Umberto Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.
18. Umberto Straccia. Answering vague queries in fuzzy dl-lite. In *Proceedings of the 11th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, (IPMU-06)*, 2006.
19. Umberto Straccia. A fuzzy description logic for the semantic web. In Elie Sanchez, editor, *Fuzzy Logic and the Semantic Web*, Capturing Intelligence, chapter 4, pages 73–90. Elsevier, 2006.
20. J. D. Ullman. *Principles of Database and Knowledge Base Systems*, volume 1,2. Computer Science Press, Potomac, Maryland, 1989.
21. M. Vardi. The complexity of relational query languages. In *Proc. of the 14th ACM SIGACT Sym. on Theory of Computing (STOC-82)*, pages 137–146, 1982.
22. L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Representing Causal Information About a Probabilistic Process

Joost Vennekens, Marc Denecker, and Maurice Bruynooghe

Dept. Computerscience, K.U. Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{joost, marcd, maurice}@cs.kuleuven.be

Abstract. We study causal information about probabilistic processes, i.e., information about *why* events occur. A language is developed in which such information can be formally represented and we investigate when this suffices to uniquely characterize the probability distribution that results from such a process. We examine both detailed representations of temporal aspects and representations in which time is implicit. In this last case, our logic turns into a more fine-grained version of Pearl's approach to causality. We relate our logic to certain probabilistic logic programming languages, which leads to a clearer view on the knowledge representation properties of these language. We show that our logic induces a semantics for disjunctive logic programs, in which these represent non-deterministic processes. We show that logic programs under the well-founded semantics can be seen as a language of deterministic causality, which we relate to McCain & Turner's causal theories.

1 Introduction

If we want to construct a probabilistic model of some domain, it is often worthwhile to wonder where precisely our uncertainty comes from. Typically, the current state of such a domain can be seen as the result of some probabilistic process, i.e., there has been some sequence of events which has lead us to this state. We can now be uncertain about the state of the domain, because, for instance, we might not precisely know the effects of some events, or because some events are inherently non-deterministic, or because we do not exactly know which events have happened. There are a number of probabilistic modeling languages that allow the dynamic evolution of a domain to be represented in full detail. In these formalisms, one defines a probability distribution by characterizing the process that has generated it. An example of this kind of approach is Halpern's work on combining knowledge and probability [3]. Often, however, we are not really interested in such a process itself, but merely want to know the probability distribution that results from it. In this case, it might suffice to only represent certain salient properties of this process. We claim that *causal* information is particularly useful for this purpose. One of the most popular causal approaches to probabilistic modeling is that of causal Bayesian networks [6]. Now, such a Bayesian network can indeed be seen as a highly abstracted representation of a

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 452–464, 2006.

probabilistic process, in which the value of every node is determined by some sequence of events that propagates the values of its parents to the node itself.

In this work, we develop a more flexible and fine-grained representation of causal information about a probabilistic process, which distinguishes itself from Bayesian networks by offering an explicit, strucural representation of causal probabilistic events. In the first part of this paper, we construct a logic that can represent knowledge about the causes and effects of probabilistic events and investigate when such information suffices to uniquely characterize a probability distribution. We also study the role of time in this language and show that, on the one hand, very detailed encodings of probabilistic processes are possible, while, on the other hand, temporal information can often also be abstracted away, leading to more static representations of causal relations.

Our treatment of causality is similar in spirit to the way causality is typically handled in logical formalisms, such as McCain & Turner's causal logic [5]. In the second part of this paper, we formally investigate this relation. This will lead to a theorem, that shows that our logic almost completely coincides with the probabilistic logic programming language of Logic Programs with Annotated Disjunctions [12], which is known to have strong ties to Poole's Independent Choice Logic [7]. As such, the previous analysis provides additional motivation for these languages, showing that they are not only meaningful combinations of logic programming and probability, but that they also arise naturally out of a desire to represent causal information about probabilistic processes.

To summarize, the contributions of this paper are the following:

- We investigate how causal information about a probabilistic process can be used to represent a probability distribution.
- We study the role of time in our logic and show that it allows both detailed representations of dynamic processes and more static, Bayesian network style models.
- We relate our logic to logic programming based approaches to probabilistic modeling. This result provide additional motivation for these approaches and helps to clarify their knowledge representation methodology.

Proofs of the theorems in this paper can be found in [11].

2 Causal Information About Probabilistic Processes

In this section, we develop a logic that represents causal information about probabilistic processes in a natural way. We begin by explicating what we mean by a probabilistic process. Such a process is a sequence of probabilistic events. Each of these events affects the state of the domain in some way. Which event happens at any particular time may depend on the state of the domain at that moment. To formally represent such a process, we assume that we have a logical vocabulary, which allows us to represent a state of the domain by a Herbrand interpretation, i.e., a set of ground atoms. We also assume that the effect of an event on the state of the domain corresponds to either no change at all, or one

ground atom becoming true. This suffices because we can easily encode other cases by choosing our vocabulary in an appropriate way. For instance, we can handle the case in which multiple properties would become true by introducing a new predicate symbols to represent the conjunction of all of these properties. To cover the case where we want to consider properties that are initially true and might become false, we can construct our vocabulary in such a way that there is an atom that represents the complement of this property. A property whose truth value might change more than once can be encoded by a set of ground atoms, one for each time point at which the truth of this property might change.

We write down such an event as $(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n)$, with the p_i ground atoms and the α_i probabilities with $\sum \alpha_i \leq 1$. Such an expression is read as: "At most one of the p_i will become true as a result of this event and the probability of each p_i becoming true is α_i." Note that an atom p_i does not represent an outcome of one particular event, but rather the effect of this outcome on the domain, i.e., if different events can have the same effect on the domain, they might share the same proposition. If an event has a deterministic effect, i.e., it always causes some atom p with probability 1, we also write p instead of $(p : 1)$.

A probabilistic process now corresponds to a tree structure, where each node s represents a particular state of the domain, i.e., to each such s there corresponds a Herbrand interpretation $\mathcal{I}(s)$. The interpretation associated to the root of this tree is $\{\}$, i.e., initially all atoms are false. Now, in every non-leaf state s, a single event $(p_1 : \alpha_1) \vee \cdots \vee (p_n : \alpha_n)$ occurs, i.e., the children of s are nodes $s_1, \ldots, s_{n+1}$, where $\mathcal{I}(s_1) = \mathcal{I}(s) \cup \{p_1\}$, $\ldots$, $\mathcal{I}(s_n) = \mathcal{I}(s) \cup \{p_n\}$, and $\mathcal{I}(s_{n+1}) = \mathcal{I}(s)$. Each such edge can also be labeled with a probability: for $1 \leq i \leq n$, the probability of going from s to s_i is α_i, and the probability of going from s to s_{n+1} is $1 - \sum \alpha_i$. Such a probabilistic process generates a probability distribution over its leaves, namely, the probability of a leaf is the product of the probabilities of all edges in the path from the root to this leaf. This leads in a straightforward way to a distribution over Herbrand interpretations: the probability of interpretation I is the sum of the probabilities of all leaves s for which $\mathcal{I}(s) = I$.

It is of course this last distribution that we are really interested in. A key observation is now that we do not need to know the entire probabilistic process in full detail, in order to know this distribution. In particular, the order in which certain events happen might be completely irrelevant. Our goal is now to develop a logic which allows one to represent enough properties of such a process to uniquely characterize this distribution, while ignoring irrelevant details. This will allow more compact definitions of probability distributions and lead to more general representations, that are less tailored to specific circumstances.

The fundamental idea behind our approach is that we will not specify *when* an event precisely happens, but rather *why* it happens. Concretely, we represent a reason for some event E by a propositional formula ϕ and write "ϕ causes E" as: $E \leftarrow \phi$. We call such a construct a *Causal Probabilistic event (CP-event)* or, alternatively, to emphasize that we are referring to a syntactical construct, simply a *rule*. Note that, even though each CP-event contains only ground atoms, there is nothing to prevent us from introducing rules with variables as abbreviations

for sets of CP-events. The formula ϕ is also allowed to be **true**, in which case it may also be omitted; in this case, the event always happens.

The head $head(r)$ of a rule $r = E \leftarrow \phi$ is the set of all pairs (p, α) appearing in the event E; the body $body(r)$ of r is the formula ϕ. By $head_{At}(r)$ we denote the set of all atoms p for which there exists an α such that $(p, \alpha) \in head(r)$. The set of all atoms that appear negatively in ϕ (i.e., within the scope of an odd number of negations), is denoted as $body_{-}(r)$, while that of all positive atoms (i.e., the complement of the set of negative atoms) is $body_{+}(r)$.

By a *CP-theory* we mean a finite set of CP-events. We now need to address the question of precisely what information such a CP-theory C gives about a probabilistic process. The following properties are quite obvious:

- An event can only occur if there is a cause for this, i.e., if some event E occurs in a state s, there should be a rule $r \in C$, such that $E = head(r)$ and $body(r)$ holds in $\mathcal{I}(s)$;
- If, in a state s, there are still CP-events $E \leftarrow \phi$ in C such that the event E has not already occurred and ϕ holds in $\mathcal{I}(s)$, then one such event should happen.

A process that satisfies these two conditions is said to be *consistent* with C. This reading of a CP-theory does not yet suffice to characterize a unique probability distribution, i.e., different consistent processes might generate different probability distributions. To illustrate this, let us consider the following example.

Example 1. A person enters a dark room. There is button which is supposed to turn on the light. However, this button is broken and only works half of the time. The person repeatedly pushes the button until the light goes on. He tries this at most two times. To model this situation, we consider three time points $0, 1, 2$, and for each time point i, $light(i)$ stands for whether the light is on at this moment and $push(i)$ stands for whether the button is pushed at time i.

$$(light(1) : 0.5) \leftarrow push(1). \quad push(1) \leftarrow \neg light(0). \quad light(2) \leftarrow light(1).$$
$$(light(2) : 0.5) \leftarrow push(2). \quad push(2) \leftarrow \neg light(1).$$

In the initial state, neither $light(0)$ nor $light(1)$ hold. According to the semantics we have defined so far, both the event that might cause $push(1)$ and the event that might cause $push(2)$ could happen. These two options lead to different probability distributions. Indeed, in the process which causes $push(2)$ before $push(1)$, the probability of $light(2)$ will (incorrectly) be 0.5, instead of 0.75.

We can resolve this ambiguity, by also taking into account the temporal information that is implicit in the rules. Concretely, because causes always happen before their effects, we can assume that, if an atom appears in the body of a CP-event, then the part of the process that determines whether or not this atom holds, takes place before this event. As such, for a rule $E \leftarrow \phi$, the event E should only happen once we are sure that all subprocesses that might affect atoms of ϕ are finished. Now, if ϕ holds, then all positive atoms must have already been derived and we can therefore assume that the processes concerning these atoms

are finished. However, for the negative atoms, we cannot make this assumption based only on the fact that ϕ holds. Indeed, it is not because an atom has not yet been derived, that it never will. We will therefore have to ensure that E does not occur while it is still possible to derive any of these negative atoms. For Example 1, the fact that $push(1)$ might still cause $light(1)$ allows us to conclude that the event $push(1) \leftarrow \neg light(0)$ should happen first.

Mathematically, we can define this as follows. For some state s, let I be $\mathcal{I}(s)$ and let $D \subseteq C$ be the set of all CP-events that have not yet happened. To find out which atoms might still be caused in s, we need to consider for which $r \in D$ the formula $body(r)$ could still possibly be satisfied. We assume that every $body(r)$ is in some normal form (e.g., CNF), in which negation appears only directly in front of atoms. We consider a negative literal $\neg p$. If p has already been caused, then it is clearly impossible for this literal to be satisfied; otherwise, this might still be possible. We therefore denote by C^I the result of replacing all negative literals $\neg p$ with $p \in I$ by **false** and all other negative literals by **true**. Now, if we already have some set S such that all atoms in S are possible, then we know that if, for some $r \in D^I$, $body(r)$ holds in S, all atoms $p \in head_{At}(r)$ are also still possible. As such, we define the set $Poss_C(I)$ of possible atoms as the smallest set S such that S contains all h for which $\exists r \in D^I$ with $S \models body(r)$ and $p \in head(r)$.[1]

We now define a *C-process* to be a process that is consistent with C and that also satisfies the condition that whenever a CP-event r occurs in a node s, none of the negative body atoms of r is still possible, i.e., $body_-(r) \cap Poss_C(I) = \{\}$. This now does suffice to characterize a unique probability distribution.

Theorem 1. *Let C be a CP-theory. Every C-process generates the same probability distribution.*

If it exists, we denote this unique distribution by π_C. Note that there can be CP-theories C which have no C-processes. If this is the case, then it is impossible to schedule the events of this theory in such a way that we can exhaust all events that might cause an atom p, before having to determine whether or not p will hold. We call a CP-theory C *valid* iff it has a C-process. In general, it is not easy to decide whether a given theory C is valid. However, there exist some simple syntactical criteria, by which it can often be concluded that this is so. For instance, if the theory does not contain negation or is stratified[2], then C is valid.

At this point, the definition of the semantics of CP-logic using the $Poss_C(I)$ construct may still seem somewhat arbitrary. In the next section, we study the role of time in CP-logic and show that our semantics gives correct results when probabilistic processes are modeled in full temporal detail, while also allowing representations in which time is abstracted away. Even though we do not have space to go into this here, the reader could convince himself that any semantics with these properties will be almost identical to the one defined here.

[1] This is similar to the definition of the stable operator of a logic program P, which maps an interpretation I to the least Herbrand model of the reduct of P by I.

[2] A CP-theory C is stratified if there is a mapping λ from ground atoms to $\mathbb{N}$, s.t. for all $r \in C$, $h \in head_{At}(r)$, $p \in body_+(r)$, $n \in body_-(r)$, $\lambda(h) \geq \lambda(p)$ and $\lambda(h) > \lambda(n)$.

2.1 The Role of Time in CP-Logic

Suppose we want to model a probabilistic process that lasts n time points. It is natural to construct the vocabulary of our theory in such a way that, for each property f that is relevant to this process, we have n ground atoms, say $f(1), \ldots, f(n)$, that refer to the truth of f at the different time points. Indeed, this is, for instance, precisely what we already did in Example 1. Now, suppose that in this domain it is the case that some property ϕ causes an event $E = (f_1(i_1) : \alpha_1) \vee \cdots \vee (f_n(i_n) : \alpha_n)$. Because causes always precede their effects, it should then be the case that all atoms belonging to ϕ refer to time points that are earlier than all of the time points i_j. In other words, for each $f(i)$ appearing in E and each $f'(i')$ appearing in ϕ, it should be the case that $i > i'$. Moreover, if this event actually occurs, this should clearly happen at some time between the maximal i' for which $f'(i')$ appears in ϕ and the minimal i for which $f(i)$ appears in E. It can be shown that our semantics respects this order. To state this formally, we define the level $lvl(r)$ of a rule r as the $\min_{f(i) \in head_{At}(r)} i$. We will show that if, in some process, the events happen according to the order dictated by lvl, then this process has to be a C-process. We could, alternatively, choose to define $lvl(r)$ as $\max_{f(i) \in body(r)} i$ and the theorem would still hold.

Theorem 2. *Let C be a CP-theory in which every ground atom is of the form $f(i)$, such that for all $r \in C$, if $f(i) \in head_{At}(r)$ and $f'(i') \in body(r)$, then $j > i$. Let $\mathcal{T}$ by a probabilistic process that is consistent with C and for which, whenever a CP-event r happens in a state s, then for all other CP-events r' that could have happened in s, i.e., no ancestor of s executes r' and $\mathcal{I}(s) \models body(r')$, it holds that $lvl(r) \leq lvl(r')$. Then $\mathcal{T}$ is a C-process and, therefore, it defines precisely the distribution π_C.*

By choosing our vocabulary in such a way that each ground atom represents the truth of some property at one particular time, we can represent a probabilistic process in quite some detail. Often, however, we would prefer to make abstraction of certain temporal information. Concretely, instead of using propositions $f(i)$ to refer to the truth of some property at time i, we would sometimes like to simply use a single atom f to represent the fact that "at some (unspecified) point in time, f holds". Formally speaking, we can ask the following question. Suppose we have a CP-theory C where, as above, all ground atoms are of the form $f(i)$ for some property f and time point i. Let C^f be the result of replacing every atom $f(i)$ by a single atom f. Now, is it the case that C and C^f generate equivalent probability distributions? To be more precise, for an interpretation I for the vocabulary of C, let us denote by I^f the result of replacing every atom $f(i)$ by f. Is it now the case that for all interpretations I' for the vocabulary of C^f, the probability $\pi_{C^f}(I')$ is equal to $\sum_{I^f = I'} \pi_C(I)$? A general, formal answer to this question falls outside the scope of this paper. We will, however, illustrate through some examples what kind of properties are relevant to this question.

In Example 1, time plays a crucial role: it matters at which time the light comes on, because this affects how many times the button is pushed, which in turn affects if and when the light might come on. Here, we cannot abstract away

time. In fact, if we try to do so, we get a theory that is not valid. Indeed, we cannot know whether the light will go on, without knowing whether the button will be pushed, which in turn depends on whether the light is on. In the following example, it is possible to make abstraction of time.

Example 2. We consider two persons, a and b. At time 0, both a and b undergo a blood transfusion, which might cause them to be infected with the HIV virus—say the probability of this is 0.1. At time 1, a and b engage in sexual intercourse, during which this virus may be transferred with a probability of 0.6.

$$(hiv(a,1):0.1). \quad (hiv(a,2):0.6) \leftarrow hiv(b,1). \quad hiv(a,2) \leftarrow hiv(a,1).$$
$$(hiv(b,1):0.1). \quad (hiv(b,2):0.6) \leftarrow hiv(a,1). \quad hiv(b,2) \leftarrow hiv(b,1).$$

Example 3. By making abstraction of time, we get:

$$(hiv(a):0.1). \quad (hiv(a):0.6) \leftarrow hiv(b). \quad hiv(a) \leftarrow hiv(a).$$
$$(hiv(b):0.1). \quad (hiv(b):0.6) \leftarrow hiv(a). \quad hiv(b) \leftarrow hiv(b).$$

This theory expresses that there are two possible causes for why a might have HIV: his blood transfusion might have infected him with probability 0.1 and, if b is infected at any time, then this might also cause $hiv(a)$. Now, this is indeed equivalent to the more detailed version in Example 2. Crucial for this equivalence is the fact that the sexual contact between a and b happens at the end of our time line, meaning that, no matter at which time b gets infected, he will still get a chance to pass on this infection to a. Also relevant is the fact that having HIV is a persistent property, which guarantees that if b ever gets infected, he will still carry the virus at the time of the sexual contact.

2.2 Causality in CP-Logic

One of the most succesful causal approaches to probabilistic modeling is that of causal Bayesian networks [6]. The intuitive reading of such a network says that, for every node, there is a causal mechanism through which the values of the parents of this node determine the value of this node itself. As such, a causal Bayesian network describes a probabilistic process in which, whenever the values of all the parents of a node have been determined, a causal event occurs that propagates these values to the node itself. There are two ways in which these processes are more restricted than those of CP-logic.

Firstly, in a causal Bayesian network, the value of a node is always determined by a single event. In CP-logic, on the other hand, many events might be involved in determining the truth of the same proposition. These events then act according to what we call the principle of *independent causation*. This says that every event affects the state of the world in a probabilistically independent way. For instance, if b is infected with HIV, then there are two events that might cause $hiv(a)$, namely a's blood transfusion and the sexual contact with b. The effect of the blood transfusion is now probabilistically independent of that of the sexual contact, i.e., the probability of $hiv(a)$ is $0.1+0.6-0.06$ (i.e., *noisy-or*$(\{0.1, 0.6\})$).

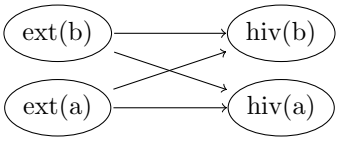
Secondly, due to the acyclic graph structure, events in a Bayesian network can only propagate values in a fixed direction. In CP-logic, on the other hand, it is possible that, e.g., under certain circumstances, $hiv(a)$ propagates to $hiv(b)$, while, under different circumstances, $hiv(b)$ might propagate to $hiv(a)$. The meaning of such a causal loop in CP-logic can be characterized by a second principle, namely that of *no deus ex machina effects*. This states that nothing happens without a cause and, moreover, that something cannot cause itself. Indeed, by itself, the loop between $hiv(a)$ and $hiv(b)$ does not cause anything, i.e., if neither a nor b has been infected by a blood transfusion, then neither has HIV.

The more general kind of events allowed by CP-logic offer some knowledge representation advantages. Firstly, they allow a better representation for effects that have a number of independent causes. For instance, in a game of Russian roulette that is being played with two guns, there are two independent causes for the death of the player. In CP-logic, we can write:

$$(death : 1/6) \leftarrow pull\_trigger(left\_gun).$$
$$(death : 1/6) \leftarrow pull\_trigger(right\_gun).$$

Here, the independence between these two causes is a *structural* property of the theory, instead of a numerical one. This improves the elaboration tolerance of the representation, since adding or removing a cause simply corresponds to adding or removing a single CP-event. Moreover, it also makes the reprentation more compact, as, for n independent causes, only n probabilities are needed instead of the 2^n in a Bayesian network table. A second advantage is that CP-logic allows cyclic causal relations to be directly represented in the same way as acyclic ones, whereas Bayesian networks require them to be encoded in a special way. For instance, to represent the cyclic relation between $hiv(a)$ and $hiv(b)$, one would introduce new atoms $ext(a)$ and $ext(b)$ to represent the possibility that a and b are infected by an external cause (i.e., one that is not part of the causal loop):

	e(a),e(b)	e(a),¬e(b)	¬e(a),e(b)	¬e(a),¬e(b)
hiv(a)	1	1	0.6	0
	e(a),e(b)	e(a),¬e(b)	¬e(a),e(b)	¬e(a),¬e(b)
hiv(b)	1	0.6	1	0



3 Logic Programs with Annotated Disjunctions

Logic Programs with Annotated Disjunctions (LPADs) are a probabilistic logic programming language, that was conceived in [12] as a straightforward extension of logic programs with probability. In this section, we relate LPADs to CP-logic. In this way, we will be able to clarify the position of CP-logic among related work, such as Poole's Independent Choice Logic and McCain and Turner's causal theories. Also, we will gain additional insight into a number of probabilistic logic

programming languages, by showing that theories in these languages can be seen as descriptions of causal information about probabilistic processes. Moreover, as we will discuss in Section 4, this also leads to an interesting way of looking at normal and disjunctive logic programs. Finally, probabilistic logic programming languages are usually motivated in a bottom-up way, i.e., along the following lines: "Logic programs are a good way of representing knowledge about relational domains, probability is a good way of representing knowledge about uncertainty; therefore, a combination of both should be useful for modeling uncertainty in a relational domain." Our results provide an additional top-down motivation, by showing that these languages are the natural way of representing causal knowledge about probabilistic processes.

We first recall the formal definition of LPADs from [12]. An LPAD is a set of rules $(h_1 : \alpha_1) \vee \cdots \vee (h_n : \alpha_n) \leftarrow l_1 \wedge \cdots \wedge l_n$, where the h_i are atoms and the l_j literals. As such, LPADs are a syntactic sublogic of CP-logic. However, their semantics is defined quite differently. Every rule of the above form represents a probability distribution over the set of logic programming rules $\{$"$h_i \leftarrow l_1 \wedge \cdots \wedge l_n$"$\mid 1 \leq i \leq n\}$. From these distributions, a probability distribution over logic programs is then derived. To formally define this distribution, we introduce the following concept of a *selection*. In this definition, we use the notation $head^*(r)$ to denote the set of pairs $head(r) \cup \{(\emptyset, 1 - \sum_{(h:\alpha) \in head(r)} \alpha)\}$, where $\emptyset$ represents the possibility that none of the h_i's are caused by the rule r.

Definition 1 (C-selection). *Let C be an LPAD. A* C-selection *is a function σ from C to $\bigcup_{r \in C} head^*(r)$, such that for all $r \in C$, $\sigma(r) \in head^*(r)$. By $\sigma^h(r)$ and $\sigma^\alpha(r)$ we denote, respectively, the first and second element of the pair $\sigma(r)$.*

The probability $\pi(\sigma)$ of a selection σ is now defined as $\prod_{r \in C} \sigma^\alpha(r)$. By C^σ we denote the logic program $\{$"$\sigma^h(r) \leftarrow body(r)$"$\mid r \in C$ and $\sigma^h(r) \neq \emptyset\}$. Such a C^σ is called an *instance* of C. These instances are interpreted according to the well-founded model semantics [10]. In general, the well-founded model $wfm(P)$ of a program P is a pair (I, J) of interpretations, where I contains all atoms that are certainly true and J contains atoms that might possibly be true. If $I = J$, the model is said to be *two-valued*. Intuitively, if $wfm(P)$ is two-valued, then the truth of all atoms can be decided, i.e., everything that is not false can be derived. In the semantics of LPADs, we want to ensure that all uncertainty is expressed by means of the annotated disjunctions. In other words, given a specific selection, there should no longer be any uncertainty. We impose the following criterion.

Definition 2 (Soundness). *An LPAD C is* sound *iff all instances of C have a two-valued well-founded model.*

For such LPADs, the following semantics can now be defined.

Definition 3 (Instance based semantics μ_C). *Let C be a sound LPAD. For an interpretation I, we denote by $W(I)$ the set of all C-selections σ for which $wfm(C^\sigma) = (I, I)$. The* instance based semantics μ_C *of C is the probability distribution on interpretations, that assigns to each I the probability $\sum_{\sigma \in W(I)} \pi(\sigma)$.*

Now, the key result of this section is that this instance based semantics coincides with the semantics defined in Section 2.

Theorem 3. *Let C be a valid CP-theory. Then C is also a sound LPAD and, moreover, for each interpretation J, $\mu_C(J) = \pi_C(J)$.*

We remark that it is not the case that every sound LPAD is also a valid CP-theory. In other words, there are some sound LPADs that cannot be seen as expressing sensible causal information about a probabilistic process.

In [12], LPADs are compared to a number of different probabilistic logic programming formalisms. For instance, it was shown that this logic is very closely related to Poole's Independent Choice Logic. Because of the above theorem, these comparisons carry over to CP-logic.

4 CP-Logic and Logic Programming

In this section, we examine some consequences of the results of the previous section from a logic programming point-of-view.

Disjunctive logic programs. In probabilistic modeling, it is often useful to consider the structure of a theory separately from its probabilistic parameters. Indeed, for instance, in machine learning, the problems of structure learning and parameter learning are two very different tasks. If we consider only the structure of a CP-theory, then, syntactically speaking, we end up with a *disjunctive logic program*[3], i.e., a set of rules $h_1 \vee \cdots \vee h_n \leftarrow \phi$. Let us now consider the class of all CP-theories C that result from adding probabilities α_i to each rule, in such a way that, for every rule, $\sum \alpha_i = 1$. Every probability distribution π_C defined by such a C induces a possible world semantics, namely the set of interpretations I for which $\pi_C(I) > 0$. This set of possible worlds does not depend on the precise values of the α_i, i.e., it is the same for all CP-theories C in this class. As such, it captures precisely the structural information in such a CP-theory.

From the point of view of disjunctive logic programming, this set of possible worlds can be seen as an alternative semantics for such a program. Under this semantics, the intuitive reading of a rule should be: "ϕ causes a non-deterministic event, that causes precisely one of $h_1,\ldots, h_n$." Clearly, this is a different informal reading than is used in the standard stable model semantics for disjunctive programs [8]. Indeed, under our reading, a rule corresponds to a causal event, whereas, under the stable model reading, it is supposed to describe an aspect of the reasoning behaviour of a rational agent. Consider, for instance, the disjunctive program $\{p \vee q.\ p.\}$. To us, this program describes a set of two non-deterministic events: One event causes either p or q and another event always causes p. Formally, this leads to two possible worlds, namely $\{p\}$ and $\{p, q\}$. Under the stable model semantics, however, this program states that an agent

[3] In most of the literature, the bodies of the rules of a disjunctive logic program must be conjunctions of literals. For our purposes, however, this restriction is not relevant.

believes either p or q and the agents believes q. In this case, he has no reason to believe q and the only stable model is $\{p\}$.

CP-logic treats disjunction in a fundamentally different way than the stable semantics. Interestingly, the *possible model semantics* [9] for disjunctive programs is very similar to our treatment. Indeed, it consists of the stable models of instances of a program. Because, as shown in Section 3, the semantics of CP-logic considers the well-founded models of instances, these two semantics are very closely related. Indeed, for a large class of programs, they coincide completely.

Normal logic programs. A normal logic program P is a set of rules $h \leftarrow \phi$, with h an atom and ϕ a formula. Syntactically, such a program is also a CP-theory. Its semantics π_P assigns a probability of 1 to a single interpretation and 0 to all other interpretations. Moreover, the results from Section 3 tell us that the interpretation with probability 1 will be precisely the well-founded model of P. As such, a logic program under the well-founded semantics can be viewed as a description of causal information about a deterministic process. Concretely, we can read a rule $h \leftarrow \phi$ as: "ϕ causes a deterministic event, that causes h."

This observation exposes an interesting connection between logic programming under the well-founded semantics and causality. Such a connection helps to explain, for instance, the usefulness of this semantics in dealing with recursive ramifications when reasoning about actions [2]. Moreover, there is also an interesting link here to the language of *ID-logic* [1]. This is an extension of classical logic, that uses logic programs under the well-founded semantics to represent inductive definitions. Inductive definitions are a well-known mathematical construct, which define a relation by describing a derivation process by which it can be constructed. It turns out that this derivation process is closely tied to the probabilistic processes described by a CP-theory. Indeed, both can be formally characterized by means of the well-founded semantics. This observation suggests that an inductive definition is actually nothing more than a representation of causal information about a process that takes place in the domain of mathematical objects.

McCain and Turner's causal theories. We compare the treatment of causality in CP-logic to McCain and Turner's *causal theories* [5]. A causal theory is a set of rules $\phi \Leftarrow \psi$, where ϕ and ψ are propositional formulas. The semantics of such a theory T is defined as follows. An interpretation I is a model of T iff I is the *unique* classical model of the theory $T^I = \{\phi \mid \text{there exists a rule } \phi \Leftarrow \psi \text{ in } T \text{ such that } I \models \psi\}$. This semantics is based on the principle of *universal causation*, which states that: "every fact that obtains is caused" [5]. We now compare this language to deterministic CP-logic, i.e., CP-logic in which every CP-event causes one atom with probability 1. The most obvious difference concerns the fundamental knowledge representation methodology of these logics. In CP-logic, a proposition represents a property that is false unless there is a cause for it to be true. For McCain & Turner, however, truth and falsity are symmetric, i.e., a property is not true unless there is a cause for it to be true and a property is also not false unless there is a cause for it to be false. It is up

to the user to make sure there is always a cause for either falsity or truth. For instance, the CP-theory $\{p \leftarrow \neg q\}$ has $\{p\}$ as its model, while the causal theory $\{p \Leftarrow \neg q\}$ has no models, because neither q nor $\neg q$ is caused. The CP-logic view that falsity is the natural state of atoms can be simulated in causal theories, by adding rules $\neg p \Leftarrow \neg p$, which say that $\neg p$ is in itself reason enough for $\neg p$. Let C' be the result of adding such rules to some original CP-theory C. As shown in [4], the models of C' are all interpretations I that consist of all heads of rules $r \in C$, for which $I \models body(r)$. In logic programming terms, these are the *supported models* of C, i.e., fixpoints of the immediate consequence operator T_C.

The difference between such a CP-theory C and its corresponding causal theory C' is, therefore, precisely the difference between the well-founded model semantics and supported model semantics. It is well-known that this lies in the treatment of loops. In our context, it can be traced back to the fundamental principles of these logics. McCain and Turner's principle of "universal causation" states that everything that holds must have a cause. This is a weaker principle than our principle of no deus ex machina effects, which states that every true proposition must have a cause *and* that something cannot cause itself. Indeed, the CP-theory $\{p \leftarrow p\}$ has $\{\}$ as its model, whereas the causal theory $\{p \Leftarrow p\}$ has $\{p\}$ as its model. In other words, in McCain and Turner's theories, it can be stated that a certain atom might be true "on its own", i.e., without any additional causal explanation being required. This can be useful to incorporate exogenous actions into a theory, i.e., actions that can simply happen, without any part of the model describing why they happen. These currently cannot be represented in CP-logic. On the other hand, McCain and Turner's approach to self-causation does not allow them to directly represent cyclic causal relations of the kind appearing in Example 3.

5 Conclusions

We have identified causal information about probabilistic processes as a useful kind of knowledge and constructed the language of CP-logic to represent it. We studied when such information suffices to characterize a probability distribution. We have analyzed the role of time in this logic and showed that it can express probabilistic processes in complete temporal detail, but is also able to represent causal relations in a more static way. We have shown how the concept of causality in this logic compares to causal Bayesian networks and McCain & Turner's causal theories. We related CP-logic to logic programming, by showing that it basically coincides with the language of LPADs. This result provides an additional motivation for an existing class of probabilistic logic programming formalisms, since it shows that these are the natural way of representing causal information about probabilistic processes. We showed that our semantics induces a possible world semantics for disjunctive programs and discussed the differences with the standard stable model semantics. Our result also shows that normal logic programs under the well-founded semantics can be seen as a language of deterministic causality, which exposes an interesting relation between causal processes and inductive definitions as formalized in ID-logic.

References

1. M. Denecker and E. Ternovska. A logic of non-monotone inductive definitions and its modularity properties. In *Proc. 7th International Conference on Logic Programming and Non-monotonitc Reasoning (LPNMR)*, pages 47–60 volume 2923 of *LNCS*, 2004.
2. M. Denecker, D. Theseider-Dupré, and K. Van Belleghem. An inductive definition approach to ramifications. *Linköping Electronic Articles in Computer and Information Science*, 3(7):1–43, 1998.
3. J. Halpern and M. Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM*, 40:917–960, 1993.
4. N. McCain. *Causality in Commonsense Reasoning about Actions*. PhD thesis, University of Texas at Austin, 1997.
5. N. McCain and H. Turner. Causal theories of action and change. In *Proc. 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 460–465, 1996.
6. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000.
7. D. Poole. The Independent Choice Logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.
8. T. C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 3/4:401–424, 1991.
9. C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13(1):145–172, 1994.
10. A. Van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.
11. J. Vennekens, M. Denecker, and M. Bruynooghe. On the equivalence of Logic Programs with Annotated Disjunctions and CP-logic. Technical report, K.U. Leuven, 2006.
12. J. Vennekens, S. Verbaeten, and M. Bruynooghe. Logic programs with annotated disjunctions. In *Logic Programming, 20th International Conference, ICLP 2004, Proceedings*, pages 431–445, volume 3132 of *LNCS*. Springer, 2004.

A Tool to Facilitate Agent Deliberation

Daniel Bryant, Paul Krause, and Sotiris Moschoyiannis

Department of Computing, University of Surrey, Guildford, GU2 7XH. UK
{d.bryant, p.krause, s.moschoyiannis}@surrey.ac.uk

Abstract. In this paper we present a prototype of a tool that demonstrates how existing limitations in ensuring an agent's compliance to an argumentation-based dialogue protocol can be overcome. We also present the implementation of compliance enforcement components for a deliberation dialogue protocol, and an application that enables two human participants to engage in an efficiently moderated dialogue, where all inappropriate utterances attempted by an agent are blocked and prevented from inclusion within the dialogue.

1 Introduction

Autonomous software agents are often cited as a key enabling technology for the next generation of distributed service provision, such as large-scale electronic commerce systems [1] and Service-Oriented Computing [2]. Key characteristics of such services are agent heterogeneity, conflicting individual goals, limited trust and a high probability of non-conformance to specifications [3]. If this vision of large-scale open multi-agent systems is to be realised then the fundamental problem of interoperability (i.e. communication between agents) must be addressed. As a result, there has been much work on agent communication languages (ACLs), and an increasing amount of this work has concentrated on argumentation-based dialogue [4]. However, for an ACL to truly be an enabling technology, it must rely on a standard or protocol to ensure that different implementations preserve the ACL's meaning [5], and in order to gain acceptance, particularly for sensitive applications such as electronic commerce, it must be possible to determine whether or not any system that claims to conform to an ACL protocol actually does so [5], [6].

In this paper we present a prototype of a tool that demonstrates how existing limitations in ensuring an agent's compliance to an argumentation-based dialogue protocol can be overcome. Dialogue protocols are enforced by means of a series of distributed "Dialogue Manager" enforcement components, implemented as a lightweight Java-based agent proxy. Our ultimate goal is to implement a generic ACL enforcement tool, but in order to keep this paper focused we will concentrate on the implementation of enforcement for a deliberation dialogue protocol, as presented in [7]. The remainder of this paper is structured as follows. First, we present an overview of the deliberation dialogue and dialogue games. Next, we summarise the implementation of our tool. We conclude the paper with an overview of the planned future work.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 465–468, 2006.

2 Deliberation and Dialogue Games

Hitchcock *et al* [7] state that a deliberation dialogue arises with a need for action in some circumstance. In general human discourse, this need may be initially expressed in governing questions which are quite open-ended, as in *where shall we go for dinner this evening?* In [7] a formal and implementable model for deliberation dialogues between autonomous agents is presented, utilising an ACL with argumentation-based social semantics [5] within a formal dialogue game. Formal dialogue games are games in which two or more participants "move" by uttering locutions, according to certain pre-defined rules (see [7] for a more detailed presentation). For each locution type in the deliberation dialogue a series of pre- and post-conditions are specified based on external observable information such as the previous utterances of each agent and current dialogical commitments. These conditions are used to axiomatise behaviour in the sense that they specify when the utterance of each locution would be considered a legal move in the dialogue.

3 Implementing the Dialogue Manager

There are many examples of existing work for verifying agent specifications and protocol compliance (which will not be cited due to space restrictions in this paper). However, many techniques rely on access to an agent's internal state (which is considered unacceptable to many researchers), are only capable of verifying the design level of an agent, or have not been practically implemented. There has also been several recent approaches to enforcing agent interaction that seek to overcome these limitations, most notably Artikis *el al's Society Visualiser* [3] and Alberti *et al's SOC-SI* [8]. Our work differs from these approaches in two fundamental ways. Firstly, we focus exclusively on enforcing argumentation-based dialogue protocols (which naturally contain a form of social semantics [5]) and as such we provide an efficient technique for translating locution pre- and post-conditions into executable code (based on formal support provided in [9] to represent locution conditions in a common format). Secondly, our enforcement mechanism has been distributed across all the participating agents, reducing the potential performance bottleneck of a monolithic mechanism.

We have implemented our tool in the form of a lightweight Java application using Sun's distributed JavaSpaces technology to act as the communication medium. At the core of the JavaSpaces system the "Linda-like" [10] tuples-based associative black board coordination model is utilised, decoupling the communicating agents both spatially and temporally. We have created a client-side "Dialogue Manager" proxy that acts as a mediator between every agent involved in a dialogue and the communication medium (based on the Controller in the LGI model [10]). We have also implemented the rules for the deliberation dialogue protocol and the pre- and post- conditions for each locution's semantics (as specified in [7]) using a flexible framework which is cleanly separated from the Dialogue Manager (analogous to the Law in LGI). This enables different dialogue protocols to be swapped and enforced at run time, and in future versions

of the tool will allow a variety of dialogue-types to be mediated. A Dialogue Manager operates essentially as follows: It intercepts all utterances that the associated agent attempts to make and, based on its own local copy of the dialogue rules and local control state (previous utterances and dialogical commitments), determines whether the locution would be appropriate at this time, blocking any inappropriate utterances from inclusion within the dialogue.

An additional client-side GUI tool has been created (Figure 1) that utilises the Dialogue Manager component so that a dialogue between two (geographically distributed) human participants can be undertaken under the protocol, with each participant taking turns to utter a locution. If a participant attempts to make an illegal move then they are informed accordingly and given the opportunity to choose an alternative move. All previous utterances and the current commitment store are displayed in the GUI and are publicly available to all agents participating in the dialogue (Figure 1). This facilitates the expedient resolution of the dialogue by allowing participants to determine which of their commitments overlap or conflict with those of other participants, and thereby identify points of agreement or determine which commitments are susceptible to an attack.

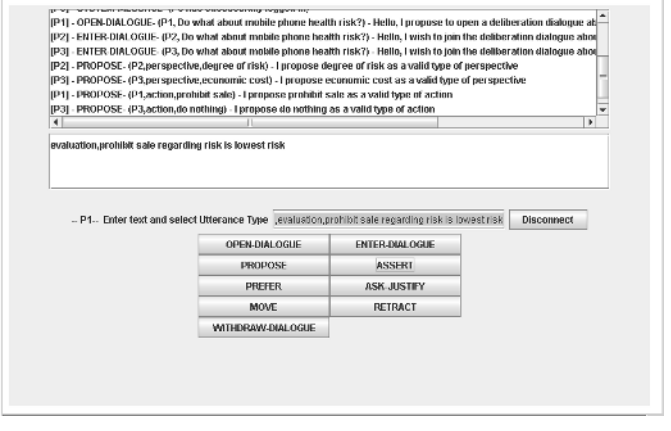


Fig. 1. Screen shot of the GUI tool allowing human participants to engage in a deliberation dialogue

4 Conclusion and Future Work

We have presented a prototype tool that demonstrates how existing limitations in ensuring an agent's compliance to an argumentation-based deliberation dialogue protocol can be overcome. Our current application utilises a flexible protocol enforcement framework, which blocks any inappropriate or illegal utterances, and does not require central control. We have also presented a GUI application that enables two human participants to engage in a moderated dialogue. Future work will focus on enhancing our application to support a dialogue framework

in which more than one kind of dialogue can be carried out (as presented in [4]). As part of this work we are currently investigating the use of a vector language (used to model component interaction in [11]) which we believe will offer a generic representation of argumentation-based dialogues in which it is possible to capture the dependencies between moves of all the participants at each step of a dialogue.

This work was partially supported by the EU IST/STReP ASPIC project, Grant 002307, and an EPSRC PhD Studentship.

References

1. C. Guilfoyle, J. Jeffcoate, and H. Stark. *Agents on the Web: Catalyst for E-Commerce*. Ovun Ltd. London, 1997.
2. M. P. Papazoglou. Service-Oriented Computing: Concepts, characteristics and directions. In *WISE '03: Proceedings of the Fourth International Conference on Web Information Systems Engineering*, page 3, Washington, DC, USA, 2003. IEEE Computer Society.
3. A. Artikis, J. Pitt, and M. Sergot. Animated specifications of computational societies. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1053–1061, New York, NY, USA, 2002. ACM Press.
4. L. Amgoud, M. Caminada, P. McBurney, H. Prakken, and G. Vreeswijk. Final Review and Report on Argumentation System. Technical Report ASPIC Deliverable 2.6, 2006.
5. M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.
6. M. Wooldridge. Verifiable Semantics for Agent Communication Languages. In Y. Demazeau, editor, *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 349–356, Paris, France, 1998. IEEE Press.
7. D. Hitchcock, P. McBurney, and S. Parsons. A Framework for Deliberation Dialogues. In *Proc. of 4th Biennial Conf. Ontario Society for the Study of Argumentation (OOSA)*, 2001.
8. M. Alberti, D. Daolio, P. Torroni, M. Gavanelli, E. Lamma, and P. Mello. Specification and verification of agent interaction protocols in a logic-based system. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 72–78, New York, NY, USA, 2004. ACM Press.
9. S. Wells and C. Reed. Formal dialectic specification. In *Proceedings of First International Workshop on Argumentation in Multi-Agent Systems (ArgMAS 2004), LNCS*, pages 31–43. Springer Berlin, 2004.
10. N. H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering and Methodology*, 9(3):273–305, 2000.
11. S. K. Moschoyiannis. *Specification and Analysis of Component-Based Software in a Concurrent Setting*. PhD thesis, University of Surrey, 2005.

An Implementation of a Lightweight Argumentation Engine for Agent Applications

Daniel Bryant and Paul Krause

Department of Computing, University of Surrey, Guildford, GU2 7XH. UK
{d.bryant, p.krause}@surrey.ac.uk

Abstract. Argumentation is becoming increasingly important in the design and implementation of autonomous software agents. In this paper we discuss our current work on a prototype lightweight Java-based argumentation engine that can be used to implement a non-monotonic reasoning component in Internet or agent-based applications. As far as possible we are aiming towards implementing a general purpose argumentation engine that can be configured to conform to one of a range of semantics.

1 Introduction

Argumentation is becoming increasingly important in the design and implementation of autonomous software agents. In particular, it has been proposed that argumentation will facilitate agent-based systems that engage in cooperative problem solving, such as automated negotiation [1] and reasoning over proposals for action [2]. In these situations classical logic-based approaches are often unsuitable. Pertinent information may be insufficient or in contrast there may be too much relevant, but partially incoherent information, and in the case of multi-agent systems, conflicts of interest are inevitable [3].

In this paper we discuss our current work on a prototype lightweight Java-based argumentation engine that can be used to implement a non-monotonic reasoning component in Internet or agent-based applications. The core engine has been built using tuProlog [4] [5], an existing open-source Prolog engine, as its foundation. Although our ultimate goal is to create a general purpose argumentation engine that can be configured to conform to one of a range of semantics, the current version of the engine implements the argumentation-based framework presented in [3] (allowing our engine to determine the acceptability of arguments and construct proofs using an argument game approach to constructing proofs of acceptance [6]), and also standard Prolog inference (allowing us to prototype a variety of metainterpretters that support other forms of argumentation). This paper is structured as follows: In Section 2 we provide motivation for our work and introduce tuProlog. Section 3 introduces the ASPIC argumentation framework, and in Section 4 we discuss how we have implemented this in our engine. We conclude the paper with an overview of planned future work.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 469–472, 2006.

2 Laying the Foundations - tuProlog

There has been much recent work on argumentation-based engines, for example, Vreeswijk's *IACAS* [7], and García and Simari's *DeLP* [8] (and later an extension to this work, *P-DeLP*, by Chesñevar and colleagues [9]). However, to our knowledge none of these engines implement support for more than one form of argumentation semantics. It is our belief that agents engaged in reasoning should have access to a general purpose argumentation engine that can be configured to conform to one of a range of semantics.

Our prototype argumentation engine has been built using tuProlog [5] as its foundation. tuProlog is a Java-based Prolog engine which has been designed from the ground up as a thin and lightweight engine that is easily deployable, dynamically configurable and easily integrated into Internet or agent applications [4]. Utilising the Prolog inference provided by the tuProlog engine we can implement a series of metainterpretters for a variety of forms of argumentation. However, this way of implementing an argumentation engine has both a serious performance overhead and a less than ideal interface. In order to avoid these problems and produce an argumentation engine that fully conforms to the spirit of a lightweight Internet enabled tool, we are re-engineering tuProlog by implementing a series of core argumentation algorithms in Java. The first algorithm we have implemented in our engine is presented in [3].

3 The Acceptability of Arguments

In [3] a framework for argument games is presented that is concerned with establishing the acceptability of arguments. Argument games between two players, a proponent (PRO) and opponent (OPP), can be interpreted as constructing proofs of acceptance utilising a dialectical structure [6]. The proponent and opponent share the same (possibly inconsistent) knowledge base and the proponent starts with a a main claim to be "proved". The proponent attempts to build an admissible set to support the claim and endeavors to defend any argument against any attack coming from the opponent. The proponent wins the game (proving acceptability of the claim) if all the attacking arguments have been defeated, and the opponent wins if they can find an attacking argument that cannot be defeated. In [3] a prototype web-based implementation (coded in RUBY) of the framework algorithms, entitled "Argumentation System" (AS), is also presented.

4 The Implementation of Our Engine

As with AS, AtuP accepts formulas in an extended first-order language and returns answers on the basis of the semantics of credulously preferred sets (as defined in [3]). Facts and beliefs can be expressed in AtuP using standard Prolog syntax with an additional numerical qualifier e.g. `london(raining) 0.8.` and rules can be expressed in a similar way, for example, `flies(X) :- bird(X) 0.8.` In both

of these cases the numerical value is a number in (0,1] that acts as the degree of belief (DOB), or the credibility, of a proposition [3]. As stated in [3], the DOB is currently provided to allow experimentation with different methods of argument evaluation and is not intended to express probabilities or represent values from other numerical theories to reason with uncertain or incomplete information. However, in future work we plan to enhance arguments with possibilities as discussed in, for example, Krause [10], Amgoud [11] or Chesñevar and colleagues [9]. Queries for the support for a claim are expressed using the standard syntax, for example, `?-flies(tweety).`

When AtuP has finished determining the support for a claim the engine generates a trace of the argument game dialogue, an example of which can be seen in Figure 1. In addition to providing an application programmers interface (API) to allow agent developers to utilise our engine, we have also modified the existing tuProlog graphical user interface to facilitate off-line experimentation with the engine (see Figure 1).

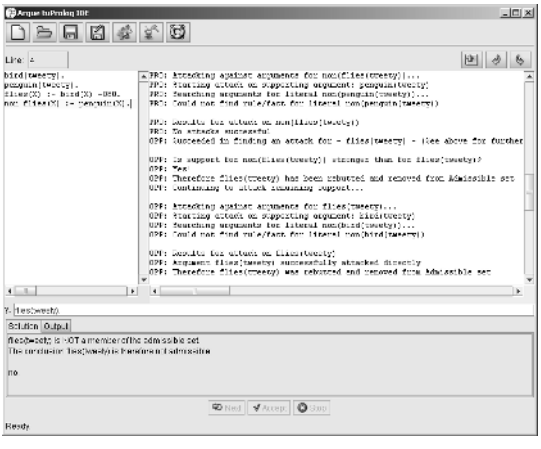


Fig. 1. Screenshot of "Argue tuProlog" GUI. (Left window) allows manipulation of the knowledge base, the (bottom window) allows query entry and displays the results, and the (right window) shows an argument game trace after a query has been executed.

5 Conclusion and Future Work

In this paper we have presented our current work on a lightweight Java-based argumentation engine. We have also discussed the integration of an argumentation-based framework for determining the acceptability of arguments, as presented in [3], into the engine. The end result is a flexible inference engine that is suitable for deployment into Internet and agent applications, and can be utilised to facilitate automated reasoning and decision-making. As far as possible we are aiming towards implementing a general purpose argumentation engine that can be configured to conform to one of a range of semantics. Our basic position is that we

have no prior disposition towards any one model of argumentation. Instead, our plan is to explore a range of models to provide an independent evaluation of their expressive power, performance and scalability.

Acknowledgements

This work was partially supported by the EU IST/STReP ASPIC project, Grant 002307, and an EPSRC PhD Studentship. We gratefully thank members of the ASPIC consortium for useful discussions and in particular Gerard Vreeswijk for his encouragement of our work. We also gratefully thank Mariam Tariq for use of her early implementation work and the tuProlog team at the University of Bologna for their enthusiastic support. Finally, we would like to thank the anonymous reviewers for the insightful and helpful comments.

References

1. I. Rahwan, S. D. Ramchurn, N. R. Jennings, P. McBurney, S. Parsons, and L. Sonenberg. Agent communication languages: Rethinking the principles. *The Knowledge Engineering Review*, 18(4):343–375, 2003.
2. K. Atkinson, T. K. Bench-Capon, and P. McBurney. A Dialogue Game Protocol for Multi-Agent Argument Over Proposals for Action. In I. Rahwam, P. Moraitis, and C. Reed, editors, *Argumentation in Multi-Agent Systems*, volume 3366 of *LNAI*, pages 149–161. Springer, 2004.
3. L. Amgoud, M. Caminada, S. Doutre, H. Prakken, and G. Vreeswijk. Draft formal semantics for ASPIC system. Technical Report ASPIC Deliverable 2.5, 2005.
4. E. Denti, A. Omicini, and A. Ricci. Multi-paradigm java-prolog integration in tuProlog. *Sci. Comput. Program.*, 57(2):217–250, 2005.
5. E. Denti, A. Omicini, and A. Ricci. tuProlog: A light-weight prolog for internet applications and infrastructures. In I. V. Ramakrishnan, editor, *PADL*, volume 1990 of *Lecture Notes in Computer Science*, pages 184–198. Springer, 2001.
6. H. Jakobovits and D. Vermeir. Dialectic semantics for argumentation frameworks. In *International Conference on Artificial Intelligence and Law*, pages 53–62, 1999.
7. G. A. W. Vreeswijk. IACAS: an implementation of Chisholm's principles of knowledge. In *The proceedings of the 2nd Dutch/German Workshop on Nonmonotonic Reasoning, Utrecht.*, pages 225–234, 1995.
8. A. J. Garcia and G. R. Simari. Defeasible logic programming: an argumentative approach. *Theory Pract. Log. Program.*, 4(2):95–138, 2004.
9. C. I. Chesnevar, G. R. Simari, T. Alsinet, and L. Godo. A logic programming framework for possibilistic argumentation with vague knowledge. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 76–84, Arlington, Virginia, United States, 2004. AUAI Press.
10. P. Krause, S. Ambler, M. Elvang-Goransson, and J. Fox. A logic of argumentation for reasoning under uncertainty. *Computational Intelligence*, 11:113–131, 1995.
11. L. Amgoud and H. Prade. Using arguments for making decisions: a possibilistic logic approach. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 10–17, Arlington, Virginia, United States, 2004. AUAI Press.

A Tool for Answering Queries on Action Descriptions*

Thomas Eiter, Michael Fink, and Ján Senko

Institute of Information Systems, Vienna University of Technology, Vienna, Austria
(eiter, michael, jan)@kr.tuwien.ac.at

1 Introduction

Action languages [1] are a formal tool for reasoning about actions, where an agent's knowledge about a domain in question is represented by a declarative action description that consists of logical formulas. For instance, consider a light bulb with a switch. When the light is off, then toggling the switch turns the light on; this can be expressed in the action description language $\mathcal{C}$ [2] by the dynamic causal law:

$$\textbf{caused } Light \textbf{ after } Toggle \wedge \neg Light. \tag{1}$$

On the other hand, at every state, if the light bulb is broken then the light is off. This can be expressed by the static causal law:

$$\textbf{caused } \neg Light \textbf{ if } Broken. \tag{2}$$

Other pieces of knowledge, like laws of inertia, may be also included:

$$\textbf{inertial } Light, \neg Light, Broken, \neg Broken.\ ^{1} \tag{3}$$

The meaning of such an action description, D, can be represented by a transition diagram, $T(D)$—a directed graph whose nodes correspond to the states of the world, $S(D)$, and the edges to the transitions, $R(D)$, describing action occurrences. For instance, the transition diagram of the above action description is shown in Figure 1. [2]

We consider the problem of revising action descriptions in the presence of conflicts between the action description and a set of conditions (axioms or observations) represented in an action query language [1]. For example, when the light bulb is broken, toggling the switch may lead to a state where the light is off; this is expressed by:

$$\textbf{possibly } \neg Light \textbf{ after } Toggle \textbf{ if } Broken. \tag{4}$$

Since at the state where the light bulb is broken and the light is off, toggling the light switch is not possible, There is a conflict between the action description and this condition. Moreover, under further conditions, like the following query:

$$\textbf{necessarily } \neg Light \textbf{ after } Toggle \textbf{ if } Light, \tag{5}$$

the conflict cannot be resolved just by dropping laws. In general, it is difficult to formalize the process of arriving at appealing "repairs", which often depend on additional knowledge or intuitions of the designer. We aim at supporting a designer in conflict and modification analysis and developed a tool that allows a user to issue a number of

* Work supported by the Austrian Science Fund (FWF) under grant P16536-N04.

[1] Here **inertial** $L_1, ..., L_k$ stands for the causal laws **caused** L_i **if** L_i **after** L_i for $i \in \{1, ..., k\}$.

[2] The action description is "buggy" (the effects of toggling the switch are improperly described).

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 473–476, 2006.

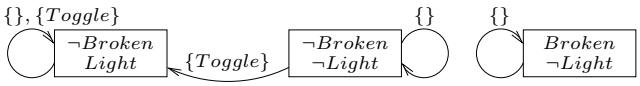


Fig. 1. Transition diagram of the action description { (1), (2), (3)}

relevant tests on an action description in $\mathcal{C}$ and its associated transition diagram in the presence of conditions. The tool computes answers to these tests by answer-set programming, revealing possible causes of conflicts or effects of certain modifications.

2 Tests

The tests a designer can issue by our tool, resemble the questions about a set of queries (conditions), Q, and D, respectively $T(D)$, as identified in [3] (see also below). Although the formal statement of these questions served as the basis for implementing a corresponding test library for the system, we confine here to an informal treatment and refer to [3] for details. As there, focusing on dynamic aspects, $S(D)$ is assumed to be correct and hence static laws need not be modified.

Tests on queries and causal laws. To better understand the reasons for conflicts, the designer may want to check whether the given queries Q make sense with respect to each other, find out which causal laws violate certain queries, or whether repairing an action description can be done without modifying some causal laws, resp. whether certain causal laws need to be modified:

- **D1:** Is Q contradictory relative to D?
- **D2:** If D does not satisfy a particular **necessarily**-query q in Q, which dynamic causal laws in D violate q?
- **D3:** Can we resolve a conflict between D and Q, without modifying a set D_0 of causal laws in D?
- **D4:** Do we have to modify a set D_0 of dynamic causal laws in D to resolve a conflict between D and Q?

Example 1. In our running example, if Q consisted of the query **possibly** $Light \wedge Broken$ **after** $Toggle$ **if** $True$ then, Q would be contradictory relative to D (no state in $S(D)$ satisfies $Light$ and $Broken$), while $Q = \{(4)(5)\}$, is not contradictory (**D1**).

Tests on states and transitions. Alternatively, the designer may want to extract information from $T(D)$. For instance, information about states, respectively transitions, violating a query q in Q, or information about candidates for transitions, that do not constitute transitions due to *under-specification* (i.e., not every fluent is causally explained):

- **T1:** Which states of $T(D)$ that satisfy a given formula ϕ, violate q?
- **T2:** Given formulas ψ and ϕ, which transitions $\langle s, A, s' \rangle$ of $T(D)$ such that s satisfies ϕ and s' satisfies ψ, violate q?
- **T3:** Given a literal L, for every state s of $T(D)$ such that s satisfies ϕ, is there some under-specified transition candidate $tc = \langle s, A, s' \rangle$ for D such that s' satisfies $\psi \wedge L$ and L is under-specified relative to tc?

T4: Which transition candidates $tc = \langle s, A, s' \rangle$ for D such that s satisfies ϕ and s' satisfies ψ are under-specified?

Example 2. In Ex. 1, if we just consider states where the light is on (i.e., $\phi = Light$). Then the only state at which a query of Q is violated is $\{Light, \neg Broken\}$ (**T1**).

3 Implementation

To compute test answers, we use disjunctive logic programming (DLP) – disjunction is actually needed due to Σ_2^p-completeness of most of the tests [3]. We translate action description, queries, and test into a DL program, and call the DLP solver DLV [3] to compute the models of this program, which encode the answer of the test performed.

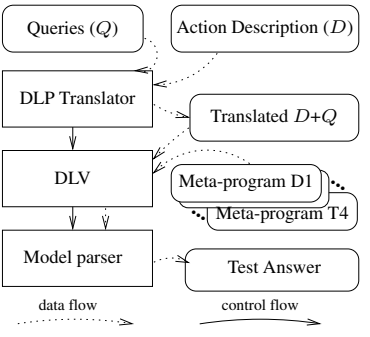


Fig. 2. Tool Architecture

The translation of an action description and queries into a logic program is uniform for all tests, and each test has been encoded in a 'meta-program' which operates on these translations, i.e., input programs. Figure 2 depicts the architecture of our tool. It is a command-line oriented Perl script consisting of two main parts: the *DLP Translator* and a *Model parser*. After pre-processing the input and translation to a DLP, calls to DLV are executed and their output is post-processed into human-readable form by the Model parser. The tool operation is controlled by the first command-line parameter that specifies the type of test to perform (e.g. `-T1`). The remaining parameters are supposed to be input files, i.e., text files, where each line either starts with one of the following keywords:

`Action/Fluent:` declares a new action or fluent literal;
`Inertial/Caused:` describes an inertia, static or dynamic law;
`Possibly/Necessarily:` describes a respective type of query;
`Initial/Successor:` describes a condition on a state (ϕ and ψ in tests);

or, otherwise, is directly copied to the output (e.g., to add background knowledge).

The *DLP Translator* compiles the input D and Q into a DLP representation on a file, which is combined with the fixed meta-program for the issued test to a single program on which DLV is invoked. The output of DLV (i.e., the answer sets) is then processed by the Model parser.

Model parsing is specific for each test: some tests yield yes/no answers by means of inconsistency (no model). E.g., no model for test **D1** means that the queries are not contradictory, whereas for some tests models encode test results such as violating states (**T1**,**T3**), violating transitions (**T2**), dynamic causal laws that violate a query (**D2**), etc.

[3] `http://www.dlvsystem.com`

While this information is encoded in DLP atoms, the Model parser prints the essential information in a human-readable format.

For user convenience, our implementation allows for generic statements as short-cuts in an action description using fluents and rules with parameters (i.e., variables). E.g., to extend our example to multiple light bulbs, one may re-write (1) as:

```
caused light(X) after toggle, -light(X) requires bulb(X).
```

where the keyword `requires` marks type information for variable X, provided by the background, e.g.: `bulb(green). bulb(yellow). bulb(red).`

4 Usage of the System

We now demonstrate a possible session of a designer using our tool. We assume that the action description consisting of (1), (2), and (3) is provided in a file `example.in`, and that the queries (4) and (5) are in files `example.pos` and `example.nec`, respectively.

First, the designer wants to check whether a query is violated at all (**T1** and **T2**):

```
./ad-query -T1 example.in example.pos
  VIOLATING STATE: broken. −light.
./ad-query -T2 example.in example.nec
  VIOLATING TRANSITION: (−broken. light.), (−broken. light.)
```

Since both queries are violated, she wonders whether they are contradictory (**D1**):

```
./ad-query -D1 example.in example.nec example.pos
  The set of queries is not contradictory.
```

Thus, the action description can be repaired such that both queries are satisfied. However, is it inevitable to modify the existing causal laws (issue **D4** with $D_0 = D$)?

Because the answer is 'yes' ((4) is violated at state $(broken, -light)$), she might ask whether at least the inertia laws can be kept by running test **D3** with $D_0 = D - \{(1)\}$. From the answer, 'yes', she eventually knows that the dynamic causal law (1) has to be modified (indeed, this law does not properly reflect the effects when the bulb is broken).

5 Conclusion

Our tool ad-query, which is available at `www.kr.tuwien.ac.at/research/ad-query/`, is to our knowledge the first tool to answer queries on action descriptions in $\mathcal{C}$ in the context of revision and design as described. The current version implements a common fragment of $\mathcal{C}$ and queries (heads of laws are literals and other formulas are conjunctions of literals). Ongoing work will extend the language and consider additional tests, as well as a methodology for using the tool.

References

1. Gelfond, M., Lifschitz, V.: Action languages. Electronic Transactions on Artificial Intelligence **3** (1998) 195–210
2. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. AAAI '98, AAAI Press (1998) 623–630
3. Eiter, T., Erdem, E., Fink, M., Senko, J.: Resolving conflicts in action descriptions. In: Proc. ECAI 2006. See `http://www.kr.tuwien.ac.at/research/ecai06.pdf`.

An Implementation for Recognizing Rule Replacements in Non-ground Answer-Set Programs*

Thomas Eiter, Patrick Traxler, and Stefan Woltran

Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter, traxler, stefan}@kr.tuwien.ac.at

1 Introduction

Answer-set programming (ASP) has emerged as an important paradigm for declarative problem solving, and provides a host for many different application domains on the basis of nonmonotonic logic programs. The increasing popularity in ASP has raised also the interest in semantic comparisons of programs in ASP [3, 4], which are nowadays recognized as a theoretical basis for program optimization, where equivalence-preserving modifications are of primary interest; in particular, rewriting rules which allow to perform a local change in a program are important. Many such rules have been considered in the propositional setting (cf., e.g., [1, 6]) but just recently have been extended to the practicably important case of non-ground programs [2].

For illustration, consider rules from an encoding of the 3-colorability problem:

$$b(X) \vee b(a) \leftarrow \mathit{edge}(X, a), \mathit{node}(X), \text{not } r(X), \text{not } g(a), \text{not } g(X) \quad (1)$$

$$r(Y) \vee b(Y) \vee g(Y) \leftarrow \mathit{node}(Y). \quad (2)$$

Results from [2] show that (i) the first rule is redundant and can be deleted in any program which contains the second rule; (ii) the entire program fragment can be rewritten into a program without disjunctions, which is equivalent for any graph specification.

In this paper, we present theoretical foundations and a practical realization for recognizing these two particular replacements, which are *rule subsumption* and *local shifting*. We describe a tool which *scans* an input program and indicates which rules can be deleted (via subsumption) and which rules apply to local shifting. As a back-end inference engine for these recognition tasks, we make use of ASP-solvers, themselves. We report first experimental evaluations, showing that our approach is feasible.

2 Replacements in Answer-Set Programming

Our objects of interest are disjunctive logic programs formulated in a language over a set $\mathcal{A}$ of *predicate symbols*, a set $\mathcal{V}$ of *variables*, and a set $\mathcal{C}$ of *constants* (also called the *domain*). Atoms, rules, and programs are defined as usual and we use, for a rule r, $H(r)$ to denote the set of atoms in the *head* of r, $B(r)$ to denote the set of literals in the *body* of r, and $B^+(r)$ (resp., $B^-(r)$) to refer to the set of positive (resp., negative) atoms in $B(r)$. Let e be an atom, rule, or a program. The set of variables occurring in

* This work was partially supported by the Austrian Science Fund (FWF) under project P18019.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 477–480, 2006.

e is denoted by $\mathcal{V}_e$, and e is called *ground* iff $\mathcal{V}_e = \emptyset$. Similarly, we use $\mathcal{C}_e$ to refer to the set of constants occurring in e. Given a rule r and a set of constants $C \subseteq \mathcal{C}$, we define $grd(r, C)$ as the set of all rules $r\vartheta$ obtained from r by all possible substitutions $\vartheta : \mathcal{V}_r \to C$. The semantics of logic programs is given in terms of answer sets as usual. $\mathcal{AS}(P)$ denotes the set of all answer sets of a program P. Programs P_1, P_2 are called *strongly* (resp., *uniformly*) *equivalent* iff, for each set S of rules (resp., facts), $\mathcal{AS}(P_1 \cup S) = \mathcal{AS}(P_2 \cup S)$. For further details we refer to [3].

We consider here two forms of replacements, *rule subsumption* and *local shifting*, cf. [2]. The former is based on the following observation, generalizing a result in [5].

Proposition 1. *Let P be a program, and r, s be different rules in P, such that there exists a substitution $\vartheta : \mathcal{V}_s \to \mathcal{V}_r \cup \mathcal{C}_r$ satisfying $H(s\vartheta) \subseteq H(r) \cup B^-(r)$ and $B(s\vartheta) \subseteq B(r)$. Then, P is strongly equivalent to $P \setminus \{r\}$.*

The aim of local shifting is motivated by the elimination of disjunctions. For an arbitrary program P, a rule $r \in P$ is called *head-cycle free* (HCF) in P iff, for each finite $C \subseteq \mathcal{C}$, all $r' \in grd(r, C)$ are head-cycle free in $grd(P, C)$; for details, see [2].

Proposition 2. *Let P be a program and $r \in P$, such that for each $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $|H(r\vartheta)| = |H(r)|$, and r is HCF in P. Then, P is uniformly equivalent to $P \setminus \{r\} \cup r^{\rightarrow}$, where*[1] *$r^{\rightarrow} = \{h \leftarrow B(r), \mathrm{not}(H(r) \setminus \{h\}) \mid h \in H(r)\}$.*

In our example from the introduction, it can be shown that (1) is subsumed by (2) by setting $\vartheta(X) = Y$. As well, both rules are HCF within the fragment, but (1) does not apply for local shifting since under $\vartheta(X) = a$, its head reduces to a single element. Next, we recapitulate the complexity of detecting rules for subsumption or shifting.

Proposition 3. *Given a program P and $r \in P$. Deciding whether (i) r is subsumed by some other rule $s \in P$ is* NP*-complete; (ii) r is HCF in P is* PSPACE*-complete.*

3 The Implemented System

For our implementation we use reductions to ASP itself to decide the problems under consideration (for details, see [7]). In particular, we provide a linear reduction for subsumption into the conjunctive query problem which is NP-complete, matching the intrinsic complexity of the encoded task. In the case of local shifting, we map (in polynomial time) this problem to that of querying a definite Horn program, which is EXPTIME-complete and thus just mildly harder than the encoded problem.

Encodings. We reduce the test whether a rule r is subsumed by a rule s as a (Boolean) conjunctive query problem (F, q), i.e., deciding, given a rule $q = b \leftarrow a_1, ..., a_m$ together with a set F of ground facts, whether the unique answer set of $F \cup \{q\}$ contains b. Given two rules r, s, we construct a set F_r of facts and a query q_s as follows, where for any rule r, r' (resp., r'') denotes the result of replacing each predicate symbol p in

[1] For a set $S = \{s_1, \ldots, s_n\}$ of atoms, not S abbreviates not $s_1, \ldots,$ not s_n.

r by a new symbol p' (resp., p''), and the substitution $\gamma : \mathcal{V} \to \mathcal{C}$ maps each variable V to a corresponding constant c_V, which does not occur in r, s:

$$F_r = H(r\gamma) \cup B^-(r\gamma) \cup B^+(r'\gamma) \cup B^-(r''\gamma);$$
$$q_s = b \leftarrow H(s), B^+(s'), B^-(s'').$$

Theorem 1. *Rule r is subsumed by rule s iff the query problem (F_r, q_s) holds.*

Concerning local shifting, let, for a program P, $C^* \supseteq \mathcal{C}_P$ be a domain of size $|C^*| = 4{\cdot}|\mathcal{C}_P|$, k be the maximal predicate-arity in P, let h, d, p and q be new predicate symbols with arities $\alpha(d) = 1$, $\alpha(h) = 2$, and $\alpha(p) = \alpha(q) = 2k + 2$, and let "_" be a new constant symbol. Define for two atoms $a = a'(t_1, \ldots, t_m)$ and $b = b'(s_1, \ldots, s_l)$ with $m, l \leq k$, and $\pi \in \{p, q\}$, the rule

$$\pi[a, b] := \pi(a', t_1, \ldots, t_m, \_, \ldots, \_, b', s_1, \ldots, s_l, \_, \ldots, \_) \leftarrow D, \tag{3}$$

such that b' appears as the $(k + 2)$nd argument in π, the _'s fill up π properly, and D denotes a sequence of $d(X)'s$, for all variables X occurring in a or b. For a program P and a set of rules $Q \subseteq P$, we define the definite Horn program

$$\begin{aligned} P_Q^* = {} & \{d(c) \mid c \in C^*\} \cup \{q[a, b] \mid a, b \in H(r), a \neq b, r \in Q\} \cup \\ & \{p[a, b] \mid a \in H(s), b \in B^+(s), s \in P, H(s) \neq \emptyset, B^+(s) \neq \emptyset\} \cup \\ & \{p(\boldsymbol{x}, \boldsymbol{z}) \leftarrow p(\boldsymbol{x}, \boldsymbol{y}), p(\boldsymbol{y}, \boldsymbol{z}); \\ & \quad h(\boldsymbol{x}, \boldsymbol{y}) \leftarrow p(\boldsymbol{x}, \boldsymbol{y}), p(\boldsymbol{y}, \boldsymbol{x}), q(\boldsymbol{x}, \boldsymbol{y}); \quad h(\boldsymbol{x}, \boldsymbol{x}) \leftarrow q(\boldsymbol{x}, \boldsymbol{x})\}; \end{aligned}$$

where $\boldsymbol{x}$ (resp., $\boldsymbol{y}$, $\boldsymbol{z}$) denotes a sequence of $k + 1$ distinct variables X_i (resp., Y_i, Z_i).

Theorem 2. *For any program P and $Q \subseteq P$, each $r \in Q$ is HCF in P iff no atom $h(\cdot, \cdot)$ is contained in the (unique) answer set of P_Q^*.*

Hence, we are able to test whether a single rule r is HCF in P (via querying $P^*_{\{r\}}$) or whether P entirely is HCF (via querying P_P^*). Moreover, inspecting atoms $h(\cdot, \cdot)$ in the answer set of P_Q^*, indicates which pair of atoms prevent rules in Q from being shifted.

System Description. The system relies on two basic steps, (i) the computation of the reductions to programs as sketched above, and (ii) the call of an ASP-solver in order to run these programs. Both reductions together with the invocation of DLV[2] are realized via perl scripts. The input program is required to be in DLV-format. Invoking `simplify program.dl`, where the file `program.dl` contains our example program, yields:

```
Scanning for Rule Subsumption...
b(X) v b(a):-edge(X,Y),node(X),not r(X),not g(a),not g(X).
[subsumed by r(Y) v b(Y) v g(Y) :- node(Y).]

Scanning for Local Shifting...
r(Y) v b(Y) v g(Y) :- node(Y).
```

indicating that Rule (1) from the program is subsumed by Rule (2), and that Rule (2) can faithfully be rewritten to a set of non-disjunctive rules, cf. Proposition 2. All scripts and further information are available at the system's homepage (see below).

[2] Available under `http://www.dlvsystem.com`.

Experiments. For first results on our approach, we set up a test series available at

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

The test for subsumption always involves a pair of rules, while the test for local shifting has to take an entire program into account. Thus, we encode for the former different NP-hards problems as pairs of rules such that subsumption holds iff the encoded problem holds. For the latter we used various application programs (some from the web) and tested whether disjunction can be eliminated in a uniform-equivalence preserving way.

Concerning subsumption, we encoded (i) graph 3-colorability and (ii) propositional satisfiability. For (i), consider a graph $G = (V, E)$ and let B_E denote the sequence of atoms $e(X_i, X_j)$, where $(v_i, v_j) \in E$. Then the rule $\leftarrow e(r, b), e(b, r), e(r, g), e(g, r), e(b, g), e(g, b)$ is subsumed by the rule $\leftarrow B_E$ iff G is 3-colorable. Our system scales well showing reasonable response times for problems generated from graphs containing up to 40 nodes (depending on the number of nodes, but within a few seconds for 30 nodes). For (ii), consider a CNF ϕ over variables $X_1, \ldots, X_n$ and represent each clause c by a triple $p(L_1, L_2, L_3)$ where $L_i = X$ (resp., $L_i = \bar{X}$) if X (resp., $\neg X$) is the i-th literal in c. Let p_ϕ be the sequence representing ϕ in this way plus pairs $v(X, \bar{X})$ for all variables X occurring in ϕ. Then, $\leftarrow p(1,1,1), p(1,1,0), p(1,0,1), p(1,0,0), p(0,1,1), p(0,1,0), p(0,0,1), v(1,0), v(0,1)$ is subsumed by the rule $\leftarrow p_\phi$ iff ϕ is satisfiable. Also in this case, our implementation provides good response times (around a few seconds) for a suite of uniform random 3-sat formulas taken from SATLIB.

Concerning the test for local shifting, we set up a suite of disjunctive programs collected from different sources, including encodings for problems as *Hamiltonian cycle*, *strategic companies*, or *diagnosis*. For all programs, our tool recognized all rules applicable to local shifting rather fast (always within a second).

The presented work has to be seen as a starting point for a more general tool considered as support for programmers in terms of offline simplification of (possibly incomplete) programs. To the best of our knowledge, our implementation is the first realization of such simplification methods working directly on non-ground programs.

References

1. S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.
2. T. Eiter, M. Fink, H. Tompits, P. Traxler, and S. Woltran. Replacements in Non-Ground Answer-Set Programming. In *Proc. KR'06*, pg. 340–351. AAAI Press, 2006.
3. T. Eiter, M. Fink, H. Tompits, and S. Woltran. Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. In *Proc. AAAI'05*, pg. 695–700. AAAI Press, 2005.
4. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
5. F. Lin and Y. Chen. Discovering Classes of Strongly Equivalent Logic Programs. In *Proc. IJCAI'05*, pages 516–521, 2005.
6. M. Osorio, J. A. Navarro, and J. Arrazola. Equivalence in Answer Set Programming. In *Proc. LOPSTR'01, Selected Papers*, vol. 2372 of *LNCS*, pg. 57–75. Springer, 2001.
7. P. Traxler. Techniques for Simplifying Disjunctive Datalog Programs with Negation. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2006.

April – An Inductive Logic Programming System

Nuno A. Fonseca[1], Fernando Silva[1], and Rui Camacho[2]

[1] DCC-FC & LIACC, Universidade do Porto
{nf, fds}@ncc.up.pt
[2] Faculdade de Engenharia & LIACC, Universidade do Porto
rcamacho@fe.up.pt

Abstract. Inductive Logic Programming (ILP) is a Machine Learning research field that has been quite successful in knowledge discovery in relational domains. ILP systems use a set of pre-classified examples (positive and negative) and prior knowledge to learn a theory in which positive examples succeed and the negative examples fail. In this paper we present a novel ILP system called April, capable of exploring several parallel strategies in distributed and shared memory machines.

1 Introduction

There is a strong connection between Inductive Logic Programming (ILP) and Logic Programming. ILP inherits from Logic Programming its representation formalism, its semantic orientation, and techniques. It is also common to see ILP systems implemented in Prolog. The major reason for using Prolog is that the inference mechanism implemented by the Prolog engine is fundamental to most ILP learning algorithms. ILP systems can therefore benefit from the extensive performance improvement work that has taken place for Prolog. On the other hand, ILP may be seen as challenging Prolog application since it often uses large sets of ground facts and requires storing a large search tree. Hence, ILP systems implemented in Prolog challenge the limits of Prolog systems due to their heavy usage of resources such as database accesses and memory usage.

The expressiveness of first-order logic gives ILP flexibility and understandability of the induced models. However, ILP systems suffer from significant limitations that reduce their applicability. First, most ILP systems execute in main memory, limiting their ability to process large databases. Second, ILP systems are computationally expensive, e.g., evaluating individual rules may take considerable time. On complex applications, ILP systems can take several hours, if not days, to return a model. Therefore, a major obstacle that ILP systems must overcome is efficiency.

In this paper we succinctly present the April ILP system, a generic purpose ILP system, implemented in Prolog with a modular design, that aims at being efficient, scalable, and flexible. April aims to be an efficient system by having low memory consumption and low response time. To this end it tries to combine and integrate several techniques to maximize efficiency (e.g., query transformations [1], randomized searches [2], coverage caching [3], lazy evaluation of

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 481–484, 2006.

examples [4], tabling [5], and parallelism [6, 7]). April's scalability is achieved by using relational databases to store the examples and ground background knowledge or by exploiting parallelism. April's ability to explore several parallelism approaches is the main difference to other systems. April aims to be flexible by providing a high level of customizations, for instance, allowing the modification of search method, heuristic, etc.

2 System Description

April can address predictive and descriptive ILP tasks. It addresses predictive learning tasks by constructing classification rules (using a MDIE-based algorithm [8]). It can also be applied to find association rules (using an algorithm similar to the one implemented by the Warmr system [9]). The ILP semantics used by April is the learning from entailment semantics. Therefore, when used to learn classification rules April follows the *normal semantic* of ILP [10]. The notion of coverage used in both tasks is intensional coverage.

April can be classified as an empirical (non-incremental), non-interactive, single predicate learning system, that does not perform predicate invention and is capable of handling noise.

April receives as input prior knowledge B (the background knowledge) and examples E, and induces a theory H that describes (explains) the examples. The examples E are represented as Prolog ground facts and the background knowledge as Prolog programs. The predicates in B can therefore be defined either intensionally or extensionally. The hypothesis language is a subset of the language of definite clauses. The hypothesis language is constrained through the use of *meta-language* declarations. April's meta-language includes determination declarations [11], mode and type declarations [8], background predicates' properties, pruning and constraints declarations, and facilities to change system parameters that may affect the hypotheses considered and the way that April operates.

April implements a *covering algorithm* to build a set of classification rules. The rules are found by performing a search through an ordered space of rules. April has two search strategies, namely top-down or stochastic, and different search methods (e.g., breadth-first, beam-search, randomized rapid restarts [2]). Several metrics are also available to score the rules, namely coverage, accuracy, etc.

A main feature of April is its ability to exploit parallelism in distributed or shared memory machines. April has several parallel algorithms built-in. The algorithms follow three main strategies: parallel exploration of the search space; parallel rule evaluation; or data parallelism [7]. One of the algorithms combines several strategies with pipelining and achieves super-linear speedups in a distributed memory computer [6]. A summary of the speedups observed in four applications are presented in the next section on Table 1.

April is implemented in Prolog and runs on top of the YAP Prolog system. Since April is implemented in Prolog the data is stored on Prolog's database (i.e., in memory). However, April has some extensions that allow the system to learn directly from relational databases. For the communication layer April

uses LAM MPI, a high-quality open-source implementation of the Message Passing Interface (MPI) specification. LAM can be used by applications running in heterogeneous clusters or in grids, but can also be used in multiple processor computers.

3 Related Work

Since the initial concept proposal of Inductive Logic Programming, in 1990, many ILP systems have been developed[1]. April is specially related to the Aleph [12] system. Like in Aleph, April's core algorithm is based on Mode Direct Inverse Entailment (MDIE), a technique initially used in the Progol [8] system. Besides the core algorithm, April also implements many features found in Aleph. Due to this close relation, April attempts to maintain high level of compatibility with the format of the input files and parameters.

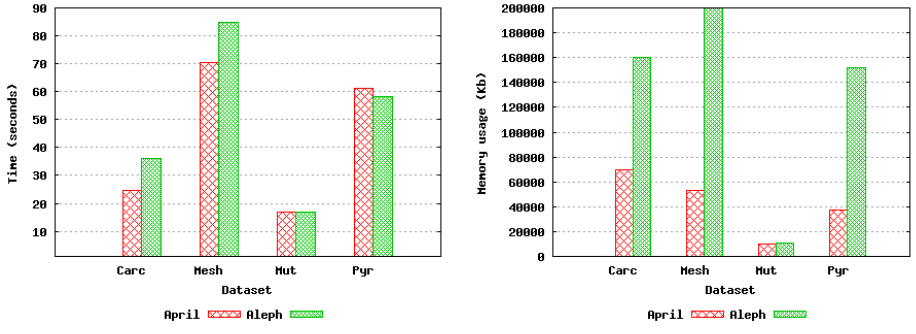


Fig. 1. Average execution time and memory consumption of April and Aleph systems in four ILP applications

A summary of an empirical comparison with Aleph is presented in Figure 1. It plots the average execution time and memory usage on four ILP applications (**Carc**inogenesis, **Mesh**, **Mut**agenesis and **Pyr**imidines) using a 10-fold cross-validation methodology. The values presented are the average of ten sequential runs to find a single rule. Figure 1 shows that sequential April is competitive against Aleph as the sequential execution time is concerned (the quality of the rules produced is also comparable). Although both systems are implemented in YAP Prolog, April's memory usage is considerably lower than Aleph. Therefore, for larger applications (number of examples or greater search spaces) April should behave better.

The main difference between April and other systems, including Aleph, resides in the ability to run in parallel using different parallel algorithms (see [6, 7]). A

[1] Srinivasan pointed out in a presentation at the ILP 2005 conference that around 100 ILP systems have been developed to date.

detailed survey of parallel ILP systems is available in [7]. Table 1 shows the speedups observed on a Beowulf cluster with one of April's parallel algorithms, the $p^2 - mdie$ parallel algorithm [6], in four ILP applications. One can observe that the speedups are good. It is important to point out that the improvements in performance obtained using the $p^2 - mdie$ parallel algorithm did not affect significantly the quality of the theories found.

Table 1. Average speedup observed for 2, 4 , 6 and 8 processors

Dataset	2	4	6	8
Carc	1.20	3.04	8.00	11.86
Mesh	1.66	4.58	6.48	7.09
Mut	3.42	6.95	6.75	8.99
Pyr	2.03	4.15	6.49	8.28

Acknowledgments. This work has been partially supported by MYDDAS (POSC/EIA/59154/2004) and funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

References

1. Vítor Santos Costa, Ashwin Srinivasan, Rui Camacho, Hendrik Blockeel, Bart Demoen, Gerda Janssens, Jan Struyf, Henk Vandecasteele, and Wim Van Laer. Query transformations for improving the efficiency of ilp systems. *Journal of Machine Learning Research*, 4:465–491, 2003.
2. F. Železný, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 333–345. Springer-Verlag, 2002.
3. James Cussens. Part-of-speech disambiguation using ilp. Technical Report PRG-TR-25-96, Oxford University Computing Laboratory, 1996.
4. Rui Camacho. As lazy as it can be. In *The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03)*, pages 47–58. Bergen, Norway, November 2003.
5. Ricardo Rocha, Nuno A. Fonseca, and Vitor Santos Costa. On Applying Tabling to Inductive Logic Programming. In *Proceedings of the 16th European Conference on Machine Learning, ECML-05*, volume 3720 of *LNAI*, pages 707–714, 2005. Springer-Verlag.
6. Nuno A. Fonseca, Fernando Silva, Vitor Santos Costa, and Rui Camacho. A pipelined data-parallel algorithm for ILP. In *Proceedings of 2005 IEEE International Conference on Cluster Computing*, 2005. IEEE.
7. Nuno A. Fonseca, Fernando Silva, and Rui Camacho. Strategies to Parallelize ILP Systems. In *Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 3625 of *LNAI*, pages 136–153, 2005. Springer-Verlag.
8. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
9. Luc Dehaspe and Hannu Toironen. *Relational Data Mining*, chapter Discovery of relational association rules, pages 189–208. Springer-Verlag, 2000.
10. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
11. T. Davies and Stuart Russell. A logical approach to reasoning by analogy. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, pages 264–270, Los Altos, California, 1987.
12. Ashwin Srinivasan. The Aleph Manual, 2003.

OPTSAT: A Tool for Solving SAT Related Optimization Problems

Enrico Giunchiglia and Marco Maratea

STAR-Lab, DIST, University of Genova
viale Francesco Causa, 13 — 16145 Genova (Italy)
{enrico, marco}@dist.unige.it

1 Introduction

Propositional satisfiability (SAT) is one of the most important and central problems in Artificial Intelligence and Computer Science. Basically, most SAT solvers are based on the well-known Davis-Logemann-Loveland (DLL) procedure. DLL is a decision procedure: given a SAT formula ϕ, it can decide if ϕ is satisfiable (and it can return a satisfying assignment μ), or not. Often, this is not sufficient, in that we would like μ to be also "optimal", i.e., that μ has also to minimize/maximize a given objective function. MAX-SAT, MIN-ONE, DISTANCE-SAT and their weighted versions are popular optimization problems. (In the following, ϕ is the input formula expressed as a set of clauses). Almost all the systems that can deal with these problems follow a classical branch&bound schema: whenever a satisfying assignment μ for ϕ with a cost c_μ is found, the search goes on looking for another satisfying assignment with a lower (or higher, depending on the problem) cost.

In this paper, we present OPTSAT (OPTimal SATisfiability), a tool for solving SAT related optimization problems based on the DLL algorithm. Here, for simplicity, we focus on MAX-SAT and MIN-ONE problems. MAX-SAT is the problem of finding an assignment (i.e., a consistent set of literals) satisfying as many clauses in ϕ as possible; MIN-ONE is the problem of determining an assignment satisfying ϕ and with as few as possible variables assigned to TRUE. MIN-SAT and MAX-ONE are defined analogously. Differently from other systems, OPTSAT does not follow a branch&bound schema, but it solves these optimization problems by imposing a partial ordering on the literals to branch on. MAX-SAT, MIN-ONE but also DISTANCE-SAT and other SAT-related optimization problems can be solved in this way. Moreover, OPTSAT is not limited, like all the other systems, to the computation of assignments which are optimal with respect to a given numeric function. OPTSAT can also solve MAX-SAT$_\subseteq$ and MIN-ONE$_\subseteq$ problems. MAX-SAT$_\subseteq$ is the problem of finding an assignment satisfying a set S of clauses and such that there is no assignment satisfying a set S' of clauses with $S \subset S' \subseteq \phi$. MIN-ONE$_\subseteq$ is defined analogously. Any solution which is "MAX-SAT"-optimal, is also "MAX-SAT$_\subseteq$"-optimal: however, in many application domains it may be sufficient to have a "MAX-SAT$_\subseteq$"-optimal solution, and this can be much easier. In the following, optimality is defined under *cardinality* (resp. *subset inclusion*) if we are considering a MAX-SAT/MIN-ONE (resp. MAX-SAT$_\subseteq$/MIN-ONE$_\subseteq$) or analogous problems.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 485–489, 2006.

2 The OPTSAT System

OPTSAT algorithm is described in details in [GM06]. Here we highlight its main points, and we present its input format, the newly implemented encodings, and report about the results obtained with them and with the integration of MINISAT.

The input format. OPTSAT input format is an extension of the well-known DIMACS format for SAT: in the comment lines (the ones starting with "c") there are all the informations that OPTSAT uses to solve the problem, i.e., the type of the problem (if it is a MAX-SAT, MIN-SAT, MAX-ONE, or MIN-ONE problem: this is specified using two flags max/min and SAT/ONE, see Example 1); and the type of optimality considered, if under cardinality or subset inclusion. The line starting with "p" is the usual problem line for DIMACS.

Example 1. We specify a MIN-ONE$_{\subseteq}$ problem with the header

```
c min ONE
c subset
p cnf n m
```

The parsing phase. Consider ϕ having n variables and m clauses. In this phase, the input file is parsed, and some operations are performed in order to define the SAT formula ϕ' that will be fed to the back-end SAT solver. First, if the problem is MAX-SAT or MIN-SAT, following a well-known method, ϕ is modified as follow: each clause $C_i \in \phi$ is replaced by C_i' defined as $\{\neg s_i \cup C_i\}$, where s_i is called *clause selector* and is a newly introduced variable. Second, if optimality is by cardinality, we compute a formula encoding the objective function. We call this function adder(S), where S is defined depending on the problem we are facing, i.e. (i) in the case of a MIN-ONE or MAX-ONE problem, S is the set of variables in ϕ or, (ii) in the case of a MIN-SAT or MAX-SAT problem, S is the set $\{s_1, \ldots, s_m\}$ of clause selectors. The goal of adder(S) is to define a sequence $b_v, \ldots, b_0$ of new variables such that for any assignment μ of the extended signature, the value of the objective function when considering the assignment μ corresponds to an assignment of $b_v, \ldots, b_0$. adder(S) can be realized, in polynomial time, in many ways. In OPTSAT, adder(S) is implemented in the following ways:

1. Bailleux/Boufkhad (BB) [BB03]. In this encoding a unary representation of integers is used: an integer x s.t. $0 \leq x \leq n$ is represented using n propositional variables $\{x_1, \ldots, x_n\}$ with (the first) "x" variables assigned to 1 (TRUE), and the others to 0 (FALSE). This representation has the property that when a variable x_j has value TRUE, all the variables x_k with $1 \leq k < j$, are TRUE as well; and similarly if x_k has value FALSE. The encoding is efficient wrt unit-propagation but it adds a quadratic number of new clauses.
2. BB$_{mod}$ (BB modified): we modified the BB encoding, in order to enforce that when b_i is assigned to FALSE (resp. TRUE), also b_{i+1} (resp. b_{i-1}) is assigned to FALSE (resp. TRUE) by unit propagation.

3. Warners [War98]. Here is used a binary representation of integers, e.g., the value x of the objective function is represented as $x = \Sigma_{i=0}^{M} 2^i x_i$, where x_i are again propositional variables, and $M = \lfloor log_2(x) \rfloor$ for $x > 0$, and $M = 0$ otherwise. This is a linear time and space encoding, that relies on sums via adder circuits and, as presented in the paper, works directly with objective functions with weights. In OPTSAT, the encoding is optimized for the non weighted case, and the size of the encoding is approximately halved.

The first two encodings are new for OPTSAT.

Solving algorithm. As we already said, OPTSAT is a modification of the DLL algorithm: it takes as input the SAT formula $\phi' = \phi \cup \mathsf{adder}(S)$ (with ϕ considered here after the introduction of clause selectors in case of MIN-SAT or MAX-SAT problem, and $\mathsf{adder(S)} = \emptyset$ if the optimality is under subset inclusion), an assignment μ (initially set to $\emptyset$), and a partial order on the set of literals. The main change that has to be made to the DLL algorithm is in the heuristic.

Consider first the case of an optimization under cardinality problem in which the sequence $b_v, \ldots, b_0$ encode the value of the objective function, b_v being the "most significant" variable. Then, the heuristic returns (i) the first not yet assigned atom b_i (i.e., the variable with the highest index i), if any, or an arbitrary variable in ϕ' (according with the heuristic of the solver); (ii) if a variable in $\{b_v, \ldots, b_0\}$ is chosen, the variable is assigned to TRUE in the case of MAX-SAT or MAX-ONE problems, and to FALSE otherwise; if the variable is chosen by the heuristic of the solver, it is left to the solver the decision about how to assign it.

In the case of an optimization under subset inclusion problem, it returns an un-assigned atom in S, if any, and assign it to FALSE in the case of MIN-SAT$_\subseteq$ or MIN-ONE$_\subseteq$ problems, and to TRUE in the other cases; or an arbitrary atom.

OPTSAT returns an optimal solution if one exists, or that no solution exists otherwise. In order to see why this is the case, observe that, in the case of minimality under cardinality, variables are preferentially and in order chosen from b_v to b_0, while in the case of minimality for subset inclusions, atom in S are chosen. Only when all these variables are assigned, the choice is delegated to the heuristic of the underlying solver. Thus, considering, e.g., a MIN-ONE or MIN-SAT problem, the algorithm first explores (assuming no literal in $\{b_v, \ldots, b_0\}$ are assigned by unit) the branches with $b_v, \ldots, b_0$ assigned by FALSE; if all such branches fail, then it explores the branches with $\{b_v, \ldots, b_1\}$ assigned to FALSE and b_0 to TRUE; if also these branches fail b_0 and b_1 are flipped and the search goes on until a satisfying assignment μ is found, or the entire search space has been explored.

One of the main property of the algorithm is that, when the first satisfying assignment is found, we are guaranteed that it is also "optimal".

DISTANCE-SAT **and problems with weights.** DISTANCE-SAT(μ) [BM06] is the problem of determining an assignment which satisfies the input formula and differing in as few as possible literals from μ. DISTANCE-SAT$_\subseteq$ is defined in the obvious way. In the weighted version of all the problems we presented, the objective function to minimize is linear function of the variables. All these problems –but also others– can be solved by OPTSAT, as shown in [GM06].

3 Experimental Results

In OPTSAT, we can now choose between zCHAFF ver. of 5.13.2004[1] and MINISAT ver. 1.14[2] to be used as back-end: each of them has been modified accordingly to the consideration made in Section 2. These are the winners of the last two SAT competitions [LS05, LS06] in the industrial categories (MINISAT together with the SAT/CNF minimizer SATELITE).

In [GM06], we showed that OPTSAT is highly competitive on MAX-SAT/MAX-SAT$_\subseteq$ and MIN-ONE/MIN-ONE$_\subseteq$ problems if compared with a variety of solvers, both tailored for a specific optimization problem, and with the solvers that showed the best performances in the PB evaluation [MR06]. To be also noticed that OPTSAT does not use any "problem-dependent" optimization, like the computation of an upper-bound of the optimal solution, using incomplete SAT solvers, performed by most of the MAX-SAT solvers. Here we extend the results in [GM06], by report about the results obtained with the new encodings and with the integration of MINISAT in our system. For lack of space we do not put any table here: the results can be found as an appendix at the system home page reported below.

In general, and as expected, because no clauses need to be added, finding a solution which is optimal under subset inclusion is easier than finding an optimal solution under cardinality. Considering the CPU times, between the BB based encodings, BB$_{mod}$ almost always is faster or competitive with BB, and in general very competitive for objective functions having a relatively low number of variables. But, when this number is high, it incurs in memory out. The use of MINISAT generally helps in reducing the time to solve a problem. The reduction is dramatic, up to 3 orders of magnitude, when considering MAX-SAT$_\subseteq$ problems. This highlights one of the main features of our approach, i.e., the possibility of levering on the enhancement that are continuously made in SAT. For this reason, we expect our system to further improve its performances thanks to the upcoming SAT race and future competitions.

Availability of the system. The binary of the system, along with benchmarks in the OPTSAT input format and a parser to the format of the PB evaluation, are available at: `http://www.star.dist.unige.it/~marco/optsat/`.

References

[BB03] Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In *9th International Conference on Principles and Practice of Constraint Programming (CP-03)*, LNCS. Springer, 2003.

[BM06] O. Bailleux and P. Marquis. Some computational aspects of DISTANCE-SAT. *Journal of Automated Reasoning (JAR), To appear*, 2006.

[1] `http://www.princeton.edu/~chaff/`

[2] `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`

[GM06] E. Giunchiglia and M. Maratea. Solving optimization problems with DLL. Accepted to ECAI 2006. Available at `http://www.star.dist.unige.it/~marco/Data/06ecai.pdf.gz`, 2006.

[LS05] D. LeBerre and L. Simon. Fifty-five solvers in vancouver: The SAT 2004 competition. In *8th International Conference on Theory an Applications of Satisfiability Testing. Selected Revised Papers.*, LNCS 3542. Springer, 2005.

[LS06] D. LeBerre and L. Simon. Preface to the special volume on the sat 2005 competitions and evaluations. *Journal of Satisfiability, Boolean Modeling and Computation (JSAT)*, 2006.

[MR06] V. M. Manquinho and O. Roussel. The first evaluation of pseudo-boolean solvers (PB05). *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:103–143, 2006.

[War98] J. P. Warners. A linear-time transformation of linear inequalities into CNF. *Information Processing Letters*, 68(2):63–69, 1998.

Automated Reasoning About Metric and Topology

Ullrich Hustadt[1], Dmitry Tishkovsky[1], Frank Wolter[1], and Michael Zakharyaschev[2]

[1] Department of Computer Science, University of Liverpool
{U.Hustadt, D.Tishkovsky, F.Wolter}@csc.liv.ac.uk
[2] School of Computer Science and Information Systems, Birkbeck College
michael@dcs.bbk.ac.uk

1 Introduction

In this paper we compare two approaches to automated reasoning about metric and topology in the framework of the logic $\mathcal{MT}$ introduced in [10]. *$\mathcal{MT}$-formulas* are built from *set variables* $p_1, p_2, \dots$ (for arbitrary subsets of a metric space) using the Booleans $\wedge$, $\vee$, $\rightarrow$, and $\neg$, the *distance operators* $\exists^{<a}$ and $\exists^{\leq a}$, for $a \in \mathbb{Q}^{>0}$, and the topological *interior* and *closure operators* $\mathbf{I}$ and $\mathbf{C}$. *Intended models* for this logic are of the form $\mathfrak{I} = (\Delta, d, p_1^{\mathfrak{I}}, p_2^{\mathfrak{I}}, \dots)$ where (Δ, d) is a metric space and $p_i^{\mathfrak{I}} \subseteq \Delta$. The *extension* $\varphi^{\mathfrak{I}} \subseteq \Delta$ of an $\mathcal{MT}$-formula φ in $\mathfrak{I}$ is defined inductively in the usual way, with $\mathbf{I}$ and $\mathbf{C}$ being interpreted as the interior and closure operators induced by the metric, and $(\exists^{<a}\varphi)^{\mathfrak{I}} = \{x \in \Delta \mid \exists y \in \varphi^{\mathfrak{I}}\ d(x,y) < a\}$. In other words, $(\mathbf{I}\varphi)^{\mathfrak{I}}$ is the interior of $\varphi^{\mathfrak{I}}$, $(\exists^{<a}\varphi)^{\mathfrak{I}}$ is the open a-neighbourhood of $\varphi^{\mathfrak{I}}$, and $(\exists^{\leq a}\varphi)^{\mathfrak{I}}$ is the closed one. A formula φ is *satisfiable* if there is a model $\mathfrak{I}$ such that $\varphi^{\mathfrak{I}} \neq \emptyset$; φ is *valid* if $\neg\varphi$ is not satisfiable.

In $\mathcal{MT}$, one can represent various basic facts about metric and topology. For example, the validity of $\exists^{<a}p \rightarrow \mathbf{I}\exists^{<a}p$ means that the open a-neighbourhood of any set is open. The non-validity of $\mathbf{C}\exists^{<a}p \rightarrow \exists^{\leq a}p$ means that there is a metric space with a subset X such that the closure of the open a-neighbourhood of X properly contains the closed a-neighbourhood of X. The logic $\mathcal{MT}$ as well as its metric fragment $\mathcal{MS}$ without the topological operators have been suggested as basic tools for reasoning about distances and similarity [9].

One obvious approach to automated reasoning with $\mathcal{MT}$ is to use the standard ε-definition of the topological interior

$$\mathbf{I}X \ = \ \{x \in X \mid \exists \varepsilon > 0\ \forall y(d(x,y) < \varepsilon \rightarrow y \in X\}$$

and translate $\mathcal{MT}$ into a two-sorted first-order language, with one sort for the real numbers and the other one for points of metric spaces. This approach allows the use of interactive systems supporting a theory of real numbers, like HOL, Isabelle, or PVS. However, it is unlikely to be a viable basis for efficient automatic reasoning about $\mathcal{MT}$ or $\mathcal{MS}$, which is our focus.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 490–493, 2006.

The alternative approach we present here is based on the results obtained in [10] which show that the intended metric models for $\mathcal{MT}$ can be (equivalently) replaced by relational Kripke-style models where the distance operators are interpreted (like in modal logic) by binary relations and the interior operator by a quasi-order. More precisely, suppose for simplicity that we are given an $\mathcal{MT}$-formula φ whose numerical parameters (in the operators $\exists^{<a}$, $\exists^{\leq a}$) are all natural numbers, and N is the maximal one. A *relational φ-model* is a structure

$$\mathfrak{R} \;=\; (W, R_1, \ldots, R_N, S_1, \ldots, S_N, R, p_1^{\mathfrak{R}}, p_2^{\mathfrak{R}}, \ldots)$$

where, for $0 < a \leq N$, (i) R_a and S_a are reflexive and symmetric binary relations on $W \neq \emptyset$, (ii) R is reflexive and transitive, (iii) $R \subseteq R_a \subseteq S_a$, (iv) $S_a \subseteq R_b$ for $a < b$, (v) xS_ayS_bz implies $xS_{a+b}z$ whenever $a + b \leq N$, (vi) xR_ayS_bz implies $xR_{a+b}z$ whenever $a+b \leq N$, (vii) xS_ayR_bz implies $xR_{a+b}z$ whenever $a+b \leq N$, and (viii) xR_ayRz implies xR_az. The operator $\exists^{<a}$ is interpreted in $\mathfrak{R}$ (in the standard Kripke style) by means of R_a, $\exists^{\leq a}$ by S_a, and $\mathbf{I}$ by R. Then, according to [10], we have the following theorem: *an $\mathcal{MT}$-formula φ is satisfiable in a metric model iff it is satisfiable in a relational φ-model.*

2 Reasoning

The relational semantics above enables a variety of reasoning techniques to be applied to the satisfiability and validity problem in $\mathcal{MT}$. Here we focus on just two: a tableau calculus, which forms the basis of our MetTeL system, and first-order translation, which allows the use of a range of existing theorem provers.

2.1 Tableau Calculus

In our tableau calculus we use a modified version of the tableau rules for hybrid logic (see e.g. [2]) with additional rules for the metric and topology operators which follow the semantics described in Section 1. For example, we use the standard tableau rules for modal logic S4 to capture the behaviour of the interior operator plus the following rules expressing interaction between the topological and metric operators (where i, j, and k are nominals):

$$\frac{@_i\exists^{<a}j \quad @_j\mathbf{I}k}{@_i\exists^{<a}k} \qquad \frac{@_i\neg\exists^{<a}j}{@_i\neg\mathbf{I}j} \qquad \frac{@_i\neg\exists^{<a}\varphi \quad @_i\mathbf{I}j}{@_j\neg\varphi} \qquad \frac{@_i\neg\exists^{\leq a}\varphi \quad @_i\mathbf{I}j}{@_j\neg\varphi}$$

Note that nominals are not part of $\mathcal{MT}$ and only serve as a technical tool in the calculus. The part of the tableau calculus related to metric operators and nominals is actually equivalent to the labelled tableau algorithm in [9]. The MetTeL system implements this tableau calculus and provides a decision procedure for $\mathcal{MT}$. MetTeL is implemented in Java 1.5.

2.2 First-Order Translation

The intuition behind the first-order encoding is to use the standard relational translation for modal logics [4] including a representation of the semantic conditions (i)–(viii) presented in Section 1. For example, occurrences of $\exists^{\leq a}\varphi$ are translated according to

$$\pi_r(\exists^{\leq a}\varphi, x) \;=\; \exists y\,(S_a(x,y) \wedge \pi_r(\varphi, y)),$$

while conditions (i) and (v), for S_a, are represented by (1) $\forall x\,(S_a(x,x))$, (2) $\forall x, y\,(S_a(x,y) \rightarrow S_a(y,x))$, and (3) $\forall x, y, z\,(S_a(x,y) \wedge S_b(y,z) \rightarrow S_c(x,z))$, for $c = a + b \leq N$. Note that the number of formulae of the form (1) and (2) which we need to include in the translation is linear in N, while for formulae of the form (3) it is quadratic in N. Our implementation of the translation also allows for the application of structural transformation and the application of alternatives to the relational translation, e.g. the axiomatic translation, [6], but in the following we restrict ourselves to the approach described above.

3 Comparisons

To establish whether the techniques presented in Section 2 provide viable means for reasoning about metric formulae with topology operators, we have devised a set of sample formulae, divided into the following groups of formulae (plus the single formula metric-axioms given by the negated conjunction of all metric axioms). (a) The formulae in the textbook group generalise the examples of interaction between the topological and metric operators from Section 1. (b) Let $\forall^{<a}\varphi = \neg\exists^{<a}\neg\varphi$. Then path-box-p.$n$ and path-box-u.n are parametric series of formulae of the shape

$$\underbrace{\forall^{<1} \ldots \forall^{<1}}_{n} p \rightarrow \forall^{<n} p \qquad \text{and} \qquad \neg(\forall^{<n} p \rightarrow \underbrace{\forall^{<1} \ldots \forall^{<1}}_{n} p),$$

respectively, with $n = 4, 8, 12, 20, 24, 32$, which test the general triangle inequality while we increase the nesting depth of $\forall^{<1}$. (c) Groups symm-box-p.n and symm-box-u.n are parametric series of formulae of the following shape:

$$p \rightarrow \forall^{<n}\neg\underbrace{\forall^{<1} \ldots \forall^{<1}}_{n}\neg p \qquad \text{and} \qquad \neg(p \rightarrow \underbrace{\forall^{<1} \ldots \forall^{<1}}_{n}\neg\forall^{<n}\neg p),$$

respectively, with $n = 4, 8, 12, 20, 24, 32$ which test the interaction of symmetry and the general triangle inequality while we increase the nesting depth of $\forall^{<1}$.

For the first-order translation approach we have used a range of state-of-the-art first-order theorem provers, Darwin 1.1 [1], DCTP 1.31 [3], E 0.91 [7], SPASS 2.2 [8], Vampire 7.0 [5]. While the last three are based on resolution calculi, Darwin and DCTP are based on the model evolution and the disconnection calculus, respectively. Each prover was executed on each sample formula with a timelimit of 1000 CPU seconds. In the case of first-order translation, the time required to perform the translation has been included. All tests were performed on a 2.8GHz Pentium 4 PC with 1024MB main memory under RedHat Linux 9. Figure 1 shows a summary of the results. Time is measured in user CPU seconds. A 'Timeout' entry indicates that the CPU timelimit was exceeded while a 'Fail' entry indicates that the reasoner failed before the timelimit was reached, e.g. because it was running out of memory.

Concerning the four parametric series of formulae, we see that increasing the parameter n increases the time required to solve a formula. The difference

Sample problem	Status	MetTeL	Darwin	DCTP	E	SPASS	Vampire
textbook.00	unsat	0.28	0.14	0.15	0.15	0.12	0.16
textbook.01	sat	0.22	0.16	0.14	0.16	0.13	0.16
textbook.02	unsat	6.63	5.16	0.16	2.34	26.68	33.42
textbook.03	sat	0.22	0.18	0.14	0.15	0.14	0.25
metric-axioms	unsat	0.88	32.01	0.68	*T/O*	*T/O*	*Fail*
path-box-p.04	sat	0.29	0.10	0.14	0.16	0.12	0.17
path-box-p.08	sat	2.86	0.28	0.14	0.20	0.39	48.03
path-box-p.12	sat	17.54	1.11	0.20	0.35	5.74	*T/O*
path-box-p.20	sat	236.63	17.57	0.55	2.25	312.68	*T/O*
path-box-p.24	sat	633.02	47.23	0.99	7.59	*T/O*	*T/O*
path-box-p.32	sat	*T/O*	230.76	3.51	45.05	*T/O*	*T/O*
path-box-u.04	unsat	0.43	0.15	0.14	0.18	0.13	0.16
path-box-u.08	unsat	2.34	0.20	0.14	1.42	0.23	0.16
path-box-u.12	unsat	14.61	0.92	0.23	14.49	1.85	0.43
path-box-u.20	unsat	194.87	14.14	0.92	977.59	73.53	3.42
path-box-u.24	unsat	575.69	35.54	1.39	*T/O*	273.85	9.87
path-box-u.32	unsat	*T/O*	171.98	4.14	*T/O*	*T/O*	408.76
symm-box-p.04	sat	0.15	0.32	0.13	1.32	0.27	0.36
symm-box-p.08	sat	0.18	104.66	0.17	*T/O*	*T/O*	*T/O*
symm-box-p.12	sat	0.19	*T/O*	0.19	*T/O*	*T/O*	*T/O*
symm-box-p.20	sat	0.22	*Fail*	0.56	*T/O*	*T/O*	*T/O*
symm-box-p.24	sat	0.20	*Fail*	1.04	*T/O*	*T/O*	*T/O*
symm-box-p.32	sat	0.23	*Fail*	3.50	*T/O*	*T/O*	*T/O*
symm-box-u.04	unsat	0.30	0.15	0.14	0.16	0.14	0.17
symm-box-u.08	unsat	2.30	0.26	0.17	1.89	0.23	0.18
symm-box-u.12	unsat	12.41	1.06	0.28	18.73	1.89	0.81
symm-box-u.20	unsat	191.92	16.36	1.24	996.37	73.66	4.62
symm-box-u.24	unsat	517.35	42.04	2.67	*T/O*	272.38	13.99
symm-box-u.32	unsat	*T/O*	191.54	7.02	*T/O*	*T/O*	480.36

Fig. 1. Performance of various provers on sample metric formulae

between the various approaches and provers is the extent to which it does so. Overall, DCTP on the relational translation performs best. The three resolution provers have more difficulty on satisfiable formulae compared to all other provers, but not uniformly so: on the path-box-p.n series, E performs better than the tableau system MetTeL. On the other hand, MetTeL performs better than E on the path-box-u.n and the symm-box-u.n series of unsatisfiable formulae, which might also be surprising. SPASS and Vampire seem roughly on par.

References

1. P. Baumgartner, A. Fuchs, and C. Tinelli. Implementing the model evolution calculus. *International Journal of Artificial Intelligence Tools*, 15(1), 2005.
2. P. Blackburn and M. Marx. Tableaux for quantified hybrid logic. In *Proc. TABLEAUX 2002*, LNAI 2381, pages 38–52, 2002.
3. R. Letz and G. Stenz. DCTP: A disconnection calculus theorem prover. In *Proc. IJCAR 2001*, LNAI 2083, pages 381–385. Springer, 2001.
4. H. J. Ohlbach, A. Nonnengart, M. de Rijke, and D. M. Gabbay. Encoding two-valued nonclassical logics in classical logic. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1403–1485. Elsevier, 2001.
5. A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Comm.*, 15(2-3):91–110, 2002.
6. R. A. Schmidt and U. Hustadt. A principle for incorporating axioms into the first-order translation of modal formulae. In *Proc. CADE-19*, LNAI 2741, pages 412–426. Springer, 2003.
7. S. Schulz. E: A Brainiac theorem prover. *AI Comm.*, 15(2/3):111–126, 2002.
8. C. Weidenbach, U. Brahm, T. Hillenbrand, E. Keen, C. Theobald, and D. Topic. SPASS version 2.0. In *Proc. CADE-18*, LNAI 2392, pages 275–279. Springer, 2002.
9. F. Wolter and M. Zakharyaschev. Reasoning about distances. In *Proc. of IJCAI 2003*, pages 1275–1280. Morgan Kaufmann, 2003.
10. F. Wolter and M. Zakharyaschev. A logic for metric and topology. *Journal of Symbolic Logic*, 70:795–828, 2005.

The QBFEVAL Web Portal

Massimo Narizzano, Luca Pulina, and Armando Tacchella*

Laboratory of Systems and Technologies for Automated Reasoning (STAR-Lab)
DIST, Università di Genova, Viale Causa, 13 – 16145 Genova, Italy
{mox, pulina, tac}@dist.unige.it

Abstract. In this paper we describe the QBFEVAL web portal, an on-line resource supporting the participants and the organizers of the yearly evaluation of QBF solvers and instances.

1 Introduction

The implementation of effective automated reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is attracting increasing attention, e.g., in formal verification, planning, and reasoning about knowledge (see, e.g., [1] for relevant references). In this context, the yearly evaluation of QBF solvers and instances [1] has been established with the aim of assessing the advancements in the field of QBF reasoning. The QBFEVAL web portal, integrated into QBFLIB [2], is motivated by the need of organizing the increasing amount of data produced by the evaluations and making it available for the community perusal.

Currently QBFEVAL automates several tasks, ranging from the submission of solvers and instances, to the generation of hyper-textual reports describing different views about the QBF evaluations. To offer these features in a flexible and scalable way, we implemented the portal on top of a three-tier architecture [3] using web services [4] to connect underlying components distributed across different hardware platforms. Although QBFEVAL is not an automated reasoning system per se, we believe that it provides an essential tool to improve the state of the art in QBF research and applications. From this point of view, the key feature of QBFEVAL is its extensive support to the manual extraction of information.

QBFEVAL is available for on-line browsing at `www.qbflib.org/qbfeval`, and the source code of the portal is downloadable from QBFLIB.

2 Architecture and Implementation

The architecture of QBFEVAL is based on the three-tier paradigm [3], whereby three separate software layers provide user interface, process logic, and data manipulation, respectively. Figure 1 presents an overview of the hardware/software

* The authors wish to thank the Italian Ministry of University and Research (MIUR) for its financial support, and the anonymous reviewers who helped to improve the original manuscript.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 494–497, 2006.

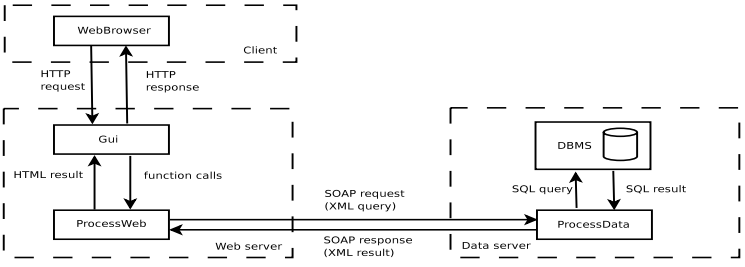


Fig. 1. QBFEVAL architecture

architecture using the following notation: dashed-outline boxes denote hardware components, solid-outline boxes denote relevant software modules, and the arrows denote communications occurring between the modules. With reference to Figure 1, QBFEVAL involves three pieces of hardware:

Client: a user's PC, i.e., a thin client containing a WebBrowser.
Web server: the front-end server, containing the Gui and a part of the process logic (ProcessWeb).
Data server: the back-end server of QBFEVAL, containing the bulk of the process logic (ProcessData), the data manipulation functionality (DBMS).

The data flow across the modules can be summarized as follows: (*i*) the user navigating QBFEVAL with her browser originates HTTP requests that the web server satisfies using the content dynamically generated by the scripts of the Gui module; (*ii*) the presentation offered by the Gui module is supported by the functions of the process layer, in particular by the set of scripts ProcessWeb that resides on the web server; (*iii*) the ProcessWeb module relies on the web services offered by the module ProcessData to perform heavy computations and data manipulations: the two process modules communicate using SOAP, whereby each service is negotiated and data is exchanged using XML; (*iv*) the ProcessData scripts are interfaced with the data base through a socket connection whereby SQL queries and results can be exchanged.

The clear cut interfaces between the layers Gui, Process[Web,Data] and DBMS allow us to modify extensively the implementation of each one without hurting the stability of the whole system. Moreover, a distinctive advantage of our implementation based on web services is that, by paying a small increase in complexity, we are able to decouple substantially the part of data presentation (hosted in the web server) and the part of data storage/manipulation (hosted in the data server). This could allow us, e.g., to transparently distribute the data part across several remote servers.

3 Interface

Figure 2 shows a hierarchical map of the QBFEVAL interface using the following notation: solid-outline boxes stand for pages containing tables with specific views

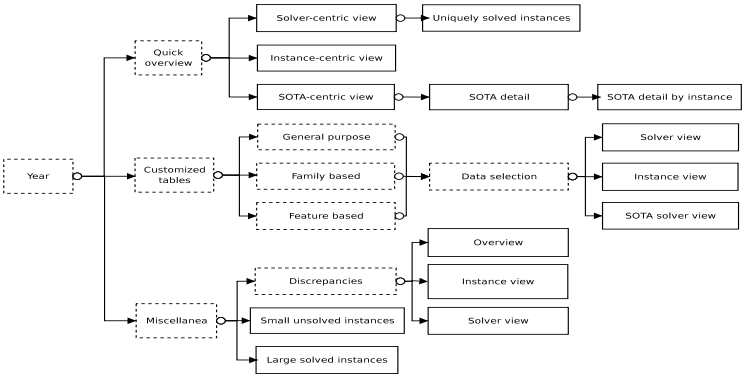


Fig. 2. QBFEVAL structure

of QBFEVAL data, dotted-outline boxes represent menus and forms by which the user can interact with QBFEVAL, and the arrows denote hyper-textual links between the components. With reference to Figure 2:

Quick Overview. The tables in this section are available according to a categorization of the instances in two classes: probabilistic structure and fixed structure (see [1] for more details). With reference to Figure 2, for each class, we report tables focusing on the relative performance of the solvers (Solver-centric view) and the relative hardness of the instances (Instance-centric view). In the SOTA solver section (SOTA-centric view), further details about the contribution of each solver to the SOTA solver can be dug out (SOTA detail), going from a general overview, down to the data related to each instance (SOTA detail by instance).

Customized tables. The tables of this section are generated dynamically according to the user's preferences. With reference to Figure 2, the General purpose form allows the user to extract specific information grouping instances by several parameters. The Family based form offers a specialized selection centered around family grouping. Finally, the Feature based form is a (still experimental) section that allows the user to extract data using features, i.e., attributes of the instances. The combination of parameters that can be specified using the Data selection wizard allows the user to obtain solver-centric, instance-centric and SOTA solver views (Solver view, Instance view, SOTA solver view).

Miscellanea. This section groups some views that should be particularly useful for developers. In particular, the Discrepancies form enables to obtain data about those instances where at least two solvers reported a different satisfiability result; the data can be arranged to give the global picture (Overview) or to obtain specific information about instances (Overview by instance) or about solvers (Overview by solver). The Small unsolved instances and the Large unsolved instances tables report data about small instances that could not be solved and about large instances that have been solved, respectively.

4 Conclusions

As far as we know, QBFEVAL is the first system in its genre to be presented in the context of automated reasoning for QBFs. QBFEVAL builds on and improves QBFLIB [2] which aims to become to the QBF community what TPTP [5] is for the automated theorem proving community. Although the work of QBFEVAL has been inspired by SATEx [6] and the on-line reporting of the SAT competition [7], we believe that our portal differs substantially from the aforementioned contributions. SATEx is a framework for continuous experimentation in SAT, i.e., developers can submit SAT solvers and instances at any time, and such submissions enter the evaluation process which is constantly summarized on the web pages. On the other hand, QBFEVAL offers a historical series of in-depth snapshots about the state of the art in QBF considering specific events; therefore, even if QBFEVAL replicates some of the underlying machinery of SATEx, the kind of reporting available in QBFEVAL is not meant to be available in SATEx. As for the on-line reporting of the SAT competition, we believe that our portal offers more customizable report generation tools, and more extensive support of the event. It is also important to notice that QBFEVAL is the only such portal featuring a complete source-code distribution available for free download. Although objectivity and reproducibility of the results made available through QBFEVAL are not its main focus, the open source distribution allows other researchers to peruse our scripts, customize them, and identify possible bugs that we could have overlooked, thus contributing to the overall robustness of the platform.

References

1. M. Narizzano, L. Pulina, and A. Tacchella. The third QBF solvers comparative evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:145–164, 2006. Available on-line at `http://jsat.ewi.tudelft.nl/` [2006-6-2].
2. E. Giunchiglia, M. Narizzano, and A. Tacchella. Quantified Boolean Formulas satisfiability library (QBFLIB), 2001. `www.qbflib.org`.
3. W. W. Eckerson. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. *Open Information Systems*, 3, 1995.
4. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture, February 2004. W3C Working Group Note.
5. G. Sutcliffe and C. Suttner. The tptp problem library for automated theorem proving. Available from `http://www.cs.miami.edu/~tptp/` [2006-6-19].
6. L. Simon and P. Chatalic. SATEx: a Web-based Framework for SAT Experimentation. In *Workshop on Theory and Applications of Satisfiability Testing*, 2001.
7. D. Le Berre and L. Simon. The SAT Competition. `http://www.satcompetition.org` [2006-6-2].

A Slicing Tool for Lazy Functional Logic Programs*

Claudio Ochoa[1], Josep Silva[2], and Germán Vidal[2]

[1] DIA, Tech. University of Madrid, 28660 Boadilla del Monte, Spain
cochoa@fi.upm.es
[2] Technical University of Valencia, Camino de Vera s/n, 46022 Valencia, Spain
{jsilva, gvidal}@dsic.upv.es

Abstract. Program slicing is a well-known technique that has been widely used for debugging in the context of imperative programming. Debugging is a particularly difficult task within lazy declarative programming. In particular, there exist very few approaches to program slicing in this context. In this paper, we describe a slicing tool for first-order lazy functional logic languages. We also illustrate its usefulness by means of an example.

1 Introduction

Program slicing is a well-known technique to extract a program fragment w.r.t. some criterion. It was first proposed as a debugging tool [3] to allow a better understanding of the portion of code which revealed an error; nowadays, it has been successfully applied to a wide variety of software engineering tasks, such as program understanding, debugging, testing, specialization, etc. Unfortunately, there are very few approaches to program slicing in the context of *declarative* languages. Basically, a *program slice* consists of those program statements which are (potentially) related to the values computed at some program point and/or variable, referred to as a *slicing criterion*.

In this work, we describe a slicing tool for first-order lazy functional logic languages. Our tool is built on top of a tracer based on *redex trails* [1], which allows the presentation of computation traces in a way easier to understand for the programmer. A clear advantage of our approach [2] is that existing tracers can be extended with slicing capabilities with a modest implementation effort, since the same data structure—the redex trail—is used for both tracing and slicing. Furthermore, it can easily be extended to cope with other language features like built-in functions, higher-order combinators, etc., since all these features are already covered by state-of-the-art debuggers based on redex trails.

* This work has been partially supported by the EU (FEDER) and the Spanish MEC under grant TIN2005-09207-C03-02, and by the ICT for EU-India Cross-Cultural Dissemination Project ALA/95/23/2003/077-054.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 498–501, 2006.

2 The Slicing Tool

In this section, we describe the structure of our slicing tool. It can be used both for debugging—by automatically extracting the program fragment which contains an error—and for program specialization—by generating executable slices w.r.t. a given slicing criterion. The technical details of this slicing technique can be found in [2].

Tracer. It is introduced in [1]. The tracer executes a program using an *instrumented* interpreter. As a side effect of the execution, the *redex trail* of the computation is stored in a file. The tracer is implemented in Haskell and accepts first-order lazy functional logic programs that can be traced either backwards or forwards. In our slicer, we only slightly extended the original tracer in order to also store in the redex trail the location (the so called *program position*), in the source program, of every reduced expression.

Viewer. Once the computation terminates (or it is aborted by the user in case of a looping computation), the viewer reads the file with the redex trail and allows the user to navigate through the entire computation. The viewer, also introduced in [1], is implemented in Curry, a conservative extension of Haskell with features from logic programming including logical variables and non-determinism. The viewer is useful in our approach to help the user to identify the slicing criterion.

Slicer. Given a redex trail and a slicing criterion, the slicer outputs a set of program positions that uniquely identify the associated program slice. The slicing tool is implemented in Curry too, and includes an editor that shows the original program and, when a slice is computed, it also highlights the expressions that belong to the computed slice.

Specializer. Similarly to the slicer, the specializer also computes a set of program positions—a slice—w.r.t. a slicing criterion. However, rather than using this information to highlight a fragment of the original program, it is used to extract an executable slice (possibly simplified) that can be seen as a specialization of the original program for the given slicing criterion.

More information on the slicing tool (including examples, benchmarks, source code) is publicly available at: `http://www.dsic.upv.es/~jsilva/slicer/`.

3 The Slicer in Practice

In order to show the usefulness of our slicer, this section presents a debugging session that combines tracing and slicing.

We consider the program shown in Fig. 1 (for the time being, the reader can safely ignore the distinction between gray and black text). In this program, the

```
data T = Hits Int Int

main = printMax (minMaxHits webSiteHits)

webSiteHits = [0, 21, 23, 45, 16, 65, 17]

printMin t = case t of (Hits x _) -> show x
printMax t = case t of (Hits _ y) -> show y

fst t = case t of (Hits x _) -> x
snd t = case t of (Hits _ y) -> y

minMaxHits xs = case xs of
                  (y:ys) -> case ys of
                              []     -> (Hits y y);
                               (z:zs) -> let m = minMaxHits (z:zs)
                                         in (Hits (min y (fst m))
                                                  (max y (snd m)))

min x y = if (leq x y) then x else y
max x y = if (leq x y) then y else x

leq x y = if x==0 then False
                  else if y==0 then False else leq (x-1) (y-1)
```

Fig. 1. Example program `minMaxHits`

function `main` returns the maximum number of hits of a given web page in a span of time. Function `main` simply calls `minMaxHits` which traverses a list containing the daily hits of a given page and returns a data structure with the minimum and maximum of such a list.

The execution of the program above should return 65, since this is the maximum number of hits in the given span of time. However, the code is faulty and prints 0 instead. We can trace this computation in order to find the source of the error. The tracer initially shows the following top-level trace:

```
0 = main
0 = printMax     (Hits _ 0)
0 = prettyPrint  0
0 = if_then_else True 0 0
0 = 0
```

Each row in the trace has the form $val = exp$, where exp is an expression and val is the computed value for this expression.

By inspecting this trace, it should be clear that the argument of `printMax` is erroneous, since it contains 0 as the maximum number of hits. Note that the minimum (represented by "`_`" in the trace) has not been computed due to the laziness of the considered language. Now, if the user selects the argument of `printMax`, the following subtrace is shown:

```
0          = printMax   (Hits _ 0)
(Hits _ 0) = minMaxHits (0:_)
(Hits _ 0) = Hits       _ 0
```

Table 1. Benchmark results

benchmark	time	orig size	slice size	reduction (%)
minmax	11 ms.	1.035 bytes	724 bytes	69.95
horseman	19 ms.	625 bytes	246 bytes	39.36
lcc	33 ms.	784 bytes	613 bytes	78.19
colormap	3 ms.	587 bytes	219 bytes	37.31
family_con	4 ms.	1453 bytes	262 bytes	18.03
family_nd	2 ms.	731 bytes	289 bytes	29.53
Average	**12 ms.**			**43.73**

From these subtraces, the user can easily conclude that the evaluation of function `minMaxHits` (rather than its definition) contains a bug since it returns `0` as the maximum of a list without evaluating the rest of the list.

At this point, the tracer cannot provide any further information about the location of the bug. This is where slicing comes into play: the programmer can use the slicer in order to isolate the slice which is responsible of the wrong result; in general, it would be much easier to locate the bug in the slice than in the complete source program. For instance, in this example, the slice would contain the black text in Fig. 1. Indeed, this slice contains a bug: the first occurrence of `False` in function `leq` (less than or equal to) should be `True`. Note that, even though the evaluation of `minMaxHits` was erroneous, its definition was correct.

4 Benchmarking the Slicer

In order to measure the specialization capabilities of our tool, we conducted some experiments over a subset of the examples listed in

`http://www.informatik.uni-kiel.de/~curry/examples`.

Some of the benchmarks are purely functional programs (`horseman`, `family_nd`), some of them are purely logic (`colormap`, `family_con`), and the rest are functional logic programs.

Results are summarized in Table 1. For each benchmark, we show the time spent to slice it, the sizes of both the benchmark and its slice, and the percentage of source code reduction after slicing. As shown in the table, an average code reduction of more than 40% is reached.

References

1. B. Braßel, M. Hanus, F. Huch, and G. Vidal. A Semantics for Tracing Declarative Multi-Paradigm Programs. In *Proc. of PPDP'04*, pages 179–190. ACM Press, 2004.
2. C. Ochoa, J. Silva, and G. Vidal. Dynamic Slicing Based on Redex Trails. In *Proc. of PEPM'04*, pages 123–134. ACM Press, 2004.
3. M.D. Weiser. *Program Slices: Formal, Psychological, and Practical Investigations of an Automatic Program Abstraction Method*. PhD thesis, U. of Michigan, 1979.

cc⊤: A Correspondence-Checking Tool for Logic Programs Under the Answer-Set Semantics

Johannes Oetsch[1], Martina Seidl[2], Hans Tompits[1], and Stefan Woltran[1]

[1] Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
{oetsch, tompits, stefan}@kr.tuwien.ac.at
[2] Institut für Softwaretechnik 188/3, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
seidl@big.tuwien.ac.at

Abstract. In recent work, a general framework for specifying correspondences between logic programs under the answer-set semantics has been defined. The framework captures different notions of equivalence, including well-known ones like ordinary, strong, and uniform equivalence, as well as refined ones based on the *projection* of answer sets where not all parts of an answer set are of relevance. In this paper, we describe an implementation to verify program correspondences in this general framework. The system, called cc⊤, relies on linear-time constructible reductions to *quantified propositional logic* and uses extant solvers for the latter language as back-end inference engines.

1 General Information

To support engineering tasks in *answer-set programming* (ASP) [4], an important issue is to determine the equivalence of different problem encodings, given by two logic programs. Various notions of equivalence between programs have been studied in the literature [7, 2, 11] including the recently proposed framework by Eiter *et al.* [3], which subsumes most of the previously introduced notions. Within this framework, correspondence between two programs, P and Q, holds iff the answer sets of $P \cup R$ and $Q \cup R$ satisfy certain specified criteria, for any program R in a specified class, called the *context*. This kind of program correspondence includes the well-known notions of *ordinary equivalence*, *strong equivalence* [7], *uniform equivalence* [2], *relativised* variants of the latter two [11], as well as the practicably important case of program comparison under *projected* answer sets as special instances. In the latter setting, not a whole answer set of a program is of interest, but only its intersection on a subset of all letters, corresponding to a removal of auxiliary letters in computation.

In this paper, we briefly describe the main features of the system cc⊤ (short for "correspondence-checking tool"), which implements correspondence problems in the framework of Eiter *et al.* [3]. Compared to similar tools which are restricted to the notions of strong and ordinary equivalence [1, 9], cc⊤ supports the user with more fine-grained equivalence notions, allowing practical comparisons useful for debugging and modular programming. Further information about cc⊤ is also available on the Web at

http://www.kr.tuwien.ac.at/research/ccT/.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 502–505, 2006.

2 System Specifics

Theoretical Background. We are concerned here with *disjunctive logic programs* with default negation over a universe $\mathcal{U}$ of propositional atoms under the *answer-set semantics* [5]. Given a program P, we denote by $\mathcal{AS}(P)$ the collection of its answer sets; moreover, P_A denotes the class of all programs given over a set $A \subseteq \mathcal{U}$ of atoms. Two programs, P and Q, are *ordinarily equivalent* iff $\mathcal{AS}(P) = \mathcal{AS}(Q)$. P and Q are *strongly equivalent* [7] iff, for any program R, $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$. In abstracting from these equivalence notions, Eiter *et al.* [3] introduce the notion of a *correspondence problem* which allows to specify, on the one hand, a *context*, i.e., a class of programs used to be added to the programs under consideration and, on the other hand, the relation that has to hold between the collections of answer sets of the extended programs. Following Eiter *et al.* [3], we focus here on correspondence problems where the context is parametrised in terms of alphabets and the comparison relation is a projection of the standard subset or set-equality relation. In formal terms, a correspondence problem, Π, (over $\mathcal{U}$) is a quadruple of form $(P, Q, \mathcal{P}_A, \rho_B)$, where $P, Q \in \mathcal{P}_\mathcal{U}$, $A, B \subseteq \mathcal{U}$ are sets of atoms, and ρ_B is either $\subseteq_B$ or $=_B$, which are defined as follows: for any sets S, S', $S \subseteq_B S'$ iff $S|_B \subseteq S'|_B$, and $S =_B S'$ iff $S|_B = S'|_B$, where $S|_B = \{I \cap B \mid I \in S\}$. We say that Π *holds* iff, for all $R \in \mathcal{P}_A$, $(\mathcal{AS}(P \cup R), \mathcal{AS}(Q \cup R)) \in \rho_B$. We call Π an *equivalence problem* if ρ_B is given by $=_B$, and an *inclusion problem* if ρ_B is given by $\subseteq_B$, for some $B \subseteq \mathcal{U}$. Note that $(P, Q, \mathcal{P}_A, =_B)$ holds iff $(P, Q, \mathcal{P}_A, \subseteq_B)$ and $(Q, P, \mathcal{P}_A, \subseteq_B)$ jointly hold.

Example 1. Consider the following two programs which both express the selection of exactly one of the atoms a, b (an atom can only be selected if it can be derived together with the context):

$$P = \{\ sel(b) \leftarrow b, not\ out(b);\quad sel(a) \leftarrow a, not\ out(a);\quad out(a) \vee out(b) \leftarrow a, b\ \}.$$

$$Q = \{\ fail \leftarrow sel(a), not\ a, not\ fail;\quad fail \leftarrow sel(b), not\ b, not\ fail;\quad sel(a) \vee sel(b) \leftarrow a;\quad sel(a) \vee sel(b) \leftarrow b\ \}.$$

Both programs use "local" atoms, $out(\cdot)$ and $fail$, respectively, which are expected not to appear in the context. We thus may consider $\Pi = (P, Q, \mathcal{P}_A, =_B)$ as a suitable equivalence problem, specifying $A = \{a, b\}$ (or, more generally, A taking as any set of atoms not containing the local atoms $out(a)$, $out(b)$, and $fail$) and $B = \{sel(a),\ sel(b)\}$. It is a straightforward matter to check that Π, defined in this way, holds. □

Implementation Methodology. The overall approach of cc⊤ is (i) to reduce correspondence problems, as introduced above, to the satisfiability problem of *quantified propositional logic*, an extension of classical propositional logic characterised by the condition that its sentences, usually referred to as *quantified Boolean formulas* (QBFs), are permitted to contain quantifications over atomic formulas, and (ii) to use extant QBF solvers as back-end inference engines for evaluating the resulting QBFs. The theoretical basis of this approach has been developed in previous work [10], where reductions constructible in *linear time and space* are provided. The motivation for adopting such a

reduction approach is due to the fact that correspondence checking is hard [3], lying on the fourth level of the polynomial hierarchy (thus, QBFs are a suitable target formalism), and since several practicably efficient QBF solvers are available (see, e.g., [6] for an overview about different QBF solvers).

Concerning the translation step, cc$\top$ implements the necessary reductions [10] (together with some simplifications, see [8] for details) from a given inclusion or equivalence problem Π to a corresponding QBF Φ such that Φ is valid iff Π holds. The reductions are designed along so-called *spoilers* [3]: The existence of a spoiler for a given inclusion problem Π indicates that Π does not hold; equivalence tests are encoded by two inclusion tests. In general, the complexity of correspondence checking is Π_4^P-complete, leading to QBFs matching this intrinsic complexity, i.e., they possess up to three quantifier alternations. However, if the specified problem falls into an easier class, cc$\top$ provides an encoding in terms of QBFs which are less involving.

For the evaluation of the resultant QBFs, the user has to employ an off-the-shelf QBF solver. Several such tools are nowadays available [6], but most of them require the input to be in a specific normal form. In such a case, the generated QBFs have to be processed according to the input syntax of the considered solver. Details about the normal-form translation employed by cc$\top$ can be found elsewhere [8].

Applying the System. The system takes as input two programs, P and Q, and two sets of atoms, A and B, where A specifies the alphabet of the context and B specifies the set of atoms used for the projection in the chosen correspondence relation. The user can select (via command-line options) between two kinds of reductions (see [10] for details), a more naive one or an optimised one, which is also the default option. As well, it can be selected whether the programs are compared with respect to an inclusion or an equivalence problem. The syntax of the programs is the basic `DLV` syntax.[1]

Let us consider the two programs P and Q from Example 1, and suppose they are stored in files `P.dl` and `Q.dl`, respectively. If we want to use cc$\top$ for checking whether P is equivalent to Q with respect to the projection to the output predicate $sel(\cdot)$, and restricting the context to programs over $\{a, b\}$, then we need to specify

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

By default, cc$\top$ writes the resulting QBF to the standard-output device. The QBF can then be processed further by QBF solvers. The output can also be piped, e.g., directly to the BDD-based QBF solver `boole`.[2] Choosing the latter way, invoking cc$\top$ on our example thus looks as follows:

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

In this case, the output (from `boole`) is `0` or `1` as answer for the input correspondence problem. In our example, the correspondence holds and the output is therefore `1`.

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

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

[2] See `http://www.cs.cmu.edu/~modelcheck/bdd.html`.

3 Discussion

In this paper, we presented an implementation for advanced program comparisons in answer-set programming via encodings into quantified propositional logic. In other work [8], we reported about initial experimental evaluations of our tool, also containing a comparison between cc⊤ and the system DLPEQ [9], which computes ordinary equivalence by means of ASP solvers. We furthermore note that, for the special case of checking strong equivalence, our system uses reductions to SAT, i.e., to the satisfiability problem of (ordinary) propositional logic. This is basically done in the same way as in the special-purpose strong-equivalence checker SELP [1]. Thus, our system can be understood as a generalisation of that approach. Compared to these other systems, cc⊤, however, processes a much broader range of correspondence problems, which are also computationally more involving.

We consider our system as a starting point for a tool box to support modular programming and offline program simplification. Future work includes an extension of the system to other classes of logic programs (like, e.g., nested logic programs) and to further correspondence notions (in particular, ones based on uniform equivalence [2]).

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

References

1. Y. Chen, F. Lin, and L. Li. SELP - A System for Studying Strong Equivalence between Logic Programs. In *Proc. LPNMR'05*, volume 3662 of *LNCS*, pages 442–446. Springer, 2005.
2. T. Eiter and M. Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proc. ICLP'03*, number 2916 in *LNCS*, pages 224–238. Springer, 2003.
3. T. Eiter, H. Tompits, and S. Woltran. On Solution Correspondences in Answer-Set Programming. In *Proc. IJCAI'05*, pages 97–102, 2005.
4. M. Gelfond and N. Leone. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence*, 138(1-2):3–38, 2002.
5. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
6. D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. The Second QBF Solvers Comparative Evaluation. In *Proc. SAT'04*, volume 3542 of *LNCS*, pages 376–392. Springer, 2005.
7. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
8. J. Oetsch, M. Seidl, H. Tompits, and S. Woltran. A Tool for Advanced Correspondence Checking in Answer-Set Programming. In *Proc. NMR'06*, pages 20–29, 2006.
9. E. Oikarinen and T. Janhunen. Verifying the Equivalence of Logic Programs in the Disjunctive Case. In *Proc. LPNMR'04*, volume 2923 of *LNCS*, pages 180–193. Springer, 2004.
10. H. Tompits and S. Woltran. Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In *Proc. ICLP'05*, volume 3668 of *LNCS*, pages 189–203. Springer, 2005.
11. S. Woltran. Characterizations for Relativized Notions of Equivalence in Answer-Set Programming. In *Proc. JELIA'04*, volume 3229 of *LNCS*, pages 161–173. Springer, 2004.

A Logic-Based Tool for Semantic Information Extraction

Massimo Ruffolo[1,3], Marco Manna[4], Lorenzo Gallucci[1], Nicola Leone[1,4], and Domenico Saccà[1,2,3]

[1] Exeura s.r.l.
[2] DEIS - Department of Electronics, Computer Science and Systems
[3] ICAR-CNR - Institute of High Performance Computing and Networking of the Italian National Research Council
[4] Department of Mathematics
University of Calabria, 87036 Arcavacata di Rende (CS), Italy
{ruffolo, sacca}@icar.cnr.it
{leone, manna}@unical.it
gallucci@exeura.it

Abstract. Recognizing and extracting meaningful information from unstructured Web documents, taking into account their semantics, is an important problem in information and knowledge management. This paper describes HiLeX, a system implementing a novel logic-based approach to information extraction from unstructured documents. The approach adopted in the HiLeX system is founded on a new two-dimensional representation of documents, and heavily exploits DLP^+ - an extension of disjunctive logic programming for ontology representation and reasoning, which has been recently implemented on top of the *DLV* system. Unlike previous systems, which are mainly syntactic, HiLeX combines both semantic and syntactic knowledge for a powerful information extraction. Ontologies, representing the semantics of the domain of the information to be extracted, are encoded in DLP^+, while the extraction patterns are encoded by regular expressions in an ad hoc two-dimensional grammar. These regular expressions are (internally) translated into DLP^+ rules, whose execution yields the actual extraction of information from the input document. HiLeX allows the semantic information extraction from both HTML pages and flat text documents. The usefulness of Hilex has been already confirmed also in practice, as the system has been successfully employed in two advanced applications in the e-health and e-finance domains.

1 Introduction

HTML and flat text documents contain a huge amount of information arranged for human readers according to syntactic, semantic, and presentation rules of a given language. This information tends to be practically unusable, both for their vastness, and for the lack of machine readability that makes existing information extraction systems unable to manage the actual knowledge that the information conveys.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 506–510, 2006.

Recognizing and extracting relevant information automatically from web documents, according to their semantics, is an important problem in the field of web information extraction.

In the recent literature a number of approaches for information extraction from unstructured documents have been proposed. An overview of the large body of existing literature and systems is given in [2, 4, 5]. The currently available systems are mainly syntactic, and are not aware of the semantics of the information they are able to extract. They principally use pattern matching mechanism exploiting the underlying HTML syntactical structure and regular expressions on textual fragments contained between HTML tags.

In this work we present HiLeX, a logic-based system which combines both syntactic and semantic knowledge for a powerful information extraction from unstructured documents. Logic-based approaches for information extraction are not new [1], however, the approach we propose is original. Its novelty is due to:

• The logic two-dimensional representation of an unstructured document. A document is viewed as a Cartesian plane composed by a set of nested rectangular regions called *portions*. Each portion, univocally identified through the cartesian coordinates of two opposite vertices, contains a piece of the input document (*element*) annotated w.r.t. an ontology instance.

• The exploitation of a logic-based knowledge representation language called DLP^+, extending DLP with object-oriented features, including classes, (multiple) inheritance, complex objects, types. $DLP^+$is well-suited for representation and powerful reasoning on ontologies; the language is supported by the DLV^+ system [7], implemented on top of DLV [6].

• The use of ontologies, encoded in DLP^+, describing the domain of the input document. A concept of the domain is represented by a DLP^+ class; each class instance is a *pattern* representing a possible way of writing the concept and is used to recognize and annotate an element contained in a portion.

• The employment of a new grammar, named HiLeX two-dimensional grammar, for specifying the (above mentioned) patterns. HiLeX grammar extends regular expressions for the representation of two-dimensional patterns (like tables, item lists, etc.), which often occur in web pages and textual tabular data. The patterns are specified through DLP^+ rules, whose execution yields the *semantic information extraction*, by associating (the part of the document embraced by) each portion to an element of the domain ontology.

It is worthwhile noting that, besides the domain ontologies, HiLeX system uses also a *core ontology*, containing (patterns for the recognition of) general linguistic elements (such as date, time, numbers, email, words, etc.); presentation elements (such as font colors, font styles, background colors, etc.); structural elements (such as table cells, item lists, paragraphs, etc.) which are not bound to a specific domain but occur generally.

The advantages of the HiLeX system over other existing approaches are mainly the following:

– The extraction of information according to their semantics and not only on the basis of their syntactic structure.

- The possibility to extract information in the same way from documents in different formats. The same extraction pattern can be used to extract data from both flat text and HTML documents. This feature is not due to a preliminary HTML-to-text translation, but it comes from the higher abstraction obtained from the transformation of the input document in a set of logical portions.
- The possibility to obtain a "semantic" classification of the input documents w.r.t. an ontology, which is much more accurate and meaningful than the syntactic classifications provided by existing systems (mainly based on counting the number of occurrences of some keywords). This feature opens the door to many relevant applications (e.g., email classification and filtering, skill classification from curricula, extraction of relevant information from medical records, etc.).

Distinctive features of the HiLeX system, summarized above, allow for a better digital contents management and fruition in different application fields such as: e-health, e-entertainment, e-commerce, e-government, e-business.

2 The HiLeX System

The architecture of the HiLeX system is depicted in Figure 1. The semantic information extraction approach can be viewed as a process composed of four main steps: knowledge representation, document preprocessing, pattern matching, and pattern extraction. The following subsections illustrate these steps.

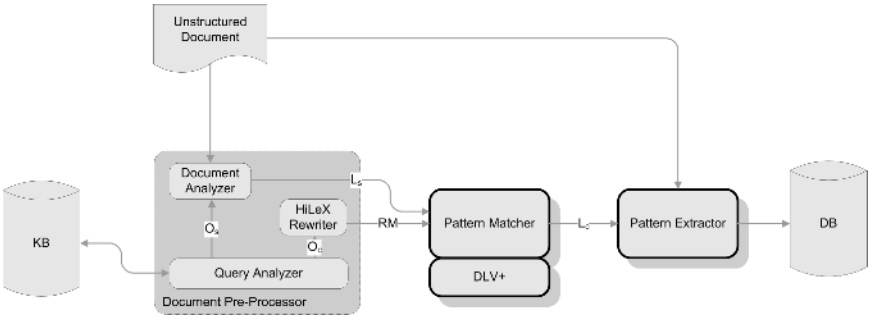


Fig. 1. The architecture of the HiLeX System

2.1 Knowledge Base

The Knowledge Base (KB) stores, using the DLV^+ system persistency layer, the core and the domain ontologies, the patterns encoding information to be extracted and the logic two-dimensional representation of unstructured documents. The KB provides an API supporting ontology querying, and pattern representation and matching.

2.2 Document Preprocessor

The document preprocessor takes as input an unstructured document, and a query containing the class-instance names representing the information that the user wishes to extract. After the execution the document preprocessor returns the logic two-dimensional document representation and a set of DLP^+ rules constituting the input for the pattern recognizer. The document preprocessing is performed by the three sub-modules described in the following.

The Query analyzer takes the user query as its input and explores the ontologies in order to identify patterns for the extraction process. The output of the query analyzer are two sets of pairs (*class-instance name*, *pattern*). The first set (O_s) contains pairs in which instances are characterized by patterns represented by regular expressions (simple elements), whereas in the second set (O_c) patterns are expressed using the HtLεX pattern representation grammar (complex elements). The set O_s is the input for the document analyzer and the set O_c is the input for the rewriter module.

The Document Analyzer gets an unstructured document and the set of pairs O_s. The document analyzer is able to recognize regular expressions, applying pattern matching mechanisms, to detect simple elements constituting the document and for each of them generates the relative *portion*. At the end of the analysis this module provides the *logic document representation* L_s which is a uniform abstract view of different document formats.

The HtLεX Rewriter takes in input the set of pairs O_c containing the extraction patterns expressed by means of the HtLεX grammar. Each pattern is translated in a set of logical DLP^+ rules. Their execution in the DLV^+ system, performed by the pattern matcher module, yields the actual semantic information extraction from unstructured documents.

2.3 Pattern Matcher

The pattern matcher is founded on the DLV^+ system. It takes as input the logic two-dimensional document representation (L_s) and the set of DLP^+ rules, resulting from the translation of the HtLεX patterns, which allow to recognize new complex elements. The output of this step is the *augmented logic two-dimensional representation* (L_c) of an unstructured document in which new document regions, containing more complex elements (e.g. tables having a certain structure and containing certain concepts, phrases having a particular meaning, etc.), are identified.

2.4 Pattern Extractor

This module takes as its input the augmented logic representation of a document (L_c) and allows for the acquisition of element instances (semantic wrapping) and/or the document classification w.r.t. the classes of the ontology.

Acquired instances can be stored in a DLP^+ ontology, in a relational database, or in an XML database. So, extracted information can be used in other applications, and more powerful query and reasoning tasks can be performed. The

extraction process causes the annotation of the documents w.r.t. the concepts in the ontology. This feature can enable, for example, semantic classification in document management contexts.

3 Current Applications

Currently two vertical applications of the system have been developed. The first application concerns the extraction of patients, diseases, therapies and drugs information from electronic medical records in unstructured format, and their storage in a structured XML format. This e-health application aids doctors to better analyze clinical information and hospitals to exchange patient and disease data.

The second application concerns the extraction of information (sub-item and the related amount in Euro) of different items contained in an unstructured section of balance-sheets. Extracted information is stored into a relational database and made available for further financial analysis.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Declarative information extraction, web crawling, and recursive wrapping with lixto. *Proc. LPNMR '01*, 2001, pp. 21–41.
2. L. Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.
3. D. Giammarresi and A. Restivo. Two-dimensional languages. *Handbook of Formal Languages*, vol. 3, Beyond Words, pages 215–267. Springer-Verlag, Berlin, 1997.
4. S. Kuhlins and R. Tredwell. Toolkits for generating wrappers – a survey of software toolkits for automated data extraction from web sites. *Proc. NODe 2002*, Germany, 2002.
5. A. Laender, B. Ribeiro-Neto, A. Silva, and J. Teixeira. A brief survey of web data extraction tools. In *SIGMOD Record*, volume 31, June 2002.
6. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV System for Knowledge Representation and Reasoning. ACM TOCL 2006 (Forthcoming).
7. F. Ricca, N. Leone, Disjunctive Logic Programming with types and objects: The DLV$^+$ System *Journal of Applied Logic*, Elsevier, 2006 (Forthcoming).

`tarfa`: Tableaux and Resolution for Finite Abduction

Fernando Soler-Toscano and Ángel Nepomuceno-Fernández

Dept. of Philosophy and Logic. University of Seville, Spain
{fsoler, nepomuce}@us.es

Abstract. n-tableaux [1] and δ-resolution [2], which are based, respectively, on semantic tableaux and resolution, have been properly used for the resolution of abductive problems. The tool we present is a Prolog implementation of an abductive solver which combines both calculi to attack first order abductive problems by reducing them to finite versions, that is, propositional rewritings of the problems which presuppose a context representable with finite models with a known cardinality.

1 Introduction

Abductive problems appear in many contexts, in which a way for reasoning with incomplete information is at call. Informally, given a *theory* Θ and an *observation* ϕ not entailed by Θ, *abduction* is the *searching* for an *explanation* α such that Θ and α together entail ϕ. When tackling first order abduction, the undecidability problem appears. There are two (not mutually exclusive) possible strategies if one wants to avoid undecidability. One is to restrict the allowed syntactical form of the formulae involved in abductive problems. On the other hand, as abduction is somehow related to the search for models, one can restrict the cardinality of those models. The tool we present follows this second option.

Our aim is to combine n-tableaux and δ-resolution calculi, which have been previously used for abduction in [1] and [2], respectively. The result is a more efficient implementation that those which use only one of the two mentioned calculi. Anyway, the program is not as efficient as other existing tools which work with restricted syntactical forms on the abductive problems. The interest of our tool may be seen in its generality, since it does not impose any of those habitual restrictions.

2 Formal Apparatus

We take $\models$ as the classical consequence relation, and $\mathcal{L}$ a first order language, without equality nor function symbols, with habitual connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$ and quantifiers $\forall$, $\exists$. Given a finite set $\Theta \subset \mathcal{L}$ and $\phi \in \mathcal{L}$, we say $\langle \Theta, \phi \rangle$ is an *abductive problem* iff $\Theta \not\models \phi$ and $\Theta \not\models \neg\phi$. Then, the set of literals $\Sigma \subset \mathcal{L}$ is an *abductive solution* of $\langle \Theta, \phi \rangle$ iff $\Theta \cup \Sigma \models \phi$, $\Theta \cup \Sigma \not\models \bot$, $\Sigma \not\models \phi$ and for every $\Sigma' \subset \Sigma$, $\Theta \cup \Sigma' \not\models \phi$. We denote by $\mathcal{A}bd(\Theta, \phi)$ the *set of abductive solutions* of an abductive problem $\langle \Theta, \phi \rangle$.

M. Fisher et al. (Eds.): JELIA 2006, LNAI 4160, pp. 511–514, 2006.

2.1 *n*-Tableaux

n-tableaux are a modification of classical semantic tableaux, which have been successfully used by [1, 3] to attack abductive problems that are intractable by the classical tableaux-based approach to abduction [4], because their theories are first order formulae which produce infinite branches in tableaux. But, in many applications of abduction, it makes sense to reduce such problems to propositional versions which are equivalent to the adaptation of those problems to first order finite domains. The solutions to the reduced versions, thought in general not valid in the original first order versions, may be useful in the concrete (finite) context of the problem.

Given a finite set $\Gamma \subset \mathcal{L}$ of formulae, its n-tableau, for a given natural number n, is constructed with classical rules for closing branches, double negation, α and β rules, plus the following versions of γ and δ rules:

$$\frac{\forall x\varphi}{\begin{array}{c}\varphi(x/c_1)\\ \vdots \\ \varphi(x/c_n)\end{array}} \qquad \frac{\neg\exists x\varphi}{\begin{array}{c}\neg\varphi(x/c_1)\\ \vdots \\ \neg\varphi(x/c_n)\end{array}}$$

$$\frac{\exists x\varphi}{\varphi(x/c_1)|\ldots|\varphi(x/c_n)} \qquad \frac{\neg\forall x\varphi}{\neg\varphi(x/c_1)|\ldots|\neg\varphi(x/c_n)}$$

the $c_1, \ldots, c_n$ are n constants of the language, which are the same for all the construction of the n-tableau and include all the constants of the original set Γ.

At the end of the construction of the n-tableau of Γ if there are m open branches, each one with the set of literals Θ_i, $i \leq m$, the formula $\bigvee_{i=1}^{m} \bigwedge_{\lambda\in\Theta_i} \lambda$, which we call $\mathcal{C}_n(\Gamma)$, is equivalent to Γ in the set of finite models with a universe of cardinality n and an interpretation function $\mathcal{I}$ such that $\mathcal{I}(c_i) \neq \mathcal{I}(c_j)$ for $1 \leq i < j \leq n$.

2.2 δ-Resolution

The δ-resolution calculus is used in [2] to produce abductive solutions with some advantages with respect to the approaches to abduction based on standard resolution or semantic tableaux. In the tool we are describing, we use δ-resolution once first order formulae have been reduced to finite-model versions, equivalent to propositional formulae. So, we will present some elements of the propositional version of the δ-resolution calculus. More details can be found in [2].

Definitions and properties of the δ-resolution calculus are dual to those of the standard resolution. First, a *δ-clause* is a set of literals, and a *δ-clausal form* a set of δ-clauses. Given a boolean valuation v, it satisfies a given δ-clause iff it satisfies *all* its literals, and v satisfies a δ-clausal form iff it satisfies *at least one* of its δ-clauses. So, the empty δ-clause, $\Diamond$, is universally valid, while the empty δ-clausal form is not satisfiable. With these semantical notions, we can

extend the relation $\models$ to δ-clauses and δ-clausal forms. Also, we can transform any formula to an equivalent δ-clausal form, by constructing its DNF (disjunctive normal form). Given $\alpha = \bigvee_{i=1}^{n} \bigwedge_{j=i}^{m} \lambda_i^j$ in DNF, we denote as $\mathcal{S}(\alpha)$ the equivalent δ-clausal form $\bigcup_{i=1}^{n} \{\Sigma_i \mid \Sigma_i = \bigcup_{j=1}^{n} \lambda_i^j\}$.

We say that a δ-clause Σ *is provable by δ-resolution* from the δ-clausal form A, represented by $A \vdash_\delta \Sigma$, if $\Sigma \in A$ or it can be obtained from only applications of the *δ-resolution rule* (which from δ-clauses $\Sigma_1 \cup \{\lambda\}$ and $\Sigma_2 \cup \{\neg\lambda\}$ produces their δ-resolvent $\Sigma_1 \cup \Sigma_2$) to other δ-clauses probable from A. Given a δ-clausal form A, we denote by A^δ the *saturation by δ-resolution* of A, that is, the minimal set that for every satisfiable Σ, if $A \vdash_\delta \Sigma$, then there is a $\Sigma' \subseteq \Sigma$ such that $\Sigma' \in A^\delta$. This set can be algorithmically constructed.

2.3 Abductive Search

To solve abductive problems $\langle \{\theta_1, \ldots, \theta_n\}, \varphi \rangle$ with n-tableaux and δ-resolution, we proceed in the following way. First, we choose a cardinality n for the n-tableaux (so also a set of n constants that will be used in all of them), and then:

Step 1: Theory Analysis. Let N_Θ be $\mathcal{S}(\mathcal{C}_n(\{\neg(\theta_1 \wedge \ldots \wedge \theta_n)\}))$. If $N_\Theta = \emptyset$, then Θ is universally valid[1] and the process stops. Else, N_Θ^δ is obtained. If $\Diamond \in N_\Theta^\delta$, then Θ is not satisfiable. Else,

Step 2: Observation Analysis. Let O be $\mathcal{S}(\mathcal{C}_n(\{\varphi\}))$. If $O = \emptyset$, then φ is not satisfiable, and the process stops. Else, O^δ is obtained. If $\Diamond \in O^\delta$, then φ is universally valid. Else,

Step 3: Refutation Search. If for any δ-clause $\Sigma \in O^\delta$ there is a $\Sigma' \in N_\Theta^\delta$ such that $\Sigma' \subseteq \Sigma$, then $\Theta \models \neg\varphi$ and the process stops. Else,

Step 4: Explanations Search. From N_Θ^δ and O^δ, the sets $(N_\Theta^\delta \cup O^\delta)$ and $(N_\Theta^\delta \cup O^\delta)^\delta$ are obtained. If $\Diamond \in (N_\Theta^\delta \cup O^\delta)^\delta$, then $\Theta \models \varphi$, so the process stops (there is no abductive problem). In other case, $\langle \Theta, \varphi \rangle$ is an abductive problem, and the process returns $(N_\Theta^\delta \cup O^\delta)^\delta - (N_\Theta^\delta \cup O^\delta)$ which is $\mathcal{A}bd(\Theta, \varphi)$.

3 Description of tarfa

The system tarfa is implemented in SWI-Prolog, and runs in the latest version 5.6.12. It uses PrologScript to work as a Unix script file (but it can be adapted to work in other platforms). The declarative character of Prolog gives us a natural way to write the operations involved in the abductive search. SWI-Prolog has been chosen because it covers a great part of the Edinburgh Prolog standard, and it has a good compatibility with other compilers. Also, it works on a wide range of 32 and 64 bit platforms and offers a flexible interface to the C and C++. Last, but not least, it is licensed under the LGPL.

The program is called from the command line according to the following syntax:

[1] When we say from now that a formula is *universally valid* or *not satisfiable*, we refer to the n-cardinality (propositional) version of the formula.

- `tarfa -n <depth> <textfile>`
- `tarfa <textfile>`

In both cases, `<textfile>` is a file which specifies the abductive problem in its first order version, previous to the finite models (propositional) reduction. Then, in the first case, the system reduces the problem to the case of models with cardinality equal to `<depth>` and attempts to solve it. If no cardinality is given, that is, when the system is called in the second way, it starts an iterative search, starting from cardinality 1, and increasing it each time in one unity until it finds a cardinality such that the original problem remains an abductive problem when translated to it. Then, it attempts to solve it.

With regard to the syntax of the text files, these comprise a non empty set of instances of the Prolog unary predicate `theory/1` and one instance of `fact/1`, all of them containing formulae. In relation to the syntax of the formulae, the connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$ and $\leftrightarrow$ are represented, respectively, by `&`, `v`, `-`, `=>` and `<=>`. For quantifiers, `all(<V>,<F>)` and `ex(<V>,<F>)` stand, respectively, for the universal and existential quantification of the variable `<V>` over the formula `<F>`. It should be taken into account that variables must be Prolog variables (starting with a capital letter) and predicates must start with a small letter. When two formulae contain the same variable, `tarfa` interprets it as one different variable in each case.

This is an example of a simple input file, which codifies the abductive problem $\langle \Theta, \phi \rangle$ when $\Theta = \{\forall x(Px \rightarrow Qx), \forall x(\neg Qx \vee Rx)\}$ and $\phi = \{Qa \wedge Rb\}$:

```
theory(all(X,p(X) => q(X))).
theory(all(X,-q(X) v r(X))).
fact(q(a) & r(b)).
```

The main procedure of `tarfa` is the implementation of the abductive search explained in subsection 2.3. The construction of the n-tableaux is done by an adaptation of leanTAP [5]. The δ-resolution calculus is implemented in Prolog following some of the propositional strategies of the Otter theorem prover.

References

1. Reyes-Cabello, A.L., Aliseda-Llera, A., Nepomuceno-Fernández, A.: Abductive reasoning in first order logic. Logic Journal of the IGPL **14**(2) (2006) 287–304
2. Soler-Toscano, F., Nepomuceno-Fernández, A., Aliseda-Llera, A.: Model-based abduction via dual resolution. Logic Journal of the IGPL **14**(2) (2006) 305–319
3. Nepomuceno, A.: Scientific explanation and modified semantic tableaux. In Magnani, L., Nerssessian, N., Pizzi, C., eds.: Logical and Computational Aspects of Model-Based Reasoning. Applied Logic Series. Kluwer Academic Publishers (2002) 181–198
4. Cialdea Mayer, M., Pirri, F.: First order abduction via tableau and sequent calculi. Bulletin of the IGPL **1** (1993) 99–117
5. Beckert, B., Posegga, J.: leanTAP: Lean tableau-based deduction. Journal of Automated Reasoning **15**(3) (1995) 339–358

Author Index