Fabio Fioravanti (Ed.)

CILC 2011

26th Italian Conference on Computational Logic
Pescara, Italy, August 31 - September 2, 2011
Proceedings

Editor's address:
Fabio Fioravanti
University of Chieti-Pescara
Department of Sciences
Viale Pindaro 42
65127 Pescara, Italy

fioravanti@sci.unich.it

Preface

From August 31th to September 2nd 2011, the University of Chieti-Pescara hosted the Italian Conference on Computational Logic, CILC 2011. It was the 26th edition of the annual meeting organized by the Italian Association for Logic Programming (GULP, Gruppo Ricercatori e Utenti di Logic Programming), which, since its first edition in 1986, constitutes the main Italian forum for computational logic researchers, users and developers to discuss work and exchange ideas.

The program of CILC 2011 featured thirty technical contributions (21 long and 9 short presentations), two invited talks and a tutorial.

The invited talks were given by Andrea Omicini, who reviewed 25 years of coordination models and languages, and Fausto Spoto, who presented a method based on abstract interpretation for nullness analysis of a Java-like language.

The tutorial was given by Fabrizio Riguzzi, who presented and compared various probabilistic logic languages.

The technical presentations concerned a number of different topics related to computational logic including game theory, description logics, answer set programming as well as constraint, abductive, inductive and probabilistic extensions of logic programming. Some of them reported successful applications of computational logic techniques to real world problems.

A selection of the accepted papers will appear in a special issue of Fundamenta Informaticae.

The complete program with links to presentation slides is available at the following address `http://www.sci.unich.it/cilc2011`.

The quality of the technical contributions and the number of participants (about fifty, most of whom were young researchers) confirms that the Italian computational logic community is very lively and active.

I wish to thank all who contributed to the success of this edition including authors, speakers, reviewers, participants and organizers.

September 2011 Fabio Fioravanti

Organizing Committee

Marco Bottalico, Università di Chieti-Pescara, Italy
Paola Campli, Università di Chieti-Pescara, Italy
Emanuele De Angelis, Università di Chieti-Pescara, Italy
Fabio Fioravanti, Università di Chieti-Pescara, Italy
Daniele Magazzeni, Università di Chieti-Pescara, Italy
Maria Chiara Meo, Università di Chieti-Pescara, Italy
Francesca Scozzari, Università di Chieti-Pescara, Italy

Program Committee

Stefano Bistarelli, Università di Perugia, Italy
Francesco Buccafurri, Università "Mediterranea" di Reggio Calabria, Italy
Amedeo Cesta, Istituto di Scienze e Tecnologie della Cognizione - CNR, Italy
Stefania Costantini, Università dell'Aquila, Italy
Alessandro Dal Palù, Università degli Studi di Parma, Italy
Wolfgang Faber, Università della Calabria, Italy
Marco Faella, Università di Napoli "Federico II", Italy
Moreno Falaschi, Università di Siena, Italy
Fabio Fioravanti, Università di Chieti-Pescara, Italy
Camillo Fiorentini, Università di Milano, Italy
Rosella Gennari, Università di Bolzano, Italy
Laura Giordano, Università del Piemonte Orientale, Italy
Francesca A. Lisi, Università di Bari, Italy
Viviana Mascardi, Università di Genova, Italy
Isabella Mastroeni, Università di Verona, Italy
Maria Chiara Meo, Università di Chieti-Pescara, Italy
Marco Montali, Università di Bologna, Italy
Marianna Nicolosi Asmundo, Università di Catania, Italy
Mimmo Parente, Università di Salerno, Italy
Carla Piazza, Università di Udine, Italy
Gian Luca Pozzato, Università di Torino, Italy
Alessandro Provetti, Università di Messina, Italy
Fabrizio Riguzzi, Università di Ferrara, Italy
Riccardo Rosati, Università di Roma "La Sapienza", Italy
Sabina Rossi, Università di Venezia, Italy
Salvatore Ruggieri, Università di Pisa, Italy
Valerio Senni, Università di Roma "Tor Vergata", Italy
Luciano Serafini, Fondazione Bruno Kessler, Trento, Italy
K. Brent Venable, Università di Padova, Italy

Contents

LONG PRESENTATIONS

The following two papers were presented at the CILC 2011 conference but have not been included in this volume because they appear elsewhere.

The paper *Conformance Checking of Executed Clinical Guidelines in presence of Basic Medical Knowledge* by Alessio Bottrighi, Federico Chesani, Paola Mello, Marco Montali, Stefania Montani and Paolo Terenziani appears in the *Proceedings of the 4th International Workshop on Process-oriented Information Systems in Healthcare* (ProHealth 11), Clermont-Ferrand, France, August 29, 2011.

The paper *Integration of abductive reasoning and constraint optimization in SCIFF* by Marco Gavanelli, Marco Alberti and Evelina Lamma appears in the *Proceedings of the 25th International Conference on Logic Programming* (ICLP '09), Lecture Notes in Computer Science, 2009, Volume 5649/2009, 387-401.

SHORT PRESENTATIONS

On modal μ-calculus in $S5$ and applications

Giovanna D'Agostino[1] and Giacomo Lenzi[2]

[1] University of Udine, Italy
[2] University of Salerno, Italy gilenzi@unisa.it

Abstract. We show that the vectorial μ-calculus model checking problem over arbitrary graphs reduces to the vectorial, existential μ-calculus model checking problem over $S5$ graphs. We also draw some consequences of this fact. Moreover, we give a proof that satisfiability of μ-calculus in $S5$ is NP-complete, and by using $S5$ graphs we give a new proof that the satisfiability problem of the existential μ-calculus is also NP-complete.

1 Introduction

Model checking is a technique widely used in verification of computer systems, be they hardware or software, see [4]. In model checking, systems are modeled as sets with one or more binary relation (in this paper we focus on systems with one relation, i.e. graphs). The desirable properties a system should have are formalized in some modal-like logic. Actually, modal logic itself is not expressive enough. For this reason, one considers more powerful formalisms. One of them is *modal μ-calculus*, introduced in [15], an extension of modal logic with least and greatest fixpoints of monotonic set-theoretic functions. Intuitively, least fixpoints correspond to inductive definitions, and greatest fixpoints correspond to coinductive definitions. Unlike plain modal logic, the μ-calculus is powerful enough to express global properties of systems, i.e. properties which depend on the whole possible history of the system. For instance, with greatest fixpoints we can capture safety properties such as "the system will never crash", whereas with least fixpoints we can capture termination properties such as "every computation of the system will terminate". More complicate properties, e.g. fairness, can be used by combining least and greatest fixpoints.

The model checking technique raises a natural computational question, which is known as the (μ-calculus) model checking problem. Formally, the μ-calculus model checking problem is: given a μ-calculus formula and a finite graph, check whether the graph satisfies the formula. Because of the importance of model checking in practice, it would be desirable to have an efficient, i.e. polynomial time computable, model checking algorithm for arbitrary (finite) graphs, but this algorithm has not been found. We know that the problem is in the complexity class UP, see [14] (and a $co-UP$ bound follows since the μ-calculus is closed under negation). Recall that the class UP (Unique P) contains the problems solved in polynomial time by nondeterministic Turing machines which have at most one accepting path on each input. So, the class UP lies somewhere between P and NP (in particular, the model checking problem is in NP). We will see

that the μ-calculus is tightly related to games, in particular parity games, and in fact a promising approach to the model checking problem is the study of various kinds of games. It must be said, however, that efficient model checking algorithms exist when the number of alternating fixpoints is bounded, and this is often the case in practice.

The other main theme of this paper is given by $S5$ graphs, i.e. graphs whose relation is an equivalence.

The modal logic of $S5$ graphs (also called modal logic $S5$) is important because it is widely recognized as a good epistemic logic, where the box operator $[\,]\phi$ means that some agent knows ϕ. When modal logic is interpreted on Kripke structures, i.e. graphs, the vertices of the structure represent possible situations, and it is reasonable that the knowledge of an agent is represented by an equivalence relation on the vertices, which indicates that certain situations are not distinguishable, in the agent's knowledge.

So, $S5$ is a way of formalizing the ideas of knowledge, and it is used in many applications such as artificial intelligence, etc. Often multimodal versions of $S5$ are considered, where different agents come into play; in this paper, however, we will focus on a single modality, representing a single agent.

We will consider also the class of all transitive graphs, called $K4$ in the modal logic literature. Many interesting relations are transitive: for instance, the relation "the event A is posterior to the event B" defines a transitive relation between events. In this paper $K4$ graphs play only a minor role; papers dedicated to the μ-calculus in $K4$ are [2], [5] and [6].

In this paper we compare the behavior of the μ-calculus on arbitrary graphs and on $S5$ graphs. It is well known that the μ-calculus is expressively equivalent to modal logic over $S5$, but this equivalence does not transfer automatically to an equivalence in complexity, neither for model checking, nor for satisfiability.

From this perspective we first show that the μ-calculus model checking problem for arbitrary graphs is as difficult as the subcase of $S5$ graphs, although the class of $S5$ graphs is significantly simpler than the class of all graphs.

Then we move to the satisfiability problem. Quite generally, recall that the satisfiability problem for a logic L on a class of models C is: given a formula ϕ in L, decide whether there is a model of ϕ which is in C.

The satisfiability problem of the μ-calculus on arbitrary graphs is settled, in the sense that it is $EXPTIME$-complete: $EXPTIME$-hardness of the problem follows from [12], and membership to $EXPTIME$ is proved in [8]. We note that $S5$ has also an application to the satisfiability problem of fragments of the μ-calculus: the satisfiability problem of the so-called existential (or box-free) μ-calculus on arbitrary graphs is as difficult as the same problem on $S5$ graphs. By using this observation we give an alternative proof of a result of [13] to the effect that the satisfiability problem for the existential μ-calculus is NP-complete. We also give a proof that satisfiability of μ-calculus in $S5$ is NP-complete, so in this respect we have a better complexity than the $EXPTIME$ complexity the full μ-calculus. Both results depend on a linear size model property for μ-calculus formulas in $S5$.

1.1 Related work

Given the relevance of $S5$ as epistemic logic, many papers in the modal logic literature are dedicated to it, and in particular on its proof theory. Finding a good axiom system for $S5$ is a longstanding open problem, see [19]. The situation is even more difficult for the modal μ-calculus in $S5$, where a recent contribution is [1].

2 Syntax

2.1 Scalar modal μ-calculus

We present here the usual modal μ-calculus, and we call it scalar because, as we will see, there is also a vectorial version of the μ-calculus. We follow the standard presentation of the formulas of modal μ-calculus:

$$\phi = A \mid \neg A \mid X \mid \phi \vee \phi \mid \phi \wedge \phi \mid \langle\ \rangle \phi \mid [\ ]\phi \mid \mu X.\phi \mid \nu X.\phi,$$

where A ranges over a set At of atoms and X ranges over a set Var of fixpoint variables. $\langle\ \rangle$ and $[\ ]$ denote the modal operators: the diamond, or the existential operator, and the box, or the universal operator.

Intuitively, $\mu X.\phi(X)$ denotes the least fixpoint of the function $\phi(X)$, and $\nu X.\phi(X)$ denotes the greatest fixpoint of this function.

A μ-calculus formula ϕ is called guarded if for every fixpoint subformula of ϕ, say $\nu X.\psi(X)$ or $\mu X.\psi(X)$, every occurrence of X in ψ is in the scope of a modal operator.

Free and bound variables are defined in analogy with first order logic, because fixpoints μX and νX are syntactically analogous to quantifiers $\exists x$ and $\forall x$ (note however that semantically, fixpoint variables correspond to monadic second order variables, i.e. variables ranging over sets, rarther than first order variables ranging over individuals).

A μ-calculus formula is called a sentence if it has no free variables. Although formulas are not closed under negation, a negation of sentences is available: the negation of a sentence is obtained by exchanging A and $\neg A$, $\wedge$ and $\vee$, $\langle\ \rangle$ and $[\ ]$, and μ with ν.

Given a formula ϕ, we denote by $|\phi|$ the size of ϕ.

2.2 Functional μ-calculus

We can generalize modal μ-calculus to functional μ-calculus, following [3]. Functional μ-calculus has n-ary function symbols, to be interpreted by monotonic functions on powersets (or more generally, on complete lattices). The syntax is

$$X|\phi \wedge \psi|\phi \vee \psi|f(\phi_1, \ldots, \phi_n)|\mu X.\phi|\nu X.\phi.$$

2.3 Vectorial μ-calculus

The most standard presentation of modal μ-calculus is in the scalar syntax of the previous section. In this section we generalize the syntax by allowing systems of equations: although this extension does not affect the expressiveness of the logic, it may increase succinctness.

We essentially follow the presentation of [3]. We restrict to powersets rather than arbitrary complete lattices. So we can consider a set V and n monotonic functions $f_1, \ldots, f_n$ from $P(V)^{n+m}$ to $P(V)$. A μ-system is a system S of n equations

$$S : \begin{cases} x_1 =_{\theta_1} f_1(x_1, \ldots, x_n, y_1, \ldots, y_m) \\ \ldots \\ x_n =_{\theta_n} f_n(x_1, \ldots, x_n, y_1, \ldots, y_m) \end{cases}$$

where $\theta_1, \ldots, \theta_n \in \{\mu, \nu\}$.

The μ-system S is by definition equivalent to a n tuple of scalar μ-calculus formulas, called the solution of S, computed inductively as follows.

If $n = 1$ then the solution is $\theta x_1.f_1(x_1, y_1, \ldots, y_m)$.

If $n > 1$, let $g_1(x_2, \ldots, x_n, y_1, \ldots, y_m) = \theta_1 x_1.f_1(x_1, \ldots, x_n, y_1, \ldots, y_m)$. The solution of S is $(g_1(h_2, \ldots, h_n, y_1, \ldots, y_m), h_2, \ldots, h_n)$, where $(h_2, \ldots, h_n)$ is the solution of the system

$$S_1 : \begin{cases} x_2 =_{\theta_2} f_2(g_1(x_2, \ldots, x_n, y_1, \ldots, y_m), \ldots, x_n, y_1, \ldots, y_m) \\ \ldots \\ x_n =_{\theta_n} f_n(g_1(x_2, \ldots, x_n, y_1, \ldots, y_m), \ldots, x_n, y_1, \ldots, y_m) \end{cases}$$

We denote by $sol_i(S)$ the i-th component of the solution of S.

A μ-system of equations is called a modal μ-system if all functions f_i are combinations of variables, atoms, negated atoms, conjunctions, disjunctions, diamonds, and boxes.

The modal μ-calculus vectorial model checking problem is: given a finite graph G and a modal μ-system S, decide whether G satisfies $sol_1(S)$.

2.4 The *LEFT* relation

Given a μ-system S, we define a relation $LEFT$ between the variables of S as follows.

Let y, z two variables of S. We say that y is at left of z, written $y \; LEFT \; z$, if there is an equation of S where y is the variable at the left hand side of the equality and z occurs at the right hand side.

A modal μ-system is called a modal system if the $LEFT$ relation on variables is acyclic. Every modal system is equivalent to a formula of modal logic.

2.5 Composition

Let $\phi(X)$ be a formula containing a free variable X and let ψ be a sentence. Then the composition $\phi[X/\psi]$ is the formula obtained by replacing X everywhere with ψ in ϕ. Note that ψ is a sentence, hence there is no variable capturing.

The usual notion of composition of formulas can be extended to μ-systems as follows.

Let S be a μ-system. The scope of a left variable y in S is the set of all variables z such that there is a $LEFT$ path from y to z.

Let S, T be two systems where the variables at left of S and T are disjoint. Let A be an atom of S. Then the composition of S and T is the system obtained by concatenating the equations of S and of T and by replacing A with the left variable of the first equation of T. Composition is possible only without capture, i.e. T must not have free variables y such that some occurrence of A is in the scope of y in S.

2.6 Vectorial alternation depth hierarchy

We define the vectorial hierarchies $VEC-\Sigma_n$, $VEC-\Pi_n$, $VEC-\Delta_n$ as follows. $VEC-\Sigma_0 = VEC-\Pi_0$ are the modal systems. $VEC-\Sigma_{n+1}$ is the closure of $VEC-\Pi_n$ under composition and adding a μ equation as a first equation of the system. $VEC-\Pi_{n+1}$ is the closure of $VEC-\Sigma_n$ under composition and adding a ν equation as a first equation of the system. $VEC-\Delta_n = VEC-\Sigma_n \cap VEC-\Pi_n$. The alternation depth of a system S is the least n such that S is in $VEC-\Delta_{n+1}$.

3 Semantics and related concepts

3.1 Graphs and models

Like it is usually done for modal logic, we give Kripke semantics to the μ-calculus by using the notion of model.

A graph (also called frame) is a pair $G = (V, E)$, where V is a set of vertices and E is a binary edge relation on V.

A graph $G = (V, E)$ is called total if $E = V^2$, i.e., all possible edges are present.

A path in a graph G from a vertex x to a vertex y is a finite sequence of vertices $z_1, \ldots, z_n$ such that $z_1 = x$, $z_n = y$ and $z_i E z_{i+1}$ for every $i < n$.

A point y is reachable from a point x if there is a path from x to y.

A model is a pair (G, Col), where G is a graph and Col is a coloring function from some domain D to the powerset of V.

3.2 Bisimulation

Intuitively, bisimulation between models indicates that the two models have the same observable behavior. Formally, we can define a bisimulation between two models (G, Col) and (G', Col') as a relation $B \subseteq V(G) \times V(G')$ such that, whenever (xBx'):

- $x \in Col(d)$ if and only if $x' \in Col'(d)$ for every $d \in D$;
- if xEy, then there is y' such that $x'E'y'$ and yBy';
- if $x'E'y'$, then there is y such that xEy and yBy'.

We say that two pointed colored graphs (G, Col, x) and (G', Col', x') are bisimilar if and only if there is a bisimulation B such that xBx'.

3.3 Special classes of graphs

We consider a few subclasses of graphs.

A graph (V, E) is called total if $E = V^2$, that is, all possible edges are present.

The class $K4$ is the class of all (vertex colored) graphs whose relation is transitive. The class $S5$ is the class of all (vertex colored) graphs whose relation is an equivalence relation. Since equivalence relations are reflexive, symmetric and transitive, $S5$ is included in $K4$ (as a class of graphs). The names $K4$ and $S5$ come from the modal logic literature.

We also speak of total models, $K4$ models and $S5$ models in the obvious sense.

Note that every total graph belongs to $S5$. Moreover, for every $S5$ model M and every vertex x of M, there is a total model M' containing x such that (M, x) and (M', x) are bisimilar: in fact M' is the submodel of M given by all points of M reachable from x.

Moreover, every $S5$ model M is bisimilar to a $S5$ graph M', colored in the same way, where every two different points have different colors: in fact, M' is M modulo the equivalence relation of having the same color, and the bisimulation is the projection function.

3.4 Semantics

The semantics of μ-calculus extends the usual Kripke semantics for modal logic. So, to give semantics to the μ-calculus, we must consider models of the form $M = (G, Val)$ where $G = (V, E)$ is a graph and Val is a valuation function from $At \cup Var$ to the powerset of V. To each model M and each formula ϕ we can associate a subset $||\phi||M$ of V, defined in this way:

- $||A||M = Val(A)$ and $||\neg A||M = V \setminus Val(A)$ if A is an atom;
- $||X||M = Val(X)$;
- $||\phi \vee \psi||M = ||\phi||M \cup ||\psi||M$;
- $||\phi \wedge \psi||M = ||\phi||M \cap ||\psi||M$;
- $||\langle\ \rangle\phi||M$ is the set of all elements of V having some successor in $||\phi||M$;
- $||[\ ]\phi||M$ is the set of all elements of V having every successor in $||\phi||M$;
- $||\mu X.\phi(X)||M$ is the smallest set $S \subseteq V$ such that $S = ||\phi||M[X := S]$, where $M[X := S]$ is obtained from M by letting $Val(X) = S$;
- $||\nu X.\phi(X)||M$ is the greatest set $S \subseteq V$ such that $S = ||\phi||M[X := S]$.

The last two items are well defined since the map sending S to $||\phi||M[X := S]$ is a monotonic function on the powerset of V and so, by the Knaster-Tarski Theorem, this map has both a least and a greatest fixpoint.

We also say that a vertex v of a model M verifies a formula ϕ, written $M, v \models \phi$, if $v \in ||\phi||M$.

4 Parity games

4.1 Definition

It is notoriously difficult to understand μ-calculus formulas, especially when there are many alternating fixpoints. A means to understand the μ-calculus is given by parity games. We will see that the semantics of μ-calculus formulas can be given in terms of parity games.

Intuitively, a parity game is a game where two players, called c and d, move in a graph (the notation, due to Arnold, suggests that c means conjunctive and d means disjunctive). The vertexes of the graph are labeled with finitely many positive integers. d wants to have many high even numbers along the play, whereas c wants to have many odd numbers.

Let us define parity games more formally. A parity game is a structure $\Gamma = (V_c, V_d, E, v_0, \Omega)$, where V_c and V_d are disjoint sets, E is a binary relation on $V_c \cup V_d$, $v_0 \in V_c \cup V_d$ is the initial vertex, and $\Omega : V_c \cup V_d \to \{1, \ldots, n\}$ is the priority function; the number n is called the index of the game.

A play of Γ is a sequence of vertices, starting from v_0, where the successor of the current vertex must be an E-successor of that vertex, and this successor is chosen by the player d, if the vertex is in V_d, and by c if the vertex is in V_c.

If the play reaches a position where either player has no moves, the other wins. If this never happens, then the play is infinite, and d wins if the greatest priority occurring infinitely often is even, and c wins otherwise.

A strategy of a player p is a function which, given an initial segment of a play ending with a p- position, determines the next move of p. A strategy is positional if the move depends only on the last vertex of the segment.

A strategy Σ of a player p is winning if every play where p moves according to Σ is won by p.

Note that parity games are Borel games, so by Borel determinacy, see [17], they enjoy determinacy: there is always one of the two players who has a winning strategy.

A well known property of parity games is positional determinacy:

Theorem 1. *(See [9], Theorem 4.4) If either player has a winning strategy in a parity game, then it has a positional winning strategy.*

However in this paper we will use a slightly stronger form of determinacy. Let us call a strategy *strongly positional* if the move on a position depends only on the successors of the position (not on the position itself). Clearly, a strongly positional strategy is positional, but the converse does not hold: if two different

nodes have the same successors, a stronlgy positional strategy gives the same answer on the two nodes, whereas a positional one need not to.

Now a careful analysis of [9] gives the following strenghtening of the previous theorem:

Corollary 1. *If either player has a winning strategy in a parity game, then it has a strongly positional winning strategy.*

Proof. Note that players c and d are called AND and OR in [9]. The successor of an OR position in the positional strategy of Theorem 4.4 of [9] is chosen in a way which does not quite depend on the OR position, but depends only on the set of positions reachable in one step from that OR position. So, the resulting strategy is actually strongly positional. □

4.2 From the μ-calculus to parity games

The semantics of the μ-calculus can be given in terms of a parity game. More precisely, given a sentence ϕ and a model $M = (V, E, Val)$, with a distinguished vertex v_0 of V, we can define an evaluation game $\Gamma(\phi, v_0, M)$ as follows (we consider sentences rather than arbitrary formulas for simplicity).

The positions of $\Gamma(\phi, v_0, M)$ are the pairs (ψ, v), where $v \in V$ and ψ is a subformula of ϕ. The d positions are the pairs of the form $(\psi \vee \chi, v)$ or $(\langle\,\rangle\psi, v)$; all other positions are c positions. (ϕ, v_0) is the initial position.

There are edges from $(\psi \vee \chi, v)$ or $(\psi \wedge \chi, v)$ to (ψ, v) and (χ, v); from $(\langle\,\rangle\psi, v)$ or $([\,]\psi, v)$ to (ψ, w) for every successor w of v in V; from $(\mu X.\psi, v)$ and $(\nu X.\psi, v)$ to (ψ, v); and if a variable occurrence X appears in a subformula $\mu X.\psi$ or $\nu X.\psi$, there is an edge from (X, v) to $(\mu X.\psi, v)$ or $(\nu X.\psi, v)$ respectively. Finally, we put also an edge from (A, v) or $(\neg A, v)$ to itself for every atom A, and from (Y, v) to itself for every variable Y free in ϕ.

To define the Ω function we proceed as follows. First we assign priorities to fixpoint subformulas of ϕ: we assign to each greatest fixpoint subformula $\nu X.\chi$ in ϕ a priority $2|\chi|$, and we assign to $\mu X.\chi$ a priority $2|\chi| + 1$. This ensures that:

- least fixpoints have odd priority;
- greatest fixpoints have even priority;
- the priority of larger subformulas is larger.

Now we let $\Omega(\psi, v)$ be the priority of ψ, if ψ is a fixpoint formula; $\Omega(A, v) = 2$ if $M, v \models A$ and $\Omega(A, v) = 1$ otherwise, if A is an atom, a negated atom or a free variable of ϕ; and $\Omega(\psi, v) = 1$ otherwise.

It results that $M, v_0 \models \phi$ if and only if player d has a winning strategy in $\Gamma(\phi, v_0, M)$.

4.3 From parity games to μ-calculus

We have seen that in a sense, μ-calculus reduces to parity games. However, as is well known, also the other way round is true: if we consider parity game of index

n as a graph vertex-colored by $c, d, 1, \ldots, n$, then there is a μ-calculus formula W_n, due to Walukiewicz, such that an arena for parity games (G, v_0) verifies W_n if and only if player d has a winning strategy in the parity game associated to (G, v_0). This formula is

$$W_n = \mu X_1 \nu X_2 \ldots \theta X_n.(d \to \langle\,\rangle \bigwedge_i (i \to X_i) \wedge (c \to [\,] \bigwedge_i (i \to X_i).$$

5 The reduction to $S5$

In [13] there is a reduction of μ-calculus model checking to box free μ-calculus model checking. Here we modify the result by specializing to $S5$ and by referring to the vectorial model checking rather than the scalar one:

Theorem 2. *Given a finite model M and a modal μ-system S, there is a finite $S5$ model M' and a box free modal μ-system S', such that M' and S' are built in time polynomial in the size of M plus the size of S, and such that M verifies $sol_1(S)$ if and only if M' verifies $sol_1(S')$.*

Proof. Let $M = (V, E, Val)$ be a model. Let S be a modal μ-system. Up to perform a polynomial time rewriting of S, we can suppose that the equations of S have one of the following forms: $X = A$, $X = \neg A$ where A is an atom, $X = Y \vee Z$, $X = Y \wedge Z$, $X = \langle\,\rangle Y$, $X = [\,]Y$, $X = Y$.

Now M' is obtained as follows. The vertices of M' are the vertices of M. Let us enumerate these vertices as $v_1, \ldots, v_n$. Let A_i be an atom which is true in G' only in the point v_i. The relation E' of M' holds for every pair of vertices of M', so M' is an $S5$ model.

Moreover S' is obtained by replacing every equation of S of the form

$$X = \langle\,\rangle Y$$

with

$$X = \bigvee \{A_i \wedge \langle\,\rangle (A_j \wedge Y) | v_i R v_j\},$$

and every equation of S of the form

$$X = [\,]Y$$

with

$$X = \bigwedge \{A_i \to \langle\,\rangle (A_j \wedge Y) | v_i R v_j\}.$$

Note that one could expect that the right hand side of $X = [\,]Y$ is replaced by the De Morgan dual of $X = \langle\,\rangle Y$, so to have $\bigwedge\{A_i \to [\,](A_j \to Y) | v_i R v_j\}$. However, the atoms A_i are interpreted as singletons, so

$$[\,](A_j \to Y)$$

is in fact equivalent to its De Morgan dual

$$\langle\ \rangle(A_j \wedge Y),$$

and this allows us to replace the box with the diamond.

□

We do not know whether Theorem 2 can be specialized to the scalar μ-calculus. In fact, the problem is that the translation from a modal μ-system to a single formula of the modal μ-calculus (i.e., the algorithm which builds the solution of a modal μ-system) takes exponential time in general.

6 Corollaries

It is well known that there is a translation of vectorial μ-calculus in $S5$ to vectorial modal logic in $S5$. In fact, given a modal μ-system in $S5$, we can first consider its solution and translate it into modal logic. From the previous theorem we obtain:

Corollary 2. *If there is a* polynomial time computable *translation from box-free vectorial μ-calculus in $S5$ to vectorial modal logic in $S5$, then the vectorial μ-calculus model checking problem is in P.*

Proof. By [16], model checking for vectorial modal logic (over arbitrary graphs) reduces in polynomial time to the problem of solving Boolean equation systems with only one type of fixpoint, and this problem is in P by [3]. By the previous theorem, vectorial model checking reduces to vectorial model checking over $S5$ in polynomial time; so if the translation in the statement exists, then by a chain of reductions, the vectorial μ-calculus model checking problem is in P. □

Considerations analogous to $S5$ hold in the larger class of graphs $K4$. In fact, from [2], [6] and [5] it follows that there is a translation from vectorial μ-calculus in $K4$ to $VEC - \Pi_2$ in $K4$. From the previous theorem we obtain:

Corollary 3. *If there is a* polynomial time computable *translation from vectorial μ-calculus in $K4$ to $VEC - \Pi_2$ in $K4$, then the μ-calculus model checking problem is in P.*

Proof. Every $S5$ graph is also a $K4$ graph, so by the previous theorem, there is a polynomial time reduction from vectorial model checking over arbitrary graphs to vectorial model checking in $K4$. Moreover, by [16], model checking for $VEC - \Pi_2$ (over arbitrary graphs) reduces in polynomial time to the problem of solving Boolean equation systems of class Π_2, and this problem is in P by [3]. So if the translation in the statement exists, then by a chain of reductions, the vectorial μ-calculus model checking problem is in P.

□

7 Satisfiability in $S5$

In this section we investigate the μ-calculus satisfiability problem for $S5$. We begin with establishing a linear size model property.

Lemma 1. *If a formula ϕ has a $S5$ model, then it has a $S5$ model of size linear in ϕ.*

Proof. Let (M, x_0) be a $S5$ model of ϕ. Up to bisimulation we can suppose that M is total. Then player d has a winning strategy in the game $\Gamma(\phi, x_0, M)$. By Corollary 1, d has a strongly positional winning strategy in the game, call it Σ. Consider a diamond position $(\langle\,\rangle\psi, y)$ of the game. Since M is total, the set of successors of $(\langle\,\rangle\psi, y)$ does not depend on y, so the choice of Σ also does not depend on y, but only on ψ. Let us denote by (ψ, x_ψ) the successor position of $(\langle\,\rangle\psi, y)$ chosen by Σ. Let N be the submodel of M given by x_0 plus all points x_ψ.

First, N has size linear in ϕ because its size is at most the number of diamond subformulas of ϕ (plus one). Moreover, we note that if player c in the game always chooses elements of N in box positions, then the game remains in N forever, and is won by d since Σ is winning on M.

□

As a consequence we have:

Theorem 3. *The satisfiability problem for the μ-calculus in $S5$ is NP-complete.*

Proof. NP-hardness holds because the μ-calculus contains propositional logic.

To show that the problem is in NP, suppose that models and formulas are encoded as strings in a convenient finite alphabet (e.g. ASCII code). We prove that there is a problem S in $PTIME$ and a polynomial p such that ϕ is satisfiable in $S5$ iff there exists a witness z such that $(\phi, z) \in S$ and $length(z) \leq p(length(\phi))$.

Since μ-calculus model checking is in NP (see [7]), we know that there exists a problem S' in $PTIME$ and a polynomial q such that, for any finite model M, M satisfies ϕ iff there exists a y with $(M, \phi, y) \in S'$ and $length(y) \leq q(length(M) + length(\phi))$. Moreover, by Lemma 1, ϕ is satisfiable in $S5$ iff ϕ is satisfiable in a model M of size linear in ϕ, which we can code with a length at most $r(length(\phi))$ for some polynomial r.

Let S be the set of tuples (ϕ, M, y) such that:

- M is an $S5$ model (i.e. the accessibility relation is an equivalence);
- $(M, \phi, y) \in S'$;
- $length(M) \leq r(length(\phi))$;
- $length(y) \leq q(length(M) + length(\phi))$.

So, ϕ is satisfiable in $S5$ if and only if there exists a witness $z = (M, y)$ such that $(\phi, z) \in S$. Note that S is in $PTIME$. Moreover, $(\phi, z) \in S$ implies $length(z) \leq p(length(\phi))$, where $p(x) = r(x) + q(r(x) + x)$. So, ϕ is satisfiable in $S5$ if and only if there exists a witness $z = (M, y)$ such that $(\phi, z) \in S$ and $length(z) \leq p(length(\phi))$. So S and p satisfy the desired properties.

□

Note that the restriction of the previous theorem to modal logic was already known, see [11].

8 On existential μ-calculus

A formula of the μ-calculus is called existential, or box-free, if it contains no box operators $[\ ]\phi$. Intuitively, existential μ-calculus is considerably simpler than general μ-calculus. In fact, the satisfiability problem for the μ-calculus is $EXPTIME$-complete, whereas, as shown in [13], the same problem for the existential μ-calculus is NP-complete. Note that this last result can be obtained in a way different from [13] as follows.

First we observe:

Lemma 2. *The satisfiability problem for existential μ-calculus is polynomial time equivalent to the same problem on $S5$.*

Proof. If an existential formula ϕ has a model M, then ϕ is also true on the reflexive, symmetric, transitive closure of M, which is an $S5$ graph of the same size as M.

□

Moreover, as a corollary of the previous section we have:

Corollary 4. *The satisfiability problem for existential μ-calculus in $S5$ is NP-complete.*

Proof. The satisfiability problem for existential μ-calculus in $S5$ is NP because, by Theorem 3, it is a particular case of an NP problem. Moreover, the problem is NP-hard because existential μ-calculus contains propositional logic. □

From the previous lemma and the previous corollary it follows:

Corollary 5. *The satisfiability problem for existential μ-calculus is NP-complete.*

9 On μ-calculus and modal logic in $S5$

It is known that μ-calculus in $S5$ is as expressive as modal logic, so in $S5$ there is no fixpoint alternation hierarchy. In this section we describe two translations from μ-calculus to modal logic; the first is due to Alberucci and Facchini, whereas the second is based on a bisimulation argument.

9.1 On the complexity of translations from the μ-calculus to modal logic over $S5$

In [2] a recursive translation of μ-calculus into modal logic in $S5$ is given. The construction is performed by induction on ordinals (rather than ordinary induction on numbers). In order to set up the construction, a notion of ordinal rank of μ-calculus formulas is introduced, with the following properties:

- $rank(A) = rank(\neg A) = 1$;
- $rank(\langle\ \rangle\phi) = rank([\ ]\phi) = rank(\phi) + 1$;
- $rank(\phi \wedge \psi) = rank(\phi \vee \psi) = max\{rank(\phi), rank(\psi)\} + 1$;
- $rank(\mu X.\phi(X)) = rank(\nu X.\phi(X)) = sup\{rank(\phi^n(X)) + 1; n \in \mathbb{N}\}$.

A μ-calculus sentence ϕ is *well named* if it is guarded and, for any variable X, no two distinct occurrences of fixpoint operators in ϕ bind X, and the atom X occurs only once in ϕ.

Using the semantical laws $\mu X.\psi(X, X) = \mu X.\mu Y.\psi(X, Y)$, $\nu X.\psi(X, X) = \nu X.\nu Y.\psi(X, Y)$ (see [3]), and renaming of bounded variables, we see that any μ-calculus formula is equivalent to a well named formula of a size which is linear in the size of ϕ. For instance, the formula $\mu X([\ ]X \wedge \langle\ \rangle X) \wedge \nu X[\ ]X$ is equivalent to the well named formula $\mu X \mu Y([\ ]X \wedge \langle\ \rangle Y) \wedge \nu Z[\ ]Z$.

The following translation t from well named μ-formulas to modal logic in $S5$ can be defined:

- $t(A) = A, t(\neg A) = \neg A$;
- $t(true) = true$, $t(false) = false$;
- $t(\langle\ \rangle\phi) = \langle\ \rangle t(\phi)$;
- $t([\ ]\phi) = [\ ]t(\phi)$;
- $t(\phi \wedge \psi) = t(\phi) \wedge t(\psi)$;
- $t(\phi \vee \psi) = t(\phi) \vee t(\psi)$;
- $t(\mu X.\phi(X)) = t((\phi(\phi(false))^*)$;
- $t(\nu X.\phi(X)) = t((\phi(\phi(true))^*)$,

where $(\phi(\phi(false))^*, (\phi(\phi(true))^*$ denote the well named formulas obtained from $\phi(\phi(false)), \phi(\phi(true))$ by renaming repeated bound variables. The translation t is given by induction on the rank, so it is well defined. Moreover:

Lemma 3. *If ϕ is a well named formula, then the length of $t(\phi)$ is at most $2^{|\phi|}$.*

Proof. We need a preliminary composition lemma:

Lemma 4. $t(\phi[X/\psi]) = t(\phi)[X/t(\psi)]$.

Proof. By induction on ϕ. □

Now the bound as in the lemma can be proved by induction on the rank of ϕ. The most delicate case is $\mu X.\psi(X)$ and $\nu X.\psi(X)$. Now, by Lemma 4, we have $t(\mu X.\psi(X)) = t(\psi(\psi(false)) = t(\psi)[X/t(\psi(false))]$, and since X occurs only once in ψ, we obtain the bound $|t(\mu X.\psi)| = |t(\psi)[X/t(\psi(false))]| \leq 2|t(\psi(false))| \leq 2 \times 2^{|\psi(false)|} = 2^{|\psi(false)|+1} \leq 2^{|\mu X.\psi(X)|}$. The case of ν is analogous. □

So, the translation t is at most exponential. We also can show that the exponential upper bound for the translation t is tight. In fact, consider the well named formulas:

$$\phi_n = \mu X_1 \ldots \mu X_n.X_1 \vee (X_2 \vee \ldots \vee X_n).$$

We can show by induction that $t(\phi_n)$ has size at least 2^n. In fact, the base case $n = 1$ is true; for the inductive case, we begin with a lemma:

Lemma 5. *The following equivalences hold:*

- $|t(\phi(false \vee \alpha))| > |t(\phi(\alpha)|$;
- *if X is a variable free in ϕ, then $|t(\phi(X \vee \alpha))| > |t(\phi(\alpha)|$.*

Proof. By induction on ϕ. □

Now consider $t(\phi_n)$ with $n > 1$. By definition of t we have

$$t(\phi_n) = t(\mu X_2 \dots \mu X_n.(\mu X_2 \dots \mu X_n.false \vee X_2 \vee \dots \vee X_n) \vee X_2 \dots \vee X_n)$$

and by Lemma 4 we obtain

$$t(\phi_n) = t(\mu X_2 \dots \mu X_n.Y \vee X_2 \dots \vee X_n)[Y/t(\mu X_2 \dots \mu X_n.false \vee X_2 \vee \dots \vee X_n).$$

By evaluating the sizes we see that

$$|t(\phi_n)| \geq |t(\mu X_2 \dots \mu X_n.Y \vee X_2 \dots \vee X_n)| + |t(\mu X_2 \dots \mu X_n.false \vee X_2 \vee \dots \vee X_n)]| - 1;$$

by Lemma 5 we obtain

$$|t(\phi_n)| \geq 2|t(\mu X_2 \dots \mu X_n.X_2 \vee \dots X_n)|,$$

and, by renaming the variables,

$$|t(\phi_n)| \geq 2|t(\phi_{n-1})|.$$

Finally, the inductive hypothesis gives $|t(\phi_{n-1})| \geq 2^{n-1}$, so $|t(\phi_n)| \geq 2^n$.

9.2 An alternative translation

A translation from μ-calculus to modal logic different from [2] is obtained as follows. Let ϕ be a μ-calculus formula containing a set At of atoms. Then, up to bisimulation, there are finitely many $S5$ models colored with At, more precisely exponentially many of them. Each bisimulation class is described by a characteristic formula, that is, the conjunction of $\langle\,\rangle\gamma$ where γ is a conjunction of atoms and negated atoms present in the model of the class, and $\neg\langle\,\rangle\gamma$ where γ is a conjunction not present in the class. So, ϕ is equivalent to the disjunction of the characteristic formulas of the bisimulation classes of the models of ϕ. Note that this alternative translation is also (at most) exponential.

9.3 A corollary

Both translations of the previous subsections give an exponential blow up in the size of the formula. It is then natural to ask whether there exists a polynomial time computable translation from the μ-calculus to modal logic over $S5$. This question is related to the results of the previous section, as the following final corollary shows:

Corollary 6. *If there is a polynomial time computable translation from the μ-calculus to modal logic over $S5$ and Theorem 2 specializes to scalar μ-calculus, then the μ-calculus model checking problem is in P.*

Proof. Given a μ-formula ϕ and a model M, first reduce the problem to a μ-formula ϕ' over an $S5$ model M', and then apply the polynomial translation in order to obtain a modal formula ϕ^* with

$$M \models \phi \Leftrightarrow M' \models \phi^*.$$

Since both M' and ϕ^* are obtained in polynomial time from M, ϕ and *modal* model checking is in P, we have done. □

10 Conclusion

In this paper we have investigated some aspects of μ-calculus on $S5$ graphs. Arguably, these graphs are quite simple. However, simplicity of $S5$ graphs gives us better satisfiability bounds than arbitrary graphs, but not better bounds on the model checking problem.

An interesting question is whether there is a direct, natural translation from modal μ systems to modal systems in $S5$: by this we mean that the translation should not go through transforming a system of equations into a single scalar term.

A similar analysis of μ-calculus model checking and satisfiability could be carried over other important classes of graphs. An example is $K4$. Since satisfiability in $K4$ efficiently reduces to general satisfiability, in $K4$ we have the same bound as in K, that is, $EXPTIME$. It would be interesting to see whether a better bound can be given. Note that for model checking, as we have seen, a better bound for $K4$ with respect to arbitrary graphs does not exist.

The same analysis could be done for other interesting classes of graphs, e.g. the longstanding Gödel-Löb class GL (i.e. the transitive wellfounded graphs), or for more recent classes such as graphs of bounded tree width or classes with forbidden minors. A good model checking algorithm for bounded tree width is e.g. [18], but we do not have yet a polynomial time model checking algorithm on graphs of bounded tree width. Apparently there is no result on satisfiability on bounded tree width.

Acknowledgments

This work has been partially supported by the PRIN project n. 20089M932N *Innovative and multi-disciplinary approaches for constraint and preference reasoning.*

References

1. L. Alberucci, Sequent Calculi for the Modal μ-Calculus over S5. J. Log. Comput. 19(6): 971–985 (2009).
2. L. Alberucci and A. Facchini, The modal μ-calculus hierarchy over restricted classes of transition systems, J. Symb. Logic 74 (2009) 1367–1400.
3. A. Arnold and D. Niwinski, Rudiments of μ-calculus, North-Holland, 2001.
4. Edmund M. Clarke, Jr., Orna Grumberg and Doron A. Peled, Model Checking, MIT Press, 1999.
5. A. Dawar and M. Otto, Modal characterisation theorems over special classes of frames, Ann. Pure Appl. Logic 161 (2009), 1–42.
6. G. D'Agostino and G. Lenzi, On the μ-calculus over transitive and finite transitive frames, Theor. Comput. Sci. 411(50): 4273–4290 (2010).
7. E. Allen Emerson: Model Checking and the Mu-calculus. Descriptive Complexity and Finite Models 1996: 185–214.
8. E. A. Emerson and C. S. Jutla, The complexity of tree automata and logics of programs. In Proc. 29th IEEE FOCS 328–337 (1988).
9. E. A. Emerson and C. S. Jutla. Tree Automata, Mu-Calculus and Determinacy (Extended Abstract). FOCS 1991: Pages 368–377.
10. Robert S. Streett and E. Allen Emerson. An Automata Theoretic Decision Procedure for the Propositional Mu-Calculus. Information and Computation 81 (1989), 249–264.
11. R. Fagin, Reasoning about knowledge, MIT Press, 2003.
12. M. J. Fischer and R. E. Ladner, Propositional dynamic logic of regular programs, J. Comput. System Sci., 18 (1979), pp. 194–211.
13. Thomas A. Henzinger, Orna Kupferman, and Rupak Majumdar, On the Universal and Existential Fragments of the Mu-Calculus, Theoretical Computer Science 354:173-186, 2006.
14. M. Jurdzinski, Deciding the winner in parity games is in $UP \cap co - UP$, Inform. Proc. Letters 68 (1998), 119-124.
15. D. Kozen, Results on the Propositional mu-Calculus, Theor. Comput. Sci. 27: 333–354 (1983).
16. A. Mader, Verification of Modal Properties using Boolean Equation Systems, Ph. D. Thesis, 1997.
17. D. A. Martin, Borel determinacy, Ann. Math., 102 (1975), pp. 363–371.
18. J. Obdrzalek, Fast Mu-Calculus Model Checking when Tree-Width Is Bounded. CAV 2003, 80–92.
19. F. Poggiolesi, A cut-free simple sequent calculus for modal logic S5, Review of Symbolic Logic, 1:3–15, 2008.
20. I. Walukiewicz, Completeness of Kozen's Axiomatisation of the Propositional Mu-Calculus, Information and Computation 157 (2000), 142–182.

MCINTYRE: A Monte Carlo Algorithm for Probabilistic Logic Programming

Fabrizio Riguzzi

ENDIF – Università di Ferrara – Via Saragat, 1 – 44122 Ferrara, Italy.
{fabrizio.riguzzi}@unife.it

Abstract. Probabilistic Logic Programming is receiving an increasing attention for its ability to model domains with complex and uncertain relations among entities. In this paper we concentrate on the problem of approximate inference in probabilistic logic programming languages based on the distribution semantics. A successful approximate approach is based on Monte Carlo sampling, that consists in verifying the truth of the query in a normal program sampled from the probabilistic program. The ProbLog system includes such an algorithm and so does the `cplint` suite. In this paper we propose an approach for Monte Carlo inference that is based on a program transformation that translates a probabilistic program into a normal program to which the query can be posed. In the transformation, auxiliary atoms are added to the body of rules for performing sampling and checking for the consistency of the sample. The current sample is stored in the internal database of the Yap Prolog engine. The resulting algorithm, called MCINTYRE for Monte Carlo INference wiTh Yap REcord, is evaluated on various problems: biological networks, artificial datasets and a hidden Markov model. MCINTYRE is compared with the Monte Carlo algorithms of ProbLog and `cplint` and with the exact inference of the PITA system. The results show that MCINTYRE is faster than the other Monte Carlo algorithms.

Keywords: Probabilistic Logic Programming, Monte Carlo Methods, Logic Programs with Annotated Disjunctions, ProbLog.

1 Introduction

Probabilistic Logic Programming (PLP) is an emerging field that has recently seen many proposals for the integration of probability in logic programming. Such an integration overcomes the limit of logic of dealing only with certain propositions and the limit of works in probability theory that consider mostly simple descriptions of domain entities instead of complex relational descriptions.

PLP is of interest also for its many application domains, the most promising of which is maybe Probabilistic Inductive Logic Programming [5] in which PLP languages are used to represent the theories that are induced from data. This allows a richer representation of the domains that often leads to increased modeling accuracy. This trend can be cast in a more general tendency in Machine

Learning to combine aspects of uncertainty with aspects of logic, as is testified by the development of the field of Statistical Relational Learning [7].

Many languages have been proposed in PLP. Among them, many share a common approach for defining the semantics, namely the so called distribution semantics [17]. This approach sees a probabilistic logic program as a description of a probability distribution over normal logic programs, from which the probability of queries is computed. Example of languages following the distribution semantics are Probabilistic Logic Programs [3], Probabilistic Horn Abduction [10], Independent Choice Logic [11], PRISM [17], Logic Programs with Annotated Disjunctions (LPADs) [21] and ProbLog [6]. These languages have essentially the same expressive power [20,4] and in this paper we consider only LPADs and ProbLog, because they stand at the extremes of syntax complexity, LPADs having the most complex syntax and ProbLog the simplest, and because most existing inference algorithms can be directly applied to them.

The problem of inference, i.e., the problem of computing the probability of a query from a probabilistic logic program, is very expensive, being #P complete [8]. Nevertheless, various exact inference algorithms have been proposed, such as the ProbLog system[1] [6], `cplint`[2] [12,13] and PITA[3] [14,16] and have been successfully applied to a variety of non-trivial problems. All of these algorithms find explanations for queries and then use Binary Decision Diagrams (BDDs) for computing the probability. This approach has been shown to be faster than algorithms not using BDDs. Reducing the time to answer a probabilistic query is important because in many applications, such as in Machine Learning, a high number of queries must be issued. To improve the speed, approximate inference algorithms have been proposed. Some compute a lower bound of the probability, as the k-best algorithm of ProbLog [8] which considers only the k most probable explanations for the query, while some compute an upper and a lower bound, as the bounded approximation algorithm of ProbLog [8] that builds an SLD tree only to a certain depth. A completely different approach for approximate inference is based on sampling the normal programs encoded by the probabilistic program and checking whether the query is true in them. This approach, called Monte Carlo, was first proposed in [8] for ProbLog, where a lazy sampling approach was used in order to avoid sampling unnecessary probabilistic facts. [1] presented algorithms for k-best, bounded approximation and Monte Carlo inference for LPADs that are all based on a meta-interpreter. In particular, the Monte Carlo approach uses the arguments of the meta-interpreter predicate to store the samples taken and to ensure consistency of the sample.

In this paper we present the algorithm MCINTYRE for Monte Carlo INference wiTh Yap REcord that computes the probability of queries by means of a program transformation technique. The disjunctive clauses of an LPAD are first transformed into normal clauses to which auxiliary atoms are added to the body for taking samples and storing the results. The internal database of the

[1] `http://dtai.cs.kuleuven.be/problog/`
[2] `http://www.ing.unife.it/software/cplint/`
[3] `https://sites.google.com/a/unife.it/ml/pita`

Yap Prolog engine is used to record all samples taken thus ensuring that samples are consistent. The truth of a query in a sampled program can be then tested by asking the query to the resulting normal program.

MCINTYRE is compared with the Monte Carlo algorithms of ProbLog and `cplint` and with the exact inference algorithm of the PITA system on various problems: biological networks, artificial datasets and a hidden Markov model. The results show that the performances of MCINTYRE overcome those of the other Monte Carlo algorithms.

The paper is organized as follows. In Section 2 we review the syntax and the semantics of PLP. Section 3 illustrates previous approaches for inference in PLP languages. Section 4 presents the MCINTYRE algorithm. Section 5 describes the experiments and Section 6 concludes the paper.

2 Probabilistic Logic Programming

One of the most interesting approaches to the integration of logic programming and probability is the distribution semantics [17], which was introduced for the PRISM language but is shared by many other languages.

A program in one of these languages defines a probability distribution over normal logic programs called *worlds*. This distribution is then extended to queries and the probability of a query is obtained by marginalizing the joint distribution of the query and the programs. We present the semantics for programs without function symbols but the semantics has been defined also for programs with function symbols [17,15].

The languages following the distribution semantics differ in the way they define the distribution over logic programs. Each language allows probabilistic choices among atoms in clauses: Probabilistic Logic Programs, Probabilistic Horn Abduction , Independent Choice Logic, PRISM and ProbLog allow probability distributions over facts, while LPADs allow probability distributions over the heads of disjunctive clauses. All these languages have the same expressive power: there are transformations with linear complexity that can convert each one into the others [20,4]. Next we will discuss LPADs and ProbLog.

Formally a *Logic Program with Annotated Disjunctions* T [21] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. In such a clause $h_{i1}, \ldots h_{in_i}$ are logical atoms and $b_{i1}, \ldots, b_{im_i}$ are logical literals, $\{\Pi_{i1}, \ldots, \Pi_{in_i}\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. $b_{i1}, \ldots, b_{im_i}$ is called the *body* and is indicated with $body(C_i)$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$ the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$. (C_i, θ_j, k) means that, for the ground clause $C_i\theta_j$, the head h_{ik} was chosen. In practice $C_i\theta_j$ corresponds to a random

variable X_{ij} and an atomic choice (C_i, θ_j, k) to an assignment $X_{ij} = k$. A set of atomic choices κ is *consistent* if $(C_i, \theta_j, k) \in \kappa, (C_i, \theta_j, l) \in \kappa \Rightarrow k = l$, i.e., only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that contains an atomic choice (C_i, θ_j, k) for each clause $C_i\theta_j$ in $ground(T)$. A selection σ identifies a normal logic program w_σ defined as $w_\sigma = \{(h_{ik} : -body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. w_σ is called a *world* of T. Since selections are composite choices, we can assign a probability to possible worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

Since the program does not have function symbols, the set of worlds is finite: $W_T = \{w_1, \ldots, w_m\}$ and, since the probabilities of the individual choices sum to 1, $P(w)$ is a distribution over worlds: $\sum_{w\in W_T} P(w) = 1$. We also assume that each world w has a two-valued well founded model $WFM(w)$. If a query Q is true in $WFM(w)$ we write $w \models Q$.

We can define the conditional probability of a query Q given a world: $P(Q|w) = 1$ if $w \models Q$ and 0 otherwise. The probability of the query can then be obtained by marginalizing over the query

$$P(Q) = \sum_w P(Q, w) = \sum_w P(Q|w)P(w) = \sum_{w\models Q} P(w)$$

Example 1. The following LPAD T encodes a very simple model of the development of an epidemic or a pandemic:

$$C_1 = epidemic : 0.6; pandemic : 0.3 : -flu(X), cold.$$
$$C_2 = cold : 0.7.$$
$$C_3 = flu(david).$$
$$C_4 = flu(robert).$$

This program models the fact that if somebody has the flu and the climate is cold, there is the possibility that an epidemic or a pandemic arises. We are uncertain about whether the climate is cold but we know for sure that David and Robert have the flu. Clause C_1 has two groundings, both with three atoms in the head, while clause C_2 has a single grounding with two atoms in the head, so overall there are $3 \times 3 \times 2 = 18$ worlds. The query *epidemic* is true in 5 of them and its probability is

$$\begin{aligned} P(epidemic) = {} & 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + \\ & 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = \\ & 0.588 \end{aligned}$$

A *ProbLog program* is composed by a set of normal clauses and a set of probabilistic facts, possibly non-ground. A probabilistic fact takes the form

$$\Pi :: f.$$

where Π is in [0,1] and f is an atom. The semantics of such program can be given by considering an equivalent LPAD containing, for each ProbLog normal

clause $h : -B$, a clause $h : 1 : -B$ and, for each probabilistic ProbLog fact, a clause

$$f : \Pi.$$

The semantics of the ProbLog program is the same as that of the equivalent LPAD.

It is also possible to translate an LPAD into a ProbLog program [4]. A clause C_i of the LPAD with variables $\overline{X}$

$$h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -B_i$$

is translated into

$$\begin{array}{l} h_{i1} : -B_i, f_{i1}(\overline{X}). \\ h_{i2} : -B_i, problog\_not(f_{i1}(\overline{X})), f_{i2}(\overline{X}). \\ \vdots \\ h_{in_i-1} : -B_i, problog\_not(f_{i1}(\overline{X})), \ldots, problog\_not(f_{in_i-2}(\overline{X})), f_{in_i-1}(\overline{X}). \\ h_{in_i} : -B_i, problog\_not(f_{i1}(\overline{X})), \ldots, problog\_not(f_{in_i-1}(\overline{X})). \\ \\ \pi_{i1} :: f_{i1}(\overline{X}). \\ \vdots \\ \pi_{in-1} :: f_{in_i-1}(\overline{X}). \end{array}$$

where $problog\_not/1$ is a ProbLog builtin predicate that implements negation for probabilistic atoms and $\pi_{i1} = \Pi_{i1}$, $\pi_{i2} = \frac{\Pi_{i2}}{1-\pi_{i1}}$, $\pi_{i3} = \frac{\Pi_{i3}}{(1-\pi_{i1})(1-\pi_{i2})}, \ldots$. In general $\pi_{ij} = \frac{\Pi_{ij}}{\prod_{k=1}^{j-1}(1-\pi_{ik})}$.

Example 2. The ProbLog program equivalent to the LPAD of Example 1 is

$$\begin{array}{l} C_{11} = epidemic : -flu(X), cold, f1(X). \\ C_{12} = pandemic : -flu(X), cold, problog\_not(f1(X)), f2(X). \\ C_{13} = 0.6 :: f1(X). \\ C_{14} = 0.75 :: f2(X). \\ C_{21} = cold : -f3. \\ C_{22} = 0.7 :: f3. \\ C_3 \ = flu(david). \\ C_4 \ = flu(robert). \end{array}$$

3 Inference Algorithms

In order to compute the probability of a query from a probabilistic logic program, [6] proposed the ProbLog system that first finds a set of explanations for the query and then computes the probability from the set by using Binary Decision Diagrams. An explanation is a set of probabilistic facts used in a derivation of the query. The set of explanations can be seen as a Boolean DNF formula in which

the Boolean propositions are random variables. Computing the probability of the formula involves solving the disjoint sum problem which is #P-complete [19]. BDDs represent an approach for solving this problem that has been shown to work well in practice [6,13,14].

[8] proposed various approaches for approximate inference that are now included in the ProbLog system. The k-best algorithm finds only the k most probable explanations for a query and then builds a BDD from them. The resulting probability is only a lower bound but if k is sufficiently high it represents a good approximation. The bounded approximation algorithm computes a lower bound and an upper bound of the probability of the query by using iterative deepening to explore the SLD tree for the query. The SLD tree is built partially, the successful derivations it contains are used to build a BDD for computing the lower bound while the successful derivations plus the incomplete ones are used to compute the upper bound. If the difference between the upper and the lower bound is above the required precision, the SLD tree is built up to a greater depth. This process is repeated until the required precision is achieved. These algorithms are implemented by means of a program transformation technique applied to the probabilistic atoms: they are turned into clauses that, when the atom is called, add the probabilistic fact to the current explanation.

[1] presented an implementation of k-best and bounded approximation for LPADs that is based on a meta-interpreter and showed that in some cases this gives good results.

[8] also presented a Monte Carlo algorithm that samples the possible programs and tests the query in the samples. The probability of the query is then given by the fraction of programs where the query is true. The Monte Carlo algorithm for ProbLog is realized by using an array with an element for each ground probabilistic fact that stores one of three values: sampled true, sampled false and not yet sampled. When a probabilistic fact is called, the algorithm first checks whether the fact has already been sampled by looking at the array. If it has not been sampled, then it samples it and stores the result in the array. Probabilistic facts that are non-ground in the program are treated differently. A position in the array is not reserved for them since their grounding is not known at the start, rather samples for groundings of these facts are stored in the internal database of Yap and the sampled value is retrieved when they are called. If no sample has been taken for a grounding, a sample is taken and recorded in the database.

[1] presents a Monte Carlo algorithm for LPADs that is based on a meta-interpreter. In order to keep track of the samples taken, two arguments of the meta-interpreter predicate are used, one for keeping the input set of choices and one for the output set of choices. This algorithm is included in the `cplint` suite available in the source tree of Yap[4].

[4] `http://www.dcc.fc.up.pt/~vsc/Yap/downloads.html`

4 MCINTYRE

MCINTYRE first transforms the program and then queries the transformed program. The disjunctive clause $C_i = h_{i1} : \Pi_{i1} \vee \ldots \vee h_{in} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. where the parameters sum to 1, is transformed into the set of clauses $MC(C_i)$:

$$MC(C_i, 1) = h_{i1} : -b_{i1}, \ldots, b_{im_i}, sample\_head(ParList, i, VC, NH), NH = 1.$$

$$\ldots$$

$$MC(C_i, n_i) = h_{in_i} : -b_{i1}, \ldots, b_{im_i}, sample\_head(ParList, i, VC, NH), NH = n_i.$$

where VC is a list containing each variable appearing in C_i and $ParList$ is $[\Pi_{i1}, \ldots, \Pi_{in_i}]$. If the parameters do not sum up to 1 the last clause (the one for *null*) is omitted. Basically, we create a clause for each head and we sample a head index at the end of the body with `sample_head/4`. If this index coincides with the head index, the derivation succeeds, otherwise it fails. Thus failure can occur either because one of the body literals fails or because the current clause is not part of the sample.

For example, clause C_1 of epidemic example becomes

$$MC(C_1, 1) = epidemic : -flu(X), cold, sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 1.$$

$$MC(C_1, 2) = pandemic : -flu(X), cold, sample\_head([0.6, 0.3, 0.1], 1, [X], NH), NH = 2.$$

The predicate `sample_head/4` samples an index from the head of a clause and uses the builtin Yap predicates `recorded/3` and `recorda/3` for respectively retrieving or adding an entry to the internal database. Since `sample_head/4` is at the end of the body and since we assume the program to be range restricted, at that point all the variables of the clause have been grounded. A program is range restricted if all the variables appearing in the head also appear in positive literals in the body. If the rule instantiation had already been sampled, `sample_head/4` retrieves the head index with `recorded/3`, otherwise it samples a head index with `sample/2`:

```
sample_head(_ParList,R,VC,NH):-
  recorded(exp,(R,VC,NH),_),!.
sample_head(ParList,R,VC,NH):-
  sample(ParList,NH),
  recorda(exp,(R,VC,NH),_).

sample(ParList, HeadId) :-
  random(Prob),
  sample(ParList, 0, 0, Prob, HeadId).

sample([HeadProb|Tail], Index, Prev, Prob, HeadId) :-
  Succ is Index + 1,
  Next is Prev + HeadProb,
  (Prob =< Next ->
```

```
    HeadId = Index
;
  sample(Tail, Succ, Next, Prob, HeadId)
).
```

Tabling can be effectively used to avoid re-sampling the same atom. To take a sample from the program we use the following predicate

```
sample(Goal):-
  abolish_all_tables,
  eraseall(exp),
  call(Goal).
```

A fixed number of samples n is taken and the fraction $\hat{p}$ of samples in which the query succeds is computed. In order to compute the confidence interval of $\hat{p}$, we use the central limit theorem to approximate the binomial distribution with a normal distribution. Then the 95% binomial proportion confidence interval is calculated as

$$\hat{p} \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}\,(1-\hat{p})}{n}}$$

where $z_{1-\alpha/2}$ is the $1-\alpha/2$ percentile of a standard normal distribution and usually $\alpha = 0.05$. If the width of the interval is below a user defined threshold δ, we stop and we return the fraction of successful samples.

This estimate of the confidence interval is good for a sample size larger than 30 and if $\hat{p}$ is not too close to 0 or 1. The normal approximation fails totally when the sample proportion is exactly zero or exactly one. Empirically, it has been observed that the normal approximation works well as long as $n\hat{p} > 5$ and $n(1-\hat{p}) > 5$.

5 Experiments

We considered three sets of benchmarks: graphs of biological concepts from [6], artificial datasets from [9] and a hidden Markov model from [2]. On these dataset, we compare MCINTYRE, the Monte Carlo algorithm of ProbLog [8], the Monte Carlo algorithm of `cplint` [1] and the exact system PITA which has been shown to be particularly fast [14]. All the experiments have been performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM. The algorithms were run on the data for 24 hours or until the program ended for lack of memory. $\delta = 0.01$ was chosen as the maximum confidence interval width for Monte Carlo algorithms. The normal approximation tests $n\hat{p} > 5$ and $n(1-\hat{p}) > 5$ were disabled in MCINTYRE because they are not present in ProbLog. For each experiment we used tabling when it gave better results.

In the graphs of biological concepts the nodes encode biological entities such as genes, proteins, tissues, organisms, biological processes and molecular functions, and the edges conceptual and probabilistic relations among them. Edges

are thus represented by ground probabilistic facts. The programs have been sampled from the Biomine network [18] containing 1,000,000 nodes and 6,000,000 edges. The sampled programs contain 200, 400, ..., 10000 edges. Sampling was repeated ten times, to obtain ten series of programs of increasing size. In each program we query the probability that the two genes HGNC_620 and HGNC_983 are related.

For MCINTYRE and ProbLog we used the following definition of path

```
path(X,X).
path(X,Y):-X\==Y, path(X,Z),arc(Z,Y).
arc(X,Y):-edge(Y,X).
arc(X,Y):-edge(X,Y).
```

For MCINTYRE, we tabled `path/2` using Yap tabling with the directive

```
:- table path/2.
```

while for ProbLog we tabled the path predicate by means of ProbLog tabling with the command

```
problog_table(path/2),
```

For PITA we used the program

```
path(X,Y):-path(X,Y,[X],Z).
path(X,X,A,A).
path(X,Y,A,R):-X\==Y, arc(X,Z), \+ member(Z,A), path(Z,Y,[Z|A],R).
arc(X,Y):-edge(Y,X).
arc(X,Y):-edge(X,Y).
```

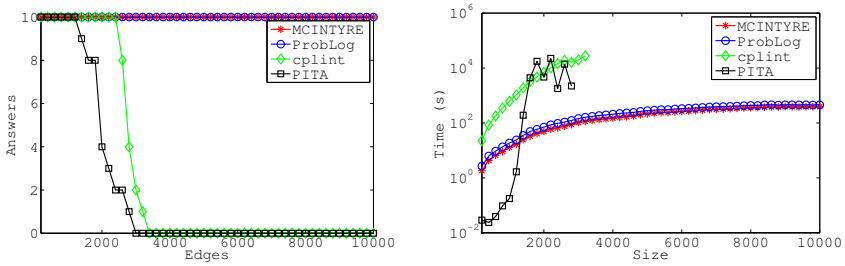
that performs loop checking by keeping a list of visited nodes rather than by using tabling because this approach gave the best results. We used the same program also for `cplint` because it does not allow to use tabling for loop checking.

Figure 1(a) shows the number of graphs for each size for which MCINTYRE, ProbLog, `cplint` and PITA were able to compute the probability. Figure 1(b) shows the execution times of the four algorithms as a function of graph size averaged over the graphs on which the algorithms succeeded.

MCINTYRE and ProbLog were able to solve all graphs, while PITA and `cplint` stopped much earlier. As regards speed, MCINTYRE is much faster than `cplint` and slightly faster than ProbLog. For non-small programs it is also faster than PITA.

The growing head dataset from [9] contains propositional programs in which the head of clauses are of increasing size. For example, the program for size 4 is

```
a0 :- a1.
a1:0.5.
a0:0.5; a1:0.5 :- a2.
a2:0.5.
a0:0.333333333333; a1:0.333333333333; a2:0.333333333333 :- a3.
a3:0.5.
```



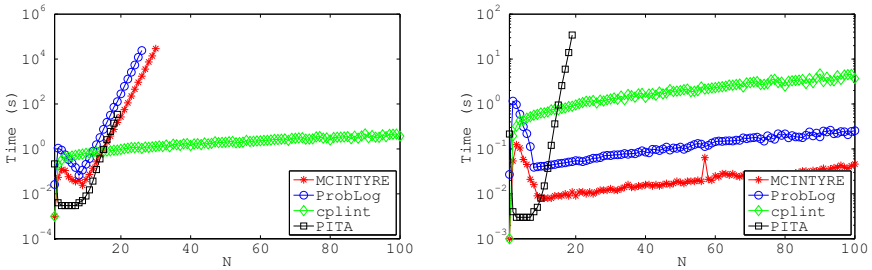
(a) Solved graphs. (b) Average execution times .

Fig. 1. Biological graph experiments.

The equivalent ProbLog program is

```
a0 :- a1.                    0.5::a1f.
a1:-a1f.                     0.5::a0_2.
a0:-a2,a0_2.
a1:-a2,problog_not(a0_2).    0.5::a2f.
a2:-a2f.
0.333333333333::a0_3.        0.5::a1_3.
a0:-a3,a0_3.
a1:-a3,problog_not(a0_3),a1_3.
a2:-a3,problog_not(a0_3),problog_not(a1_3).
0.5::a3f.
a3:-a3f.
```

In this dataset no predicate is tabled for both MCINTYRE and ProbLog. Figure 2(a) shows the time for computing the probability of `a0` as a function of the size. MCINTYRE is faster than ProbLog and PITA for non-small programs but all of them are much slower and less scalable than `cplint`. The reason why



(a) Sampling last. (b) Sampling first.

Fig. 2. Growing head from [9].

`cplint` performs so well is that the meta-interpreter checks for the consistency of the sample when choosing a clause to resolve with the goal, rather than after having resolved all the body literals as in MCINTYRE and ProbLog. However, since the clauses are ground, the sampling predicates of MCINTYRE can be put at the beginning of the body, simulating `cplint` behavior. Similarly, the probabilistic atoms can be put at the beginning of the body of ProbLog clauses. With this approach, we get the timings depicted in Figure 2(b) which shows that now MCINTYRE and ProbLog are faster than `cplint` and MCINTYRE is the fastest.

The blood type dataset from [9] determines the blood type of a person on the basis of her chromosomes that in turn depend on those of her parents. The blood type is given by clauses of the form

```
bloodtype(Person,a):0.90 ; bloodtype(Person,b):0.03 ;
bloodtype(Person,ab):0.03 ; bloodtype(Person,null):0.04 :-
  pchrom(Person,a),mchrom(Person,a).
```

where `pchrom/2` indicates the chromosome inherited from the father and `mchrom /2` that inherited from the mother. There is one such clause for every combination of the values {`a, b, null`} for the father and mother chromosomes. In turn, the chromosomes of a person depend from those of her parents, with clauses of the form

```
mchrom(Person,a):0.90 ; mchrom(Person,b):0.05 ;
mchrom(Person,null):0.05 :-
  mother(Mother,Person), pchrom(Mother,a), mchrom(Mother,a).
```

There is one such clause for every combination of the values {`a, b, null`} for the father and mother chromosomes of the mother and similarly for the father chromosome of a person. In this dataset we query the blood type of a person on the basis of that of its ancestors. We consider families with an increasing number of components: each program adds two persons to the previous one. The chromosomes of the parentless ancestors are given by disjunctive facts of the form

```
mchrom(p,a):0.3 ; mchrom(p,b):0.3 ; mchrom(p,null):0.4.
pchrom(p,a):0.3 ; pchrom(p,b):0.3 ; pchrom(p,null):0.4.
```

For both MCINTYRE and ProbLog all the predicates are tabled.

Figure 3 shows the execution times as a function of the family size. Here MCINTYRE is faster than ProbLog but slower than the exact inference of PITA. This is probably due to the fact that the bodies of clauses with the same atoms in the head are mutually exclusive in this dataset and the goals in the bodies are independent, making BDD operations particularly fast.

In the growing body dataset [9] the clauses have bodies of increasing size. For example, the program for size 4 is,

```
a0:0.5 :-  a1.
```

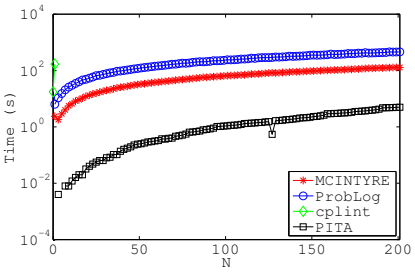
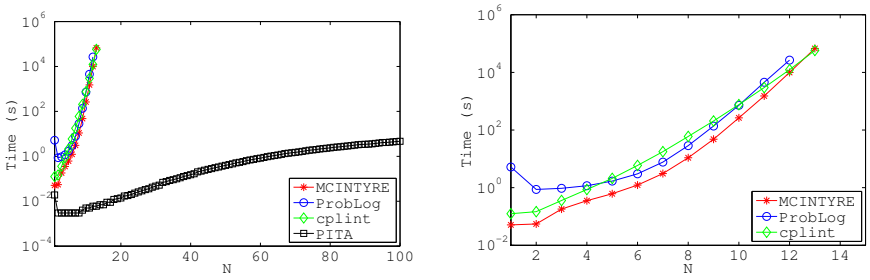


Fig. 3. [Bloodtype from [9].

```
a0:0.5 :- \+ a1,  a2.
a0:0.5 :- \+ a1, \+ a2,  a3.
a1:0.5 :-  a2.
a1:0.5 :- \+ a2,  a3.
a2:0.5 :-  a3.
a3:0.5.
```

In this dataset as well no predicate is tabled for both MCINTYRE and ProbLog and the sampling predicates of MCINTYRE and the probabilistic atoms of ProbLog have been put at the beginning of the body since the clauses are ground. Figure 4(a) shows the execution time for computing the probability of `a0`. Here PITA is faster and more scalable than Monte Carlo algorithms, again probably due to the fact that the bodies of clauses with the same heads are mutually exclusive thus simplifying BDD operations. Figure 4(b) shows the execution time of the Monte Carlo algorithms only, where it appears that MCINTYRE is faster than ProbLog and `cplint`.

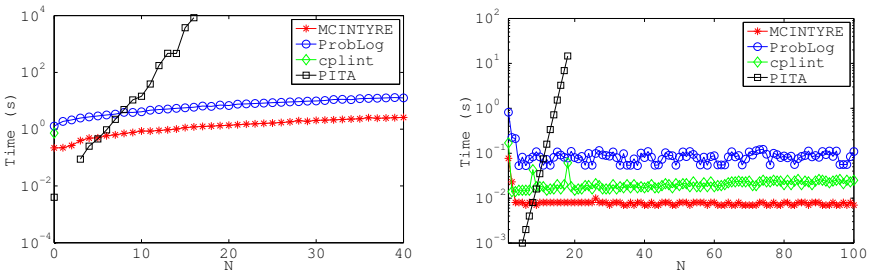


(a) All algorithms. (b) Monte Carlo algorithms.

Fig. 4. Growing body. from [9].

The UWCSE dataset [9] describes a university domain with predicates such as `taught_by/2`, `advised_by/2`, `course_level/2`, `phase/2`, `position/2`, `course/1`,

`professor/1`, `student/1` and others. Programs of increasing size are considered by adding facts for the `student/1` predicate, i.e., by considering an increasing number of students. For both MCINTYRE and ProbLog all the predicates are tabled. The time for computing the probability of the query `taught_by(c1,p1)` as a function of the number of students is shown in Figure 5(a). Here MCINTYRE is faster than ProbLog and both scale much better than PITA.



(a) UWCSE. (b) Hidden Markov Model.

Fig. 5. UWCSE and Hidden Markov Model

The last experiment involves the Hidden Markov model for DNA sequences from [2]: bases are the output symbols and three states are assumed, of which one is the end state. The following program generates base sequences.

```
hmm(O):-hmm1(_,O).
hmm1(S,O):-hmm(q1,[],S,O).
hmm(end,S,S,[]).
hmm(Q,S0,S,[L|O]):- Q\= end, next_state(Q,Q1,S0), letter(Q,L,S0),
  hmm(Q1,[Q|S0],S,O).
next_state(q1,q1,_S):1/3;next_state(q1,q2,_S):1/3;
  next_state(q1,end,_S):1/3.
next_state(q2,q1,_S):1/3;next_state(q2,q2,_S):1/3;
  next_state(q2,end,_S):1/3.
letter(q1,a,_S):0.25;letter(q1,c,_S):0.25;letter(q1,g,_S):0.25;
  letter(q1,t,_S):0.25.
letter(q2,a,_S):0.25;letter(q2,c,_S):0.25;letter(q2,g,_S):0.25;
  letter(q2,t,_S):0.25.
```

The algorithms are used to compute the probability of `hmm(O)` for random sequences `O` of increasing length. Tabling was not used for MCINTYRE nor for ProbLog.

Figure 5(b) show the time taken by the various algorithms as a function of the sequence length. Since the probability of such a sequence goes rapidly to zero, the derivations of the goal terminate mostly after a few steps only and

all Monte Carlo algorithms take constant time with MCINTYRE faster that ProbLog and `cplint`.

6 Conclusions

Probabilistic Logic Programming is of high interest for its many application fields. The distribution semantics is one of the most popular approaches to PLP and underlies many languages, such as LPADs and ProbLog. However, exact inference is very expensive, being #P complete and thus approximate approaches have to be investigated. In this paper we propose the algorithm MCINTYRE that performs approximate inference by means of a Monte Carlo technique, namely random sampling. MCINTYRE transforms an input LPAD into a normal program that contains a clause for each head of an LPAD clause. The resulting clauses contain in the body auxiliary predicates that perform sampling and check for the consistency of the sample.

MCINTYRE has been tested on graphs of biological concepts, on four artificial datasets from [9] and on a hidden Markov model. In all cases it turned out to be faster than the Monte Carlo algorithms of ProbLog and `cplint`. It is also faster and more scalable than exact inference except in two datasets, blood type and growing body, that however possess peculiar characteristics. MCINTYRE is available in the `cplint` package of the source tree of Yap and instructions on its use are available at `http://www.ing.unife.it/software/cplint/`.

In the future we plan to investigate other approximate inference techniques such as lifted belief propagation and variational methods.

References

1. Bragaglia, S., Riguzzi, F.: Approximate inference for logic programs with annotated disjunctions. In: International Conference on Inductive Logic Programming. LNAI, vol. 6489, pp. 30–37. Springer (2011)
2. Christiansen, H., Gallagher, J.P.: Non-discriminating arguments and their uses. In: International Conference on Logic Programming. LNCS, vol. 5649, pp. 55–69. Springer (2009)
3. Dantsin, E.: Probabilistic logic programs and their semantics. In: Russian Conference on Logic Programming. LNCS, vol. 592, pp. 152–164. Springer (1991)
4. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In: Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J. (eds.) Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, in NIPS (2008)
5. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.): Probabilistic Inductive Logic Programming - Theory and Applications, LNCS, vol. 4911. Springer (2008)
6. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: International Joint Conference on Artificial Intelligence. pp. 2462–2467. AAAI Press (2007)

7. Getoor, L., Taskar, B. (eds.): Introduction to Statistical Relational Learning. MIT Press (2007)
8. Kimmig, A., Demoen, B., De Raedt, L., Costa, V.S., Rocha, R.: On the implementation of the probabilistic logic programming language ProbLog. Theory and Practice of Logic Programming 11(2-3), 235–262 (2011)
9. Meert, W., Struyf, J., Blockeel, H.: CP-Logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In: International Conference on Inductive Logic Programming. LNCS, vol. 5989, pp. 96–109. Springer (2010)
10. Poole, D.: Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities. New Generation Computing 11(3-4), 377–400 (1993)
11. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artificial Intelligence 94(1-2), 7–56 (1997)
12. Riguzzi, F.: A top-down interpreter for LPAD and CP-Logic. In: Congress of the Italian Association for Artificial Intelligence. LNCS, vol. 4733, pp. 109–120. Springer (2007)
13. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Logic Journal of the IGPL 17(6), 589–629 (2009)
14. Riguzzi, F., Swift, T.: Tabling and Answer Subsumption for Reasoning on Logic Programs with Annotated Disjunctions. In: International Conference on Logic Programming. LIPIcs, vol. 7, pp. 162–171. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2010)
15. Riguzzi, F., Swift, T.: An extended semantics for logic programs with annotated disjunctions and its efficient implementation. In: Italian Conference on Computational Logic. No. 598 in CEUR Workshop Proceedings, Sun SITE Central Europe (2010)
16. Riguzzi, F., Swift, T.: The PITA system: Tabling and answer subsumption for reasoning under uncertainty. Theory and Practice of Logic Programming, International Conference on Logic Programming Special Issue 11(4-5) (2011)
17. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
18. Sevon, P., Eronen, L., Hintsanen, P., Kulovesi, K., Toivonen, H.: Link discovery in graphs derived from biological databases. In: International Workshop on Data Integration in the Life Sciences. LNCS, vol. 4075, pp. 35–49. Springer (2006)
19. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
20. Vennekens, J., Verbaeten, S.: Logic programs with annotated disjunctions. Tech. Rep. CW386, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2003)
21. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)

Nonmonotonic Extensions of Low Complexity DLs: Complexity Results and Proof Methods

Laura Giordano[1], Valentina Gliozzi[2], Nicola Olivetti[3], and Gian Luca Pozzato[2]

[1] Dip. di Informatica - U. Piemonte O. - Alessandria - Italy - laura@mfn.unipmn.it
[2] Dip. Informatica - Univ. di Torino - Italy {gliozzi,pozzato}@di.unito.it
[3] LSIS-UMR CNRS 6168 - Marseille - France - nicola.olivetti@univ-cezanne.fr

Abstract. In this paper we propose nonmonotonic extensions of low complexity Description Logics $\mathcal{EL}^{\bot}$ and *DL-Lite*$_{core}$ for reasoning about typicality and defeasible properties. The resulting logics are called $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ and *DL-Lite*$_c\mathbf{T}_{min}$. We summarize complexity results for such extensions recently studied. Entailment in *DL-Lite*$_c\mathbf{T}_{min}$ is in Π_2^p, whereas entailment in $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ is EXPTIME-hard. However, considering the known fragment of Left Local $\mathcal{EL}^{\bot}\mathbf{T}_{min}$, we have that the complexity of entailment drops to Π_2^p. Furthermore, we present tableau calculi for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ (focusing on Left Local knowledge bases) and *DL-Lite*$_c\mathbf{T}_{min}$. The calculi perform a two-phase computation in order to check whether a query is minimally entailed from the initial knowledge base. The calculi are sound, complete and terminating. Furthermore, they represent decision procedures for Left Local $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ knowledge bases and *DL-Lite*$_c\mathbf{T}_{min}$ knowledge bases, whose complexities match the above mentioned results.

1 Introduction

The family of description logics (DLs) is one of the most important formalisms of knowledge representation. They have a well-defined semantics based on first-order logic and offer a good trade-off between expressivity and complexity. DLs have been successfully implemented by a range of systems and they are at the base of languages for the semantic web such as OWL. A DL knowledge base (KB) comprises two components: the TBox, containing the definition of concepts (and possibly roles), and a specification of inclusion relations among them, and the ABox containing instances of concepts and roles. Since the very objective of the TBox is to build a taxonomy of concepts, the need of representing prototypical properties and of reasoning about defeasible inheritance of such properties naturally arises.

Nonmonotonic extensions of Description Logics (DLs) have been actively investigated since the early 90s, [15, 4, 2, 3, 7, 12, 10, 9, 6]. A simple but powerful nonmonotonic extension of DLs is proposed in [12, 10, 9]: in this approach "typical" or "normal" properties can be directly specified by means of a "typicality" operator $\mathbf{T}$ enriching the underlying DL; the typicality operator $\mathbf{T}$ is essentially characterised by the core properties of nonmonotonic reasoning axiomatized by *preferential logic* [13]. In $\mathcal{ALC} + \mathbf{T}$ [12], one can consistently express defeasible inclusions and exceptions such as: typical students do not pay taxes, but working students do typically pay taxes, but working students having children normally do not: $\mathbf{T}(Student) \sqsubseteq \neg TaxPayer$; $\mathbf{T}(Student \sqcap Worker) \sqsubseteq TaxPayer$; $\mathbf{T}(Student \sqcap Worker \sqcap \exists HasChild.\top) \sqsubseteq \neg TaxPayer$. Although the operator $\mathbf{T}$ is nonmonotonic in itself, the logic $\mathcal{ALC} + \mathbf{T}$, as

well as the logic $\mathcal{EL}^{+\perp}\mathbf{T}$ [10] extending $\mathcal{EL}^{\perp}$, is monotonic. As a consequence, unless a KB contains explicit assumptions about typicality of individuals (e.g. that john is a typical student), there is no way of inferring defeasible properties of them (e.g. that john does not pay taxes). In [9], a non monotonic extension of $\mathcal{ALC}+\mathbf{T}$ based on a minimal model semantics is proposed. The resulting logic, called $\mathcal{ALC}+\mathbf{T}_{min}$, supports typicality assumptions, so that if one knows that john is a student, one can nonmonotonically assume that he is also a *typical* student and therefore that he does not pay taxes. As an example, for a TBox specified by the inclusions above, in $\mathcal{ALC}+\mathbf{T}_{min}$ the following inference holds: TBox $\cup\ \{Student(john)\} \models_{\mathcal{ALC}+\mathbf{T}_{min}} \neg TaxPayer(john)$.

Similarly to other nonmonotonic DLs, adding the typicality operator with its minimal-model semantics to a standard DL, such as $\mathcal{ALC}$, leads to a very high complexity (namely query entailment in $\mathcal{ALC}+\mathbf{T}_{min}$ is in CO-NEXP$^{\mathrm{NP}}$ [9]). This fact has motivated the study of nonmonotonic extensions of low complexity DLs such as *DL-Lite*$_{core}$ [5] and $\mathcal{EL}^{\perp}$ of the $\mathcal{EL}$ family [1] which are nonetheless well-suited for encoding large knowledge bases (KBs).

In this paper, we hence consider the extensions of the low complexity logics *DL-Lite*$_{core}$ and $\mathcal{EL}^{\perp}$ with the typicality operator based on the minimal model semantics introduced in [9]. We summarize complexity upper bounds for the resulting logics $\mathcal{EL}^{\perp}\mathbf{T}_{min}$ and *DL-Lite*$_c\mathbf{T}_{min}$ studied in [11]. For $\mathcal{EL}^{\perp}$, it turns out that its extension $\mathcal{EL}^{\perp}\mathbf{T}_{min}$ is unfortunately EXPTIME-hard. This result is analogous to the one for *circumscribed* $\mathcal{EL}^{\perp}$ KBs [3]. However, the complexity decreases to Π_2^p for the fragment of *Left Local* $\mathcal{EL}^{\perp}$ KBs, corresponding to the homonymous fragment in [3]. The same complexity upper bound is obtained for *DL-Lite*$_c\mathbf{T}_{min}$.

We also present tableau calculi for *DL-Lite*$_c\mathbf{T}_{min}$ as well as for the Left Local fragment of $\mathcal{EL}^{\perp}\mathbf{T}_{min}$ for deciding minimal entailment in Π_2^p. Our calculi perform a two-phase computation: in the first phase, candidate models (complete open branches) falsifying the given query are generated, in the second phase the minimality of candidate models is checked by means of an auxiliary tableau construction. The latter tries to build a model which is "more preferred" than the candidate one: if it fails (being closed) the candidate model is minimal, otherwise it is not. Both tableaux constructions comprise some non-standard rules for existential quantification in order to constrain the domain (and its size) of the model being constructed. The second phase makes use in addition of special closure conditions to prevent the generation of non-preferred models. The calculi are very simple and do not require any blocking machinery in order to achieve termination. It comes as a surprise that the modification of the existential rule is sufficient to match the Π_2^p complexity.

2 The typicality operator $\mathbf{T}$ and the Logic $\mathcal{EL}^{\perp}\mathbf{T}_{min}$

Before describing $\mathcal{EL}^{\perp}\mathbf{T}_{min}$, let us briefly recall the underlying monotonic logic $\mathcal{EL}^{+\perp}\mathbf{T}$ [10], obtained by adding to $\mathcal{EL}^{\perp}$ the typicality operator $\mathbf{T}$. The intuitive idea is that $\mathbf{T}(C)$ selects the *typical* instances of a concept C. In $\mathcal{EL}^{+\perp}\mathbf{T}$ we can therefore distinguish between the properties that hold for all instances of concept C ($C \sqsubseteq D$), and those that only hold for the normal or typical instances of C ($\mathbf{T}(C) \sqsubseteq D$).

Formally, the $\mathcal{EL}^{+\perp}\mathbf{T}$ language is defined as follows.

Definition 1. *We consider an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individuals $\mathcal{O}$. Given $A \in \mathcal{C}$ and $R \in \mathcal{R}$, we define*

$$C := A \mid \top \mid \bot \mid C \sqcap C \qquad C_R := C \mid C_R \sqcap C_R \mid \exists R.C \qquad C_L := C_R \mid \mathbf{T}(C)$$

A KB is a pair (TBox, ABox). TBox contains a finite set of general concept inclusions (or subsumptions) $C_L \sqsubseteq C_R$. ABox contains assertions of the form $C_L(a)$ and $R(a, b)$, where $a, b \in \mathcal{O}$.

The semantics of $\mathcal{EL}^{+\bot}\mathbf{T}$ [10] is defined by enriching ordinary models of $\mathcal{EL}^{\bot}$ by a *preference relation* $<$ on the domain, whose intuitive meaning is to compare the "typicality" of individuals: $x < y$, means that x is more typical than y. Typical members of a concept C, that is members of $\mathbf{T}(C)$, are the members x of C that are minimal with respect to this preference relation.

Definition 2 (Semantics of T). *A model $\mathcal{M}$ is any structure $\langle \Delta, <, I\rangle$ where Δ is the domain; $<$ is an irreflexive and transitive relation over Δ that satisfies the following* Smoothness Condition*: for all $S \subseteq \Delta$, for all $x \in S$, either $x \in Min_<(S)$ or $\exists y \in Min_<(S)$ such that $y < x$, where $Min_<(S) = \{u : u \in S$ and $\nexists z \in S$ s.t. $z < u\}$. Furthermore, $<$ is multilinear: if $u < z$ and $v < z$, then either $u = v$ or $u < v$ or $v < u$. I is the extension function that maps each concept C to $C^I \subseteq \Delta$, and each role r to $r^I \subseteq \Delta^I \times \Delta^I$. For concepts of $\mathcal{EL}^{\bot}$, C^I is defined in the usual way. For the $\mathbf{T}$ operator: $(\mathbf{T}(C))^I = Min_<(C^I)$.*

Given a model $\mathcal{M}$, I can be extended so that it assigns to each individual a of $\mathcal{O}$ a distinct element a^I of the domain Δ. We say that $\mathcal{M}$ satisfies an inclusion $C \sqsubseteq D$ if $C^I \subseteq D^I$, and that $\mathcal{M}$ satisfies $C(a)$ if $a^I \in C^I$ and aRb if $(a^I, b^I) \in R^I$. Moreover, $\mathcal{M}$ satisfies TBox if it satisfies all its inclusions, and $\mathcal{M}$ satisfies ABox if it satisfies all its formulas. $\mathcal{M}$ satisfies a KB (TBox,ABox), if it satisfies both its TBox and its ABox.

The operator $\mathbf{T}$ [12] is characterized by a set of postulates that are essentially a reformulation of the KLM [13] axioms of *preferential logic* $\mathbf{P}$. $\mathbf{T}$ has therefore all the "core" properties of nonmonotonic reasoning as it is axiomatised by $\mathbf{P}$. The semantics of the typicality operator can be specified by modal logic. The interpretation of $\mathbf{T}$ can be split into two parts: for any x of the domain Δ, $x \in (\mathbf{T}(C))^I$ just in case (i) $x \in C^I$, and (ii) there is no $y \in C^I$ such that $y < x$. Condition (ii) can be represented by means of an additional modality $\Box$, whose semantics is given by the preference relation $<$ interpreted as an accessibility relation. Observe that by the Smoothness Condition, $\Box$ has the properties of Gödel-Löb modal logic of provability G. The interpretation of $\Box$ in $\mathcal{M}$ is as follows: $(\Box C)^I = \{x \in \Delta \mid$ for every $y \in \Delta$, if $y < x$ then $y \in C^I\}$. We immediately get that $x \in (\mathbf{T}(C))^I$ if and only if $x \in (C \sqcap \Box\neg C)^I$. From now on, we consider $\mathbf{T}(C)$ as an abbreviation for $C \sqcap \Box\neg C$.

As mentioned in the Introduction, the main limit of $\mathcal{EL}^{+\bot}\mathbf{T}$ is that it is *monotonic*. Even if the typicality operator $\mathbf{T}$ itself is nonmonotonic (i.e. $\mathbf{T}(C) \sqsubseteq E$ does not imply $\mathbf{T}(C \sqcap D) \sqsubseteq E$), what is inferred from an $\mathcal{EL}^{+\bot}\mathbf{T}$ KB can still be inferred from any KB' with KB $\subseteq$ KB'. In order to perform nonmonotonic inferences, as done in [9], we strengthen the semantics of $\mathcal{EL}^{+\bot}\mathbf{T}$ by restricting entailment to a class of minimal (or preferred) models. We call the new logic $\mathcal{EL}^{\bot}\mathbf{T}_{min}$. Intuitively, the idea is to restrict our consideration to models that *minimize the non typical instances of a concept*.

Given a KB, we consider a finite set $\mathcal{L}_{\mathbf{T}}$ of concepts: these are the concepts whose non typical instances we want to minimize. We assume that the set $\mathcal{L}_{\mathbf{T}}$ contains at least all concepts C such that $\mathbf{T}(C)$ occurs in the KB or in the query F, where a *query* F is either an assertion $C(a)$ or an inclusion relation $C \sqsubseteq D$. As we have just said, $x \in C^I$ is typical if $x \in (\Box\neg C)^I$. Minimizing the non typical instances of C therefore means to minimize the objects not satisfying $\Box\neg C$ for $C \in \mathcal{L}_{\mathbf{T}}$. Hence, for a given model $\mathcal{M} = \langle \Delta, <, I\rangle$, we define:

$$\mathcal{M}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} = \{(x, \neg\Box\neg C) \mid x \notin (\Box\neg C)^I, \text{ with } x \in \Delta, C \in \mathcal{L}_{\mathbf{T}}\}.$$

Definition 3 (Preferred and minimal models). *Given a model $\mathcal{M} = \langle \Delta <, I\rangle$ of a knowledge base KB, and a model $\mathcal{M}' = \langle \Delta', <', I'\rangle$ of KB, we say that $\mathcal{M}$ is preferred to $\mathcal{M}'$ with respect to $\mathcal{L}_{\mathbf{T}}$, and we write $\mathcal{M} <_{\mathcal{L}_{\mathbf{T}}} \mathcal{M}'$, if (i) $\Delta = \Delta'$, (ii) $\mathcal{M}^{\Box^-}_{\mathcal{L}_{\mathbf{T}}} \subset \mathcal{M}'^{\Box^-}_{\mathcal{L}_{\mathbf{T}}}$, (iii) $a^I = a^{I'}$ for all $a \in \mathcal{O}$. $\mathcal{M}$ is a* minimal model *for KB (with respect to $\mathcal{L}_{\mathbf{T}}$) if it is a model of KB and there is no other model $\mathcal{M}'$ of KB such that $\mathcal{M}' <_{\mathcal{L}_{\mathbf{T}}} \mathcal{M}$.*

Definition 4 (Minimal Entailment in $\mathcal{EL}^{\bot}\mathbf{T}_{min}$). *A query F is minimally entailed in $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ by KB with respect to $\mathcal{L}_{\mathbf{T}}$ if F is satisfied in all models of KB that are minimal with respect to $\mathcal{L}_{\mathbf{T}}$. We write KB $\models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} F$.*

Example 1. The KB of the Introduction can be reformulated as follows in $\mathcal{EL}^{+^{\bot}}\mathbf{T}$: $TaxPayer \sqcap NotTaxPayer \sqsubseteq \bot$; $Parent \sqsubseteq \exists HasChild.\top$; $\exists HasChild.\top \sqsubseteq Parent$; $\mathbf{T}(Student) \sqsubseteq NotTaxPayer$; $\mathbf{T}(Student \sqcap Worker) \sqsubseteq TaxPayer$; $\mathbf{T}(Student \sqcap Worker \sqcap Parent) \sqsubseteq NotTaxPayer$. Let $\mathcal{L}_{\mathbf{T}} = \{Student, Student \sqcap Worker, Student \sqcap Worker \sqcap Parent\}$. Then TBox $\cup \{Student(john)\} \models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} NotTaxPayer(john)$, since $john^I \in (Student \sqcap \Box\neg Student)^I$ for all minimal models $\mathcal{M} = \langle \Delta <, I\rangle$ of the KB. In contrast, by the nonmonotonic character of minimal entailment, TBox $\cup \{Student(john), Worker(john)\} \models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} TaxPayer(john)$. Last, notice that TBox $\cup \{\exists HasChild.(Student \sqcap Worker)(jack)\} \models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} \exists HasChild.TaxPayer(jack)$. The latter shows that minimal consequence applies to implicit individuals *as well, without any ad-hoc mechanism.*

Theorem 1 (Complexity for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ KBs (Theorem 3.1 in [11])). *The problem of deciding whether KB $\models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} \alpha$ is* EXPTIME*-hard.*

In order to lower the complexity of minimal entailment in $\mathcal{EL}^{\bot}\mathbf{T}_{min}$, we consider a syntactic restriction on the KB called Left Local KBs. This restriction is similar to the one introduced in [3] for circumscribed $\mathcal{EL}^{\bot}$ KBs.

Definition 5 (Left Local knowledge base). *A Left Local KB only contains subsumptions $C^{LL}_L \sqsubseteq C_R$, where C and C_R are as in Definition 1 and:*

$C^{LL}_L := C \mid C^{LL}_L \sqcap C^{LL}_L \mid \exists R.\top \mid \mathbf{T}(C)$

There is no restriction on the ABox.

Observe that the KB in the Example 1 is Left Local, as no concept of the form $\exists R.C$ with $C \neq \top$ occurs on the left hand side of inclusions. In [11] an upper bound for the complexity of $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ Left Local KBs is provided by a small model theorem. Intuitively, what allows us to keep the size of the small model polynomial is that we reuse the same world to verify the same existential concept throughout the model. This allows us to conclude that:

Theorem 2 (Complexity for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ Left Local KBs (Theorem 3.12 in [11])). *If KB is Left Local, the problem of deciding whether KB $\models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} \alpha$ is in Π_2^p.*

3 The Logic *DL-Lite*$_c\mathbf{T}_{min}$

In this section we present the extension of the logic *DL-Lite*$_{core}$ [5] with the $\mathbf{T}$ operator. We call the resulting logic *DL-Lite*$_c\mathbf{T}_{min}$. The language of *DL-Lite*$_c\mathbf{T}_{min}$ is defined as follows.

Definition 6. *We consider an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individuals $\mathcal{O}$. Given $A \in \mathcal{C}$ and $r \in \mathcal{R}$, we define*

$$C_L := A \mid \exists R.\top \mid \mathbf{T}(A) \qquad R := r \mid r^- \qquad C_R := A \mid \neg A \mid \exists R.\top \mid \neg\exists R.\top$$

A DL-Lite$_c\mathbf{T}_{min}$ *KB is a pair (TBox, ABox). TBox contains a finite set of concept inclusions of the form $C_L \sqsubseteq C_R$. ABox contains assertions of the form $C(a)$ and $r(a, b)$, where C is a concept C_L or C_R, $r \in \mathcal{R}$, and $a, b \in \mathcal{O}$.*

As for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$, a model $\mathcal{M}$ for *DL-Lite*$_c\mathbf{T}_{min}$ is any structure $\langle \Delta, <, I\rangle$, defined as in Definition 2, where I is extended to take care of inverse roles: given $r \in \mathcal{R}$, $(r^-)^I = \{(a, b) \mid (b, a) \in r^I\}$.

In [11] it has been shown that a small model construction similar to the one for Left Local $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ KBs can be made also for *DL-Lite*$_c\mathbf{T}_{min}$. As a difference, in this case, we exploit the fact that, for each atomic role r, the same element of the domain can be used to satisfy all occurrences of the existential $\exists r.\top$. Also, the same element of the domain can be used to satisfy all occurrences of the existential $\exists r^-.\top$.

Theorem 3 (Complexity for *DL-Lite*$_c\mathbf{T}_{min}$ KBs (Theorem 4.6 in [11])). *The problem of deciding whether KB $\models_{\textit{DL-Lite}_c\mathbf{T}_{min}} \alpha$ is in Π_2^p.*

4 The Tableau Calculus for Left Local $\mathcal{EL}^{\bot}\mathbf{T}_{min}$

In this section we present a tableau calculus $\mathcal{TAB}_{min}^{\mathcal{EL}^{\bot}\mathbf{T}}$ for deciding whether a query F is minimally entailed from a Left Local knowledge base in the logic $\mathcal{EL}^{\bot}\mathbf{T}_{min}$. It performs a two-phase computation: in the first phase, a tableau calculus, called $\mathcal{TAB}_{PH1}^{\mathcal{EL}^{\bot}\mathbf{T}}$, simply verifies whether KB $\cup \{\neg F\}$ is satisfiable in an $\mathcal{EL}^{\bot}\mathbf{T}$ model, building candidate models; in the second phase another tableau calculus, called $\mathcal{TAB}_{PH2}^{\mathcal{EL}^{\bot}\mathbf{T}}$, checks whether the candidate models found in the first phase are *minimal* models of KB, i.e. for each open branch of the first phase, $\mathcal{TAB}_{PH2}^{\mathcal{EL}^{\bot}\mathbf{T}}$ tries to build a model of KB which is preferred to the candidate model w.r.t. Definition 3. The whole procedure $\mathcal{TAB}_{min}^{\mathcal{EL}^{\bot}\mathbf{T}}$ is formally defined at the end of this section (Definition 8).

As usual, $\mathcal{TAB}_{min}^{\mathcal{EL}^{\bot}\mathbf{T}}$ tries to build an open branch representing a minimal model satisfying KB $\cup \{\neg F\}$. The negation of a query $\neg F$ is defined as follows: if $F \equiv C(a)$, then $\neg F \equiv (\neg C)(a)$; if $F \equiv C \sqsubseteq D$, then $\neg F \equiv (C \sqcap \neg D)(x)$, where x does not occur in KB. Notice that we introduce the connective $\neg$ in a very "localized" way. This is very different from introducing the negation all over the knowledge base, and indeed it does not imply that we jump out of the language of $\mathcal{EL}^{\bot}\mathbf{T}_{min}$.

$\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ makes use of labels, which are denoted with $x, y, z, \ldots$. Labels represent either a variable or an individual of the ABox, that is to say an element of $\mathcal{O} \cup \mathcal{V}$. These labels occur in *constraints* (or *labelled* formulas), that can have the form $x \xrightarrow{R} y$ or $x : C$, where x, y are labels, R is a role and C is either a concept or the negation of a concept of $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ or has the form $\Box\neg D$ or $\neg\Box\neg D$, where D is a concept.

Let us now analyze the two components of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$, starting with $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$.

4.1 First Phase: the tableaux calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{\mathbf{PH1}}$

A tableau of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ is a tree whose nodes are tuples $\langle S \mid U \mid W\rangle$. S is a set of constraints, whereas U contains formulas of the form $C \sqsubseteq D^L$, representing subsumption relations $C \sqsubseteq D$ of the TBox. L is a list of labels, used in order to ensure the termination of the tableau calculus. W is a set of labels x_C used in order to build a "small" model, matching the construction of Theorem 3.11 in [11]. A branch is a sequence of nodes $\langle S_1 \mid U_1 \mid W_1\rangle, \langle S_2 \mid U_2 \mid W_2\rangle, \ldots, \langle S_n \mid U_n \mid W_n\rangle \ldots$, where each node $\langle S_i \mid U_i \mid W_i\rangle$ is obtained from its immediate predecessor $\langle S_{i-1} \mid U_{i-1} \mid W_{i-1}\rangle$ by applying a rule of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, having $\langle S_{i-1} \mid U_{i-1} \mid W_{i-1}\rangle$ as the premise and $\langle S_i \mid U_i \mid W_i\rangle$ as one of its conclusions. A branch is closed if one of its nodes is an instance of a (Clash) axiom, otherwise it is open. A tableau is closed if all its branches are closed.

The calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ is different in two respects from the calculus $\mathcal{ALC} + \mathbf{T}_{min}$ presented in [9]. First, the rule $(\exists^+)$ is split in the following two rules:

$$\frac{\langle S, u : \exists R.C \mid U \mid W\rangle}{\langle S, u \xrightarrow{R} x_C, x_C : C \mid U \mid W \cup \{x_C\}\rangle \quad \langle S, u \xrightarrow{R} y_1, y_1 : C \mid U \mid W\rangle \cdots \langle S, u \xrightarrow{R} y_m, y_m : C \mid U \mid W\rangle}(\exists^+)_1$$

if $x_C \notin W$ and $y_1, \ldots, y_m$ are all the labels occurring in S

$$\frac{\langle S, u : \exists R.C \mid U \mid W\rangle}{\langle S, u \xrightarrow{R} x_C \mid U \mid W\rangle \quad \langle S, u \xrightarrow{R} y_1, y_1 : C \mid U \mid W\rangle \cdots \langle S, u \xrightarrow{R} y_m, y_m : C \mid U \mid W\rangle}(\exists^+)_2$$

if $x_C \in W$ and $y_1, \ldots, y_m$ are all the labels occurring in S

When the rule $(\exists^+)_1$ is applied to a formula $u : \exists R.C$, it introduces a new label x_C only when the set W does not already contain x_C. Otherwise, since x_C has been already introduced in that branch, $u \xrightarrow{R} x_C$ is added to the conclusion of the rule rather than introducing a new label. As a consequence, in a given branch, $(\exists^+)_1$ only introduces a new label x_C for each concept C occurring in the initial KB in some $\exists R.C$, and no blocking machinery is needed to ensure termination. As it will become clear in the proof of Theorem 4, this is possible since we are considering Left Local KBs, which have small models; in these models all existentials $\exists R.C$ occurring in KB are made true by reusing a single witness x_C (Theorem 3.12 in [11]). Notice also that the rules $(\exists^+)_1$ and $(\exists^+)_2$ introduce a branching on the choice of the label used to realize the existential restriction $u : \exists R.C$: just the leftmost conclusion of $(\exists^+)_1$ introduces a new label (as mentioned, the x_C such that $x_C : C$ and $u \xrightarrow{R} x_C$ are added to the branch); in all the other branches, each one of the other labels y_i occurring in S may be chosen.

Second, in order to build multilinear models of Definition 2, the calculus adopts a strengthened version of the rule $(\Box^-)$ used in $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ [9]. We write $\overline{S}$ as an

abbreviation for $S, u : \neg\Box\neg C_1, \ldots, u : \neg\Box\neg C_n$. Moreover, we define $S^M_{u\rightarrow y} = \{y : \neg D, y : \Box\neg D \mid u : \Box\neg D \in S\}$ and, for $k = 1, 2, \ldots, n$, we define $\overline{S}^{\Box^{-}k}_{u\rightarrow y} = \{y : \neg\Box\neg C_j \sqcup C_j \mid u : \neg\Box\neg C_j \in \overline{S} \wedge j \neq k\}$. The strengthened rule $(\Box^-)$ is as follows:

$$\frac{\langle S, u : \neg\Box\neg C_1, u : \neg\Box\neg C_2, \ldots, u : \neg\Box\neg C_n \mid U \mid W\rangle}{\langle S, x : C_k, x : \Box\neg C_k, S^M_{u\rightarrow x}, \overline{S}^{\Box^{-}k}_{u\rightarrow x} \mid U \mid W\rangle \quad \langle S, y_1 : C_k, y_1 : \Box\neg C_k, S^M_{u\rightarrow y_1}, \overline{S}^{\Box^{-}k}_{u\rightarrow y_1} \mid U \mid W\rangle \;\cdots\; \langle S, y_m : C_k, y_m : \Box\neg C_k, S^M_{u\rightarrow y_m}, \overline{S}^{\Box^{-}k}_{u\rightarrow y_m} \mid U \mid W\rangle} \; (\Box^-)$$

for all $k = 1, 2, \ldots, n$, where $y_1, \ldots, y_m$ are all the labels occurring in S and x is new.

Rule $(\Box^-)$ contains: n branches, one for each $u : \neg\Box\neg C_k$ in $\overline{S}$; in each branch a *new* typical C_k individual x is introduced (i.e. $x : C_k$ and $x : \Box\neg C_k$ are added), and for all other $u : \neg\Box\neg C_j$, either $x : C_j$ holds or the formula $x : \neg\Box\neg C_j$ is recorded; - other $n \times m$ branches, where m is the number of labels occurring in S, one for each label y_i and for each $u : \neg\Box\neg C_k$ in $\overline{S}$; in these branches, a given y_i is chosen as a typical instance of C_k, that is to say $y_i : C_k$ and $y_i : \Box\neg C_k$ are added, and for all other $u : \neg\Box\neg C_j$, either $y_i : C_j$ holds or the formula $y_i : \neg\Box\neg C_j$ is recorded. This rule is sound with respect to multilinear models. The advantage of this rule over the $(\Box^-)$ rule in the calculus $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ is that all the negated box formulas labelled by u are treated in one step, introducing only a new label x in (some of) the conclusions. Notice that in order to keep $\overline{S}$ readable, we have used $\sqcup$. This is the reason why our calculi contain the rule for $\sqcup$, even if this constructor does not belong to $\mathcal{EL}^{\bot}\mathbf{T}_{min}$.

In order to check the satisfiability of a KB, we build its *corresponding constraint system* $\langle S \mid U \mid \emptyset\rangle$, and we check its satisfiability. Given *KB*=(TBox,ABox), its *corresponding constraint system* $\langle S \mid U \mid \emptyset\rangle$ is defined as follows: $S = \{a : C \mid C(a) \in ABox\} \cup \{a \xrightarrow{R} b \mid R(a, b) \in ABox\}$; $U = \{C \sqsubseteq D^{\emptyset} \mid C \sqsubseteq D \in TBox\}$.

Definition 7 (Model satisfying a constraint system). *Let $\mathcal{M} = \langle\Delta, I, <\rangle$ be a model as in Definition 2. We define a function α which assigns to each variable of $\mathcal{V}$ an element of Δ, and assigns every individual $a \in \mathcal{O}$ to $a^I \in \Delta$. $\mathcal{M}$ satisfies a constraint F under α, written $\mathcal{M} \models_\alpha F$, as follows: (i) $\mathcal{M} \models_\alpha x : C$ iff $\alpha(x) \in C^I$; (ii) $\mathcal{M} \models_\alpha x \xrightarrow{R} y$ iff $(\alpha(x), \alpha(y)) \in R^I$. A constraint system $\langle S \mid U \mid W\rangle$ is satisfiable if there is a model $\mathcal{M}$ and a function α such that $\mathcal{M}$ satisfies every constraint in S under α and that, for all $C \sqsubseteq D^L \in U$ and for all $x \in \Delta$, we have that if $x \in C^I$ then $x \in D^I$.*

Given a KB=(TBox,ABox), it is satisfiable if and only if its corresponding constraint system $\langle S \mid U \mid \emptyset\rangle$ is satisfiable. In order to verify the satisfiability of KB $\cup \{\neg F\}$, we use $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ to check the satisfiability of the constraint system $\langle S \mid U \mid \emptyset\rangle$ obtained by adding the constraint corresponding to $\neg F$ to S', where $\langle S' \mid U \mid \emptyset\rangle$ is the corresponding constraint system of KB. To this purpose, the rules of the calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ are applied until either a contradiction is generated (Clash) or a model satisfying $\langle S \mid U \mid \emptyset\rangle$ can be obtained from the resulting constraint system.

Given a node $\langle S \mid U \mid W\rangle$, for each subsumption $C \sqsubseteq D^L \in U$ and for each label x that appears in the tableau, we add to S the constraint $x : \neg C \sqcup D$: we refer to this mechanism as *unfolding*. As mentioned above, each formula $C \sqsubseteq D$ is equipped with a list L of labels in which it has been unfolded in the current branch. This is needed to

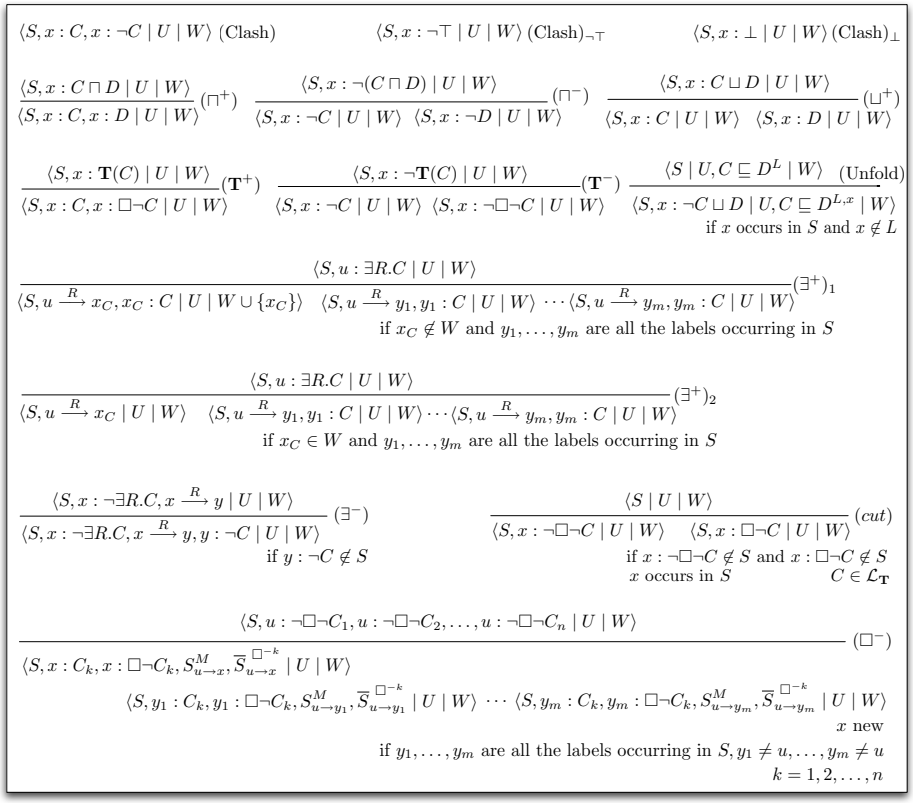


Fig. 1. The calculus $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH1}$.

avoid multiple unfolding of the same subsumption by using the same label, generating infinite branches.

Before introducing the rules of $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH1}$ we need some more definitions. First, we define an ordering relation $\prec$ to keep track of the temporal ordering of insertion of labels in the tableau, that is to say if y is introduced in the tableau, then $x \prec y$ for all labels x that are already in the tableau. Furthermore, if x is the label occurring in the query F, then $x \prec y$ for all y occurring in the constraint system corresponding to the initial KB. The rules of $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH1}$ are presented in Figure 1. Rules $(\exists^+_1)$ and $(\Box^-)$ are called *dynamic* since they can introduce a new variable in their conclusions. The other rules are called *static*. We do not need any extra rule for the positive occurrences of the $\Box$ operator, since these are taken into account by the computation of $S^M_{x\to y}$ of $(\Box^-)$. The (cut) rule ensures that, given any concept $C \in \mathcal{L}_{\mathbf{T}}$, an open branch built by $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH1}$ contains either $x : \Box\neg C$ or $x : \neg\Box\neg C$ for each label x: this is needed in order to allow $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH2}$ to check the minimality of the model corresponding to the open branch.

The rules of $\mathcal{TAB}^{\mathcal{EL}^{\perp}\mathbf{T}}_{PH1}$ are applied with the following *standard strategy*: 1. apply a rule to a label x only if no rule is applicable to a label y such that $y \prec x$; 2. apply

dynamic rules only if no static rule is applicable. In [8] it has been shown that the calculus is sound and complete with respect to the semantics in Definition 7 and it ensures termination:

Theorem 4 (Soundness and completeness of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ [8]). *If KB $\not\models_{\mathcal{EL}^{\bot}\mathbf{T}_{min}} F$, then the tableau for the constraint system corresponding to KB $\cup \{\neg F\}$ contains an open saturated branch, which is satisfiable (via an injective assignment from labels to domain elements) in a minimal model of KB. Given a constraint system $\langle S \mid U \mid W\rangle$, if it is unsatisfiable, then it has a closed tableau in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$.*

Theorem 5 (Termination of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ [8]). *Any tableau generated by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ for $\langle S \mid U \mid \emptyset\rangle$ is finite.*

Let us conclude this section by estimating the complexity of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$. Let n be the size of the initial KB, i.e. the length of the string representing KB, and let $\langle S \mid U \mid \emptyset\rangle$ be its corresponding constraint system. We assume that the size of F and $\mathcal{L}_{\mathbf{T}}$ is $O(n)$. The calculus builds a tableau for $\langle S \mid U \mid \emptyset\rangle$ whose branches's size is $O(n)$. This immediately follows from the fact that dynamic rules $(\exists^{+})_1$ and $(\Box^{-})$ generate at most $O(n)$ labels in a branch. Indeed, the rule $(\exists^{+})_1$ introduces a new label x_C for each concept C occurring in KB, then at most $O(n)$ labels. Concerning $(\Box^{-})$, consider a branch generated by its application to a constraint system $\langle S, u : \neg\Box\neg C_1 \ldots, u : \neg\Box\neg C_n \mid U \mid W\rangle$. In the worst case, a new label x_1 is introduced. Suppose also that the branch under consideration is the one containing $x_1 : C_1$ and $x_1 : \Box\neg C_1$. The $(\Box^{-})$ rule can then be applied to formulas $u : \neg\Box\neg C_k$, introducing also a further new label x_2. However, by the presence of $x_1 : \Box\neg C_1$, the rule $(\Box^{-})$ can no longer consistently introduce $x_2 : \neg\Box\neg C_1$, since $x_2 : \Box\neg C_1 \in S^{M}_{x_1 \to x_2}$. Therefore, $(\Box^{-})$ is applied to $\neg\Box\neg C_1 \ldots \neg\Box\neg C_n$ in u. This application generates (at most) one new world x_1 that labels (at most) $n-1$ negated boxed formulas. A further application of $(\Box^{-})$ to $\neg\Box\neg C_1 \ldots \neg\Box\neg C_{n-1}$ in x_1 generates (at most) one new world x_2 that labels (at most) $n-2$ negated boxed formulas, and so on. Overall, at most $O(n)$ new labels are introduced by $(\Box^{-})$ in each branch. For each of these labels, static rules apply at most $O(n)$ times: (Unfold) is applied at most $O(n)$ times for each $C \sqsubseteq D \in U$, one for each label introduced in the branch. The rule (cut) is also applied at most $O(n)$ times for each label, since $\mathcal{L}_{\mathbf{T}}$ contains at most $O(n)$ formulas. As the number of different concepts in KB is at most $O(n)$, in all steps involving the application of boolean rules, there are at most $O(n)$ applications of these rules. Therefore, the length of the tableau branch built by the strategy is $O(n^2)$. Finally, we observe that all the nodes of the tableau contain a number of formulas which is polynomial in n, therefore to test whether a node is an instance of a (Clash) axiom has at most complexity polynomial in n.

Theorem 6 (Complexity of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$). *Given a KB and a query F, the problem of checking whether KB $\cup \{\neg F\}$ in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ is satisfiable is in* NP.

4.2 The tableaux calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$

Let us now introduce the calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ which, for each open branch **B** built by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, verifies whether it represents a minimal model of the KB. Given an open

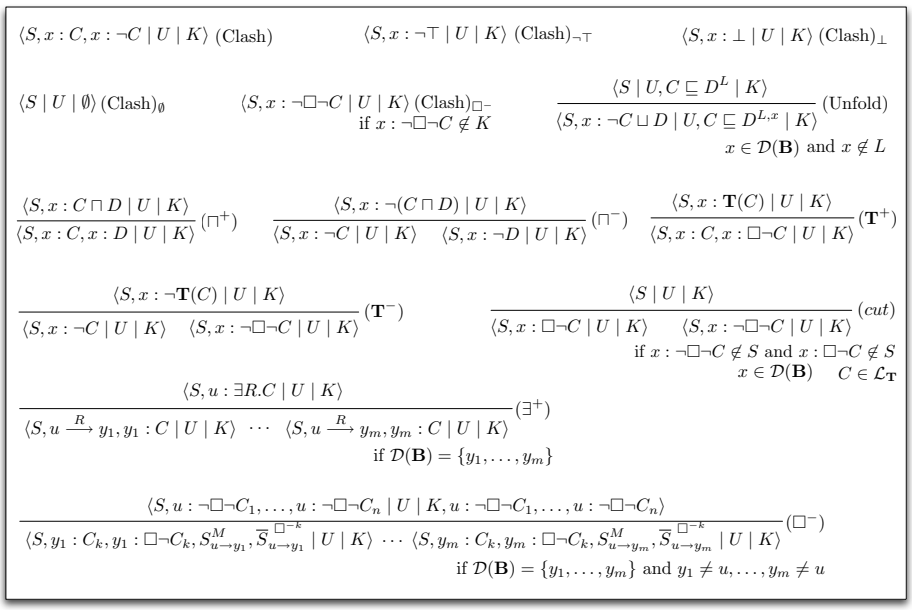


Fig. 2. The calculus $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$. To save space, we omit the rule $(\sqcup^+)$.

branch $\mathbf{B}$ of a tableau built from $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH1}$, let $\mathcal{D}(\mathbf{B})$ be the set of labels occurring on $\mathbf{B}$. Moreover, let $\mathbf{B}^{\Box^-}$ be the set of formulas $x : \neg\Box\neg C$ occurring in $\mathbf{B}$, that is to say $\mathbf{B}^{\Box^-} = \{x : \neg\Box\neg C \mid x : \neg\Box\neg C \text{ occurs in } \mathbf{B}\}$.

A tableau of $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ is a tree whose nodes are tuples of the form $\langle S \mid U \mid K\rangle$, where S and U are defined as in a constraint system, whereas K contains formulas of the form $x : \neg\Box\neg C$, with $C \in \mathcal{L}_{\mathbf{T}}$. The basic idea of $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ is as follows. Given an open branch $\mathbf{B}$ built by $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH1}$ and corresponding to a model $\mathcal{M}^{\mathbf{B}}$ of KB $\cup \{\neg F\}$, $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ checks whether $\mathcal{M}^{\mathbf{B}}$ is a minimal model of KB by trying to build a model of KB which is preferred to $\mathcal{M}^{\mathbf{B}}$. To this purpose, it keeps track (in K) of the negated box used in $\mathbf{B}$ ($\mathbf{B}^{\Box^-}$) in order to check whether it is possible to build a model of KB containing less negated box formulas. The tableau built by $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ closes if it is not possible to build a model smaller than $\mathcal{M}^{B}$, it remains open otherwise. Since by Definition 3 two models can be compared only if they have the same domain, $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ tries to build an open branch containing all the labels appearing on $\mathbf{B}$, i.e. those in $\mathcal{D}(\mathbf{B})$. To this aim, the dynamic rules use labels in $\mathcal{D}(\mathbf{B})$ instead of introducing new ones in their conclusions. The rules of $\mathcal{TAB}^{\mathcal{ELO}^{\bot}\mathbf{T}}_{PH2}$ are shown in Fig. 2.

More in detail, the rule $(\exists^+)$ is applied to a constraint system containing a formula $x : \exists R.C$; it introduces $x \xrightarrow{R} y$ and $y : C$ where $y \in \mathcal{D}(\mathbf{B})$, instead of y being a new label. The choice of the label y introduces a branching in the tableau construction. The rule (Unfold) is applied to *all the labels of* $\mathcal{D}(\mathbf{B})$ (and not only to those appearing in the branch). The rule $(\Box^-)$ is applied to a node $\langle S, u : \neg\Box\neg C_1, \ldots, u : \neg\Box\neg C_n \mid U \mid K\rangle$, when $\{u : \neg\Box\neg C_1, \ldots, u : \neg\Box\neg C_n\} \subseteq K$, i.e. when the negated box formulas

$u : \neg\Box\neg C_i$ also belong to the open branch **B**. Even in this case, the rule introduces a branch on the choice of the individual $y_i \in \mathcal{D}(\mathbf{B})$ to be used in the conclusion. In case a tableau node has the form $\langle S, x : \neg\Box\neg C \mid U \mid K\rangle$, and $x : \neg\Box\neg C \notin K$, then $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ detects a clash, called $(\text{Clash})_{\Box^-}$: this corresponds to the situation where $x : \neg\Box\neg C$ does not belong to **B**, while the model corresponding to the branch being built contains $x : \neg\Box\neg C$, and hence is *not* preferred to the model represented by **B**.

The calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ also contains the clash condition $(\text{Clash})_{\emptyset}$. Since each application of $(\Box^-)$ removes the negated box formulas $x : \neg\Box\neg C_i$ from the set K, when K is empty all the negated boxed formulas occurring in **B** also belong to the current branch. In this case, the model built by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ satisfies the same set of $x : \neg\Box\neg C_i$ (for all individuals) as **B** and, thus, it is not preferred to the one represented by **B**.

Theorem 7 (Soundness and completeness of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ [8]). *Given a* KB *and a query F, let $\langle S' \mid U \mid \emptyset\rangle$ be the corresponding constraint system of* KB*, and $\langle S \mid U \mid \emptyset\rangle$ the corresponding constraint system of* $\text{KB} \cup \{\neg F\}$*. An open branch* **B** *built by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ for $\langle S \mid U \mid \emptyset\rangle$ is satisfiable by an injective mapping in a minimal model of* KB *iff the tableau in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ for $\langle S' \mid U \mid \mathbf{B}^{\Box^-}\rangle$ is closed.*

$\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ always terminates. Termination is ensured by the fact that dynamic rules make use of labels belonging to $\mathcal{D}(\mathbf{B})$, which is finite, rather than introducing "new" labels in the tableau.

Theorem 8 (Termination of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$). *Let $\langle S' \mid U \mid \mathbf{B}^{\Box^-}\rangle$ be a constraint system starting from an open branch* **B** *built by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, then any tableau generated by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ is finite.*

It is possible to show that the problem of verifying that a branch **B** represents a minimal model for KB in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ is in NP in the size of **B**.

The overall procedure $\mathcal{TAB}^{\mathcal{ALC}+\mathbf{T}}_{min}$ is defined as follows:

Definition 8. *Let KB be a knowledge base whose corresponding constraint system is $\langle S \mid U \mid \emptyset\rangle$. Let F be a query and let S' be the set of constraints obtained by adding to S the constraint corresponding to $\neg F$. The calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ checks whether a query F is minimally entailed from a* KB *by means of the following procedure:* (phase 1) *the calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ is applied to $\langle S' \mid U \mid \emptyset\rangle$; if, for each branch* **B** *built by $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, either (i)* **B** *is closed or (ii)* (phase 2) *the tableau built by the calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ for $\langle S \mid U \mid \mathbf{B}^{\Box^-}\rangle$ is open, then* $\text{KB} \models^{\mathcal{L}_{\mathbf{T}}}_{min} F$*, otherwise* $\text{KB} \not\models^{\mathcal{L}_{\mathbf{T}}}_{min} F$*.*

Theorem 9 (Soundness and completeness of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ [8]). *$\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ is a sound and complete decision procedure for verifying if* $\text{KB} \models^{\mathcal{L}_{\mathbf{T}}}_{min} F$*.*

The complexity of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ matches the results of Theorem 2. Consider the complementary problem: $\text{KB} \not\models^{\mathcal{L}_{\mathbf{T}}}_{min} F$. This problem can be solved according to the procedure in Definition 8: by nondeterministically generating an open branch of polynomial length in the size of KB in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$ (a model $\mathcal{M}^{\mathbf{B}}$ of $\text{KB} \cup \{\neg F\}$), and then by

calling an NP oracle which verifies that $\mathcal{M}^{\mathbf{B}}$ is a minimal model of KB. In fact, the verification that $\mathcal{M}^{\mathbf{B}}$ is not a minimal model of the KB can be done by an NP algorithm which nondeterministically generates a branch in $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH2}$ of polynomial size in the size of $\mathcal{M}^{\mathbf{B}}$ (and of KB), representing a model $\mathcal{M}^{B'}$ of KB preferred to $\mathcal{M}^{\mathbf{B}}$. Hence, the problem of verifying that KB $\not\models^{\mathcal{L}_{\mathbf{T}}}_{min} F$ is in NP^{NP}, i.e. in Σ^p_2, and the problem of deciding whether KB $\models^{\mathcal{L}_{\mathbf{T}}}_{min} F$ is in CO-NP^{NP}, i.e. in Π^p_2.

Theorem 10 (Complexity of $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$). *The problem of deciding whether* KB $\models^{\mathcal{L}_{\mathbf{T}}}_{min} F$ *by means of* $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{min}$ *is in* Π^p_2.

5 A Tableau Calculus for *DL-Lite*$_c\mathbf{T}_{min}$

In this section we present a tableau calculus $\mathcal{TAB}^{Lite_c\mathbf{T}}_{min}$ for deciding query entailment in the logic *DL-Lite*$_c\mathbf{T}_{min}$. The calculus is similar to the one for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$ in the previous section, however it contains a few significant differences. Let us analyze in detail the two components of $\mathcal{TAB}^{Lite_c\mathbf{T}}_{min}$.

5.1 First Phase: the tableaux calculus $\mathcal{TAB}^{Lite_c\mathbf{T}}_{PH1}$

The calculus $\mathcal{TAB}^{Lite_c\mathbf{T}}_{PH1}$ is significantly different in three respects from the calculus for $\mathcal{EL}^{\bot}\mathbf{T}_{min}$. We try to explain such differences in detail. First of all, given a set of constraints S and a role $r \in \mathcal{R}$, we define $r(S) = \{x \xrightarrow{r} y \mid x \xrightarrow{r} y \in S\}$.

1. The rule $(\exists^+)$ is split in the following two rules:

$$\frac{\langle S, x : \exists r.\top \mid U\rangle}{\langle S, x \xrightarrow{r} y \mid U\rangle \quad \langle S, x \xrightarrow{r} y_1 \mid U\rangle \cdots \langle S, x \xrightarrow{r} y_m \mid U\rangle}\,(\exists^+)^r_1$$

if $r(S) = \emptyset$, y new, if $y_1, \ldots, y_m$ are all the labels occurring in S

$$\frac{\langle S, x : \exists r.\top \mid U\rangle}{\langle S, x \xrightarrow{r} y_1 \mid U\rangle \ \cdots \ \langle S, x \xrightarrow{r} y_m \mid U\rangle}\,(\exists^+)^r_2$$

if $r(S) \neq \emptyset$, if $y_1, \ldots, y_m$ are all the labels occurring in S

As in the calculus $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, the split of the $(\exists^+)$ in the two rules above reflects the main idea of the construction of a small model at the base of Theorem 4.5 in [11]. Such small model theorem essentially shows that *DL-Lite*$_c\mathbf{T}_{min}$ KBs can have small models in which all existentials $\exists R.\top$ occurring in KB are made true in the model by reusing a single witness y. In the calculus we use the same idea: when the rule $(\exists^+)^r_1$ is applied to a formula $x : \exists r.\top$, it introduces a new label y and the constraint $x \xrightarrow{r} y$ only when there is no other previous constraint $u \xrightarrow{r} v$ in S, i.e. $r(S) = \emptyset$. Otherwise, rule $(\exists^+)^r_2$ is applied and it introduces $x \xrightarrow{r} y$. As a consequence, $(\exists^+)^r_2$ does not introduce any new label in the branch whereas $(\exists^+)^r_1$ only introduces a new label y for each role r occurring in the initial KB in some $\exists r.\top$ or $\exists r^-.\top$, and no blocking machinery is needed to ensure termination.

2. In order to keep into account inverse roles, two further rules for existential formulas are introduced:

$$\frac{\langle S, x : \exists r^-.\top \mid U\rangle}{\langle S, y \xrightarrow{r} x \mid U\rangle\ \langle S, y_1 \xrightarrow{r} x \mid U\rangle \cdots \langle S, y_m \xrightarrow{r} x \mid U\rangle}\,(\exists^+)^{r^-}_1$$

if $r(S) = \emptyset$, y new, if $y_1, \ldots, y_m$ are all the labels occurring in S

$$\frac{\langle S, x : \exists r^-.\top \mid U\rangle}{\langle S, y_1 \xrightarrow{r} x \mid U\rangle \cdots \langle S, y_m \xrightarrow{r} x \mid U\rangle}\,(\exists^+)^{r^-}_2$$

if $r(S) \neq \emptyset$, if $y_1, \ldots, y_m$ are all the labels occurring in S

These rules work similarly to $(\exists^+)^r_1$ and $(\exists^+)^r_2$ in order to build a branch representing a small model: when the rule $(\exists^+)^{r^-}_1$ is applied to a formula $x : \exists r^-.\top$, it introduces a new label y and the constraint $y \xrightarrow{r} x$ only when there is no other constraint $u \xrightarrow{r} v$ in S. Otherwise, since a constraint $y \xrightarrow{r} u$ has been already introduced in that branch, $y \xrightarrow{r} x$ is added to the conclusion of the rule.

3. Negated existential formulas can occur in a branch, but only having the form (i) $x : \neg\exists r.\top$ or (ii) $x : \neg\exists r^-.\top$. (i) means that x has no relationships with other individuals via the role r, i.e. we need to detect a contradiction if both (i) and, for some y, $x \xrightarrow{r} y$ belong to the same branch, in order to mark the branch as closed. The clash condition $(\text{Clash})_r$ is added to the calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$ in order to detect such a situation. Analogously, (ii) means that there is no y such that y is related to x by means of r, then $(\text{Clash})_{r^-}$ is introduced in order to close a branch containing both (ii) and, for some y, a constraint $y \xrightarrow{r} x$. These clash conditions are as follows:

$$\langle S, x \xrightarrow{r} y, x : \neg\exists r.\top \mid U\rangle \ (\text{Clash})_r \qquad\qquad \langle S, y \xrightarrow{r} x, x : \neg\exists r^-.\top \mid U\rangle \ (\text{Clash})_{r^-}$$

The rules of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$ are presented in Figure 3. The calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$ is sound, complete and terminating.

$$\langle S, x : C, x : \neg C \mid U\rangle \ (\text{Clash}) \qquad \langle S, x \xrightarrow{r} y, x : \neg\exists r.\top \mid U\rangle \ (\text{Clash})_r \qquad \langle S, y \xrightarrow{r} x, x : \neg\exists r^-.\top \mid U\rangle \ (\text{Clash})_{r^-}$$

$$\frac{\langle S, x : \exists r.\top \mid U\rangle}{\langle S, x \xrightarrow{r} y \mid U\rangle \ \langle S, x \xrightarrow{r} y_1 \mid U\rangle \cdots \langle S, x \xrightarrow{r} y_m \mid U\rangle}(\exists^+)^r_1 \quad \begin{array}{l}\text{if } r(S) = \emptyset\\ y \text{ new}\\ \text{if } y_1, \ldots, y_m \text{ are all the labels occurring in } S\end{array}$$

$$\frac{\langle S, x : \exists r.\top \mid U\rangle}{\langle S, x \xrightarrow{r} y_1 \mid U\rangle \cdots \langle S, x \xrightarrow{r} y_m \mid U\rangle}(\exists^+)^r_2 \quad \begin{array}{l}\text{if } r(S) \neq \emptyset\\ \text{if } y_1, \ldots, y_m \text{ are all the labels occurring in } S\end{array}$$

$$\frac{\langle S, x : \mathbf{T}(C) \mid U\rangle}{\langle S, x : C, x : \Box\neg C \mid U\rangle}(\mathbf{T}^+)$$

$$\frac{\langle S, x : \neg\mathbf{T}(C) \mid U\rangle}{\langle S, x : \neg C \mid U\rangle \ \langle S, x : \neg\Box\neg C \mid U\rangle}(\mathbf{T}^-)$$

$$\frac{\langle S \mid U\rangle}{\langle S, x : \Box\neg C \mid U\rangle \ \langle S, x : \neg\Box\neg C \mid U\rangle}(cut) \quad \begin{array}{l}\text{if } x : \neg\Box\neg C \notin S \text{ and } x : \Box\neg C \notin S\\ C \in \mathcal{L}_{\mathbf{T}}\\ x \text{ occurs in } S\end{array}$$

$$\frac{\langle S \mid U, C \sqsubseteq D^L\rangle}{\langle S, x : \neg C \sqcup D \mid U, C \sqsubseteq D^{L,x}\rangle}(\text{Unfold}) \quad \text{if } x \text{ occurs in } S \text{ and } x \notin L$$

$$\frac{\langle S, x : \exists r^-.\top \mid U\rangle}{\langle S, y \xrightarrow{r} x \mid U\rangle \ \langle S, y_1 \xrightarrow{r} x \mid U\rangle \cdots \langle S, y_m \xrightarrow{r} x \mid U\rangle}(\exists^+)^{r^-}_1 \quad \begin{array}{l}\text{if } r(S) = \emptyset\\ y \text{ new}\\ \text{if } y_1, \ldots, y_m \text{ are all the labels occurring in } S\end{array}$$

$$\frac{\langle S, x : \exists r^-.\top \mid U\rangle}{\langle S, y_1 \xrightarrow{r} x \mid U\rangle \cdots \langle S, y_m \xrightarrow{r} x \mid U\rangle}(\exists^+)^{r^-}_2 \quad \begin{array}{l}\text{if } r(S) \neq \emptyset\\ \text{if } y_1, \ldots, y_m \text{ are all the labels occurring in } S\end{array}$$

$$\frac{\langle S, x : C \sqcup D \mid U\rangle}{\langle S, x : C \mid U\rangle \ \langle S, x : D \mid U\rangle}(\sqcup^+)$$

$$\frac{\langle S, x : \neg\Box\neg C_1, \ldots, \neg\Box\neg C_n \mid U\rangle}{\langle S, y : C_k, y : \Box\neg C_k, S^M_{x\to y}, \overline{S}^{\Box^-k}_{x\to y} \mid U\rangle \ \langle S, y_1 : C_k, y_1 : \Box\neg C_k, S^M_{x\to y_1}, \overline{S}^{\Box^-k}_{x\to y_1} \mid U\rangle \cdots \langle S, y_m : C_k, y_m : \Box\neg C_k, S^M_{x\to y_m}, \overline{S}^{\Box^-k}_{x\to y_m} \mid U\rangle}(\Box^-)$$

$$\begin{array}{l}y \text{ new}\\ \text{if } y_1, \ldots, y_m \text{ are all the labels occurring in } S, y_1 \neq x, \ldots, y_m \neq x\\ \forall\, k = 1, 2, \ldots, n\end{array}$$

Fig. 3. The calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$.

Theorem 11 (Soundness and completeness of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$). *If KB $\models_{\text{DL-Lite}_c\mathbf{T}_{min}} F$, then the tableau for the constraint system corresponding to KB $\cup \{\neg F\}$ contains an open saturated branch, which is satisfiable (via an injective assignment from labels to domain elements) in a minimal model of KB. Given a constraint system $\langle S \mid U\rangle$, if it is unsatisfiable, then it has a closed tableau in $\mathcal{TAB}^{\text{Lite}_c\mathbf{T}}_{PH1}$.*

Theorem 12 (Termination of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$). *Any tableau generated by $\mathcal{TAB}^{\text{Lite}_c\mathbf{T}}_{PH1}$ for $\langle S \mid U\rangle$ is finite.*

Reasoning as we have done for $\mathcal{TAB}^{\mathcal{EL}^{\bot}\mathbf{T}}_{PH1}$, we can show that:

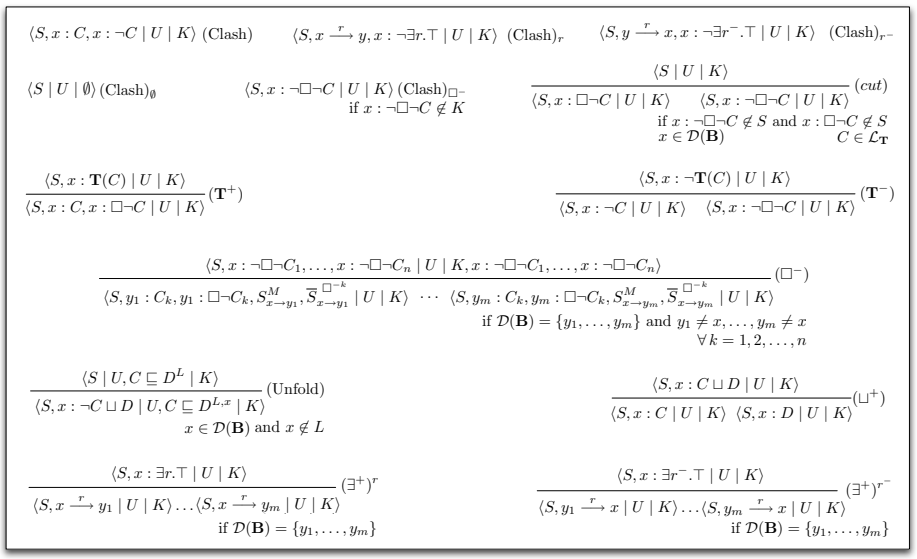


Fig. 4. The calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$.

Theorem 13 (Complexity of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$). *Given a KB and a query F, the problem of checking whether KB $\cup \{\neg F\}$ in $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{PH1}$ is satisfiable is in* NP.

5.2 The tableaux calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$

Let us now introduce the calculus $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$. Exactly as for $\mathcal{TAB}^{\mathcal{EL}^\bot\mathbf{T}}_{PH2}$, for each open saturated branch $\mathbf{B}$ built by $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH1}$, it verifies whether it represents a minimal model of the KB. The rules of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$ are shown in Figure 4. The rules $(\exists^+)^r$ and $(\exists^+)^{r^-}$ introduce $x \xrightarrow{r} y$ and $y \xrightarrow{r} x$, respectively, where $y \in \mathcal{D}(\mathbf{B})$, instead of y being a new label.

Theorem 14 (Soundness and completeness of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$). *Given a* KB *and a query F, let $\langle S' \mid U\rangle$ be the corresponding constraint system of* KB, *and $\langle S \mid U\rangle$ the corresponding constraint system of* KB $\cup \{\neg F\}$. *An open saturated branch $\mathbf{B}$ built by $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{PH1}$ for $\langle S \mid U\rangle$ is satisfiable by an injective mapping in a minimal model of* KB *iff the tableau in $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{PH2}$ for $\langle S' \mid U \mid \mathbf{B}^{\Box^-}\rangle$ is closed.*

Theorem 15 (Termination of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{PH2}$). *Let $\langle S' \mid U \mid \mathbf{B}^{\Box^-}\rangle$ be a constraint system starting from an open saturated branch $\mathbf{B}$ built by $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{PH1}$, then any tableau generated by $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{PH2}$ is finite.*

By reasoning exactly as done for $\mathcal{TAB}^{\mathcal{EL}^\bot\mathbf{T}}_{min}$, we prove that:

Theorem 16 (Complexity of $\mathcal{TAB}^{\mathit{Lite}_c\mathbf{T}}_{min}$). *The problem of deciding whether* KB $\models^{\mathcal{L}_\mathbf{T}}_{min} F$ *by means of $\mathcal{TAB}^{\mathrm{Lite}_c\mathbf{T}}_{min}$ is in Π^p_2.*

6 Conclusions

We have proposed a nonmonotonic extension of low complexity DLs $\mathcal{EL}^\bot$ and *DL-Lite*$_{core}$ for reasoning about typicality and defeasible properties. We have summarized complexity results recently studied for such extensions [11], namely that entailment is EXPTIME-hard for $\mathcal{EL}^\bot\mathbf{T}_{min}$, whereas it drops to Π_2^p when considering the Left Local Fragment of $\mathcal{EL}^\bot\mathbf{T}_{min}$. The same Π_2^p complexity has been found for *DL-Lite*$_c\mathbf{T}_{min}$. These results match the complexity upper bounds of the same fragments in circumscribed KBs [3]. We have also provided tableau calculi for checking minimal entailment in the Left Local fragment of $\mathcal{EL}^\bot\mathbf{T}_{min}$ as well as in *DL-Lite*$_c\mathbf{T}_{min}$. The proposed calculi match the complexity results above. Of course, many optimizations are possible and we intend to study them in future work.

As mentioned in the Introduction, several nonmonotonic extensions of DLs have been proposed in the literature [15, 4, 2, 3, 7, 12, 10, 9, 6] and we refer to [12] for a survey. Concerning nonmonotonic extensions of low complexity DLs, the complexity of *circumscribed* fragments of the $\mathcal{EL}^\bot$ and *DL-lite* families have been studied in [3]. Recently, a fragment of $\mathcal{EL}^\bot$ for which the complexity of circumscribed KBs is polynomial has been identified in [14]. In future work, we shall investigate complexity of minimal entailment and proof methods for such a fragment extended with $\mathbf{T}$ and possibly the definition of a calculus for it.

References

1. F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ envelope. In *IJCAI*, pages 364–369, 2005.
2. F. Baader and B. Hollunder. Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *JAR*, 15(1):41–68, 1995.
3. P. Bonatti, M. Faella, and L. Sauro. Defeasible inclusions in low-complexity dls: Preliminary notes. In *IJCAI*, pages 696–701, 2009.
4. P. A. Bonatti, C. Lutz, and F. Wolter. DLs with circumscription. In *KR*, p. 400–410, 2006.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable Reasoning and Efficient Query Answering in DLs: The DL-Lite Family. *JAR*, 39(3):385429, 2007.
6. G. Casini and U. Straccia. Rational closure for defeasible DLs. In *JELIA*, p. 77–90, 2010.
7. F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Trans. Comput. Log.*, 3(2):177–225, 2002.
8. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. A tableau calculus for a nonmonotonic extension of $\mathcal{EL}^\bot$. In *TABLEAUX*, pages 164–179, 2011.
9. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Reasoning About Typicality in Preferential Description Logics. In *JELIA*, pages 192–205, 2008.
10. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Prototypical reasoning with low complexity Description Logics: Preliminary results. In *LPNMR*, pages 430–436, 2009.
11. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Reasoning about typicality in low complexity DLs: the logics $\mathcal{EL}^\bot\mathbf{T}_{min}$ and *DL-Lite*$_c\mathbf{T}_{min}$. In *IJCAI*, pages 894–899, 2011.
12. L. Giordano, V. Gliozzi, N. Olivetti, and G.L. Pozzato. $\mathcal{ALC}+\mathbf{T}_{min}$: a preferential extension of description logics. *Fundamenta Informaticae*, 96:1–32, 2009.
13. S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.
14. P.A.Bonatti, M. Faella, and L. Sauro. $\mathcal{EL}$ with default attributes and overriding. In *ISWC*, pages 64–79, 2010.
15. U. Straccia. Default inheritance reasoning in hybrid kl-one-style logics. In *IJCAI*, pages 676–681, 1993.

An Inductive Logic Programming Approach to Learning Inclusion Axioms in Fuzzy Description Logics

Francesca A. Lisi[1] and Umberto Straccia[2]

[1] Dipartimento di Informatica, Università degli Studi di Bari "Aldo Moro", Italy
lisi@di.uniba.it
[2] ISTI - CNR, Pisa, Italy
straccia@isti.cnr.it

Abstract. Fuzzy Description Logics (DLs) are logics that allow to deal with vague structured knowledge. Although a relatively important amount of work has been carried out in the last years concerning the use of fuzzy DLs as ontology languages, the problem of automatically managing fuzzy ontologies has received very little attention so far. We report here our preliminary investigation on this issue by describing a method for inducing inclusion axioms in a fuzzy DL-Lite like DL.

1 Introduction

Description Logics (DLs) [1] play a key role in the design of *ontologies*. An ontology consists of a hierarchical description of important concepts in a particular domain, along with the description of the properties (of the instances) of each concept. In this context, DLs are important as they are essentially the theoretical counterpart of the *Web Ontology Language OWL 2* [3], the current standard language to represent ontologies, and its profiles. [4] E.g., DL-Lite [2] is the DL behind the *OWL 2 QL* profile and is especially aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task.

On the other hand, it is well-known that "classical" ontology languages are not appropriate to deal with *vague knowledge*, which is inherent to several real world domains [21]. So far, several fuzzy extensions of DLs can be found in the literature (see the survey in [14]), which includes, among others a fuzzy DL-Lite like DL [23] which has been implemented in the SoftFacts system [23] [5].

Although a relatively important amount of work has been carried out in the last years concerning the use of fuzzy DLs as ontology languages, the problem of automatically managing fuzzy ontologies has received very little attention so far. In this work, we report our preliminary investigation on this issue by describing

[3] http://www.w3.org/TR/2009/REC-owl2-overview-20091027/
[4] http://www.w3.org/TR/owl2-profiles/.
[5] See, http://www.straccia.info/software/SoftFacts/SoftFacts.html

	Łukasiewicz logic	Gödel logic	Product logic
$a \otimes b$	$\max(a+b-1,0)$	$\min(a,b)$	$a \cdot b$
$a \oplus b$	$\min(a+b,1)$	$\max(a,b)$	$a+b-a \cdot b$
$a \Rightarrow b$	$\min(1-a+b,1)$	$\begin{cases} 1 & \text{if } a \leqslant b \\ b & \text{otherwise} \end{cases}$	$\min(1, b/a)$
$\ominus a$	$1-a$	$\begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}$

Table 1. Combination functions of various fuzzy logics.

a method for inducing inclusion axioms in a fuzzy DL-Lite like DL. The method follows the machine learning approach known as *Inductive Logic Programming* (ILP) by adapting known results in ILP concerning crisp rules to the novel case of fuzzy DL inclusion axioms.

The paper is structured as follows. Section 2 is devoted to preliminaries on Mathematical Fuzzy Logic, Fuzzy DLs and ILP. Section 3 describes our preliminary contribution to the problem in hand, also by means of an illustrative example. Section 4 concludes the paper with final remarks and comparison with related work.

2 Background

2.1 Mathematical Fuzzy Logic Basics

In *Mathematical Fuzzy Logic* [7], the convention prescribing that a statement is either true or false is changed and is a matter of degree measured on an ordered scale that is no longer $\{0,1\}$, but the $[0,1]$. This degree is called *degree of truth* (or *score*) of the logical statement ϕ in the interpretation $\mathcal{I}$. In this section, *fuzzy statements* have the form $\phi[r]$, where $r \in [0,1]$ (see, *e.g.* [6,7]) and ϕ is a statement, which encode that the degree of truth of ϕ is *greater or equal* r.

A *fuzzy interpretation* $\mathcal{I}$ maps each basic statement p_i into $[0,1]$ and is then extended inductively to all statements: $\mathcal{I}(\phi \wedge \psi) = \mathcal{I}(\phi) \otimes \mathcal{I}(\psi)$, $\mathcal{I}(\phi \vee \psi) = \mathcal{I}(\phi) \oplus \mathcal{I}(\psi)$, $\mathcal{I}(\phi \rightarrow \psi) = \mathcal{I}(\phi) \Rightarrow \mathcal{I}(\psi)$, $\mathcal{I}(\neg\phi) = \ominus\mathcal{I}(\phi)$, $\mathcal{I}(\exists x.\phi(x)) = \sup_{a \in \Delta^{\mathcal{I}}} \mathcal{I}(\phi(a))$, $\mathcal{I}(\forall x.\phi(x)) = \inf_{a \in \Delta^{\mathcal{I}}} \mathcal{I}(\phi(a))$, where $\Delta^{\mathcal{I}}$ is the domain of $\mathcal{I}$, and $\otimes$, $\oplus$, $\Rightarrow$, and $\ominus$ are so-called *t-norms*, *t-conorms*, *implication functions*, and *negation functions*, respectively, which extend the Boolean conjunction, disjunction, implication, and negation, respectively, to the fuzzy case.

One usually distinguishes three different logics, namely Łukasiewicz, Gödel, and Product logics [7], whose combination functions are reported in Table 1. The operators for Zadeh logic, namely $a \otimes b = \min(a,b)$, $a \oplus b = \max(a,b)$, $\ominus a = 1-a$ and $a \Rightarrow b = \max(1-a,b)$, can be expressed in Łukasiewicz logic[6].

[6] More precisely, $\min(a,b) = a \otimes_L (a \Rightarrow_L b), \max(a,b) = 1 - \min(1-a, 1-b)$.

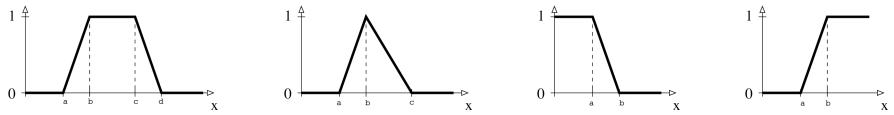


Fig. 1. (a) Trapezoidal function $trz(x; a, b, c, d)$, (b) triangular function $tri(x; a, b, c)$, (c) left shoulder function $ls(x; a, b)$, and (d) right shoulder function $rs(x; a, b)$.

A *fuzzy set* R over a countable crisp set X is a function $R\colon X \to [0, 1]$. The trapezoidal (Fig. 1 (a)), the triangular (Fig. 1 (b)), the L-function (left-shoulder function, Fig. 1 (c)), and the R-function (right-shoulder function, Fig. 1 (d)) are frequently used to specify membership degrees. In particular, the left-shoulder function is defined as

$$ls(x; a, b) = \begin{cases} 1 & \text{if } x \leqslant a \\ 0 & \text{if } x \geqslant b \\ (b - x)/(b - a) & \text{if } x \in [a, b] \end{cases} \tag{1}$$

The notions of satisfiability and logical consequence are defined in the standard way. A fuzzy interpretation $\mathcal{I}$ *satisfies* a fuzzy statement $\phi[r]$ or $\mathcal{I}$ is a *model* of $\phi[r]$, denoted $\mathcal{I} \models \phi[r]$ iff $\mathcal{I}(\phi) \geqslant r$.

2.2 DL-Lite like description logic and its fuzzy extensions

For computational reasons, the logic we adopt is based on a fuzzy extension of the DL-Lite DL without negation [23]. It supports at the intensional level unary relations (called *concepts*) and binary relations (called *roles*), while supports n-ary relations (relational tables) at the extensional level.

Formally, a *knowledge base* $\mathcal{K} = \langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ consists of a *facts component* $\mathcal{F}$, an *ontology component* $\mathcal{O}$ and an *abstraction component* $\mathcal{A}$, which are defined as follows (for a detailed account of the semantics, see [22]). Information can be retrieved from the knowledge base by means of an appropriate *query language* discussed later.

Facts Component. The facts component $\mathcal{F}$ is a finite set of expressions of the form

$$R(c_1, \ldots, c_n)[s] \ , \tag{2}$$

where R is an n-ary relation, every c_i is a constant, and s is a degree of truth (or *score*) in $[0, 1]$ indicating to which extent the tuple $\langle c_1, \ldots, c_n \rangle$ is an instance of relation R.[7] Facts are stored in a relational database. We may omit the score component and in such case the value 1 is assumed.

[7] The score s may have been computed by some external tool, such as a classifier, etc.

Ontology Component. The ontology component is used to define the relevant abstract concepts and relations of the application domain by means of inclusion axioms. Specifically, $\mathcal{O}$ is a finite set of *inclusion axioms* having the form

$$Rl_1 \sqcap \ldots \sqcap Rl_m \sqsubseteq Rr \ , \tag{3}$$

where $m \geqslant 1$, all Rl_i and Rr have the same arity and each Rl_i is a so-called *left-hand relation* and Rr is a *right-hand relation*[8]. We assume that relations occurring in $\mathcal{F}$ do not occur in inclusion axioms (so, we do not allow that database relation names occur in $\mathcal{O}$). Also we recall that from a semantics point of view, Gödel logic is adopted. The intuition for one such semantics is that if a tuple $\mathbf{c}$ is instance of each relation Rl_i to degree s_i then $\mathbf{c}$ is instance of Rr to degree $\min(s_1, \ldots, s_m)$.

The exact syntax of the relations appearing on the left-hand and right-hand side of inclusion axioms is specified below:

$$\begin{array}{l} Rl \longrightarrow A \mid R[i_1, i_2] \\ Rr \longrightarrow A \mid R[i_1, i_2] \mid \exists R.A \end{array} \tag{4}$$

where A is an *atomic concept* and R is a role with $1 \leqslant i_1, i_2 \leqslant 2$. Here $R[i_1, i_2]$ is the projection of the relation R on the columns i_1, i_2 (the order of the indexes matters). Hence, $R[i_1, i_2]$ has arity 2. Additionally, $\exists R.A$ is a so-called qualified existential quantification on roles which corresponds to the FOL formula $\exists y.R(x, y) \wedge A(y)$ where $\wedge$ is interpreted as the t-norm in the Gödel logic (see Table 1).

Abstraction Component. $\mathcal{A}$ (similarly to [3,17]) is a set of "abstraction statements" that allow to connect atomic concepts and roles to physical relational tables. Essentially, this component is used as a wrapper to the underlying database and, thus, prevents that relational table names occur in the ontology. Formally, an *abstraction statement* is of the form

$$R \mapsto (c_1, \ldots, c_n)[c_{score}].sql \ , \tag{5}$$

where *sql* is a SQL statement returning n-ary tuples $\langle c_1, \ldots, c_n \rangle$ $(n \leqslant 2)$ with score determined by the c_{score} column. The tuples have to be ranked in decreasing order of score and, as for the fact component, we assume that there cannot be two records $\langle \mathbf{c}, s_1 \rangle$ and $\langle \mathbf{c}, s_2 \rangle$ in the result set of *sql* with $s_1 \neq s_2$ (if there are, then we remove the one with the lower score). The score c_{score} may be omitted and in that case the score 1 is assumed for the tuples. We assume that R occurs in $\mathcal{O}$, while all of the relational tables occurring in the SQL statement occur in $\mathcal{F}$. Finally, we assume that there is at most one abstraction statement for each abstract relational symbol R.

[8] Note that recursive inclusion axioms are allowed.

Query Language. The query language enables the formulation of conjunctive queries with a scoring function to rank the answers. More precisely, a *ranking query* [13] is of the form

$$q(\mathbf{x})[s] \leftarrow \exists \mathbf{y}\ R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l], \\ \mathsf{OrderBy}(s = f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h)), \mathsf{Limit}(k) \tag{6}$$

where

1. q is an n-ary relation, every R_i is a n_i-ary relation ($1 \leqslant n_i \leqslant 2$). $R_i(\mathbf{z_i})$ may also be of the form $(z \leqslant v), (z < v), (z \geqslant v), (z > v), (z = v), (z \neq v)$, where z is a variable, v is a value of the appropriate concrete domain;
2. $\mathbf{x}$ are the *distinguished variables.*
3. $\mathbf{y}$ are existentially quantified variables called the *non-distinguished variables.* We omit to write $\exists \mathbf{y}$ when $\mathbf{y}$ is clear from the context;
4. $\mathbf{z_i}, \mathbf{z'_j}$ are tuples of constants or variables in $\mathbf{x}$ or $\mathbf{y}$;
5. $s, s_1, \ldots, s_l$ are distinct variables and different from those in $\mathbf{x}$ and $\mathbf{y}$;
6. p_j is an n_j-ary *fuzzy predicate* assigning to each n_j-ary tuple $\mathbf{c}_j$ a *score* $p_j(\mathbf{c}_j) \in [0, 1]$. We require that an n-ary fuzzy predicate p is *safe*, that is, 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 $f\colon ([0, 1])^{l+h} \to [0, 1]$, which combines the scores of the l relations $R_i(\mathbf{c}'_i)$ and the n fuzzy predicates $p_j(\mathbf{c}''_j)$ into an overall score s to be assigned to $q(\mathbf{c})$. We assume that f is *monotone*, that is, for each $\mathbf{v}, \mathbf{v}' \in ([0, 1])^{l+h}$ such that $\mathbf{v} \leqslant \mathbf{v}'$, it holds $f(\mathbf{v}) \leqslant f(\mathbf{v}')$, where $(v_1, \ldots, v_{l+h}) \leqslant (v'_1, \ldots, v'_{l+h})$ iff $v_i \leqslant v'_i$ for all i. We also assume that the computational cost of f and all fuzzy predicates p_i is bounded by a constant;
8. $\mathsf{Limit}(k)$ indicates the number of answers to retrieve and is optional. If omitted, all answers are retrieved.

We call $q(\mathbf{x})[s]$ its *head*, $\exists \mathbf{y}.R_1(\mathbf{z}_1)[s_1], \ldots, R_l(\mathbf{z}_l)[s_l]$ its *body* and $\mathsf{OrderBy}(s = f(s_1, \ldots, s_l, p_1(\mathbf{z}'_1), \ldots, p_h(\mathbf{z}'_h))$ the *scoring atom.* We also allow the scores $[s]$, $[s_1]$, $\ldots$, $[s_l]$ and the scoring atom to be omitted. In this case we assume the value 1 for s_i and s instead. The informal meaning of such a query is: if $\mathbf{z}_i$ is an instance of R_i to degree at least or equal to s_i, then $\mathbf{x}$ is an instance of q to degree at least or equal to s, where s has been determined by the scoring atom.

The answer set $ans_{\mathcal{K}}(q)$ over $\mathcal{K}$ of a query q is the set of tuples $\langle \mathbf{t}, s \rangle$ such that $\mathcal{K} \models q(\mathbf{t})[s]$ with $s > 0$ (informally, $\mathbf{t}$ satisfies the query to non-zero degree s) and the score s is as high as possible, *i.e.* if $\langle \mathbf{t}, s \rangle \in ans_{\mathcal{K}}(q)$ then (i) $\mathcal{K} \not\models q(\mathbf{t})[s']$ for any $s' > s$; and (ii) there cannot be another $\langle \mathbf{t}, s' \rangle \in ans_{\mathcal{K}}(q)$ with $s > s'$.

2.3 Learning rules with ILP

Inductive Logic Programming (ILP) was born at the intersection between Concept Learning and Logic Programming [16].

From Logic Programming it has borrowed the Knowledge Representation (KR) framework, i.e. the possibility of expressing facts and rules in the form of Horn clauses. In the following, rules are denoted by

$$B(\mathbf{x}) \rightarrow H(\mathbf{x}) \tag{7}$$

where $\mathbf{x}$ is the vector of the n variables that appear in the rule, $B(\mathbf{x}) = B_0(\mathbf{x}) \wedge \ldots \wedge B_q(\mathbf{x})$ represents the antecedent (called the *body*) of the rule, and $H(\mathbf{x})$ is the consequent (called *head*) of the rule. The predicate H pertains to the concept to be learnt (called *target*). Given an attribute domain $\mathcal{D}$ and a vector $\mathbf{t} \in \mathcal{D}^n$ of n values of the domain, we denote the ground substitution of the variable $\mathbf{x}$ with $\mathbf{t}$ by $H(\mathbf{t}) = \sigma[\mathbf{x}/\mathbf{t}]H(\mathbf{x})$. Then $H(\mathbf{t})$ is true or false in a given interpretation.

From Concept Learning it has inherited the inferential mechanisms for induction, the most prominent of which is *generalisation*. A distinguishing feature of ILP with respect to other forms of Concept Learning is the use of prior domain knowledge in the background during the induction process. The classical ILP problem is described by means of two logic programs: (i) the *background theory* $\mathcal{K}$ which is a set of ground facts and rules; (ii) the *training set* $\mathcal{E}$ which is a set of ground facts, called *examples*, pertaining to the predicate to be learnt. It is often split in $\mathcal{E}^+$ and $\mathcal{E}^-$, which correspond respectively to positive and negative examples. If only $\mathcal{E}^+$ is given, $\mathcal{E}^-$ can be deduced by using the Closed World Assumption (CWA). A rule r *covers* an example $e \in \mathcal{E}$ iff $\mathcal{K} \cup \{r\} \models e$. The task of induction is to find, given $\mathcal{K}$ and $\mathcal{E}$, a set $\mathcal{H}$ of rules such that: (i) $\forall e \in \mathcal{E}^+, \mathcal{K} \cup \mathcal{H} \models e$ (completeness of $\mathcal{H}$) and (ii) $\forall e \in \mathcal{E}^-, \mathcal{K} \cup \mathcal{H} \not\models e$ (consistency of $\mathcal{H}$). Two further restrictions hold naturally. One is that $\mathcal{K} \not\models \mathcal{E}^+$ since, in such a case, $\mathcal{H}$ would not be necessary to explain $\mathcal{E}^+$. The other is $\mathcal{K} \cup \mathcal{H} \not\models \bot$, which means that $\mathcal{K} \cup \mathcal{H}$ is a consistent theory. Usually, rule induction fits with the idea of providing a compression of the information contained in $\mathcal{E}$.

A popular ILP algorithm for learning sets of rules is FOIL [18]. It performs a greedy search in order to maximise a gain function. The rules are induced until all examples are covered or no more rules are found that overcome the threshold. When a rule is induced, the positive examples covered by the rule are removed from $\mathcal{E}$. This is the *sequential covering* approach underlying the function FOIL-Learn-Sets-of-Rules shown in Figure 2. For inducing a rule, the function FOIL-Learn-One-Rule reported in Figure 3 starts with the most general clause ($\top \rightarrow H(\mathbf{x})$) and specialises it step by step by adding literals to the antecedent. The rule r is accepted when its confidence degree $cf(r)$ (see later on) overcomes a fixed threshold θ and it does not cover any negative example.

The Gain function is computed by the formula:

$$\text{Gain}(r', r) = p * (log_2(cf(r')) - log_2(cf(r))) \ , \tag{8}$$

where p is the number of distinct positive examples covered by the rule r that are still covered by r'. Thus, the gain is positive iff r' is more informative in the sense of Shannon's information theory (i.e. iff the confidence degree increases). If there are some literals to add which increase the confidence degree, the gain tends to favor the literals that offer the best compromise between the confidence degree and the number of examples covered.

function FOIL-LEARN-SETS-OF-RULES(H, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{K}$): $\mathcal{H}$
begin
1. $\mathcal{H} \leftarrow \emptyset$;
2. **while** $\mathcal{E}^+ \neq \emptyset$ **do**
3. $\quad r \leftarrow$ FOIL-LEARN-ONE-RULE(H, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{K}$);
4. $\quad \mathcal{H} \leftarrow \mathcal{H} \cup \{r\}$;
5. $\quad \mathcal{E}^+_r \leftarrow \{e \in \mathcal{E}^+ | \mathcal{K} \cup r \models e\}$;
6. $\quad \mathcal{E}^+ \leftarrow \mathcal{E}^+ \setminus \mathcal{E}^+_r$;
7. **endwhile**
8. **return** $\mathcal{H}$
end

Fig. 2. Algorithm for learning sets of rules in FOIL

Given a Horn clause $B(\mathbf{x}) \rightarrow H(\mathbf{x})$, the confidence degree is given by:

$$cf(B(\mathbf{x}) \rightarrow H(\mathbf{x})) = P(B(\mathbf{x}) \wedge H(\mathbf{x}))/P(B(\mathbf{x})) \ . \tag{9}$$

Confidence degrees are computed in the spirit of domain probabilities [8]. Input data in ILP are supposed to describe one interpretation under CWA. We call $\mathcal{I}_{ILP}$ this interpretation. So, given a fact f, we define:

$$\mathcal{I}_{ILP} \models f \text{ iff } \mathcal{K} \cup \mathcal{E} \models f \ . \tag{10}$$

The domain $\mathcal{D}$ is the Herbrand domain described by $\mathcal{K}$ and $\mathcal{E}$. We take P as a uniform probability on $\mathcal{D}$. So the confidence degree in a clause $B(\mathbf{x}) \rightarrow H(\mathbf{x})$ is:

$$cf(B(\mathbf{x}) \rightarrow H(\mathbf{x})) = \frac{|\mathbf{t} \in \mathcal{D}^n \mid \mathcal{I}_{ILP} \models B(\mathbf{t}) \text{ and } H(\mathbf{t}) \in \mathcal{E}^+|}{|\mathbf{t} \in \mathcal{D}^n \mid \mathcal{I}_{ILP} \models B(\mathbf{t}) \text{ and } H(\mathbf{t}) \in \mathcal{E}|} \tag{11}$$

where $|\cdot|$ denotes set cardinality. Testing all possible $\mathbf{t} \in \mathcal{D}^n$ is not tractable in practice. However, we can equivalently restrict the computation to the substitutions that map variables to constants in their specific domains. In fact, this computation is equivalent to a database query and thus, we can also use some optimization strategy such as indexing or query ordering. This makes the computation tractable although it remains costly.

3 Towards Learning Fuzzy DL-Lite like Inclusion Axioms

In this section we consider a learning problem where:

- the target concept H is a DL-Lite atomic concept;
- the background theory $\mathcal{K}$ is a DL-Lite like knowledge base $\langle \mathcal{F}, \mathcal{O}, \mathcal{A} \rangle$ of the form described in Section 2.2;
- the training set $\mathcal{E}$ is a collection of fuzzy DL-Lite like facts of the form (2) and labeled as either positive or negative examples for H. We assume that $\mathcal{F} \cap \mathcal{E} = \emptyset$;

function FOIL-LEARN-ONE-RULE(H, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{K}$): r
begin
1. $B(\mathbf{x}) \leftarrow \top$;
2. $r \leftarrow \{B(\mathbf{x}) \rightarrow H(\mathbf{x})\}$;
3. $\mathcal{E}_r^- \leftarrow \mathcal{E}^-$;
4. **while** $cf(r) < \theta$ **and** $\mathcal{E}_r^- \neq \emptyset$ **do**
5. $\quad B_{best}(\mathbf{x}) \leftarrow B(\mathbf{x})$;
6. $\quad maxgain \leftarrow 0$;
7. $\quad$ **foreach** $l \in \mathcal{K}$ **do**
8. $\quad\quad gain \leftarrow \text{GAIN}(B(\mathbf{x}) \wedge l(\mathbf{x}) \rightarrow H(\mathbf{x}), B(\mathbf{x}) \rightarrow H(\mathbf{x}))$;
9. $\quad\quad$ **if** $gain \geqslant maxgain$ **then**
10. $\quad\quad\quad maxgain \leftarrow gain$;
11. $\quad\quad\quad B_{best}(\mathbf{x}) \leftarrow B(\mathbf{x}) \wedge l(\mathbf{x})$;
12. $\quad\quad$ **endif**
13. $\quad$ **endforeach**
14. $\quad r \leftarrow \{B_{best}(\mathbf{x}) \rightarrow H(\mathbf{x})\}$;
15. $\quad \mathcal{E}_r^- \leftarrow \mathcal{E}_r^- \setminus \{e \in \mathcal{E}^- | \mathcal{K} \cup r \models e\}$;
16. **endwhile**
17. **return** r
end

Fig. 3. Algorithm for learning one rule in FOIL

– the target theory $\mathcal{H}$ is a set of inclusion axioms of the form

$$B \sqsubseteq H \tag{12}$$

where H is an atomic concept, $B = C_1 \sqcap \ldots \sqcap C_m$, and each concept C_i has syntax

$$C \longrightarrow A \mid \exists R.A \mid \exists R.\top \ . \tag{13}$$

Note that the language of hypotheses $\mathcal{L}_\mathcal{H}$ differs from the language of the background theory $\mathcal{L}_\mathcal{K}$ as for the form of axioms. Yet the alphabet underlying $\mathcal{L}_\mathcal{H}$ is a subset of the alphabet for $\mathcal{L}_\mathcal{K}$. Note also that $\mathcal{H}$, in order to be acceptable, must be complete and consistent w.r.t. $\mathcal{E}$, *i.e.* it must cover all the positive examples and none of the negative examples.

3.1 The FOIL-like algorithm

We now show how we may learn inclusion axioms of the form (12). To this aim, we adapt (10) to our case and define for $C \neq H$

$$\mathcal{I}_{ILP} \models C(t) \text{ iff } \mathcal{K} \cup \mathcal{E} \models C(t)[s] \text{ and } s > 0 \ . \tag{14}$$

That is, we write $\mathcal{I}_{ILP} \models C(t)$ iff it can be inferred from $\mathcal{K}$ and $\mathcal{E}$ that t is an instance of concept C to a non-zero degree. Note that $\mathcal{E}$ is split into $\mathcal{E}^+$ and $\mathcal{E}^-$. In order to distinguish between the two sets while using a uniform representation

with $\mathcal{K}$, we introduce two additional concepts, H^+ and H^-, whose intension coincide with the sets $\mathcal{E}^+$ and $\mathcal{E}^-$, respectively, as well as the axioms $H^+ \sqsubseteq H$ and $H^- \sqsubseteq H$. We call $\mathcal{K}'$ the background theory augmented with the training set represented this way, *i.e.* $\mathcal{K}' = \mathcal{K} \cup \mathcal{E}$.

Now, in order to account for multiple fuzzy instantiations of fuzzy predicates occurring in the inclusion axioms of interest to us, we customise (11) into the following formula for computing the confidence degree:

$$cf(B \sqsubseteq H) = \frac{\sum_{t \in P} B(t) \Rightarrow H(t)}{|D|} \tag{15}$$

where

- $P = \{t \mid \mathcal{I}_{ILP} \models C_i(t) \text{ and } H(t)[s] \in \mathcal{E}^+\}$, *i.e.* P is the set of instances for which the implication covers a positive example;
- $D = \{t \mid \mathcal{I}_{ILP} \models C_i(t) \text{ and } H(t)[s] \in \mathcal{E}\}$, *i.e.* D is the set of instances for which the implication covers an example (either positive or negative);
- $B(t) \Rightarrow H(t)$ denotes the degree to which the implication holds for a certain instance t;
- $B(t) = \min(s_1, \ldots, s_n)$, with $\mathcal{K} \cup \mathcal{E} \models C_i(t)[s_i]$;
- $H(t) = s$ with $H(t)[s] \in \mathcal{E}$.

Clearly, the more positive instances supporting the inclusion axiom, the higher the confidence degree of the axiom.

Note that the confidence score can be determined easily by submitting appropriate queries via the query language described in Section 2.2. More precisely, proving the fuzzy entailment in (14) for each C_i is equivalent to answering a unique ranking query whose body is the conjunction of the relations R_l resulting from the transformation of C_i's into FOL predicates and whose score s is given by the minimum between s_l's.

From an algorithm point of view, it suffices to change FOIL-Learn-One-Rule at step 7., where now l may be of any of the forms allowed in (13). More precisely, in line with the tradition in ILP and in conformance with the search direction in FOIL, we devise a specialization operator, *i.e.* an operator for traversing the hypotheses space top down, with the following refinement rules:

1. Add atomic concept (A)
2. Add complex concept by existential role restriction ($\exists R.\top$)
3. Add complex concept by qualified existential role restriction ($\exists R.A$)
4. Replace atomic concept (A replaced by A' if $A' \sqsubseteq A$)
5. Replace complex concept ($\exists R.A$ replaced by $\exists R.A'$ if $A' \sqsubseteq A$)

The rules are numbered according to an order of precedence, *e.g.* the addition of an atomic concept has priority over the addition of a complex concept obtained by existential role restriction. A rule can be applied when the preceding one in the list can not be applied anymore. Concept and role names in the alphabet underlying $\mathcal{L}_\mathcal{H}$ are themselves ordered. This implies that, *e.g.*, the addition of an atomic concept is not possible anymore when all the atomic concepts in the alphabet have been already used in preceding applications of the rule.

3.2 An illustrative example

For illustrative purposes we consider the following case involving the classification of hotels as good ones. We assume to have a background theory $\mathcal{K}$ with a relational database $\mathcal{F}$ (reported in Figure 4), an ontology $\mathcal{O}$ [9] (illustrated in Figure 5) which encompasses the following inclusion axioms

$Park \sqsubseteq Attraction$
$Tower \sqsubseteq Attraction$
$Attraction \sqsubseteq Site$
$Hotel \sqsubseteq Site$

and the following set $\mathcal{A}$ of abstraction statements:

$Hotel \mapsto (h.id)$. SELECT h.id
FROM HotelTable h

$hasRank \mapsto (h.id, h.rank)$. SELECT h.id, h.rank
FROM HotelTable h

$cheapPrice \mapsto (h.id, r.price)[score]$. SELECT h.id, r.price, *cheap*(r.price) AS score
FROM HotelTable h, RoomTable r
WHERE h.id = r.hotel
ORDER BY score

$closeTo \mapsto (from, to)[score]$. SELECT d.from, d.to *closedistance*(d.time) AS score
FROM DistanceTable d
ORDER BY score

where $cheap(p)$ is a function determining how cheap a hotel room is given its price, modelled as *e.g.* a so-called left-shoulder function $cheap(p) = ls(p; 50, 100)$, while $closedistance(d) = ls(d; 5, 25)$.

Assume now that:

- $H = GoodHotel$;
- $\mathcal{E}^+ = \{GoodHotel(h1)[0.6], GoodHotel(h2)[0.8]\}$;
- $\mathcal{E}^- = \{GoodHotel(h3)[0.4]\}$.

In order to have a uniform representation of the examples w.r.t. the background theory, we transform $\mathcal{E}$ as follows:

$GoodHotel^+ \sqsubseteq GoodHotel$
$GoodHotel^- \sqsubseteq GoodHotel$
$GoodHotel^+(h1)[0.6]$
$GoodHotel^+(h2)[0.8]$
$GoodHotel^-(h3)[0.4]$

[9] http://donghee.info/research/SHSS/ObjectiveConceptsOntology(OCO).html

HotelTable		
id	rank	noRooms
h1	3	21
h2	5	123
h3	4	95

RoomTable			
id	price	roomType	hotel
r1	60	single	h1
r2	90	double	h1
r3	80	single	h2
r4	120	double	h2
r5	70	single	h3
r6	90	double	h3

Tower
id
t1

Park
id
p1
p2

DistanceTable			
id	from	to	time
d1	h1	t1	10
d2	h2	p1	15
d3	h3	p2	5

Fig. 4. Hotel database

and call $\mathcal{K}'$ the background theory augmented with the training set represented this way.

The following inclusion axioms:

$r_0 : \top \sqsubseteq GoodHotel$
$r_1 : Hotel \sqsubseteq GoodHotel$
$r_2 : Hotel \sqcap \exists cheapPrice.\top \sqsubseteq GoodHotel$
$r_3 : Hotel \sqcap \exists cheapPrice.\top \sqcap \exists closeTo.Attraction \sqsubseteq GoodHotel$
$r_4 : Hotel \sqcap \exists cheapPrice.\top \sqcap \exists closeTo.Park \sqsubseteq GoodHotel$
$r_5 : Hotel \sqcap \exists cheapPrice.\top \sqcap \exists closeTo.Tower \sqsubseteq GoodHotel$

belong to $\mathcal{L}_{\mathcal{H}} = \{r | C_i \in \{\top, Hotel, \exists cheapPrice, \exists closeTo\}\} \subset \mathcal{L}_{\mathcal{K}}$. They can be read as:

r_0 : Everything is a good hotel
r_1 : Every hotel is a good hotel
r_2 : Hotels having a cheap price are good hotels
r_3 : Hotels having a cheap price and close to an attraction are good hotels
r_4 : Hotels having a cheap price and close to a park are good hotels
r_5 : Hotels having a cheap price and close to a tower are good hotels

thus highlighting the possibility of generating "extreme" hypotheses about good hotels such as r_0 and r_1. Of course, some of them will be discarded on the basis of their confidence degree.

Before showing how hypotheses evaluation is performed in our adaptation of FOIL, we illustrate the computation of the confidence degree for r_3. It can be verified that for $\mathcal{K}'$

1. The query

$$\begin{aligned} q_P(h)[s] \leftarrow\ & GoodHotel^+(h), \\ & cheapPrice(h,p)[s_1], \\ & closeTo(h,a)[s_2], Attraction(a), \\ & s = \min(s_1, s_2) \end{aligned}$$

has answer set $ans_{\mathcal{K}'}(q_P) = \{\langle h1, 0.75\rangle, \langle h2, 0.4\rangle\}$ over $\mathcal{K}'$;

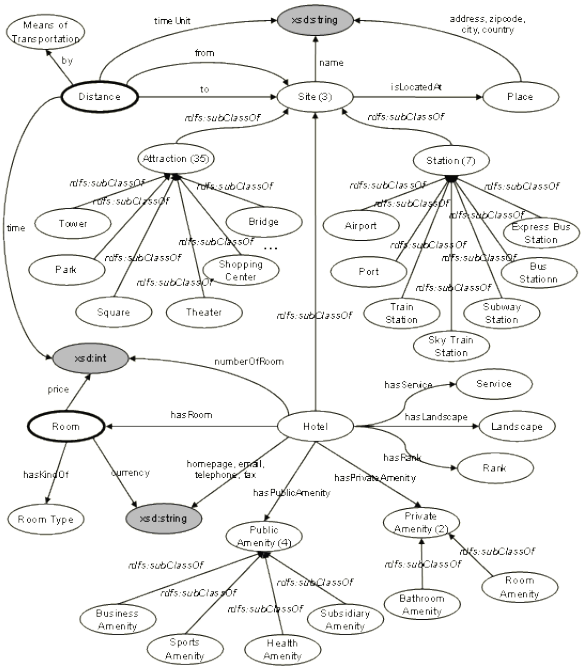


Fig. 5. Hotel ontology.

2. The query

$$q_D(h)[s] \leftarrow GoodHotel(h), \\ cheapPrice(h,p)[s_1], \\ closeTo(h,a)[s_2], Attraction(a), \\ s = \min(s_1, s_2)$$

has answer set $ans_{\mathcal{K}'}(q_D) = \{\langle h1, 0.75\rangle, \langle h2, 0.4\rangle, \langle h3, 0.6\rangle\}$ over $\mathcal{K}'$;
3. Therefore, according to (15), $P = \{h1, h2\}$, while $D = \{h1, h2, h3\}$;
4. As a consequence,

$$cf(r_3) = \frac{0.75 \Rightarrow 0.6 + 0.4 \Rightarrow 0.8}{3} = \frac{0.6 + 1.0}{3} = 0.5333 \ .$$

Note that in q_P the literals $Hotel(h)$ and $GoodHotel(h)$ are removed from the body in favour of $GoodHotel^+(h)$ because the concepts $Hotel$ and $GoodHotel$ subsume $GoodHotel$ (due to a derived axiom) and $GoodHotel^+$ (due to an asserted axiom) respectively. Analogously, in q_D, the literal $Hotel(h)$ is superseded by $GoodHotel(h)$.

Analogously, we can obtain:

$$cf(r_2) = \frac{0.8 \Rightarrow 0.6 + 0.4 \Rightarrow 0.8}{3} = \frac{0.6 + 0.4}{3} = 0.3333 \ .$$

$$cf(r_4) = \frac{0.4 \Rightarrow 0.8}{2} = \frac{0.4}{2} = 0.2 \ .$$

$$cf(r_5) = \frac{0.8 \Rightarrow 0.6}{2} = \frac{0.6}{2} = 0.3 \ .$$

The function FOIL-Learn-One-Rule starts from r_0 which is then specialized into r_1 by applying the refinement rule which adds an atomic concept, $Hotel$, to the left-hand side of the axiom. As aforementioned, r_0 and r_1 are trivial hypotheses, therefore we can skip the computation steps for them and go ahead. In particular, the algorithm generates r_2 from r_1 by adding a complex concept obtained as existential restriction of the role *cheapPrice*. This hypothesis is not consistent with the training set, therefore it must be specialized in order not to cover the negative example. Considering that r_3, r_4 and r_5 are both possible specializations of r_2, we can now compute the information gain for each of them according to (8):

$$\textsc{Gain}(r_3, r_2) = 2{*}(log_2(0.5333) - log_2(0.3333)) = 2{*}(-0.907 + 1.5851) = 1.3562 \ ,$$

$$\textsc{Gain}(r_4, r_2) = 1 * (log_2(0.2) - log_2(0.3333)) = (-2.3219 + 1.5851) = -0.7368 \ ,$$

$$\textsc{Gain}(r_5, r_2) = 1 * (log_2(0.3) - log_2(0.3333)) = (-1.7369 + 1.5851) = -0.1518 \ ,$$

The algorithm will prefer r_3. Yet r_3 still covers the negative example, therefore it must be further refined, e.g. by strengthening the qualified restriction of $closeTo$. The algorithm then generates once again the axioms r_4 and r_5 which have the following values of information gain over r_3:

$$\textsc{Gain}(r_4, r_3) = 1 * (log_2(0.2) - log_2(0.5333)) = (-2.3219 + 0.907) = -1.4149 \ ,$$

$$\textsc{Gain}(r_5, r_3) = 1 * (log_2(0.3) - log_2(0.5333)) = (-1.7369 + 0.907) = -0.8299 \ ,$$

The axiom r_5 is more informative than r_4, therefore it is preferred to r_4. Also it does not cover the negative example. Indeed, the literal $\exists closeTo.Tower$ is a discriminant feature. Therefore, r_5 becomes part of the target theory. Since one positive example is still uncovered, the computation continues within the function FOIL-Learn-Sets-of-Rules aiming at finding a complete theory, *i.e.* a theory which explains all the positive examples.

4 Final remarks

In this paper we have proposed a method for inducing ontology inclusion axioms within the KR framework of a fuzzy DL-Lite like DL where vagueness is dealt with the Gödel logic. The method extends FOIL, a popular ILP algorithm for learning sets of crisp rules, in a twofold direction: from crisp to fuzzy and from rules to inclusion axioms. Indeed, related FOIL-like algorithms are reported in the literature [20,5,19] but they can only learn fuzzy rules. Another relevant work is the formal study of fuzzy ILP contributed by [10]. Yet, it is less promising than our proposal from the practical side. Close to our application domain, [9] faces the problem of inducing equivalence axioms in a fragment of OWL corresponding to the $\mathcal{ALC}$ DL. Last, the work reported in [11] is based on an ad-hoc translation

of fuzzy Łukasiewicz $\mathcal{ALC}$ DL constructs into LP and then uses a conventional ILP method to lean rules. The method is not sound as it has been recently shown that the traduction from fuzzy DLs to LP is incomplete [15] and entailment in Łukasiewicz $\mathcal{ALC}$ is undecidable [4].

For the future we intend to study more formally the proposed specialization operator for the fuzzy DL being considered. Also we would like to investigate in depth the impact of Open World Assumption (holding in DLs) on the proposed ILP setting, and implement and experiment our method. Finally, it can be interesting to analyze the effect of the different implication functions on the learning process.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In P. Doherty, J. Mylopoulos, and C. A. Welty, editors, *Proc. of the Tenth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR-06)*, pages 260–270, 2006.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi. Data integration through DL-Lite$_\text{a}$ ontologies. In K.-D. Schewe and B. Thalheim, editors, *Semantics in Data and Knowledge Bases*, number 4925 in Lecture Notes in Computer Science, pages 26–47. Springer Verlag, 2008.
4. M. Cerami and U. Straccia. On the Undecidability of Fuzzy Description Logics with GCIs with Lukasiewicz t-norm. Technical report, Computing Research Repository, 2011. Available as CoRR technical report at http://arxiv.org/abs/1107.4212.
5. M. Drobics, U. Bodenhofer, and E.-P. Klement. FS-FOIL: an inductive learning method for extracting interpretable fuzzy descriptions. *Int. J. Approximate Reasoning*, 32(2-3):131–152, 2003.
6. R. Hähnle. Advanced many-valued logics. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition*, volume 2. Kluwer, 2001.
7. P. Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, 1998.
8. J.Y. Halpern. An Analysis of First-Order Logics of Probability. *Artificial Intelligence*, 46(3):311–350, 1990.
9. S. Hellmann, J. Lehmann, and S. Auer. Learning of OWL Class Descriptions on Very Large Knowledge Bases. *International Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.
10. T. Horváth and P. Vojtás. Induction of fuzzy and annotated logic programs. In S. Muggleton, R. P. Otero, and A. Tamaddoni-Nezhad, editors, *Inductive Logic Programming*, volume 4455 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2007.
11. S. Konstantopoulos and A. Charalambidis. Formulating description logic learning as an inductive logic programming task. In *Proceedings of the 19th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2010)*, pages 1–7. IEEE Press, 2010.
12. G.J. Klir and Bo Yuan. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1995.

13. T. Lukasiewicz and U. Straccia. Top-k retrieval in description logic programs under vagueness for the semantic web. In H. Prade, V.S. Subrahmanian, editors, *Scalable Uncertainty Management*, number 4772 in Lecture Notes in Computer Science, pages 16–30. Springer Verlag, 2007.
14. T. Lukasiewicz and U. Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Journal of Web Semantics*, 6:291–308, 2008.
15. B. Motik and R. Rosati. A faithful integration of description logics with logic programming. In M.M. Veloso, editor, *IJCAI 2007, Proc. of the 20th Int. Joint Conf. on Artificial Intelligence*, pages 477–482, 2007.
16. S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Artificial Intelligence*. Springer, 1997.
17. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *Journal of Data Semantics*, 10:133–173, 2008.
18. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
19. M. Serrurier and H. Prade. Improving expressivity of inductive logic programming by learning different kinds of fuzzy rules. *Soft Computing*, 11(5):459–466, 2007.
20. D. Shibata, N. Inuzuka, S. Kato, T. Matsui, and H. Itoh. An induction algorithm based on fuzzy logic programming. In N. Zhong and L. Zhou, editors, *Methodologies for Knowledge Discovery and Data Mining*, volume 1574 of *Lecture Notes in Computer Science*, pages 268–273. Springer, 1999.
21. U. Straccia. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.
22. U. Straccia. Softfacts: a top-k retrieval engine for a tractable description logic accessing relational databases. Technical report, 2009.
23. U. Straccia. Softfacts: A top-k retrieval engine for ontology mediated access to relational databases. In *Proceedings of the 2010 IEEE International Conference on Systems, Man and Cybernetics (SMC-10)*, pages 4115–4122. IEEE Press, 2010.

On the satisfiability problem for a 4-level quantified syllogistic and some applications to modal logic

Domenico Cantone and Marianna Nicolosi Asmundo

Dipartimento di Matematica e Informatica, Università di Catania
Viale A. Doria 6, I-95125 Catania, Italy
e-mail: cantone@dmi.unict.it, nicolosi@dmi.unict.it

Abstract. We introduce a fragment of multi-sorted stratified syllogistic, called $4LQS^R$, admitting variables of four sorts and a restricted form of quantification, and prove that it has a solvable satisfiability problem by showing that it enjoys a small model property. Then, we consider the sublanguage $(4LQS^R)^k$ of $4LQS^R$, where the length of quantifier prefixes (over variables of sort 1) is bounded by $k \geq 0$, and prove that its satisfiability problem is NP-complete. Finally we show that modal logics S5 and K45 can be expressed in $(4LQS^R)^1$.

1 Introduction

Most of the decidability results in computable set theory concern one-sorted multi-level syllogistics, namely collections of formulae admitting variables of one sort only, which range over the von Neumann universe of sets (see [6, 8] for a thorough account of the state-of-art until 2001). Only a few stratified syllogistics, where variables of several sorts are allowed, have been investigated, despite the fact that in many fields of computer science and mathematics often one has to deal with multi-sorted languages. For instance, in modal logics, one has to consider entities of different types, namely worlds, formulae, and accessibility relations.

In [10] an efficient decision procedure was presented for the satisfiability of the Two-Level Syllogistic language ($2LS$). $2LS$ has variables of two sorts and admits propositional connectives together with the basic set-theoretic operators $\cup, \cap, \setminus$, and the predicates $=, \in$, and $\subseteq$. Then, in [2], it was shown that the extension of $2LS$ with the singleton operator and the Cartesian product operator is decidable. Tarski's and Presburger's arithmetics extended with sets have been analyzed in [4]. Subsequently, in [3], a three-sorted language $3LSSPU$ (Three-Level Syllogistic with Singleton, Powerset and general Union) has been proved decidable. Recently, in [7], it was shown that the Three-Level Quantified Syllogistic with Restricted quantifiers language ($3LQS^R$) is decidable. $3LQS^R$ admits variables of three sorts and a restricted form of quantification. Its vocabulary contains only the predicate symbols $=$ and $\in$. In spite of that, $3LQS^R$ allows to express several constructs of set theory. Among them, the most comprehensive

one is the set former, which in turn enables one to express other operators like the powerset operator, the singleton operator, and so on.

In this paper we present a decidability result for the satisfiability problem of the set-theoretic language $4LQS^R$ (Four-Level Quantified Syllogistic with Restricted quantifiers). $4LQS^R$ is an extension of $3LQS^R$ which admits variables of four sorts and a restricted form of quantification over variables of the first three sorts. Its vocabulary contains the pairing operator $\langle \cdot, \cdot \rangle$ and the predicate symbols $=$ and $\in$.

We will prove that $4LQS^R$ enjoys a small model property by showing how one can extract, out of a given model satisfying a $4LQS^R$-formula ψ, another model of ψ but of bounded finite cardinality. The construction of the finite model extends the decision algorithm described in [7]. Concerning complexity issues, we will show that the satisfiability problem for each of the sublanguages $(4LQS^R)^k$ of $4LQS^R$, whose formulae are restricted to have quantifier prefixes over variables of sort 1 of length at most $k \geq 0$, is NP-complete.

Clearly, $4LQS^R$ can express all the set-theoretical constructs which are already expressible by $3LQS^R$. In addition, in $4LQS^R$ one can plainly formalize several properties of binary relations also needed to define accessibility relations of well-known modal logics. $4LQS^R$ can also express Boolean operations over relations and the inverse operation over binary relations. Finally, we will show that the modal logics S5 and K45 can be easily formalized in the $(4LQS^R)^1$ language. Since the satisfiability problems for S5 and K45 are NP-complete, in terms of computational complexity the algorithm we present here can be considered optimal for both logics.

2 The language $4LQS^R$

Before defining the language $4LQS^R$, in Section 2.1 we present the syntax and the semantics of a more general four-level quantified fragment, denoted $4LQS$. Then, in Section 2.2, we introduce some restrictions over the quantified formulae of $4LQS$ which characterize $4LQS^R$-formulae.

2.1 The more general language $4LQS$

Syntax of $4LQS$. The four-level quantified language $4LQS$ involves four collections $\mathcal{V}_0$, $\mathcal{V}_1$, $\mathcal{V}_2$, and $\mathcal{V}_3$ of variables.

(i) $\mathcal{V}_0$ contains variables of *sort 0*, denoted by $x, y, z, \ldots$;
(ii) $\mathcal{V}_1$ contains variables of *sort 1*, denoted by $X^1, Y^1, Z^1, \ldots$;
(iii) $\mathcal{V}_2$ contains variables of *sort 2*, denoted by $X^2, Y^2, Z^2, \ldots$;
(iv) $\mathcal{V}_3$ contains variables of *sort 3*, denoted by $X^3, Y^3, Z^3, \ldots$.

$4LQS$ quantifier-free atomic formulae are classified as:

level 0: $x = y$, $x \in X^1$, for $x, y \in \mathcal{V}_0, X^1 \in \mathcal{V}_1$;
level 1: $X^1 = Y^1$, $X^1 \in X^2$, for $X^1, Y^1 \in \mathcal{V}_1, X^2 \in \mathcal{V}_2$;

level 2: $X^2 = Y^2$, $\langle x, y\rangle = X^2$, $\langle x, y\rangle \in X^3$, $X^2 \in X^3$, for $X^2, Y^2 \in \mathcal{V}_2$, $x, y \in \mathcal{V}_0$, $X^3 \in \mathcal{V}_3$.

$4LQS$ *quantified atomic formulae* are classified as:

level 1: $(\forall z_1)\dots(\forall z_n)\varphi_0$, with φ_0 any propositional combination of quantifier-free atomic formulae, and $z_1, \dots, z_n$ variables of sort 0;
level 2: $(\forall Z_1^1)\dots(\forall Z_m^1)\varphi_1$, where $Z_1^1, \dots, Z_m^1$ are variables of sort 1, and φ_1 is any propositional combination of quantifier-free atomic formulae and of quantified atomic formulae of level 1;
level 3: $(\forall Z_1^2)\dots(\forall Z_p^2)\varphi_2$, with φ_2 any propositional combination of quantifier-free atomic formulae and of quantified atomic formulae of levels 1 and 2, and $Z_1^2, \dots, Z_p^2$ variables of sort 2.

Finally, the formulae of $4LQS$ are all the propositional combinations of quantifier-free atomic formulae of levels 0, 1, 2, and of quantified atomic formulae of levels 1, 2, 3.

Semantics of $4LQS$. A $4LQS$-*interpretation* is a pair $\boldsymbol{\mathcal{M}} = (D, M)$, where D is any *nonempty* collection of objects, called the *domain* or *universe* of $\boldsymbol{\mathcal{M}}$, and M is an assignment over the variables of $4LQS$ such that

- $Mx \in D$, for each $x \in \mathcal{V}_0$;
- $MX^1 \in \mathrm{pow}(D)$, for each $X^1 \in \mathcal{V}_1$;
- $MX^2 \in \mathrm{pow}(\mathrm{pow}(D))$, for all $X^2 \in \mathcal{V}_2$;
- $MX^3 \in \mathrm{pow}(\mathrm{pow}(\mathrm{pow}(D)))$, for all $X^3 \in \mathcal{V}_3$.[1]

Moreover we put $M\langle x, y\rangle = \{\{Mx\}, \{Mx, My\}\}$. Let

- $\boldsymbol{\mathcal{M}} = (D, M)$ be a $4LQS$-interpretation,
- $x_1, \dots, x_n \in \mathcal{V}_0$,
- $X_1^1, \dots, X_m^1 \in \mathcal{V}_1$,
- $X_1^2, \dots, X_p^2 \in \mathcal{V}_2$,
- $u_1, \dots, u_n \in D$,
- $U_1^1, \dots, U_m^1 \in \mathrm{pow}(D)$,
- $U_1^2, \dots, U_p^2 \in \mathrm{pow}(\mathrm{pow}(D))$.

By $\boldsymbol{\mathcal{M}}[x_1/u_1, \dots, x_n/u_n, X_1^1/U_1^1, \dots, X_m^1/U_m^1, X_1^2/U_1^2, \dots, X_p^2/U_p^2]$, we denote the interpretation $\boldsymbol{\mathcal{M}}' = (D, M')$ such that $M'x_i = u_i$, for $i = 1, \dots, n$, $M'X_j^1 = U_j^1$, for $j = 1, \dots, m$, $M'X_k^2 = U_k^2$, for $k = 1, \dots, p$, and which otherwise coincides with $\boldsymbol{\mathcal{M}}$ on all remaining variables. Throughout the paper we use the abbreviations: $\mathcal{M}^z$ for $\boldsymbol{\mathcal{M}}[z_1/u_1, \dots, z_n/u_n]$, $\mathcal{M}^{Z^1}$ for $\boldsymbol{\mathcal{M}}[Z_1^1/U_1^1, \dots, Z_m^1/U_m^1]$, and $\mathcal{M}^{Z^2}$ for $\boldsymbol{\mathcal{M}}[Z_1^2/U_1^2, \dots, Z_p^2/U_p^2]$.

Let φ be a $4LQS$-formula and let $\boldsymbol{\mathcal{M}} = (D, M)$ be a $4LQS$-interpretation. The notion of *satisfiability* of φ by $\boldsymbol{\mathcal{M}}$ (denoted by $\boldsymbol{\mathcal{M}} \models \varphi$) is defined inductively over the structure of the formula. Quantifier-free atomic formulae are interpreted in the standard way according to the usual meaning of the predicates '$=$' and '$\in$', and quantified atomic formulae are evaluated as follows:

[1] We recall that, for any set s, $\mathrm{pow}(s)$ denotes the *powerset* of s, i.e., the collection of all subsets of s.

1. $\mathcal{M} \models (\forall z_1) \ldots (\forall z_n)\varphi_0$ iff $\mathcal{M}[z_1/u_1, \ldots, z_n/u_n] \models \varphi_0$, for all $u_1, \ldots, u_n \in D$;
2. $\mathcal{M} \models (\forall Z_1^1) \ldots (\forall Z_m^1)\varphi_1$ iff $\mathcal{M}[Z_1^1/U_1^1, \ldots, Z_m^1/U_m^1] \models \varphi_1$, for all $U_1^1, \ldots, U_m^1 \in \mathrm{pow}(D)$;
3. $\mathcal{M} \models (\forall Z_1^2) \ldots (\forall Z_p^2)\varphi_2$ iff $\mathcal{M}[Z_1^2/U_1^2, \ldots, Z_p^2/U_p^2] \models \varphi_2$, for all $U_1^2, \ldots, U_p^2 \in \mathrm{pow}(\mathrm{pow}(D))$.

Finally, evaluation of compound formulae plainly follows the standard rules of propositional logic. Let ψ be a $4LQS$-formula, if $\mathcal{M} \models \psi$, i.e. $\mathcal{M}$ *satisfies* ψ, then $\mathcal{M}$ is said to be a $4LQS$-*model* for ψ. A $4LQS$-formula is said to be *satisfiable* if it has a $4LQS$-model. A $4LQS$-formula is *valid* if it is satisfied by all $4LQS$-interpretations.

2.2 Characterizing $4LQS^R$

$4LQS^R$ is the subcollection of the formulae ψ of $4LQS$ for which the following restrictions hold.

I. For *every* atomic formula $(\forall Z_1^1), \ldots, (\forall Z_m^1)\varphi_1$ of level 2 occurring in ψ and *every* level 1 atomic formula of the form $(\forall z_1) \ldots (\forall z_n)\varphi_0$ occurring in φ_1, φ_0 is a propositional combination of level 0 atoms and the condition

$$\neg\varphi_0 \rightarrow \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m} z_i \in Z_j^1 \tag{1}$$

is a valid $4LQS$-formula (in this case we say that the atom $(\forall z_1) \ldots (\forall z_n)\varphi_0$ is *linked* to the variables $Z_1^1, \ldots, Z_m^1$).

II. *Every* atomic formula of level 3 in ψ is either of type $(\forall Z_1^2), \ldots, (\forall Z_p^2)\varphi_2$, where φ_2 is a propositional combination of quantifier-free atomic formulae, or of type $(\forall Z^2)(Z^2 \in X^3 \leftrightarrow \neg(\forall z_1)(\forall z_2)\neg(\langle z_1, z_2\rangle = Z^2)$.

Restriction (I) is similar to the one described in [7]. In particular, following [7], we recall that condition (1) guarantees that if a given interpretation assigns to $z_1, \ldots, z_n$ elements of the domain that make φ_0 false, then such elements must be contained in at least one of the sets assigned to $Z_1^1, \ldots, Z_m^1$. This fact is needed in the proof of statement (ii) of Lemma 5 to make sure that satisfiability is preserved in a suitable finite submodel (details, however, are not reported here and can be found in [7]).

Through several examples, in [7] it is argued that condition (1) is not particularly restrictive. Indeed, to establish whether a given $4LQS$-formula is a $4LQS^R$-formula, since condition (1) is a $2LS$-formula, its validity can be checked using the decision procedure in [10], as $4LQS$ is a conservative extension of $2LS$. In addition, in many cases of interest, condition (1) is just an instance of the simple propositional tautology $\neg(A \rightarrow B) \rightarrow A$, and thus its validity can be established just by inspection.

Restriction (II) has been introduced to be able to express binary relations and several operations on relations keeping low, at the same time, the complexity of the decision procedure of Section 3.2.

Finally, we observe that though the semantics of $4LQS^R$ plainly coincides with the one given above for $4LQS$-formulae, in what follows we prefer to refer to $4LQS$-interpretations of $4LQS^R$-formulae as *$4LQS^R$-interpretations*.

3 The satisfiability problem for $4LQS^R$-formulae

We will solve the satisfiability problem for $4LQS^R$, i.e. the problem of establishing for any given formula of $4LQS^R$ whether it is satisfiable or not, as follows:

(i) firstly, we will show how to reduce effectively the satisfiability problem for $4LQS^R$-formulae to the satisfiability problem for *normalized* $4LQS^R$-*conjunctions* (these will be defined below);
(ii) secondly, we will prove that normalized $4LQS^R$-conjunctions enjoy a small model property.

From (i) and (ii), the solvability of the satisfiability problem for $4LQS^R$ follows immediately. Additionally, by further elaborating on point (i), it could easily be shown that indeed the whole collection of $4LQS^R$-formulae enjoys a small model property.

3.1 Normalized $4LQS^R$-conjunctions

Let ψ be a formula of $4LQS^R$ and let ψ_{DNF} be a disjunctive normal form of ψ. Then ψ is satisfiable if and only if at least one of the disjuncts of ψ_{DNF} is satisfiable. We recall that the disjuncts of ψ_{DNF} are conjunctions of literals, namely atomic formulae or their negation. In view of the previous observations, without loss of generality, we can suppose that our formula ψ is a conjunction of level $0, 1, 2$ quantifier-free literals and of level $1, 2, 3$ quantified literals. In addition, we can also assume that no variable occurs both bound and free in ψ and that distinct occurrences of quantifiers bind distinct variables.

For decidability purposes, negative quantified conjuncts occurring in ψ can be removed as follows. Let $\boldsymbol{\mathcal{M}} = (D, M)$ be a model for ψ, and let $\neg(\forall z_1)\dots(\forall z_n)\varphi_0$ be a negative quantified literal of level 1 in ψ. Since $\boldsymbol{\mathcal{M}} \models \neg(\forall z_1)\dots(\forall z_n)\varphi_0$ if and only if $\mathcal{M}[z_1/u_1, \dots, z_n/u_n] \models \neg\varphi_0$, for some $u_1, \dots, u_n \in D$, we can replace $\neg(\forall z_1)\dots(\forall z_n)\varphi_0$ in ψ by $\neg(\varphi_0)^{z_1,\dots,z_n}_{z'_1,\dots,z'_n}$, where $z'_1, \dots, z'_n$ are newly introduced variables of sort 0. Negative quantified literals of levels 2 and 3 can be dealt with much in the same way and hence, we can further assume that ψ is a conjunction of literals of the following types:

(1) quantifier-free literals of any level;
(2) quantified atomic formulae of level 1;
(3) quantified atomic formulae of levels 2 and 3 satisfying the restrictions given in Section 2.2.

We call these formulae *normalized $4LQS^R$-conjunctions*.

3.2 A small model property for normalized $4LQS^R$-conjunctions

In view of the above reductions, we can limit ourselves to consider the satisfiability problem for normalized $4LQS^R$-conjunctions only.

Thus, let ψ be a normalized $4LQS^R$-conjunction and assume that $\boldsymbol{\mathcal{M}} = (D, M)$ is a model for ψ.

We show how to construct, out of $\boldsymbol{\mathcal{M}}$, a finite $4LQS^R$-interpretation $\boldsymbol{\mathcal{M}}^* = (D^*, M^*)$ which is a model of ψ and such that the size of D^* depends solely on the size of ψ. We will proceed as follows. First we outline a procedure for the construction of a suitable nonempty finite universe $D^* \subseteq D$. Then we show how to relativize $\boldsymbol{\mathcal{M}}$ to D^* according to Definition 1 below, thus defining a finite $4LQS^R$-interpretation $\boldsymbol{\mathcal{M}}^* = (D^*, M^*)$. Finally, we prove that $\boldsymbol{\mathcal{M}}^*$ satisfies ψ.

Construction of the universe D^*. Let us denote by $\mathcal{V}'_0$, $\mathcal{V}'_1$, and $\mathcal{V}'_2$ the collections of variables of sort 0, 1, and 2 occurring free in ψ, respectively. Then we construct D^* according to the following steps:

Step 1: Let $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, where
- $\mathcal{F}_1$ 'distinguishes' the set $S = \{MX^2 : X^2 \in \mathcal{V}'_2\}$, in the sense that $K \cap \mathcal{F}_1 \neq K' \cap \mathcal{F}_1$ for every distinct $K, K' \in S$. Such a set $\mathcal{F}_1$ can be constructed by the procedure *Distinguish* described in [5]. As shown in [5], we can also assume that $|\mathcal{F}_1| \leq |S| - 1$.
- $\mathcal{F}_2$ satisfies $|MX^2 \cap \mathcal{F}_2| \geq \min(3, |MX^2|)$, for every $X^2 \in \mathcal{V}'_2$. Plainly, we can also assume that $|\mathcal{F}_2| \leq 3 \cdot |\mathcal{V}'_2|$.

Step 2: Let $\{F_1, \ldots, F_k\} = \mathcal{F} \backslash \{MX^1 : X^1 \in \mathcal{V}'_1\}$ and let $\mathcal{V}_1^F = \{X_1^1, \ldots, X_k^1\} \subseteq \mathcal{V}_1$ be such that $\mathcal{V}_1^F \cap \mathcal{V}'_1 = \emptyset$ and $\mathcal{V}_1^F \cap \mathcal{V}_1^B = \emptyset$, where $\mathcal{V}_1^B$ is the collection of bound variables in ψ. Let $\overline{\boldsymbol{\mathcal{M}}}$ be the interpretation $\boldsymbol{\mathcal{M}}[X_1^1/F_1, \ldots, X_k^1/F_k]$. Since the variables in $\mathcal{V}_1^F$ do not occur in ψ (neither free nor bound), their evaluation is immaterial for ψ and therefore, from now on, we identify $\overline{\boldsymbol{\mathcal{M}}}$ and $\boldsymbol{\mathcal{M}}$.

Step 3: Let $\Delta = \Delta_1 \cup \Delta_2$, where
- Δ_1 distinguishes the set $T = \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)\}$ and $|\Delta_1| \leq |T| - 1$ holds (cf. Step 1 above).
- Δ_2 satisfies $|J \cap \Delta_2| \geq \min(3, |J|)$, for every $J \in \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)\}$. Plainly, we can assume that $|\Delta_2| \leq 3 \cdot |\mathcal{V}'_1 \cup \mathcal{V}_1^F|$.

We then initialize D^* by putting

$$D^* := \{Mx : x \text{ in } \mathcal{V}'_0\} \cup \Delta\,.$$

Step 4: Let $\psi_1, \ldots, \psi_r$ be the conjuncts of ψ. To each conjunct ψ_i of the form $(\forall Z^1_{i,h_1}) \ldots (\forall Z^1_{i,h_{m_i}})\varphi_i$ we associate the collection $\varphi_{i,k_1}, \ldots, \varphi_{i,k_{\ell_i}}$ of atomic formulae of the form $(\forall z_1) \ldots (\forall z_n)\varphi_0$ present in the matrix of ψ_i, and call the variables $Z^1_{i,h_1}, \ldots, Z^1_{i,h_{m_i}}$ the *arguments of* $\varphi_{i,k_1}, \ldots, \varphi_{i,k_{\ell_i}}$. Let us put

$$\Phi = \{\varphi_{i,k_j} : 1 \leq j \leq \ell_i \text{ and } 1 \leq i \leq r\}.$$

Then, for each $\varphi \in \Phi$ of the form $(\forall z_1)\dots(\forall z_n)\varphi_0$ having $Z^1_1, \dots, Z^1_m$ as arguments, and for each ordered m-tuple $(X^1_{h_1}, \dots, X^1_{h_m})$ of variables in $\mathcal{V}'_1 \cup \mathcal{V}^F_1$, if $M(\varphi_0)^{Z^1_1, \dots, Z^1_m}_{X^1_{h_1}, \dots, X^1_{h_m}} = \mathbf{false}$ we insert in D^* elements $u_1, \dots, u_n \in D$ such that

$$M[z_1/u_1, \dots, z_n/u_n](\varphi_0)^{Z^1_1, \dots, Z^1_m}_{X^1_{h_1}, \dots, X^1_{h_m}} = \mathbf{false}\,,$$

otherwise we leave D^* unchanged.

Relativized interpretations. We introduce the notion of *relativized interpretation*, to be used together with the domain D^* constructed above, to define, out of a model $\boldsymbol{\mathcal{M}} = (D, M)$ for a $4LQS^R$-formula ψ, a finite interpretation $\boldsymbol{\mathcal{M}}^* = (D^*, M^*)$ of bounded size satisfying ψ as well.

Definition 1. *Let $\boldsymbol{\mathcal{M}} = (D, M)$ be a $4LQS^R$-interpretation. Let D^*, $\mathcal{V}'_1, \mathcal{V}^F_1$, and $\mathcal{V}'_2$ be as above, and let $d^* \in D^*$. The* relativized *interpretation of $\boldsymbol{\mathcal{M}}$ with respect to D^*, d^*, $\mathcal{V}'_1$, $\mathcal{V}^F_1$, and $\mathcal{V}'_2$,* $\mathrm{Rel}(\boldsymbol{\mathcal{M}}, D^*, d^*, \mathcal{V}'_1, \mathcal{V}^F_1, \mathcal{V}'_2) = (D^*, M^*)$*, is the interpretation such that*

$$\begin{aligned}
M^*x &= \begin{cases} Mx\,, \text{ if } Mx \in D^* \\ d^*\,, \quad \text{otherwise}\,, \end{cases} \\
M^*X^1 &= MX^1 \cap D^*\,, \\
M^*X^2 &= ((MX^2 \cap \mathrm{pow}(D^*)) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1)\}) \\
&\qquad \cup \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1),\ MX^1 \in MX^2\}\,, \\
M^*\langle x, y\rangle &= \{\{M^*x\}, \{M^*x, M^*y\}\}\,, \\
M^*X^3 &= ((MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^*X^2 : X^2 \in \mathcal{V}'_2\})\,, \\
&\qquad \cup \{M^*X^2 : X^2 \in \mathcal{V}'_2,\ MX^2 \in MX^3\}\,.
\end{aligned}$$

Concerning M^*X^2 and M^*X^3, we observe that they have been defined in such a way that all the membership relations between variables of ψ of sorts 2 and 3 are the same in both the interpretations $\boldsymbol{\mathcal{M}}$ and $\boldsymbol{\mathcal{M}}^*$. This fact will be proved in the next section.

For ease of notation, we will often omit the reference to the element $d^* \in D^*$ and write simply $\mathrm{Rel}(\boldsymbol{\mathcal{M}}, D^*, \mathcal{V}'_1, \mathcal{V}^F_1, \mathcal{V}'_2)$ in place of $\mathrm{Rel}(\boldsymbol{\mathcal{M}}, D^*, d^*, \mathcal{V}'_1, \mathcal{V}^F_1, \mathcal{V}'_2)$, when d^* is clear from the context.

The following useful properties are immediate consequences of the construction of D^*:

(A) if $MX^1 \neq MY^1$, then $(MX^1 \mathbin{\triangle} MY^1) \cap D^* \neq \emptyset$,[2]
(B) if $MX^2 \neq MY^2$, there is a $J \in (MX^2 \mathbin{\triangle} MY^2) \cap \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}'_F)\}$ such that $J \cap D^* \neq \emptyset$,
(C) if $M\langle x, y\rangle \neq MX^2$, there is a $J \in (MX^2 \mathbin{\triangle} M\langle x, y\rangle) \cap \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}'_F)\}$ such that $J \cap D^* \neq \emptyset$, and if $J \in MX^2$, $J \cap D^* \neq \{Mx\}$ and $J \cap D^* \neq \{Mx, My\}$,

[2] We recall that for any sets s and t, $s \mathbin{\triangle} t$ denotes the symmetric difference of s and of t, namely the set $(s \setminus t) \cup (t \setminus s)$.

for any $x, y \in \mathcal{V}'_0$, $X^1, Y^1 \in \mathcal{V}'_1$, and $X^2, Y^2 \in \mathcal{V}'_2$.

3.3 Soundness of the relativization

Let $\boldsymbol{\mathcal{M}} = (D, M)$ be a $4LQS^R$-interpretation satisfying a given $4LQS^R$-formula ψ, and let D^*, $\mathcal{V}'_1$, $\mathcal{V}'^F_1$, $\mathcal{V}'_2$, and $\boldsymbol{\mathcal{M}}^*$ be defined as above. The main result of this section is Theorem 1 which states that if $\boldsymbol{\mathcal{M}}$ satisfies ψ, then $\boldsymbol{\mathcal{M}}^*$ satisfies ψ as well. The proof of Theorem 1 exploits the technical Lemmas 1, 2, 3, 4, and 5 below. In particular, Lemma 1 states that $\boldsymbol{\mathcal{M}}$ satisfies a quantifier-free atomic formula φ fulfilling conditions (A), (B), and (C), if and only if $\boldsymbol{\mathcal{M}}^*$ satisfies φ too. Lemmas 2, 3, and 4 claim that suitably constructed variants of $\boldsymbol{\mathcal{M}}^*$ and the small models resulting by applying the construction of Section 3.2 to the corresponding variants of $\boldsymbol{\mathcal{M}}$ can be considered identical. Finally, Lemma 5, stating that if $\boldsymbol{\mathcal{M}}$ satisfies a quantified conjunction of ψ, then $\boldsymbol{\mathcal{M}}^*$ satisfies it as well, is proved by applying Lemmas 1, 2, 3, and 4.

Proofs of Lemmas 1, 2, 3, and 4 are routine and can be found in Appendices A.1, A.2, A.3, and A.4, respectively.

Lemma 1. *The following statements hold:*

(a) $\boldsymbol{\mathcal{M}}^* \models x = y$ *iff* $\boldsymbol{\mathcal{M}} \models x = y$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$;
(b) $\boldsymbol{\mathcal{M}}^* \models x \in X^1$ *iff* $\boldsymbol{\mathcal{M}} \models x \in X^1$, *for all* $X^1 \in \mathcal{V}_1$ *and* $x \in \mathcal{V}_0$ *such that* $Mx \in D^*$;
(c) $\boldsymbol{\mathcal{M}}^* \models X^1 = Y^1$ *iff* $\boldsymbol{\mathcal{M}} \models X^1 = Y^1$, *for all* $X^1, Y^1 \in \mathcal{V}_1$ *such that condition (A) holds;*
(d) $\boldsymbol{\mathcal{M}}^* \models X^1 \in X^2$ *iff* $\boldsymbol{\mathcal{M}} \models X^1 \in X^2$, *for all* $X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}'_F)$, $X^2 \in \mathcal{V}_2$;
(e) $\boldsymbol{\mathcal{M}}^* \models X^2 = Y^2$ *iff* $\boldsymbol{\mathcal{M}} \models X^2 = Y^2$, *for all* $X^2, Y^2 \in \mathcal{V}_2$ *such that condition (B) holds;*
(f) $\boldsymbol{\mathcal{M}}^* \models \langle x, y\rangle = X^2$ *iff* $\boldsymbol{\mathcal{M}} \models \langle x, y\rangle = X^2$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that condition (C) holds;*
(g) $\boldsymbol{\mathcal{M}}^* \models \langle x, y\rangle \in X^3$ *iff* $\boldsymbol{\mathcal{M}} \models \langle x, y\rangle \in X^3$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that condition (C) holds;*
(h) $\boldsymbol{\mathcal{M}}^* \models X^2 \in X^3$ *iff* $\boldsymbol{\mathcal{M}} \models X^2 \in X^3$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that conditions (B) and (C) hold.* □

In view of the next technical lemmas, we introduce the following notations. Let $u_1, \ldots, u_n \in D^*$, $U^1_1, \ldots, U^1_m \in \mathrm{pow}(D^*)$, and $U^2_1, \ldots, U^2_p \in \mathrm{pow}(\mathrm{pow}(D^*))$. Then we put

$$\begin{aligned}
\boldsymbol{\mathcal{M}}^{*,z} &= \boldsymbol{\mathcal{M}}^*[z_1/u_1, \ldots, z_n/u_n],\\
\boldsymbol{\mathcal{M}}^{*,Z^1} &= \boldsymbol{\mathcal{M}}^*[Z^1_1/U^1_1, \ldots, Z^1_m/U^1_m],\\
\boldsymbol{\mathcal{M}}^{*,Z^2} &= \boldsymbol{\mathcal{M}}^*[Z^2_1/U^2_1, \ldots, Z^2_p/U^2_p],
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{\mathcal{M}}^{z,*} &= \mathrm{Rel}(\boldsymbol{\mathcal{M}}^z, D^*, \mathcal{V}'_1, \mathcal{V}'^F_1, \mathcal{V}'_2),\\
\boldsymbol{\mathcal{M}}^{Z^1,*} &= \mathrm{Rel}(\boldsymbol{\mathcal{M}}^{Z^1}, D^*, \mathcal{V}'_1 \cup \{Z^1_1, \ldots, Z^1_m\}, \mathcal{V}'^F_1, \mathcal{V}'_2),\\
\boldsymbol{\mathcal{M}}^{Z^2,*} &= \mathrm{Rel}(\boldsymbol{\mathcal{M}}^{Z^2}, D^*, \mathcal{F}^*, \mathcal{V}'_1, \mathcal{V}'^F_1, \mathcal{V}'_2 \cup \{Z^2_1, \ldots, Z^2_p\}).
\end{aligned}$$

The next three lemmas claim that, under certain conditions, the following pairs of $4LQS^R$-interpretations $\mathcal{M}^{*,z}$ and $\mathcal{M}^{z,*}$, $\mathcal{M}^{*,Z^1}$ and $\mathcal{M}^{Z^1,*}$, $\mathcal{M}^{*,Z^2}$ and $\mathcal{M}^{Z^2,*}$ can be identified.

Lemma 2. *Let $u_1, \ldots, u_n \in D^*$, and let $z_1, \ldots, z_n \in \mathcal{V}_0$. Then, for every $x, y \in \mathcal{V}_0$, $X^1 \in \mathcal{V}_1$, $X^2 \in \mathcal{V}_2$, $X^3 \in \mathcal{V}_3$, we have:*

(i) $M^{*,z}x = M^{z,*}x$,
(ii) $M^{*,z}X^1 = M^{z,*}X^1$,
(iii) $M^{*,z}X^2 = M^{z,*}X^2$,
(iv) $M^{*,z}X^3 = M^{z,*}X^3$. □

Lemma 3. *Let $Z_1^1, \ldots, Z_m^1 \in \mathcal{V}_1 \setminus (\mathcal{V}_1' \cup \mathcal{V}_1^F)$ and $U_1^1, \ldots, U_m^1 \in \mathrm{pow}(D^*) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}$. Then, the $4LQS^R$-interpretations $\mathcal{M}^{*,Z^1}$ and $\mathcal{M}^{Z^1,*}$ coincide.* □

Lemma 4. *Let $Z_1^2, \ldots, Z_p^2 \in \mathcal{V}_2 \setminus \mathcal{V}_2'$ and $U_1^2, \ldots, U_p^2 \in \mathrm{pow}(\mathrm{pow}(D^*)) \setminus \{M^*X^2 : X^2 \in \mathcal{V}_2'\}$. Then the $4LQS^R$-interpretations $\mathcal{M}^{*,Z^2}$ and $\mathcal{M}^{Z^2,*}$ coincide.* □

The following lemma proves that satisfiability is preserved in the case of quantified atomic formulae.

Lemma 5. *Let $(\forall z_1)\ldots(\forall z_n)\varphi_0$, $(\forall Z_1^1)\ldots(\forall Z_m^1)\varphi_1$, $(\forall Z_1^2)\ldots(\forall Z_p^2)\varphi_2$, and $(\forall Z^2)(Z^2 \in X^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2))$ be conjuncts of ψ. Then*

(i) if $\mathcal{M} \models (\forall z_1)\ldots(\forall z_n)\varphi_0$, then $\mathcal{M}^* \models (\forall z_1)\ldots(\forall z_n)\varphi_0$;
(ii) if $\mathcal{M} \models (\forall Z_1)\ldots(\forall Z_m)\varphi_1$, then $\mathcal{M}^* \models (\forall Z_1)\ldots(\forall Z_m)\varphi_1$;
(iii) if $\mathcal{M} \models (\forall Z_1^2)\ldots(\forall Z_p^2)\varphi_2$, then $\mathcal{M}^* \models (\forall Z_1^2)\ldots(\forall Z_p^2)\varphi_2$;
(iv) if $\mathcal{M} \models (\forall Z^2)(Z^2 \in X^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2))$, then $\mathcal{M}^* \models (\forall Z^2)(Z^2 \in X^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2))$.

Proof. (i) Assume by contradiction that there exist $u_1, \ldots, u_n \in D^*$ such that $\mathcal{M}^{*,z} \not\models \varphi_0$. Then, there must be an atomic formula φ_0' in φ_0 that is interpreted differently in $\mathcal{M}^{*,z}$ and in $\mathcal{M}^z$. Recalling that φ_0 is a propositional combination of quantifier-free atomic formulae of any level, we can suppose that φ_0' is $X^2 = Y^2$ and, without loss of generality, assume that $\mathcal{M}^{*,z} \not\models X^2 = Y^2$. Then $M^{*,z}X^2 \neq M^{*,z}Y^2$, so that, by Lemma 2, $M^{z,*}X^2 \neq M^{z,*}Y^2$. Then, Lemma 1 yields $M^zX^2 \neq M^zY^2$, a contradiction. The other cases are proved in an analogous way.

(ii) This case can proved much along the same lines as the proof of case (ii) of Lemma 4 in [7]. Here, one has only to take care of the fact that the collection of relevant variables of sort 1 for ψ are not just the variables occurring free in ψ, namely the ones in $\mathcal{V}_1'$, but also the variables in $\mathcal{V}_1^F$, introduced to denote the elements distinguishing the sets M^*X^2, for $X^2 \in \mathcal{V}_2'$.

(iii) The proof is carried out as in case (ii).

(iv) Assume by contradiction that there exists a $U \in \mathrm{pow}(\mathrm{pow}(D^*))$ such that $\boldsymbol{\mathcal{M}}^{*,Z^2} \not\models (Z^2 \in X^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2))$. We can distinguish two cases:

1. If there is a $X^2 \in \mathcal{V}_2'$ such that $M^*X^2 = U$, then $\boldsymbol{\mathcal{M}}^* \not\models (X^2 \in X^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2))$ and either $X^2 \in X^3$ or $\neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$ must be interpreted differently in $\boldsymbol{\mathcal{M}}^*$ and in $\boldsymbol{\mathcal{M}}$.
   By Lemma 1, $X^2 \in X^3$ is interpreted in the same way in $\boldsymbol{\mathcal{M}}^*$ and in $\boldsymbol{\mathcal{M}}$. By case (i) of this lemma, if $\boldsymbol{\mathcal{M}}^* \models \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$ then $\boldsymbol{\mathcal{M}} \models \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$. Thus, the only case to be considered is when $\boldsymbol{\mathcal{M}}^* \models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$. Assume that $\boldsymbol{\mathcal{M}} \models \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$. Then MX^2 must be a pair $\{\{u\}, \{u, v\}\}$, for some $u, v \in D$. But then by the construction of the universe D^*, we have $u, v \in D^*$, contradicting the hypothesis that $\boldsymbol{\mathcal{M}}^* \models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = X^2)$.
2. If $U \neq M^*X^2$, for every $X^2 \in \mathcal{V}_2'$, either $Z^2 \in X^3$ or $\neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$ has to be interpreted in a different way in $\boldsymbol{\mathcal{M}}^{*,Z^2}$ and in $\boldsymbol{\mathcal{M}}^{Z^2}$.
   By Lemmas 4 and 1, and by case (i) of this lemma, $Z^2 \in X^3$ has the same evaluation in $\boldsymbol{\mathcal{M}}^{*,Z^2}$ and in $\boldsymbol{\mathcal{M}}^{Z^2}$, and if $\boldsymbol{\mathcal{M}}^{*,Z^2} \models \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$ then $\boldsymbol{\mathcal{M}}^{Z^2} \models \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$. The only case that still has to be analyzed is when $\boldsymbol{\mathcal{M}}^{*,Z^2} \models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$. By Lemma 4, $\boldsymbol{\mathcal{M}}^{Z^2,*} \models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$. Let us assume that $\boldsymbol{\mathcal{M}}^{Z^2} \not\models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$. Then U must be a pair $\{\{u\}, \{u, v\}\}$, $u, v \in D$. Since $U \in \mathrm{pow}(\mathrm{pow}(D^*))$, then $u, v \in D^*$, contradicting that $\boldsymbol{\mathcal{M}}^{Z^2,*} \models (\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)$. ■

Next, we can state our main result.

Theorem 1. *Let $\boldsymbol{\mathcal{M}}$ be a $4LQS^R$-interpretation satisfying ψ. Then $\boldsymbol{\mathcal{M}}^* \models \psi$.*

Proof. We have to prove that $\boldsymbol{\mathcal{M}}^* \models \psi'$ for each literal ψ' occurring in ψ. Each ψ' must be of one of the types introduced in Section 3.1. By applying Lemmas 1 or 5 to every ψ' (according to its type) we obtain the thesis. ■

From the above reduction and relativization steps, it is not hard to derive the following result:

Corollary 1. *The fragment $4LQS^R$ enjoys a small model property (and therefore its satisfiability problem is solvable).* □

3.4 Complexity issues

Let $(4LQS^R)^k$ be the sublanguage of $4LQS^R$ in which the quantifier prefixes of quantified atoms of level 2 have length not exceeding k. Then the following result holds.

Lemma 6. *The satisfiability problem for* $(4LQS^R)^k$ *is* NP*-complete, for any* $k \in \mathbb{N}$.

Proof. NP-hardness is trivially proved by reducing an instance of the satisfiability problem of propositional logic to our problem.

To prove that our problem is in NP, we reason as follows. Let φ be a satisfiable $(4LQS^R)^k$-formula. Let φ_{DNF} be a disjunctive normal form of φ. Then there is a disjunct ψ of φ_{DNF} that is satisfied by a $(4LQS^R)^k$-interpretation $\boldsymbol{\mathcal{M}} = (D, M)$. After the normalization step, ψ is a normalized $(4LQS^R)^k$-conjunction satisfied by $\boldsymbol{\mathcal{M}}$ and, according to the procedure of Section 3.2, we can construct a small interpretation $\boldsymbol{\mathcal{M}}^* = (D^*, M^*)$ satisfying ψ and such that $|D^*|$ is polynomial in the size of ψ. This can be shown by recalling that $|\mathcal{F}_1| \leq |S| - 1 \leq |\mathcal{V}'_2| - 1$ and that $|\mathcal{F}_2| \leq 3|\mathcal{V}'_2|$ (cf. Step 1 of the procedure in Section 3.2). Thus, clearly, $|\mathcal{F}| \leq 4|\mathcal{V}'_2| - 1$. Analogously, from Step 3, $|\Delta| \leq 4(|\mathcal{V}'_1| + (4|\mathcal{V}'_2| - 1)) - 1$, and $|D^*|$ (in the initialization phase) is bounded by $|\mathcal{V}'_0| + 4|\mathcal{V}'_1| + 16|\mathcal{V}'_2| - 5$. Finally, after Step 4, if we let L_n denote the maximal length of the quantifier prefix of $\varphi = (\forall z_1) \ldots (\forall z_n)\varphi_0$, with φ varying in Φ, then $|D^*| \leq |\mathcal{V}'_0| + 4|\mathcal{V}'_1| + 16|\mathcal{V}'_2| - 5 + ((|\mathcal{V}'_1| + 4|\mathcal{V}'_2| - 1)^k L_n)|\Phi|$. Thus the size of D^* is polynomial in the size of ψ. Since $\boldsymbol{\mathcal{M}}^* \models \psi$ can be verified in polynomial time and the size of ψ is polynomial w.r.t. the size of φ, it results that the satisfiability problem for $(4LQS^R)^k$ is in NP, and therefore it is NP-complete. ∎

4 Expressiveness of the language $4LQS^R$

As discussed in [7], $4LQS^R$ can express a restricted variant of the set former, which in turn allows to express other significant set operators such as binary union, intersection, set difference, the singleton operator, the powerset operator (over subsets of the universe only), etc. More specifically, atomic formulae of type $X^1 = \{z : \varphi(z)\}$ or $X^i = \{X^{i-1} : \varphi(X^{i-1})\}$, for $i \in \{2, 3\}$, can be expressed in $4LQS^R$ by the formulae

$$(\forall z)(z \in X^1 \leftrightarrow \varphi(z)) \tag{2}$$

$$(\forall X^{i-1})(X^{i-1} \in X^i \leftrightarrow \varphi(X^{i-1})) \tag{3}$$

provided that they satisfy the syntactic constraints of $4LQS^R$.

Since $4LQS^R$ is a superlanguage of $3LQS^R$, as shown in [7] $4LQS^R$ can express the stratified syllogistic $2LS$ and the sublanguage $3LSSP$ of $3LSSPU$ not involving the set-theoretic construct of general union. We recall that $3LSSPU$ admits variables of three sorts and, besides the usual set-theoretical constructs, it involves the 'singleton set' operator $\{\cdot\}$, the powerset operator pow, and the general union operator *Un*.

$3LSSP$ can plainly be decided by the decision procedure presented in [3] for the whole $3LSSPU$.

Other constructs of set theory which are expressible in the $4LQS^R$ formalism, as shown in [7], are:

Binary relation	$(\forall Z^2)(Z^2 \in R^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2))$
Reflexive	$(\forall z_1)(\langle z_1, z_1\rangle \in R^3)$
Symmetric	$(\forall z_1, z_2)(\langle z_1, z_2\rangle \in R^3 \rightarrow \langle z_2, z_1\rangle \in R^3)$
Transitive	$(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_2, z_3\rangle \in R^3) \rightarrow \langle z_1, z_3\rangle \in R^3)$
Euclidean	$(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_1, z_3\rangle \in R^3) \rightarrow \langle z_2, z_3\rangle \in R^3)$
Weakly-connected	$(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_1, z_3\rangle \in R^3) \rightarrow (\langle z_2, z_3\rangle \in R^3 \vee z_2 = z_3 \vee \langle z_3, z_2\rangle \in R^3))$
Irreflexive	$(\forall z_1)\neg(\langle z_1, z_1\rangle \in R^3)$
Intransitive	$(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_2, z_3\rangle \in R^3) \rightarrow \neg\langle z_1, z_3\rangle \in R^3)$
Antisymmetric	$(\forall z_1, z_2)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_2, z_1\rangle \in R^3) \rightarrow (z_1 = z_2))$
Asymmetric	$(\forall z_1, z_2)(\langle z_1, z_2\rangle \in R^3 \rightarrow \neg(\langle z_2, z_1\rangle \in R^3))$

Table 1. $4LQS^R$ formalization of conditions of accessibility relations

Intersection	$R^3 = R_1^3 \cap R_2^3$	$(\forall Z^2)(Z^2 \in R^3 \leftrightarrow (Z^2 \in R_1^3 \wedge Z^2 \in R_2^3))$
Union	$R^3 = R_1^3 \cup R_2^3$	$(\forall Z^2)(Z^2 \in R^3 \leftrightarrow (Z^2 \in R_1^3 \vee Z^2 \in R_2^3))$
Complement	$R_1^3 = \overline{R_2^3}$	$(\forall Z^2)(Z^2 \in R_1^3 \leftrightarrow \neg(Z^2 \in \overline{R_2^3}))$
Set difference	$R^3 = R_1^3 \setminus R_2^3$	$(\forall Z^2)(Z^2 \in R^3 \leftrightarrow (Z^2 \in R_1^3 \wedge \neg(Z^2 \in R_2^3)))$
Set inclusion	$R_1^3 \subseteq R_2^3$	$(\forall Z^2)(Z^2 \in R_1^3 \rightarrow Z^2 \in R_2^3)$

Table 2. $4LQS^R$ formalization of Boolean operations over relations

- the literal $X^2 = \mathrm{pow}_{\leq h}(X^1)$, where $\mathrm{pow}_{\leq h}(X^1)$ denotes the collection of all the subsets of X^1 having at most h elements;
- the literal $X^2 = \mathrm{pow}_{=h}(X^1)$, where $\mathrm{pow}_{=h}(X^1)$ denotes the collection of subsets of X^1 with exactly h elements;
- the unordered Cartesian product $X^2 = X_1^1 \otimes \ldots \otimes X_n^1$;
- the literal $A = \mathrm{pow}^*(X_1^1, \ldots, X_n^1)$, where $\mathrm{pow}^*(X_1^1, \ldots, X_n^1)$ is a variant of the powerset which denotes the collection

$$\{Z : Z \subseteq \bigcup_{i=1}^{n} X_i^1 \text{ and } Z \cap X_i^1 \neq \emptyset, \text{ for all } 1 \leq i \leq n\}$$

  introduced in [1].

4.1 Other applications of $4LQS^R$

Within the $4LQS^R$ language it is also possible to define binary relations over elements of a domain together with several conditions on them which characterize accessibility relations of well-known modal logics. These formalizations are illustrated in Table 1.

Usual Boolean operations over relations can be defined as shown in Table 2. Within the $4LQS^R$ fragment it is also possible to define the inverse of a given binary relation R_1^3, namely $R_2^3 = (R_1^3)^{-1}$, by means of the $4LQS^R$-formula $(\forall z_1, z_2)(\langle z_1, z_2\rangle \in R_1^3 \leftrightarrow \langle z_2, z_1\rangle \in R_2^3)$.

In the next section we will show how the $4LQS^R$ fragment can be used to formalize some normal modal logics.

4.2 Some normal modal logics expressible in $4LQS^R$

The *modal language* L_M is based on a countably infinite set of propositional letters $\mathcal{P} = \{p_1, p_2, \ldots\}$, the classical propositional connectives '$\neg$', '$\wedge$' , and '$\vee$', the modal operators '$\Box$', '$\Diamond$' (and the parentheses). L_M is the smallest set such that $\mathcal{P} \subseteq \mathsf{L}_M$, and such that if $\varphi, \psi \in \mathsf{L}_M$, then $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\Box\varphi$, $\Diamond\varphi \in \mathsf{L}_M$. Lower case letters like p denote elements of $\mathcal{P}$ and Greek letters like φ and ψ represent formulae of L_M. Given a formula φ of L_M, we indicate with $SubF(\varphi)$ the set of the subformulae of φ. The *modal depth* of a formula φ is the maximum nesting depth of modalities occurring in φ.

A *normal modal logic* is any subset of L_M which contains all the tautologies and the axiom

$$\mathsf{K}: \ \Box(p_1 \rightarrow p_2) \rightarrow (\Box p_1 \rightarrow \Box p_2)\,,$$

and which is closed with respect to modus ponens, substitution, and necessitation (the reader may consult a text on modal logic like [9] for more details).

A *Kripke frame* is a pair $\langle W, R\rangle$ such that W is a nonempty set of possible worlds and R is a binary relation on W called *accessibility relation*. If $R(w, u)$ holds, we say that the world u is accessible from the world w. A *Kripke model* is a triple $\langle W, R, h\rangle$, where $\langle W, R\rangle$ is a Kripke frame and h is a function mapping propositional letters into subsets of W. Thus, $h(p)$ is the set of all the worlds where p is true.

Let $K = \langle W, R, h\rangle$ be a Kripke model and let w be a world in K. Then, for every $p \in \mathcal{P}$ and for every $\varphi, \psi \in \mathsf{L}_M$, the relation of satisfaction $\models$ is defined as follows:

- $K, w \models p$ iff $w \in h(p)$;
- $K, w \models \varphi \vee \psi$ iff $K, w \models \varphi$ or $K, w \models \psi$;
- $K, w \models \varphi \wedge \psi$ iff $K, w \models \varphi$ and $K, w \models \psi$;
- $K, w \models \neg\varphi$ iff $K, w \not\models \varphi$;
- $K, w \models \Box\varphi$ iff $K, w' \models \varphi$, for every $w' \in W$ such that $(w, w') \in R$;
- $K, w \models \Diamond\varphi$ iff there is a $w' \in W$ such that $(w, w') \in R$ and $K, w' \models \varphi$.

A formula φ is said to be *satisfied* at w in K if $K, w \models \varphi$; φ is said to be *valid* in K (and we write $K \models \varphi$), if $K, w \models \varphi$, for every $w \in W$.

The smallest normal modal logic is K, which contains only the modal axiom K and whose accessibility relation R can be any binary relation. The other normal modal logics admit together with K other modal axioms drawn from the ones in Table 3.

Translation of a normal modal logic into the $4LQS^R$ language is based on the semantics of propositional and modal operators. For any normal modal logic, the formalization of the semantics of modal operators depends on the axioms that characterize the logic. In the case of the logics $\mathsf{S5}$ and $\mathsf{K45}$, proved to be NP-complete in [11], and introduced next, the $4LQS^R$ formalization of the modal formulae $\Box\varphi$ and $\Diamond\varphi$ turns out to be straightforward and thus these logics can be entirely translated into the $4LQS^R$ language. This is illustrated in what follows.

Axiom	Schema	Condition on R (see Table 1)
T	$\Box p \to p$	Reflexive
5	$\Diamond p \to \Box\Diamond p$	Euclidean
B	$p \to \Box\Diamond p$	Symmetric
4	$\Box p \to \Box\Box p$	Transitive
D	$\Box p \to \Diamond p$	Serial: $(\forall w)(\exists u)R(w,u)$

Table 3. Axioms of normal modal logics

The logic S5. Modal logic S5 is the strongest normal modal system. It can be obtained from the logic K in several ways. One of them consists in adding axioms T and **5** from Table 3 to the logic K. Given a formula φ, a Kripke model $K = \langle W, R, h\rangle$, and a world $w \in W$, the semantics of the modal operators can be defined as follows:

- $K, w \models \Box\varphi$ iff $K, v \models \varphi$, for every $v \in W$,
- $K, w \models \Diamond\varphi$ iff $K, v \models \varphi$, for some $v \in W$.

This makes it possible to translate a formula φ of S5 into the $4LQS^R$ language.

For the purpose of simplifying the definition of the translation function τ_{S5} given below, the concept of "empty formula" is introduced, to be denoted by Λ, and not interpreted in any particular way. The only requirement on Λ needed for the definition given next is that $\Lambda \wedge \psi$ and $\psi \wedge \Lambda$ are to be considered as syntactic variations of ψ, for any $4LQS^R$-formula ψ.

For every propositional letter p, let $\tau^1_{\mathsf{S5}}(p) = X^1_p$, where $X^1_p \in \mathcal{V}_1$, and let $\tau^2_{\mathsf{S5}} : \mathsf{S5} \to 4LQS^R$ be the function defined recursively as follows:

- $\tau^2_{\mathsf{S5}}(p) = \Lambda$,
- $\tau^2_{\mathsf{S5}}(\neg\varphi) = (\forall z)(z \in X^1_{\neg\varphi} \leftrightarrow \neg(z \in X^1_\varphi)) \wedge \tau^2_{\mathsf{S5}}(\varphi)$,
- $\tau^2_{\mathsf{S5}}(\varphi_1 \wedge \varphi_2) = (\forall z)(z \in X^1_{\varphi_1\wedge\varphi_2} \leftrightarrow (z \in X^1_{\varphi_1} \wedge z \in X^1_{\varphi_2})) \wedge \tau^2_{\mathsf{S5}}(\varphi_1) \wedge \tau^2_{\mathsf{S5}}(\varphi_2)$,
- $\tau^2_{\mathsf{S5}}(\varphi_1 \vee \varphi_2) = (\forall z)(z \in X^1_{\varphi_1\vee\varphi_2} \leftrightarrow (z \in X^1_{\varphi_1} \vee z \in X^1_{\varphi_2})) \wedge \tau^2_{\mathsf{S5}}(\varphi_1) \wedge \tau^2_{\mathsf{S5}}(\varphi_2)$,
- $\tau^2_{\mathsf{S5}}(\Box\varphi) =$
  $(\forall z)(z \in X^1_\varphi) \to (\forall z)(z \in X^1_{\Box\varphi}) \wedge \neg(\forall z)(z \in X^1_\varphi) \to (\forall z)\neg(z \in X^1_{\Box\varphi}) \wedge \tau^2_{\mathsf{S5}}(\varphi)$,
- $\tau^2_{\mathsf{S5}}(\Diamond\varphi) =$
  $\neg(\forall z)\neg(z \in X^1_\varphi) \to (\forall z)(z \in X^1_{\Diamond\varphi}) \wedge (\forall z)\neg(z \in X^1_\varphi) \to (\forall z)\neg(z \in X^1_{\Diamond\varphi}) \wedge \tau^2_{\mathsf{S5}}(\varphi)$,

where Λ is the empty formula and $X^1_{\neg\varphi}, X^1_\varphi, X^1_{\varphi_1\wedge\varphi_2}, X^1_{\varphi_1\vee\varphi_2}, X^1_{\varphi_1}, X^1_{\varphi_2} \in \mathcal{V}_1$.

Finally, for every φ in S5, if φ is a propositional letter in $\mathcal{P}$ we put $\tau_{\mathsf{S5}}(\varphi) = \tau^1_{\mathsf{S5}}(\varphi)$, otherwise $\tau_{\mathsf{S5}}(\varphi) = \tau^2_{\mathsf{S5}}(\varphi)$.

Even though the accessibility relation R is not used in the translation, we can give its formalization in the $4LQS^R$ fragment. Let U be defined so that $(\forall z)(z \in U)$, then R can be defined in the following two ways:

1. as a variable of sort 2, R^2, such that
   $(\forall Z^1)(Z^1 \in R^2 \leftrightarrow (Z^1 \in \mathrm{pow}_{=1}(U) \vee Z^1 \in \mathrm{pow}_{=2}(U)))$,
2. as a variable of sort 3, R^3, such that
   $(\forall Z^2)(Z^2 \in R^3 \leftrightarrow \neg(\forall z_1, z_2)\neg(\langle z_1, z_2\rangle = Z^2)) \wedge (\forall z_1)(\langle z_1, z_1\rangle \in R^3)$
   $\wedge(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_1, z_3\rangle \in R^3) \rightarrow \langle z_2, z_3\rangle \in R^3)$.

Correctness of the above translation is guaranteed by the following lemma, whose proof can be found in Appendix A.5.

Lemma 7. *For every formula φ of the logic $\mathsf{S5}$, φ is satisfiable in a model $K = \langle W, R, h\rangle$ iff there is a $4LQS^R$-interpretation satisfying $x \in X_\varphi$.* □

It can be checked that $\tau_{\mathsf{S5}}(\varphi)$ is polynomial in the size of φ and that its satisfiability can be verified in nondeterministic polynomial time since it belongs to $(4LQS^R)^1$. Consequently, the decision algorithm presented in this paper together with the translation function introduced above can be considered an optimal procedure (in terms of its computational complexity class) to decide the satisfiability of any formula φ of $\mathsf{S5}$. Moreover, it can be noticed that if we apply the first definition of R, $\mathsf{S5}$ can be expressed by the language $3LQS^R$ presented in [7].

The logic K45. The normal modal logic $\mathsf{K45}$ is obtained from the logic K by adding axioms **4** and **5** described in Table 3 to K. Semantics of the modal operators $\Box$ and $\Diamond$ for the logic $\mathsf{K45}$ can be described as follows. Given a formula φ of $\mathsf{K45}$ and a Kripke model $K = \langle W, R, h\rangle$,

- $K \models \Box\varphi$ iff $K, v \models \varphi$, for every $v \in W$ s.t. there is a $w' \in W$ with $(w', v) \in R$,
- $K \models \Diamond\varphi$ iff $K, v \models \varphi$, for some $v \in W$ s.t. there is a $w' \in W$ with $(w', v) \in R$.

It is convenient, before translating $\mathsf{K45}$ into the $4LQS^R$ fragment, to introduce the $4LQS^R$-formula which describes the semantics of the accessibility relation R of the logic $\mathsf{K45}$:

$$(\forall Z^2)(Z^2 \in R^3 \leftrightarrow \neg(\forall z_1)(\forall z_2)\neg(\langle z_1, z_2\rangle = Z^2))$$
$$\wedge(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_2, z_3\rangle \in R^3) \rightarrow \langle z_1, z_3\rangle \in R^3)$$
$$\wedge(\forall z_1, z_2, z_3)((\langle z_1, z_2\rangle \in R^3 \wedge \langle z_1, z_3\rangle \in R^3) \rightarrow \langle z_2, z_3\rangle \in R^3).$$

The transformation function $\tau_{\mathsf{K45}} : \mathsf{K45} \rightarrow 4LQS^R$ is constructed as for $\mathsf{S5}$. For every $\varphi \in \mathsf{K45}$ we put $\tau_{\mathsf{K45}}(\varphi) = \tau^1_{\mathsf{K45}}(\varphi)$, if φ is a propositional letter and $\tau_{\mathsf{K45}}(\varphi) = \tau^2_{\mathsf{K45}}(\varphi)$ otherwise. $\tau^1_{\mathsf{K45}}(p) = X^1_p$, with $X^1_p \in \mathcal{V}_1$, for every propositional letter p, and $\tau^2_{\mathsf{K45}}(\varphi)$ is defined inductively over the structure of φ. We report the definition of $\tau^2_{\mathsf{K45}}(\varphi)$ only when $\varphi = \Box\psi$ and $\varphi = \Diamond\psi$, as the other cases are identical to $\tau^2_{\mathsf{S5}}(\varphi)$, defined in the previous section:

– $\tau^2_{\mathsf{K45}}(\Box\psi) = (\forall z_1)((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \rightarrow z_1 \in X^1_\psi) \rightarrow (\forall z)(z \in X^1_{\Box\psi})$
$\wedge\neg(\forall z_1)\neg((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \wedge \neg(z_1 \in X^1_\psi)) \rightarrow (\forall z)\neg(z \in X^1_{\Box\psi}) \wedge \tau^2_{\mathsf{K45}}(\psi)$;

– $\tau^2_{\mathsf{K45}}(\Diamond\psi) = \neg(\forall z_1)\neg((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \wedge z_1 \in X^1_\psi) \rightarrow (\forall z)(z \in X^1_{\Diamond\psi})$
$\wedge(\forall z_1)(((\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \vee \neg(z_1 \in X^1_\psi))) \rightarrow (\forall z)\neg(z \in X^1_{\Diamond\psi}) \wedge \tau^2_{\mathsf{K45}}(\psi)$.

The following lemma, proved in Appendix A.6, shows the correctness of the translation.

Lemma 8. *For every formula φ of the logic τ_{K45}, φ is satisfiable in a model $K = \langle W, R, h\rangle$ iff there is a $4LQS^R$-interpretation satisfying $x \in X_\varphi$.* □

As for S5, it can be checked that $\tau_{\mathsf{K45}}(\varphi)$ is polynomial in the size of φ and that its satisfiability can be verified in nondeterministic polynomial time since it belongs to the sublanguage $(4LQS^R)^1$ of $4LQS^R$. Thus, the decision algorithm we have presented and the translation function introduced above represent an optimal procedure (in terms of its computational complexity class) to decide satisfiability of any formula φ of K45.

5 Conclusions

We have presented a decidability result for the satisfiability problem for the fragment $4LQS^R$ of multi-sorted stratified syllogistic embodying variables of four sorts and a restricted form of quantification. As the semantics of the modal formulae $\Box\varphi$ and $\Diamond\varphi$ in the modal logics S5 and K45 can be easily formalized in $4LQS^R$, it follows that $4LQS^R$ can express both logics S5 and K45.

Currently, in the case of modal logics characterized by having a liberal accessibility relation like K, we are not able to translate the modal formulae $\Box\varphi$ and $\Diamond\varphi$ in $4LQS^R$. The same problem concerns also the composition operation on binary relations and the set-theoretical operation of general union. We intend to investigate such a question more in depth and verify whether a formalization of these constructs is still possible in $4LQS^R$ or if an extension of the language $4LQS^R$ is required. In the same direction, we aim at finding a characterization of the conditions that an accessibility relation has to fulfil in order for a modal logic to be formalized in $4LQS^R$. We also intend to find classes of modal formulae with bounded modal nesting and multi-modal logics that can be embedded in the $4LQS^R$ framework. Finally, since $4LQS^R$ is able to express Boolean operations on relations, we plan to investigate the possibility of translating fragments of Boolean modal logics into $4LQS^R$.

References

1. D. Cantone. Decision procedures for elementary sublanguages of set theory: X. Multilevel syllogistic extended by the singleton and powerset operators. *Journal of Automated Reasoning*, volume 7, number 2, pages 193–230, Kluwer Academic Publishers, Hingham, MA, USA, 1991.

2. D. Cantone and V. Cutello. A decidable fragment of the elementary theory of relations and some applications. In *ISSAC '90: Proceedings of the international symposium on Symbolic and algebraic computation*, pages 24–29, New York, NY, USA, 1990. ACM Press.
3. D. Cantone and V. Cutello. Decision procedures for stratified set-theoretic syllogistics. In Manuel Bronstein, editor, *Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, ISSAC'93 (Kiev, Ukraine, July 6-8, 1993)*, pages 105–110, New York, 1993. ACM Press.
4. D. Cantone, V. Cutello, and J. T. Schwartz. Decision problems for Tarski and Presburger arithmetics extended with sets. In *CSL '90: Proceedings of the 4th Workshop on Computer Science Logic*, pages 95–109, London, UK, 1991. Springer-Verlag.
5. D. Cantone and A. Ferro Techniques of computable set theory with applications to proof verification. *Comm. Pure Appl. Math.*, pages 901–945, vol. XLVIII, 1995. Wiley.
6. D. Cantone, A. Ferro, and E. Omodeo. *Computable set theory.* Clarendon Press, New York, NY, USA, 1989.
7. D. Cantone and M. Nicolosi Asmundo. On the satisfiability problem for a 3-level quantified syllogistic. In *Proceedings of CEDAR'08"*. Sydney, Australia, 11 August, 2008.
8. D. Cantone, E. Omodeo, and A.Policriti. *Set Theory for Computing - From decision procedures to declarative programming with sets.* Springer-Verlag, Texts and Monographs in Computer Science, 2001.
9. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic.* Cambridge University Press, Cambridge Tracts in Theoretical Computer Science, 2001.
10. A. Ferro and E.G. Omodeo. *An efficient validity test for formulae in extensional two-level syllogistic.* Le Matematiche, 33:130–137, 1978.
11. R. Ladner. The computational complexity of provability in systems of modal propositional logic. *SIAM Journal of Computing,* 6: 467-480, 1977.

A Proofs of some lemmas

A.1 Proof of Lemma 1

Lemma 1. *The following statements hold:*

(a) $\boldsymbol{\mathcal{M}}^* \models x = y$ *iff* $\boldsymbol{\mathcal{M}} \models x = y$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$;
(b) $\boldsymbol{\mathcal{M}}^* \models x \in X^1$ *iff* $\boldsymbol{\mathcal{M}} \models x \in X^1$, *for all* $X^1 \in \mathcal{V}_1$ *and* $x \in \mathcal{V}_0$ *such that* $Mx \in D^*$;
(c) $\boldsymbol{\mathcal{M}}^* \models X^1 = Y^1$ *iff* $\boldsymbol{\mathcal{M}} \models X^1 = Y^1$, *for all* $X^1, Y^1 \in \mathcal{V}_1$ *such that condition (A) holds;*
(d) $\boldsymbol{\mathcal{M}}^* \models X^1 \in X^2$ *iff* $\boldsymbol{\mathcal{M}} \models X^1 \in X^2$, *for all* $X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}'_F)$, $X^2 \in \mathcal{V}_2$;
(e) $\boldsymbol{\mathcal{M}}^* \models X^2 = Y^2$ *iff* $\boldsymbol{\mathcal{M}} \models X^2 = Y^2$, *for all* $X^2, Y^2 \in \mathcal{V}_2$ *such that condition (B) holds;*
(f) $\boldsymbol{\mathcal{M}}^* \models \langle x, y\rangle = X^2$ *iff* $\boldsymbol{\mathcal{M}} \models \langle x, y\rangle = X^2$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that condition (C) holds;*
(g) $\boldsymbol{\mathcal{M}}^* \models \langle x, y\rangle \in X^3$ *iff* $\boldsymbol{\mathcal{M}} \models \langle x, y\rangle \in X^3$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that condition (C) holds;*
(h) $\boldsymbol{\mathcal{M}}^* \models X^2 \in X^3$ *iff* $\boldsymbol{\mathcal{M}} \models X^2 \in X^3$, *for all* $x, y \in \mathcal{V}_0$ *such that* $Mx, My \in D^*$ *and* $X^2 \in \mathcal{V}_2$ *such that conditions (B) and (C) hold.*

Proof. (a) Let $x, y \in \mathcal{V}_0$ be such that $Mx, My \in D^*$. Then $M^*x = Mx$ and $M^*y = My$, so we have immediately that $\mathcal{M}^* \models x = y$ iff $\mathcal{M} \models x = y$.
(b) Let $X^1 \in \mathcal{V}_1$ and let $x \in \mathcal{V}_0$ be such that $Mx \in D^*$. Then $M^*x = Mx$, so that $M^*x \in M^*X^1$ iff $Mx \in MX^1 \cap D^*$ iff $Mx \in MX^1$.
(c) If $MX^1 = MY^1$, then plainly $M^*X^1 = M^*Y^1$. On the other hand, if $MX^1 \neq MY^1$, then, by condition (A), $(MX^1 \mathbin{\triangle} MY^1) \cap D^* \neq \emptyset$ and thus $M^*X^1 \neq M^*Y^1$.
(d) If $MX^1 \in MX^2$, then $M^*X^1 \in M^*X^2$. On the other hand, suppose by contradiction that $MX^1 \notin MX^2$ and $M^*X^1 \in M^*X^2$. Then, there must necessarily be a $Z^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)$ with $MZ^1 \in MX^2$, $MZ^1 \neq MX^1$, and $M^*X^1 = M^*Z^1$. Since $MZ^1 \neq MX^1$ and $(MZ^1 \mathbin{\triangle} MX^1) \cap D^* \neq \emptyset$, by condition (A), we have $M^*X^1 \neq M^*Z^1$, which is a contradiction.
(e) If $MX^2 = MY^2$, then $M^*X^2 = M^*Y^2$. On the other hand, if $MX^2 \neq MY^2$, by condition (B), there is a $J \in (MX^2 \mathbin{\triangle} MY^2) \cap \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)\}$ such that $J \cap D^* \neq \emptyset$. Let $J = MX^1$, for some $X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)$, and suppose without loss of generality that $MX^1 \in MX^2$ and $MX^1 \notin MY^2$. Then, by (d), $M^*X^1 \in M^*X^2$ and $M^*X^1 \notin M^*Y^2$ and hence $M^*X^2 \neq M^*Y^2$.
(f) If $M\langle x, y\rangle = MX^2$, then $M^*\langle x, y\rangle = M^*X^2$. If $M\langle x, y\rangle \neq MX^2$, then there is a $J \in (MX^2 \mathbin{\triangle} M\langle x, y\rangle) \cap \{MX^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)\}$ satisfying the constraints of condition (C). Let $J = MX^1$, for some $X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}_1^F)$, and suppose that $MX^1 \in MX^2$ and $MX^1 \notin M\langle x, y\rangle$. Then $M^*X^1 \in M^*X^2$ and since $M^*X^1 \neq \{Mx\}$ and $M^*X^1 \neq \{Mx, My\}$, it follows that $M^*X^1 \notin M^*\langle x, y\rangle$. On the other hand, if $MX^1 \in M\langle x, y\rangle$ and $MX^1 \notin MX^2$, then either $MX^1 = \{Mx\}$ or $MX^1 = \{Mx, My\}$. In both cases $MX^1 = M^*X^1$ and thus if $MX^1 \notin MX^2$, it plainly follows that $M^*X^1 \notin M^*X^2$.

(g) Let $x, y \in \mathcal{V}_0$ and $X^3 \in \mathcal{V}_3$ be such that $M\langle x, y\rangle \in MX^3$. Then $M^*\langle x, y\rangle \in M^*X^3$. On the other hand, suppose by contradiction that $M\langle x, y\rangle \notin MX^3$ and $M^*\langle x, y\rangle \in M^*X^3$. Then, there must be an $X^2 \in \mathcal{V}_2'$ such that $M^*X^2 \in M^*X^3$, $M^*X^2 = M^*\langle x, y\rangle$, and $MX^2 \neq M\langle x, y\rangle$. But this is impossible by (f).

(h) If $MX^2 \in MX^3$ then $M^*X^2 \in M^*X^3$. Now suppose by contradiction that $MX^2 \notin MX^3$ and that $M^*X^2 \in M^*X^3$. Then, either there is a $Y^2 \in \mathcal{V}_2'$ such that $MX^2 \neq MY^2$ and $M^*X^2 = M^*Y^2$, which is not possible by (e), or there is a $\langle x, y\rangle$, with $x, y \in \mathcal{V}_0$, $Mx, My \in D^*$, such that $MX^2 \neq M\langle x, y\rangle$ and $M^*X^2 = M^*\langle x, y\rangle$, but this is absurd by (f). ■

A.2 Proof of Lemma 2

Lemma 2. *Let $u_1, \ldots, u_n \in D^*$, and let $z_1, \ldots, z_n \in \mathcal{V}_0$. Then, for every $x, y \in \mathcal{V}_0$, $X^1 \in \mathcal{V}_1$, $X^2 \in \mathcal{V}_2$, $X^3 \in \mathcal{V}_3$, we have:*

(i) $M^{*,z}x = M^{z,*}x$,
(ii) $M^{*,z}X^1 = M^{z,*}X^1$,
(iii) $M^{*,z}X^2 = M^{z,*}X^2$,
(iv) $M^{*,z}X^3 = M^{z,*}X^3$.

Proof. (i) Since $u_1, \ldots, u_n \in D^*$, the thesis follows immediately.
(ii) Let $X^1 \in \mathcal{V}_1$, then $M^{*,z}X^1 = M^*X^1 = MX^1 \cap D^* = M^zX^1 \cap D^* = M^{z,*}X^1$.
(iii) Let $X^2 \in \mathcal{V}_2$, then we have the following equalities:

$$\begin{aligned} M^{*,z}X^2 = M^*X^2 &= ((MX^2 \cap \mathrm{pow}(D^*)) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\quad \cup \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F),\ MX^1 \in MX^2\}, \\ &= ((M^zX^2 \cap \mathrm{pow}(D^*)) \setminus \{M^{z,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\quad \cup \{M^{z,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F), M^zX^1 \in M^zX^2\} \\ &= M^{z,*}X^2. \end{aligned}$$

(iv) Let $X^3 \in \mathcal{V}_3$, then the following holds:

$$\begin{aligned} M^{*,z}X^3 = M^*X^3 &= ((MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^*X^2 : X^2 \in \mathcal{V}_2'\}) \\ &\quad \cup \{M^*X^2 : X^2 \in \mathcal{V}_2', MX^2 \in MX^3\}, \\ &= ((M^zX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^{z,*}X^2 : X^2 \in \mathcal{V}_2'\}) \\ &\quad \cup \{M^{z,*}X^2 : X^2 \in \mathcal{V}_2', M^zX^2 \in M^zX^3\} \\ &= M^{z,*}X^3. \end{aligned}$$

■

A.3 Proof of Lemma 3

Lemma 3. *Let* $Z^1_1, \ldots, Z^1_m \in \mathcal{V}_1 \setminus (\mathcal{V}'_1 \cup \mathcal{V}^F_1)$ *and* $U^1_1, \ldots, U^1_m \in \mathrm{pow}(D^*) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1)\}$. *Then, the* $4LQS^R$*-interpretations* $\mathcal{M}^{*,Z^1}$ *and* $\mathcal{M}^{Z^1,*}$ *coincide.*

Proof. We prove the lemma by showing that $\mathcal{M}^{*,Z^1}$ and $\mathcal{M}^{Z^1,*}$ agree over variables of all sorts.

1. Clearly $M^{*,Z^1}x = M^*x = M^{Z^1,*}x$, for all individual variables $x \in \mathcal{V}_0$.
2. Let $X^1 \in \mathcal{V}_1$. If $X^1 \notin \{Z^1_1, \ldots, Z^1_m\}$, then

$$M^{Z^1,*}X^1 = M^{Z^1}X^1 \cap D^* = MX^1 \cap D^* = M^*X^1 = M^{*,Z^1}X^1\,.$$

On the other hand, if $X^1 = Z^1_j$ for some $j \in \{1, \ldots, m\}$, we have

$$M^{Z^1,*}Z^1_j = M^{Z^1}Z^1_j \cap D^* = U^1_j \cap D^* = U^1_j = M^{*,Z^1}Z^1_j\,.$$

3. Let $X^2 \in \mathcal{V}_2$. Then we have

$$\begin{aligned}
M^{*,Z^1}X^2 = M^*X^2 &= ((MX^2 \cap \mathrm{pow}(D^*)) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1)\}) \\
&\quad \cup \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1),\ MX^1 \in MX^2\}\,,
\end{aligned}$$

$$\begin{aligned}
M^{Z^1,*}X^2 &= ((M^{Z^1}X^2 \cap \mathrm{pow}(D^*)) \\
&\qquad \setminus \{M^{Z^1,*}X^1 : X^1 \in ((\mathcal{V}'_1 \cup \mathcal{V}^F_1) \cup \{Z^1_1, \ldots, Z^1_m\})\}) \\
&\quad \cup \{M^{Z^1,*}X^1 : X^1 \in ((\mathcal{V}'_1 \cup \mathcal{V}^F_1) \cup \{Z^1_1, \ldots, Z^1_m\}), \\
&\qquad M^{Z^1}X^1 \in M^{Z^1}X^2\} \\
&= ((MX^2 \cap \mathrm{pow}(D^*)) \\
&\qquad \setminus (\{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1)\} \cup \{U_j : j = 1, \ldots, m\})) \\
&\quad \cup (\{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1),\ MX^1 \in MX^2\} \\
&\quad \cup (\{U_j : j = 1, \ldots, m\} \cap MX^2))\,.
\end{aligned}$$

By putting

$$\begin{aligned}
&P_1 = MX^2 \cap \mathrm{pow}(D^*), \\
&P_2 = \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1)\}, \\
&P_3 = \{U_j : j = 1, \ldots, m\}, \\
&P_4 = \{M^*X^1 : X^1 \in (\mathcal{V}'_1 \cup \mathcal{V}^F_1),\ MX^1 \in MX^2\}, \\
&P_5 = \{U_j : j = 1, \ldots, m\} \cap MX^2,
\end{aligned}$$

the above relations can be rewritten as

$$\begin{aligned}
M^{*,Z^1}X^2 &= (P_1 \setminus P_2) \cup P_4 \\
M^{Z^1,*}X^2 &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup P_5\,.
\end{aligned}$$

Moreover, it is easy to verify that the following relations hold:

$$\begin{aligned} &P_2 \cap P_3 = \emptyset \\ &P_5 = P_1 \cap P_3 \\ &P_4 \subseteq P_2\,. \end{aligned}$$

Therefore we have

$$\begin{aligned} (P_1 \setminus P_2) \cup P_4 &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup (P_1 \cap P_3) \\ &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup P_5 \end{aligned}$$

i.e., we have $M^{*,Z^1}X^2 = M^{Z^1,*}X^2$.

4. Let $X^3 \in \mathcal{V}_3$, then $M^{*,Z^1}X^3 = M^*[Z_1^1/U_1^1,\ldots,Z_m^1/U_m^1]X^3 = M^*X^3$ and

$$\begin{aligned} M^{Z^1,*}X^3 &= ((M^{Z^1}X^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^{Z^1,*}X^2 : X^2 \in \mathcal{V}_2'\}) \\ &\qquad \cup \{M^{Z^1,*}X^2 : X^2 \in \mathcal{V}_2', M^{Z^1}X^2 \in M^{Z^1}X^3\} \\ &= ((MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^*X^2 : X^2 \in \mathcal{V}_2'\}) \\ &\qquad \cup \{M^*X^2 : X^2 \in \mathcal{V}_2', MX^2 \in MX^3\} \\ &= M^*X^3\,. \end{aligned}$$

Since $M^{*,Z^1}X^3 = M^{Z^1,*}X^3$ the thesis follows. ■

A.4 Proof of Lemma 4

Lemma 4. *Let* $Z_1^2,\ldots,Z_p^2 \in \mathcal{V}_2 \setminus \mathcal{V}_2'$ *and* $U_1^2,\ldots,U_p^2 \in \mathrm{pow}(\mathrm{pow}(D^*)) \setminus \{M^*X^2 : X^2 \in \mathcal{V}_2'\}$. *Then the* $4LQS^R$*-interpretations* $\boldsymbol{\mathcal{M}}^{*,Z^2}$ *and* $\boldsymbol{\mathcal{M}}^{Z^2,*}$ *coincide.*

Proof. We show that $\mathcal{M}^{*,Z^2}$ and $\mathcal{M}^{Z^2,*}$ coincide by proving that they agree over variables of all sorts.

1. Plainly $M^{*,Z^2}x = M^*x = M^{Z^2,*}x$, for every $x \in \mathcal{V}_0$.
2. Let $X^1 \in \mathcal{V}_1$, then $M^{*,Z^2}X^1 = M^*X^1 = M^{Z^2,*}X^1$.
3. Let $X^2 \in \mathcal{V}_2$ such that $X^2 \notin \{Z_1^2,\ldots,Z_p^2\}$, then

$$M^{*,Z^2}X^2 = M^*[Z_1^2/U_1^2,\ldots,Z_p^2/U_p^2]X^2 = M^*X^2,$$

and

$$\begin{aligned} M^{Z^2,*}X^2 &= ((M^{Z^2}X^2 \cap \mathrm{pow}(D^*)) \setminus \{M^{Z^2,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\qquad \cup \{M^{Z^2,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F), M^{Z^2}X^1 \in M^{Z^2}X^2\} \\ &= ((MX^2 \cap pow(D^*)) \setminus \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\qquad \cup \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F), MX^1 \in MX^2\} \\ &= M^*X^2\,. \end{aligned}$$

Since $M^{*,Z^2}X^2 = M^{Z^2,*}X^2$ the thesis follows. On the other hand, if $X^2 \in \{Z_1^2, \ldots, Z_p^2\}$, say $X^2 = Z_j^2$, then $M^{*,Z^2}X^2 = U_j^2$, and

$$\begin{aligned} M^{Z^2,*}X^2 &= ((M^{Z^2}X^2 \cap \mathrm{pow}(D^*)) \setminus \{M^{Z^2,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\qquad \cup \{M^{Z^2,*}X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F), M^{Z^2}X^1 \in M^{Z^2}X^2\} \\ &= (U_j^2 \setminus \{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F)\}) \\ &\qquad \cup (\{M^*X^1 : X^1 \in (\mathcal{V}_1' \cup \mathcal{V}_1^F), MX^1 \in U_j^2\}) \\ &= U_j^2 \,. \end{aligned}$$

Clearly the thesis follows also in this case.

4. Let $X^3 \in \mathcal{V}_3$. Then we have

$$\begin{aligned} M^{*,Z^2}X^3 = M^*X^3 &= ((MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \setminus \{M^*X^2 : X^2 \in \mathcal{V}_2'\}) \\ &\qquad \cup \{M^*X^2 : X^2 \in \mathcal{V}_2',\ MX^2 \in MX^3\} \\ M^{Z^2,*}X^3 &= ((M^{Z^2}X^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \\ &\qquad \setminus \{M^{Z^2,*}X^2 : X^2 \in \mathcal{V}_2' \cup \{Z_1^2, \ldots, Z_p^2\}\}) \\ &\qquad \cup \{M^{Z^2,*}X^2 : X^2 \in \mathcal{V}_2' \cup \{Z_1^2, \ldots, Z_p^2\},\ M^{Z^2}X^2 \in M^{Z^2}X^3\} \\ &= ((MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*))) \\ &\qquad \setminus (\{M^*X^2 : X^2 \in \mathcal{V}_2'\} \cup \{U_j^2 : j = 1, \ldots, p\})) \\ &\qquad \cup \{M^*X^2 : X^2 \in \mathcal{V}_2',\ MX^2 \in MX^3\} \\ &\qquad \cup (\{U_j^2 : j = 1, \ldots, p\} \cap MX^3) \,. \end{aligned}$$

By putting

$$\begin{aligned} P_1 &= MX^3 \cap \mathrm{pow}(\mathrm{pow}(D^*)) \\ P_2 &= \{M^*X^2 : X^2 \in \mathcal{V}_2'\} \\ P_3 &= \{U_j^2 : j = 1, \ldots, p\} \\ P_4 &= \{M^*X^2 : X^2 \in \mathcal{V}_2',\ MX^2 \in MX^3\} \\ P_5 &= \{U_j^2 : j = 1, \ldots, p\} \cap MX^3 \end{aligned}$$

then the above relations can be rewritten as

$$\begin{aligned} M^{*,Z^2}X^3 &= (P_1 \setminus P_2) \cup P_4 \\ M^{Z^2,*}X^3 &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup P_5 \,. \end{aligned}$$

Moreover, it is easy to verify that the following relations hold:

$$\begin{aligned} P_2 \cap P_3 &= \emptyset \\ P_5 &= P_1 \cap P_3 \\ P_4 &\subseteq P_2 \,. \end{aligned}$$

Therefore we have

$$\begin{aligned} (P_1 \setminus P_2) \cup P_4 &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup (P_1 \cap P_3) \\ &= (P_1 \setminus (P_2 \cup P_3)) \cup P_4 \cup P_5 \end{aligned}$$

i.e., we have $M^{*,Z^2}X^3 = M^{Z^2,*}X^3$. ■

A.5 Proof of Lemma 7

Lemma 7. *For every formula φ of the logic* S5*, φ is satisfiable in a model $K = \langle W, R, h\rangle$ iff there is a $4LQS^R$-interpretation satisfying $x \in X_\varphi$.*

Proof. Let $\bar{w}$ be a world in W. We construct a $4LQS^R$-interpretation $\boldsymbol{\mathcal{M}} = (W, M)$ as follows:

- $Mx = \bar{w}$,
- $MX^1_p = h(p)$, where p is a propositional letter and $X^1_p = \tau_{\mathsf{S5}}(p)$,
- $M\tau_{\mathsf{S5}}(\psi) = \mathbf{true}$, for every $\psi \in SubF(\varphi)$, where ψ is not a propositional letter.

To prove the lemma, it would be enough to show that $K, \bar{w} \models \varphi$ iff $M \models x \in X^1_\varphi$. However, it is more convenient to prove the following more general property:

Given a $w \in W$, if $y \in \mathcal{V}_0$ is such that $My = w$, then

$$K, w \models \varphi \text{ iff } M \models y \in X^1_\varphi,$$

which we do by structural induction on φ.

Base case: If φ is a propositional letter, by definition, $K, w \models \varphi$ iff $w \in h(\varphi)$. But this holds iff $My \in MX^1_\varphi$, which is equivalent to $M \models y \in X^1_\varphi$.

Inductive step: We consider only the cases in which $\varphi = \Box\psi$ and $\varphi = \Diamond\psi$, as the other cases can be dealt with similarly.

- If $\varphi = \Box\psi$, assume first that $K, w \models \Box\psi$. Then $K, w \models \psi$ and, by inductive hypothesis, $M \models y \in X^1_\psi$. Since $M \models \tau_{\mathsf{S5}}(\Box\psi)$, it holds that $M \models (\forall z_1)(z_1 \in X^1_\psi) \rightarrow (\forall z_2)(z_2 \in X^1_{\Box\psi})$. Then we have $M[z_1/w, z_2/w] \models (z_1 \in X^1_\psi) \rightarrow (z_2 \in X^1_{\Box\psi})$ and, since $My = w$, we have also that $M \models (y \in X^1_\psi) \rightarrow (y \in X^1_{\Box\psi})$. By the inductive hypothesis and by modus ponens we obtain $M \models y \in X^1_{\Box\psi}$, as required.
  On the other hand, if $K, w \not\models \Box\psi$, then $K, w \not\models \psi$ and, by inductive hypothesis, $M \not\models y \in X^1_\psi$. Since $M \models \tau_{\mathsf{S5}}(\Box\psi)$, then $M \models \neg(\forall z_1)(z_1 \in X^1_\psi) \rightarrow (\forall z_2)\neg(z_2 \in X^1_{\Box\psi})$. By the inductive hypothesis and some predicate logic manipulations, we have $M \models \neg(y \in X^1_\psi) \rightarrow \neg(y \in X^1_{\Box\psi})$, and by modus ponens we infer $M \models \neg(y \in X^1_{\Box\psi})$, as we wished to prove.
- Let $\varphi = \Diamond\psi$ and, to begin with, assume that $K, w \models \Diamond\psi$. Then, there is a w' such that $K, w' \models \psi$, and a $y' \in \mathcal{V}_0$ such that $My' = w'$. Thus, by inductive hypothesis, we have $M \models y' \in X^1_\psi$ and, by predicate logic, $M \models \neg(\forall z_1)\neg(z_1 \in X^1_\psi)$. By the very definition of M, $M \models \tau_{\mathsf{S5}}(\Diamond\psi)$ and thus $M \models \neg(\forall z_1)\neg(z_1 \in X^1_\psi) \rightarrow (\forall z_2)(z_2 \in X^1_{\Diamond\psi})$. Then, by modus ponens we obtain $M \models (\forall z_2)(z_2 \in X^1_{\Diamond\psi})$ and finally, by predicate logic, $M \models y \in X^1_{\Diamond\psi}$.
  On the other hand, if $K, w \not\models \Diamond\psi$, then $K, w' \not\models \psi$, for any $w' \in W$ and, since $w' = My'$ for any $y' \in \mathcal{V}_0$, it holds that $M \not\models y' \in X^1_\psi$ and thus, by predicate logic, $M \models (\forall z_1)\neg(z_1 \in X^1_\psi)$.

Reasoning as above, $M \models (\forall z_1)\neg(z_1 \in X^1_\psi) \rightarrow (\forall z_2)\neg(z_2 \in X^1_{\Diamond\psi})$ and, by modus ponens, $M \models (\forall z_2)\neg(z_2 \in X^1_{\Diamond\psi})$. Finally, by predicate logic, $M \not\models y \in X^1_{\Diamond\psi}$, as required. ■

A.6 Proof of Lemma 8

Lemma 8. *For every formula φ of the logic τ_{K45}, φ is satisfiable in a model $K = \langle W, R, h\rangle$ iff there is a $4LQS^R$-interpretation satisfying $x \in X_\varphi$.*

Proof. We proceed as in the proof of Lemma 7, by constructing a $4LQS^R$-interpretation $\boldsymbol{\mathcal{M}} = (W, M)$ which has the following property:

Given a $w \in W$ and a $y \in \mathcal{V}_0$ such that $My = w$, it holds that

$$K, w \models \varphi \text{ iff } M \models y \in X^1_\varphi.$$

We proceed by structural induction on φ. As with Lemma 7, we consider only the cases in which $\varphi = \Box\psi$ and $\varphi = \Diamond\psi$.

- Let $\varphi = \Box\psi$ and assume that $K, w \models \Box\psi$. Let v be a world of W such that there is a $u \in W$ with $\langle u, v\rangle \in R^3$, and let $x_1, x_2 \in \mathcal{V}_0$ be such that $v = Mx_1$ and $u = Mx_2$. We have that $K, v \models \psi$ and, by inductive hypothesis, $M \models x_1 \in X^1_\psi$. Since $M \models \tau_{\mathsf{K45}}(\Box\psi)$, then $M \models (\forall z_1)((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \rightarrow z_1 \in X^1_\psi) \rightarrow (\forall z)(z \in X^1_{\Box\psi})$. Hence $M[z_1/v, z_2/u, z/w] \models (\langle z_2, z_1\rangle \in R^3 \rightarrow z_1 \in X^1_\psi) \rightarrow z \in X^1_{\Box\psi}$ and thus $M \models (\langle x_2, x_1\rangle \in R^3 \rightarrow x_1 \in X^1_\psi) \rightarrow y \in X^1_{\Box\psi}$. Since $M \models \langle x_2, x_1\rangle \in R^3 \rightarrow x_1 \in X^1_\psi$, by modus ponens we have the thesis. The thesis follows also in the case in which there is no u such that $\langle u, v\rangle \in R^3$. In fact, in that case $M \models \langle x_2, x_1\rangle \in R^3 \rightarrow x_1 \in X^1_\psi$ holds for any $x_2 \in \mathcal{V}_0$.
  Consider next the case in which $K, w \not\models \Box\psi$. Then, there must be a $v \in W$ such that there is a u with $\langle u, v\rangle \in R^3$ and $K, v \not\models \psi$. Let $x_1, x_2 \in \mathcal{V}_0$ be such that $Mx_1 = v$ and $Mx_2 = u$. Then, by inductive hypothesis, $M \not\models x_1 \in X^1_\psi$. By definition of M, we have $M \models \neg(\forall z_1)\neg((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \wedge \neg(z_1 \in X^1_\psi)) \rightarrow (\forall z)\neg(z \in X^1_{\Box\psi})$. By the above instantiations and by the hypotheses, we have that $M \models ((\langle x_2, x_1\rangle \in R^3) \wedge \neg(x_1 \in X^1_\psi)) \rightarrow \neg(y \in X^1_{\Box\psi})$ and $M \models (\langle x_2, x_1\rangle \in R^3) \wedge \neg(x_1 \in X^1_\psi)$. Thus, by modus ponens, we obtain the thesis.
- Let $\varphi = \Diamond\psi$ and assume that $K, w \models \Diamond\psi$. Then there are $u, v \in W$ such that $\langle u, v\rangle \in R$ and $K, v \models \psi$. Let $x_1, x_2 \in \mathcal{V}_0$ be such that $Mx_1 = v$ and $Mx_2 = u$. Then, by inductive hypothesis, $M \models x_1 \in X^1_\psi$. Since $M \models \tau_{\mathsf{K45}}(\Diamond\psi)$, it follows that $M \models \neg(\forall z_1)\neg((\neg(\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \wedge z_1 \in X^1_\psi) \rightarrow (\forall z)(z \in X^1_{\Diamond\psi})$. By the hypotheses and the variable instantiations above it follows that $M \models ((\langle x_2, x_1\rangle \in R^3) \wedge x_1 \in X^1_\psi) \rightarrow y \in X^1_{\Diamond\psi}$ and $M \models (\langle x_2, x_1\rangle \in R^3) \wedge x_1 \in X^1_\psi$. Finally, by an application of modus ponens the thesis follows.

On the other hand, if $K, w \not\models \Diamond\psi$, then for every $v \in W$, either there is no $u \in W$ such that $\langle u, v\rangle \in R$, or $K, v \not\models \psi$. Let $x_1, x_2 \in \mathcal{V}_0$ be such that $Mx_1 = v$ and $Mx_2 = u$. If $K, v \not\models \psi$, by inductive hypothesis, we have that $M \not\models y \in X^1_\psi$.
Since $M \models (\forall z_1)(((\forall z_2)\neg(\langle z_2, z_1\rangle \in R^3)) \vee \neg(z_1 \in X^1_\psi)) \rightarrow (\forall z)\neg(z \in X^1_{\Diamond\psi})$, by the hypotheses and by the variable instantiations above we get $M \models (\neg(\langle x_2, x_1\rangle \in R^3) \vee \neg(x_1 \in X^1_\psi)) \rightarrow \neg(y \in X^1_{\Diamond\psi})$ and $M \models (\neg(\langle x_2, x_1\rangle \in R^3) \vee \neg(x_1 \in X^1_\psi))$. Finally, by modus ponens we infer the thesis. ■

The Birth of a WASP: Preliminary Report on a New ASP Solver*

Carmine Dodaro, Mario Alviano, Wolfgang Faber, Nicola Leone,
Francesco Ricca, and Marco Sirianni

Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
carminedodaro@gmail.com,
{alviano,faber,leone,ricca,sirianni}@mat.unical.it

Abstract. We present a new ASP solver for ground ASP programs that builds upon related techniques, originally introduced for SAT solving, which have been extended to cope with disjunctive logic programs under the stable model semantics. We describe the key components of this solving strategy, namely: learning, restarts, heuristics based on look-back concepts, and backjumping. At the same time, we introduce a new heuristics based on a mixed approach between look-back and look-ahead techniques. Moreover, we present the results of preliminary experiments that we conducted in order to assess the impact of these techniques on both random and structured instances (used also in the last ASP Competition 2011). In particular, we compared our system with both DLV and ClaspD.

1 Introduction

Answer Set Programming (ASP) [1] is a declarative programming paradigm which has been proposed in the area of non-monotonic reasoning and logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use a solver to find them [2].

The ASP language considered here allows disjunction in rule heads and nonmonotonic negation in rule bodies. These features make ASP very expressive; all problems in the second level of the polynomial hierarchy are indeed expressible in ASP [3]. Therefore, ASP is strictly more expressive than SAT (unless $P = NP$). Despite the intrinsic complexity of the evaluation of ASP, after twenty years of research many efficient ASP systems have been developed (e.g. [4–11]). The availability of robust implementations made ASP a powerful tool for developing advanced applications in the areas of Artificial Intelligence, Information Integration, or Knowledge Management; for example, ASP has been used in applications for team-building [12], semantic-based information extraction [13], and e-tourism [14]. These applications of ASP have confirmed the viability of the use of ASP. Nonetheless, the interest in developing more effective and faster systems is still a crucial and challenging research topic, as witnessed by the results of the ASP Contests series [15–17].

* Partly supported by Regione Calabria and EU under POR Calabria FESR 2007-2013 and within the PIA project of DLVSYSTEM s.r.l., and by MIUR under the PRIN project LoDeN. We also thank the anonymous reviewers for their valuable comments.

This paper provides a contribution in the aforementioned context. In particular, we provide a preliminary report on the development of a new ASP solver for propositional programs called $wasp$. The new system is inspired by several techniques that were originally introduced for SAT solving, like the Davis-Putnam-Logemann-Loveland (DPLL) backtracking search algorithm [18], *clause learning* [19, 20], *backjumping* [21, 22], *restarts* [23], and *conflict-driven heuristics* [24] in the style of Berkmin [25]. The mentioned SAT-solving methods have been adapted and combined with state-of-the-art pruning techniques adopted by modern native disjunctive ASP systems [4]. In particular, the role of Boolean Constraint Propagation in SAT-solvers (based on the simple *unit propagation* inference rule) is taken by a procedure combining a set of inference rules. Those rules combine an extension of the well-founded operator for disjunctive programs with a number of techniques based on ASP program properties (see, e.g., [26]). Moreover, $wasp$ uses a new branching heuristics tailored for ASP programs, which is based on a mixed approach between Berkmin-like heuristics and look-ahead, which takes into account minimality of answer sets (a requirement not present in SAT solving). Finally, stable model checking, which is a co-NP-complete problem for disjunctive logic programs, is efficiently implemented relying on the rewriting method of [27], by calling Minisat [28] as suggested by [29].

In the following, after briefly introducing ASP, we describe the new system $wasp$. We start from the solving strategy and present the design choices regarding propagation, constraint learning, restarts, and the new heuristics. Moreover, we present the results of some experiments conducted for assessing the impact of these techniques, on both random and structured instances; some of these instances had been used in the last ASP Competition [17]. In particular, we compared our system with both DLV and ClaspD. The obtained results are encouraging: the new prototype system is already competitive with state-of-the-art solvers, even if there is still room for improvements in both the implementation (e.g., through the optimization and tuning of data structures and heuristic parameters), and in the supported language features (notably aggregates and weak constraints).

2 Preliminaries

In this paper we consider propositional programs, so an atom p is a member of a countable set $\mathcal{A}$. A *literal* is either an atom p (a positive literal), or an atom preceded by the *negation as failure* symbol not (a negative literal). A *rule* r is of the form

$$p_1 \vee \cdots \vee p_n \texttt{ :- } q_1, \ldots, q_j, \mathrm{not}\ q_{j+1}, \ldots, \mathrm{not}\ q_m \tag{1}$$

where $p_1, \ldots, p_n, q_1, \ldots, q_m$ are atoms and $n \geq 0, m \geq j \geq 0$. The disjunction $p_1 \vee \cdots \vee p_n$ is the *head* of r, while the conjunction $q_1, \ldots, q_j, \mathrm{not}\ q_{j+1}, \ldots, \mathrm{not}\ q_m$ is the *body* of r. Moreover, $H(r)$ denotes the set of head atoms, while $B(r)$ denotes the set of body literals. We also use $B^+(r)$ and $B^-(r)$ for denoting the set of atoms appearing in positive and negative body literals, respectively, and $At(r)$ for the set $H(r) \cup B^+(r) \cup B^-(r)$. A rule r is normal (or disjunction-free) if $|H(r)| \leq 1$, positive (or negation-free) if $B^-(r) = \emptyset$, a *fact* if both $B(r) = \emptyset$ and $|H(r)| = 1$, a *constraint* if $|H(r)| = 0$.

A *program* $\mathcal{P}$ is a finite set of rules; if all rules in it are positive (resp. normal), then $\mathcal{P}$ is a positive (resp. normal) program.

Let $\overline{L}$ denote the complement of a literal L, i.e., $\overline{a} = \text{not } a$ and $\overline{\text{not } a} = a$ for an atom a. We extend this to sets of literals and will use $\overline{S}$ for denoting $\{\overline{L} \mid L \in S\}$. An *interpretation* I is a subset of $\mathcal{A} \cup \overline{\mathcal{A}}$. An interpretation I is total if for each $a \in \mathcal{A}$ either $a \in I$ or $\text{not } a \in I$; otherwise, I is partial. An interpretation I is inconsistent if there exists $a \in \mathcal{A}$ such that $\{a, \text{not } a\} \subseteq I$; otherwise, I is consistent. An interpretation thus associates each ASP structure (atom, literal, head or body) with a truth value in the set $\{\mathcal{T}, \mathcal{F}, \mathcal{U}\}$, which extends to $H(r)$ and $B(r)$ in the standard way.

An interpretation I *satisfies* a rule $r \in \mathcal{P}$ if $H(r)$ is true w.r.t. I whenever $B(r)$ is true w.r.t. I, while I *violates* r if $H(r)$ is false but $B(r)$ is true. A total interpretation I is a *model* of a program $\mathcal{P}$ if I satisfies all the rules in $\mathcal{P}$. Given an interpretation I for a program $\mathcal{P}$, the reduct of $\mathcal{P}$ w.r.t. I, denoted by $\mathcal{P}^I$, is obtained by deleting from $\mathcal{P}$ all the rules r with $B^-(r) \cap I \neq \emptyset$, and then by removing all the negative literals from the remaining rules. The semantics of a program $\mathcal{P}$ is given by the set $\mathcal{AS}(\mathcal{P})$ of the answer sets (or stable models) of $\mathcal{P}$, where a total interpretation M is an answer set (or stable model) for $\mathcal{P}$ if and only if M is a subset-minimal model of $\mathcal{P}^M$.

3 Model Generator

In this section we sketch the main model generator function MG (cf. Fig. 1), which is able to perform learning and restart techniques. MG is similar to the Davis-Putnam procedure in SAT solvers. For reasons of presentation, we have considerably simplified the procedure in order to focus on its main ideas. For example, the version described here computes only one answer set, but modifying it to compute all or n stable models is straightforward.

In the sequel, $\mathcal{P}$ will refer to the input program. Initially, the MG function is invoked with $I = \emptyset$, and $bj\_level = -1$ (but it will become 0 immediately), and the global variable numberOfConflicts is set to 0. MG returns true if the program $\mathcal{P}$ has an answer set, and sets I to the computed answer set; otherwise it returns false.

MG first calls a function Propagate, which extends I with those literals that can be deterministically inferred, and keeps track of the reason of each inference by building a representation of the so-called implication graph [24]. Propagate is similar to unit propagation as employed by SAT solvers, but exploits the peculiarities of ASP for making further inferences (e.g., it uses the knowledge that every answer set is a minimal model). Propagate, described in more detail in Section 3.1, returns false if an inconsistency (or conflict) is detected (i.e., the complement of a true literal is inferred to be true), true otherwise.

If Propagate returns true and no undefined atom is left in I, MG invokes CheckModel to verify that the current total interpretation is also an answer set; the CheckModel function implements the techniques described in [27]. If the stability check succeeds, MG returns true. [1] If Propagate returned true but I is still partial, an undefined literal L is selected according to a heuristic criterion and MG is recursively called. The atom L corresponds to a *branching variable* in SAT solvers.

[1] This is a co-NP-complete task in case of general disjunctive ASP programs.

If Propagate returns false, function ResolveConflict is called, which calculates the Unique Implication Point (UIP) of the implication graph (see Section 3.1), and exploits it to *learn* a constraint representing the inconsistency (see Section 3.2), which is added to the input program. As a by-product, ResolveConflict returns the recursion level to go back to (backjumping) in order to continue the search in the first branch of the search that is free of the just-detected conflict.

After a certain number of conflicts, ResolveConflict may decide to restart the entire search, if the total number of conflicts found during the search reached a certain threshold. It is important to note that after each restart MG works on a program composed of the original input program and the learned constraints. Our restart policy is based on the sequence of thresholds (32, 32, 64, 32, 32, 64, 128, ...) introduced in [30].

If the recursive call returned true, MG just returns true as well. If it returned false, the corresponding branch is inconsistent, bj_level is set to the recursion level to backtrack or backjump to. Now, if bj_level is less than the current level, this indicates a backjump, and we return. If not, then we have reached the level to go to, and the search continues.

```
bool MG (Interpretation& I, int& bj_level )
    int curr_level = ++ bj_level;

    if ( ! Propagate( I ) )
      bj_level = ResolveConflict();
      return false;
    if ( "no atom is undefined in I" )
      if ( CheckModel( I ) ) return true;
      else
        bj_level = ResolveConflict();
        return false;

    Select an undefined atom A using a heuristic;

    if ( MG( I ∪ {A}, bj_level ) ) return true;
    if (bj_level < curr_level) return false;

    if ( MG( I ∪ {not A}, bj_level ) ) return true;
    if (bj_level < curr_level) return false;

    return false;

int  ResolveConflict()
    int level = calculateFirstUIP();
    learning();
    if(inRestartSequence(numberOfConflict)) return 0;
    return level;
```

Fig. 1. Computation of answer sets

3.1 Propagation

WASP implements a number of deterministic inference rules for pruning the search space during the computation of stable models. These inference rules are named *forward inference*, *Kripke-Kleene negation*, *contraposition for true heads*, *contraposition for false heads* and *well-founded negation*. All of these inference rules are briefly described in this section.

During the propagation of deterministic inferences, implication relationships among atoms are stored in a graph $\mathcal{G}$ named Implication Graph. This graph has a node $\langle a, t\rangle$ for each atom a and truth value t such that a has been assigned t. Each node of the graph is associated with a *decision level*, which is set to the level of the backtracking tree when t is assigned to a. Moreover, $\mathcal{G}$ has a directed arc connecting a node $\langle a, t\rangle$ to a node $\langle a', t'\rangle$ whenever $\langle a, t\rangle$ is one of the reasons that lead to the derivation of the truth value t' for the atom a'. Note that $\mathcal{G}$ will contain at most one node for each atom of the program, unless a conflict is derived. The way of building $\mathcal{G}$ is described below.

Forward Inference. This is essentially modus ponens. When the body of a rule r is true w.r.t. the current partial interpretation, and all but one of the head atoms of r are false and the remaining one is undefined, then there is only one way to satisfy r, by deriving the remaining head atom as true.

Concerning the Implication Graph $\mathcal{G}$, it is updated as follows. Let r be of the form (1) and let p_i be the undefined atom in $H(r)$. The following elements are added to $\mathcal{G}$: a node $\langle p_i, \mathcal{T}\rangle$; arcs $(\langle q_k, \mathcal{T}\rangle, \langle p_i, \mathcal{T}\rangle)$ ($k = 1, \ldots, j$); arcs $(\langle q_k, \mathcal{F}\rangle, \langle p_i, \mathcal{T}\rangle)$ ($k = j+1, \ldots, m$); arcs $(\langle p_k, \mathcal{F}\rangle, \langle p_i, \mathcal{T}\rangle)$ ($k = 1, \ldots, n$ and $k \neq i$).

Kripke-Kleene Negation. This derives negative information by using supportedness, the fact that each atom a which is true in a stable model M must occur in at least one rule r such that $B(r)$ is true w.r.t. M and a is the only atom in $H(r)$ which is true w.r.t. M. Hence, atoms with no candidate supporting rules can be derived to be false. So, if all of the rules r such that $a \in H(r)$ are satisfied because of a false body literal or because of a true head atom different from a, atom a is inferred as false.

Concerning $\mathcal{G}$, a node $\langle a, \mathcal{F}\rangle$ is introduced. Moreover, for each rule r with $a \in H(r)$, let L be the first literal (in chronological order of derivation) that satisfied r. If $L \in B^+(r)$, an arc $(\langle L, \mathcal{F}\rangle, \langle a, \mathcal{F}\rangle)$ is added to $\mathcal{G}$; otherwise, if $L \in H(r)$, an arc $(\langle L, \mathcal{T}\rangle, \langle a, \mathcal{F}\rangle)$ is added to $\mathcal{G}$; otherwise, $\overline{L} \in B^-(r)$ and thus an arc $(\langle \overline{L}, \mathcal{T}\rangle, \langle a, \mathcal{F}\rangle)$ is added to $\mathcal{G}$.

Contraposition for True Heads. Supportedness is also used by this inference rule: If an atom a that has been derived as true has only one candidate supporting rule r, the truth of all literals in $B(r)$ and the falsity of all atoms in $H(r)$ different from a are inferred.

Concerning $\mathcal{G}$, the following new nodes and arcs are introduced: $\langle b, \mathcal{T}\rangle$ (for each $b \in B^+(r)$); $\langle b, \mathcal{F}\rangle$ (for each $b \in B^-(r) \cup H(r) \setminus \{a\}$); for each new node $\langle b, v\rangle$ an arc $(\langle a, \mathcal{T}\rangle, \langle b, v\rangle)$. Moreover, for each rule r' such that $a \in H(r')$, let L be the first literal (in chronological order of derivation) that satisfied r'. If $L \in B^+(r')$, an arc $(\langle L, \mathcal{F}\rangle, \langle b, v\rangle)$ is added to $\mathcal{G}$, otherwise, if $L \in B^-(r') \cup H(r') \setminus \{a\}$, an arc $(\langle L, \mathcal{T}\rangle, \langle b, v\rangle)$ is added to $\mathcal{G}$; this is done for each new node $\langle b, v\rangle$ introduced by the application of the inference rule for r.

Contraposition for False Heads. This inference rule is essentially modus tollens. When for a rule r all head atoms are false, the only way to satisfy r is by having a false body. In case all but one body literals of r are true, falsity of the remaining L is inferred.

Concerning $\mathcal{G}$, a node $\langle a, v\rangle$ is added, where a is the atom in L and $v = \mathcal{F}$ if $L = a$ or $v = \mathcal{T}$ if $L = \text{not } a$. Moreover, the following arcs are added to $\mathcal{G}$: $(\langle b, \mathcal{F}\rangle, \langle a, v\rangle)$ (for each $b \in H(r) \cup B^-(r) \setminus \{a\}$; $(\langle b, \mathcal{T}\rangle, \langle a, v\rangle)$ (for each $b \in B^+(r) \setminus \{a\}$).

Well-founded Negation. Unfounded sets are sets of unsupported or self-supporting atoms, that is, atoms that can have supporting rules only if their own truth is assumed. It is well-known that unfounded sets are disjoint from stable models, which allows for assuming the falsity of all the atoms that belong to some unfounded set. Hence, after the propagation process has been carried out, $wasp$ determines the set X of all the atoms belonging to some unfounded set and derives the falsity of these atoms; if this set is empty, the rule does not apply.

In order to model such a lack of external supporting rules, a number of nodes and arcs is added to $\mathcal{G}$. For each $a \in X$, a node $\langle a, \mathcal{F}\rangle$ is added. Arcs are introduced according to the following schema: Let C be the set of atoms in X that were previously derived as true, and let c be a randomly selected atom in C. For each $a \in X \setminus C$, an arc $(\langle a, \mathcal{F}\rangle, \langle c, \mathcal{F}\rangle)$ is added to $\mathcal{G}$. Moreover, for each $b \in C \setminus \{c\}$ and for each rule r such that $b \in H(r)$, let L be the first literal (in chronological order of derivation) that satisfied r. If $L \in B^+(r)$, an arc $(\langle L, \mathcal{F}\rangle, \langle c, \mathcal{F}\rangle)$ is added to $\mathcal{G}$; otherwise, if $L \in H(r)$, an arc $(\langle L, \mathcal{T}\rangle, \langle c, \mathcal{F}\rangle)$ is added to $\mathcal{G}$; otherwise, $\overline{L} \in B^-(r)$ and thus an arc $(\langle \overline{L}, \mathcal{T}\rangle, \langle c, \mathcal{F}\rangle)$ is added to $\mathcal{G}$.

3.2 Constraint Learning

Constraint learning means acquiring information that avoids arriving again at a conflict that was already encountered during the search. Our learning schema is based on the concept of the first Unique Implication Point (UIP) [24]. A node n in the Implication Graph is a UIP for a decision level d iff all paths from the literal chosen at the level d to a conflict atom pass through n. Intuitively, a UIP is the most concise reason for the conflict of a certain decision level. We calculate the first UIP only for the decision level of the conflict. By definition the chosen literal is always a UIP, but since several UIPs may exist, we calculate the UIP closest to the conflict, the first UIP. After each conflict at the decision level d, a constraint is learned that contains the first UIP and all atoms of lower levels that are connected to a node between the first UIP and the conflict.

Since the number of learned constraints may become exponential in the size of the program, we adopt the standard technique of expiring learned constraints. Our policy is similar to Minisat's [28]: Each learned constraint has an activity value, measuring how much it is involved in conflicts. If a learned constraint has recently been used for propagation, we do not delete it. If the number of learned constraints is greater than one third of the input program, then we delete half of the learned constraints. Moreover, we also delete all learned constraints with an activity value lower than a threshold value.

4 Heuristics

Clearly, a crucial issue in the Model Generator function in Fig. 1 is the selection of a literal when all inferences have been made and there are still undefined atoms. It is clear that the correctness of the algorithm reported in Fig. 1 does not depend on the strategy in which this selection is made, but making a "good" choice is very important for practical efficiency. However, strategies which perform very well on some domains may perform very bad for other domains, and of course an optimal strategy seems unlikely to be found. For this reason, some heuristic must be adopted; the quality of the adopted heuristic can often only be assessed empirically.

Heuristics can be classified in two main classes, *look-ahead* based and *look-back* based. Look-ahead heuristics estimate the effects of assigning a specific truth value to a given undefined atom, for any truth value and for a set of undefined atoms (which might also be the set of all undefined atoms). Once the effects of all candidate assumptions have been estimated, a look-ahead heuristic selects the most promising undefined atom and truth value according to some function. Look-back heuristics, instead, rely on the information on conflicts derived in the computation so far.

The heuristic implemented in $wasp$ is based on a mixed approach. In fact, a look-back approach is used for selecting an undefined atom and, in some cases, a look-ahead step is performed for choosing the truth value for the selected atom. More specifically, statistics on previously detected conflicts are analyzed and atoms that have caused most conflicts are preferred. Also the "age" of conflicts is taken into account in the selection process, and more recent conflicts are given greater importance. This approach has already been adopted in the context of SAT, for example in the BerkMin solver [31]. In this sense, our heuristic could be seen as an extension of the heuristic implemented in BerkMin to the framework of ASP.

In the remainder of this section, we will provide a few additional details on the strategy adopted by $wasp$ for selecting undefined atoms and truth values to be assumed during the computation of stable models.

A counter $cl(L)$ is associated with each literal L. Initially, all of these counters are set to zero. When a new constraint is learned, counters for all literals occurring in the constraint are increased by one. In this way, $wasp$ keeps track of those literals occurring more frequently in learned constraints. Moreover, counters are also updated during the computation of the First UIP: If a literal L is traversed in the implication graph, the associated counter $cl(L)$ is increased by one. In this way, those literals that mainly caused the derivation of a conflict are identified. Finally, every 100 conflicts, all these counters are divided by 4 (this is an experimentally determined parameter), which gives more importance to recently active literals. Our heuristic will first select an atom and then a truthvalue for this atom. To this end, we will use $cv(a) := cl(a) + cl(\mathrm{not}\ a)$, for each propositional atom a.

Learned constraints are stored in chronological order. The atom selection is first restricted to those undefined atoms that occur in the first (if any) learned constraint r with undefined body. Among those, the atom with the highest $cv(\cdot)$ value is chosen. In case of ties, the atom removing the highest number of supporting rules is selected[2].

[2] An atom removes a supporting rule if it makes the body of r false or the head of r true

If two or more atoms remove the same number of supporting rules, the first processed atom is chosen. In this way, the chances of achieving a conflict increases, and this may help the learning process. If no learned constraints with undefined body exist, the undefined atom with the highest $cv(\cdot)$ value is selected. In case of ties, the first processed atom is selected. If there are no learned constraints, e.g. in the beginning of the solving process, the atom occurring in most rules is picked.

After selecting an atom a according to the strategy described above, $wasp$ chooses a truth value for a. For this purpose, we only distinguish two cases, namely whether a learned constraint r with undefined body exists or not. If a learned constraint r with undefined body exists, additional counters are considered for choosing a truth value for a. In particular, a counter $gcl(L)$ is associated with each literal L for estimating the global contribution of L to all of the conflicts derived during the computation. For each literal L, $gcl(L)$ is initially set to zero and increased whenever $cl(L)$ is increased. The difference to $cl(L)$ is that $gcl(L)$ is never decreased, that is, $gcl(L)$ is unchanged when $cl(L)$ is divided by 4. Thus, in this case $wasp$ assumes the truth of a if $gcl(a) > gcl(\text{not } a)$; otherwise, if $gcl(a) \leq gcl(\text{not } a)$, the falsity of a is assumed. It is important to emphasize that this counter is not used when the atom removing the highest number of supporting rules was chosen. In fact, in this case the literal removing the highest number of supporting rules is picked. In the other case, that is, if all learned constraints have false bodies, a look-ahead step is performed and both a and $\text{not } a$ are propagated (i.e., the function Propagate is invoked). The literal appearing in more rules is propagated before the other one. During these propagations, $wasp$ estimates the impact of the two assumptions on the computation of answer sets. In particular, $wasp$ counts the number of inferred atoms and the number of rules that have been satisfied by the two propagations. The truth of a is then assumed if the impact of the propagation of a is greater than the impact of the propagation of $\text{not } a$, while a is assumed to be false in other case, that is, if the impact of the propagation of $\text{not } a$ is greater than the impact of the propagation of a. If the impact is equal then a is assumed to be false. It is important to note that when a conflict is derived in one of the two propagations, a deterministic inference is determined. That is, if a conflict is derived during the propagation of a, the falsity of a is determined, while the truth of a is determined whenever a conflict is derived during the propagation of $\text{not } a$.

Example 1. We will now provide an example of the way our heuristic works. In the example, we will consider the following rules r_1–r_4 and learned constraints c_1–c_2 (listed in chronological order):

$$r_1: \quad a \texttt{ :- } c. \qquad r_3: \quad a \vee c \texttt{ :- } e. \qquad c_1: \quad \texttt{:- } a,\ b.$$
$$r_2: \quad a \vee b \texttt{ :- } d. \qquad r_4: \quad e \vee b \texttt{ :- } c. \qquad c_2: \quad \texttt{:- } a,\ \text{not } c,\ d.$$

Moreover, let us assume a partial interpretation $I_1 = \{a,\ \text{not } b\}$ and the following counter values: $cl(a) = 2$, $cl(\text{not } a) = 2$, $cl(b) = 1$, $cl(\text{not } b) = 0$, $cl(c) = 1$, $cl(\text{not } c) = 2$, $cl(d) = 3$ and $cl(\text{not } d) = 0$.

Note that constraint c_1 is satisfied because b is false. Thus, the first learned constraint (according to the chronological order) which is not satisfied is c_2. Indeed, two undefined literals occur in the body of c_2, namely $\text{not } c$ and d. We then consider the counters

$cv(c) = cl(c) + cl(\text{not } c)$ and $cv(d) = cl(d) + cl(\text{not } d)$, which are both equal to 3. The heuristics then examines the removal of supporting rules:

Two supporting rules would be removed (r_1 and r_4) by setting c false, and one supporting rule (r_3) would be removed by setting c true, for a total of 3 supporting rules removed. Concerning d, one supporting rule (r_2) would be removed by setting d false, and no rules would be removed by setting d true, for a total of 1 supporting rule removed. Therefore c removes more supporting rules than d, and therefore our heuristic will choose c and it first will be set to false.

5 Experiments

In this section we report the results of an experimental analysis we carried out in order to assess the performance of $wasp$. As a comparison, we also ran the suite of our benchmarks on two state-of-the-art ASP solvers, namely DLV and ClaspD;[3] a discussion on the difference between $wasp$ and these two systems is provided in Section 6.

The machine used for the experiments is a two-processor Intel Xeon "Woodcrest" (quad core) 3GHz machine with 4MB of L2 Cache and 4GB of RAM, running Debian GNU Linux 4.0. As our ASP system focuses on the Model Generation phase, only the time for evaluating ground programs (previously produced by the DLV instantiator from the original non-ground instances) have been considered. In the following, we briefly describe both benchmark problems and data.

5.1 Benchmark Problems and Data

In our experiments, we considered problems from the most recent ASP Competition [17] and other problems which have already been employed for assessing performance of the ASP solver DLV [4]. Our experiments consist of 36 instances in 15 different domains. The instances and encodings are those that were used in the competitions or in the other publicly available suites. In the following we describe the benchmark problems.

Labyrinth. Ravensburger's Labyrinth game deals with guiding an avatar through a dynamically changing labyrinth to certain fields. A solution is represented by pushes of the labyrinth's rows and columns such that the avatar can reach the goal field (which changes its location when pushed) from its starting field (which also changes its location when pushed) by a move along some path after each push.

Knight-tour. Given a chessboard, the problem is to find a tour for a knight piece that starts at any square, travels all squares, and comes back to the origin, following the knight move rules of chess.

Graph coloring. Given an undirected graph and a set of n colors, we are interested in checking whether there is an assignment of colors to nodes such that no adjacent nodes share the same color.

[3] Winners of the disjunctive tracks in the last ASP Competitions [15–17].

Maze-Generation. A maze is an $m \times n$ grid, in which each cell is empty or a wall and two distinct cells on the edges are indicated as entrance and exit, satisfying the following conditions: (1) each cell on the edge of the grid is a wall, except entrance and exit that are empty; (2) there is no 2×2 square of empty cells or walls; (3) if two walls are on a diagonal of a 2×2 square, then not both of their common neighbors are empty; (4) no wall is completely surrounded by empty cells; (5) there is a path from the entrance to every empty cell. The problem has been proved to be NP-complete in [32].

Strategic Companies. Strategic companies is a well-known NP^{NP}-complete problem that has often been used for system comparisons, also in the previous ASP Competitions. In the Strategic Companies problem, a collection $C = c_1, \ldots, c_m$ of companies is given, for some $m \geq 1$. Each company produces some goods in a set G, and each company c_i in C is possibly controlled by a set of owner companies O_i (where O_i is a subset of C, for each $i = 1, \ldots, m$). In this context, a set C' of companies (i.e., a subset of C) is a *strategic set* if it is minimal among all the sets satisfying the following conditions: (i) Companies in C' produce all goods in G; (ii) if O_i is a subset of C', the associated company c_i must belong to C' (for each $i = 1, \ldots, m$). We considered a random instance having 7500 companies and 22500 products.

2-QBF. The problem consists of checking the validity of a quantified boolean formula $\Phi = \exists X \forall Y \phi$, where X and Y are disjoint sets of propositional variables and $\phi = C_1 \vee \ldots \vee C_k$ is a DNF on variables X and Y. In our benchmark, we used the transformation from 2-QBF to ASP presented in [4], which is based on a reduction presented in [33]. The instance considered has 1000 universal variables, 20 existential variables, 10000 clauses, and is a 5-DNF.

Prime Implicants. In Boolean logic, an implicant is a "covering" (sum term or product term) of one or more minterms (a product term in which each of the n variables appears once) in a sum of products, or, maxterms (a sum term in which each of the n variables appears once) in a product of sums, of a boolean function. Formally, a product term P in a sum of products is an implicant of the Boolean function F if P implies F. A prime implicant of a function is an implicant that cannot be covered by a more general (more reduced - meaning with fewer literals) implicant. The instance we considered consists of 180 variables and 774 clauses.

3-Colorability. This well-known problem asks for an assignment of three colors to the nodes of a graph, in such a way that adjacent nodes always have different colors. One simplex graph was generated with the Stanford GraphBase library [34], by using the function $simplex(600, 600, -2, 0, 0, 0, 0)$. Another ladder graph was generated having 11998 edges, and 8000 nodes.

Hamiltonian Cycle. A classical NP-complete problem in graph theory, which can be expressed as follows: given a directed graph $G = (V, E)$ and a node $a \in V$ of this graph, does there exist a path in G starting at a and passing through each node in V exactly once? One random graph was generated with the Stanford GraphBase library [34], by using the function $random\_graph(85, 700, 0, 0, 0, 0, 0, 1, 1, 33)$, having 700 edges and 85 nodes; the other instances has been generating using the function $random\_graph(80, 456, 0, 0, 0, 0, 0, 1, 1, 33)$, having 456 edges and 80 nodes.

Blocks World. Blocks world is one of the most famous planning domains in artificial intelligence. We have a set of cubes (blocks) sitting on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved. The four instances considered are by Esra Erdem and taken from the ccalc homepage (http://www.cs.utexas.edu/users/tag/cc/).

3SAT. The satisfiability problem (SAT) is a decision problem, whose instance is a propositional formula. The question is: given the formula, is there some assignment of $\mathcal{T}$ and $\mathcal{F}$ values to the variables that will make the entire expression true? SAT is the best-known NP-complete problem. 3-satisfiability is a special case of SAT, where each formula is a CNF in which each clause contains exactly three literals. We considered two random instances with 280 variables and 1204 clauses.

Towers of Hanoi. The Towers of Hanoi (ToH) problem has three pegs and n disks. Initially, all n disks are on the left-most peg. The goal is to move all n disks to the right-most peg with the help of the middle peg. The rules are: (1) move one disk at a time; (2) only the top disk on a peg can be moved; (3) a larger disk cannot be placed on top of a smaller one. The instance we considered has 6 disks, and we check whether a plan of length 64 exists.

Ramsey Numbers. The Ramsey number $ramsey(k, m)$ is the least integer n such that, no matter how the edges of the complete undirected graph (clique) with n nodes are colored using two colors, say red and blue, there is a red clique with k nodes (a red k-clique) or a blue clique with m nodes (a blue m-clique).The encoding of this problem consists of one rule and two constraints. For the experiments, the problem was considered of deciding whether, for $k = 3$, $m = 7$, $n = 21$, and for $k = 4$, $m = 6$, $n = 26$, n is the Ramsey number $ramsey(k, m)$.

n-Queens. The 8-queens puzzle is the problem of putting eight chess queens on an 8x8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The n-queens puzzle is the more general problem of placing n queens on an nxn chessboard ($n \geq 4$). The instance considered is for $n = 23$.

Timetabling. The problem is determining a timetable for some university lectures that have to be given in one week to some groups of students. The timetable must respect a number of given constraints concerning availability of rooms, teachers, and other issues related to the overall organization of the lectures.

5.2 Experimental Results

The results of our experiment are summarized in Table 1, reporting, for each considered instance the execution times in seconds elapsed by each considered system. For each instance of the benchmark problems, we allowed a maximum of 600 seconds of execution time. Timeouts are indicated by means of the word TIME in Table 1. In the last rows we report, for each system, the total number of solved instances, the average execution time for solving all the 36 considered instances (timeouts are counted 600s each), and the number of instances in which each solver resulted to be the fastest.

Table 1. Benchmark Results on ASP competition suite

Problem	$wasp$	DLV	*ClaspD*
LABYRINTH-1	0,39	0,02	0,03
LABYRINTH-2	299,74	3,17	65,84
LABYRINTH-3	415,14	56,19	113,04
LABYRINTH-4	TIME	25,76	561,93
LABYRINTH-5	14,47	29,15	490,04
KNIGHT-TOUR-1	0,07	0,21	0,15
KNIGHT-TOUR-2	0,14	1,64	0,34
KNIGHT-TOUR-3	0,65	14,45	2,84
KNIGHT-TOUR-4	0,67	56,31	10,56
KNIGHT-TOUR-5	7,44	TIME	179,48
GRAPH-COLOURING-1	153,67	TIME	3,05
GRAPH-COLOURING-2	TIME	TIME	TIME
MAZE-GENERATION-1	0,28	0,93	0,79
MAZE-GENERATION-2	46,84	104,47	1,76
MAZE-GENERATION-3	47,37	261,57	3,94
MAZE-GENERATION-4	94,17	TIME	9,64
MAZE-GENERATION-5	123,40	TIME	23,49
STRATCOMP	179,06	2,33	5,71
2QBF	0,11	3,31	0,92

Problem	$wasp$	DLV	*ClaspD*
PRIMEIMPL	3,24	1,33	0,21
3COL-SIMPLEX	23,02	33,58	TIME
3COL-LADDER	2,29	91,24	34,08
HAMCYCLE-RANDOM	5,29	1,50	2,52
HAMCYCLE-FREE	106,89	31,37	0,47
BLOCKS-WORLD-1	224,09	6,48	1,92
BLOCKS-WORLD-2	340,84	11,84	1,75
BLOCKS-WORLD-3	0,76	8,87	1,67
BLOCKS-WORLD-4	129,28	11,05	0,83
3SAT-1	78,31	9,59	65,84
3SAT-2	31,07	5,43	0,06
TOWERS-OF-HANOI	3,81	8,46	437,55
RAMSEY-1	3,03	9,84	24,01
RAMSEY-2	4,87	15,74	40,28
23-QUEENS	0,10	41,10	0,54
SCHOOL-TIMETABLING	7,45	61,09	224,93
TOTAL SOLVED	**34**	**31**	**34**
WEIGHTED AVERAGE	**98,08**	**108,87**	**98,17**
WINS	**15**	**9**	**15**

Overall, the results of the preliminary experimental analysis are encouraging: the performance of $wasp$ is comparable to ClaspD (same number of wins and cumulative average time), and it is often faster than DLV (only 9 wins vs 15 of $wasp$ and ClaspD). In more detail, for the Labyrinth problem $wasp$ was able to solve four instances out of five in the allowed time, while the other systems solved all five instances; the system is always outperformed by the competitors, except for one instance in which it is the best performer. Regarding the Knight Tour problem, $wasp$ always outperforms the competitor systems, solving the hardest instance (on which DLV timed out) in only $7,44$ seconds compared to $179,48$ seconds for ClaspD. Concerning the Graph Coloring problem, $wasp$ was slower than ClaspD, but solved one instance more than DLV. Also for the Maze Generation benchmarks, $wasp$ was slightly slower than ClaspD, but always outperformed DLV. Considering the other benchmarks, $wasp$ outperformed the other two ASP solvers on 2QBF, Ramsey Numbers, N-Queens, School Timetabling, 3Colorability, and Towers of Hanoi. In the remaining benchmarks, the system remains competitive, with the single exception of Strategic Companies. For this, we hypothesize that a reason might be that $wasp$ does not implement yet a model-checking-driven backjumping technique, which proved to be very effective on this particular benchmark [35].

6 Related Work and Conclusion

In this paper we provided a preliminary report on a new ASP solver for propositional programs called $wasp$. The new system is inspired by several techniques that were originally introduced for SAT solving, like the Davis-Putnam-Logemann-Loveland (DPLL) backtracking search algorithm [18], *clause learning* [19, 20], *backjumping* [21, 22], *restarts* [23], and *conflict-driven heuristics* [24] in the style of Berkmin [25]. Actually, some of the techniques adopted in $wasp$, including *backjumping* and *look back heuristics* were first introduced for Constraint Satisfaction [21, 22, 36] and successively successfully applied in SAT [37, 38, 25, 24] and QBF solving [39–42]. Some of these tech-

niques were already adapted in modern non-disjunctive ASP solvers like Smodels$_{cc}$ [43, 44], Clasp [8], and solvers supporting disjunction like CModels3 [10], GnT [45], and DLV [46, 47].

Concerning other ASP solvers, we differ from non-native solvers like Cmodels3 [10], in the sense that we do not rely on a rewriting into a propositional formula and an external SAT solver, but use native ASP techniques. Among native solvers, similarities with DLV [4] can be found in the propagation rules, in the computation of the greatest unfounded set, and in the model checking technique. However, we clearly differ from DLV as it does not implement many of the look-back techniques borrowed from CP and SAT. The prototypical version of DLV presented in [46] and extended in [47], implements backjumping and some forms of look back heuristics, but it does not include clause learning, restarts, and does not use an implication graph for determining the reasons of the conflicts. Similar considerations hold for GnT [45], which, as DLV, implements a systematic backtracking without learning and look-ahead heuristics.

Comparing our system with ClaspD (a disjunction-supporting version built upon Clasp) more similarities can be found, as it includes similar techniques, e.g. backjumping, clause learning, restarts, and look-back heuristics. There are nonetheless several differences with $wasp$. First of all, $wasp$ performs the unfounded set checking by means of the well-founded operator, while ClaspD relies on the computation of loop formulas. Moreover, ClaspD implements an alternative version of the implication graph that is more similar to SAT solvers, since it relies on unit propagation of nogoods (minimality is handled via loop formula learning). Furthermore, ClaspD, as $wasp$, adopts a branching heuristics based on Berkmin [25]; however, $wasp$ extends the original Berkmin heuristics by exploiting a lookahaed technique in place of the *"two"* function calculating the number of binary clauses in the neighborhood of literal L, together with an additional criterion based on minimality of answer sets. In particular, to deal with the case of two atoms with the same heuristic value, $wasp$ chooses the atom that introduces the maximum number of unsatisfied supporting rules.

It is worth pointing out that the implementation of $wasp$ is still in a preliminary phase, yet the results obtained up to now are encouraging. Our system is able to compete with the state-of-the-art solvers, and even outperform them in some of the considered benchmarks.

Concerning future work, we plan to extend the prototypical system by introducing new language constructs such as aggregates [48, 49] and weak constraints [50], which are currently missing from $wasp$. Moreover, the current implementation can be improved in several respects: parameter tuning of the heuristics, fine tuning of the source code, a model-checking-driven backjumping [35] as well as support for multi-threading are also planned.

References

1. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC **9** (1991) 365–385
2. Lifschitz, V.: Answer Set Planning. In: ICLP'99, Las Cruces, New Mexico, USA, The MIT Press (1999) 23–37
3. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS **22** (1997) 364–418

4. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL **7** (2006) 499–562
5. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. AI **138** (2002) 181–234
6. Lin, F., Zhao, Y.: ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In: AAAI-2002, Edmonton, Alberta, Canada, AAAI Press / MIT Press (2002)
7. Babovich, Y., Maratea, M.: Cmodels-2: Sat-based answer sets solver enhanced to non-tight programs. `http://www.cs.utexas.edu/users/tag/cmodels.html` (2003)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007,(2007) 386–392
9. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL **7** (2006) 1–37
10. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: LPNMR'05. LNCS 3662, (2005) 447–451
11. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In: Proc. of KR 2008, Sydney, Australia, AAAI Press (2008) 422–432
12. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with Answer Set Programming in the Gioia-Tauro Seaport. TPLP. *CUP* (2011) To appear.
13. Manna, M., Ruffolo, M., Oro, E., Alviano, M., Leone, N.: The HiLeX System for Semantic Information Extraction. Transactions on Large-Scale Data and Knowledge-Centered Systems. *Berlin/Heidelberg* (2011) To appear.
14. Ricca, F., Alviano, M., Dimasi, A., Grasso, G., Ielpa, S.M., Iiritano, S., Manna, M., Leone, N.: A Logic-Based System for e-Tourism. FI. *IOS Press* **105** (2010) 35–55
15. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: LPNMR'07. LNCS 4483, (2007) 3–17
16. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The second answer set programming competition. In: Proc. of LPNMR '09, Berlin, Heidelberg, (2009) 637–654
17. Calimeri, F., Ianni, G., Ricca, F., Alviano, M., Bria, A., Catalano, G., Cozza, S., Faber, W., Febbraro, O., Leone, N., Manna, M., Martello, A., Panetta, C., Perri, S., Reale, K., Santoro, M.C., Sirianni, M., Terracina, G., Veltri, P.: The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In: Proc. of LPNMR11., LNCS (2003) 388–403
18. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. Communications of the ACM **5** (1962) 394–397
19. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In: ICCAD 2001. (2001) 279–285
20. Pipatsrisawat, K., Darwiche, A.: On Modern Clause-Learning Satisfiability Solvers. JAIR **44** (2010) 277–301
21. Gaschnig, J.: Performance measurement and analysis of certain search algorithms. PhD thesis, CMU (1979) Tech. Report CMU-CS-79-124.
22. Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence **9** (1993) 268–299
23. Gomes, C.P., Selman, B., Kautz, H.A.: Boosting Combinatorial Search Through Randomization. In: Proceedings of AAAI/IAAI 1998, AAAI Press (1998) 431–437
24. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: DAC 2001 (2001) 530–535
25. Goldberg, E., Novikov, Y.: BerkMin: A Fast and Robust Sat-Solver. In: Design, Automation and Test in Europe Conference and Exposition (DATE 2002), Paris, France, IEEE Computer Society (2002) 142–149

26. Faber, W., Leone, N., Pfeifer, G.: Pushing Goal Derivation in DLP Computations. In: LPNMR'99. LNCS 1730, (1999) 177–191
27. Koch, C., Leone, N., Pfeifer, G.: Enhancing Disjunctive Logic Programming Systems by SAT Checkers. AI **15** (2003) 177–212
28. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003., LNCS (2003) 502–518
29. Maratea, M., Ricca, F., Veltri, P.: DLV^C: Enhanced Model Checking in DLV. In: Proceedings of Logics in Artificial Intelligence, JELIA 2010. (2010) 365–368
30. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of las vegas algorithms. Inf. Process. Lett. **47** (1993) 173–180
31. Goldberg, E., Novikov, Y.: Berkmin: A fast and robust sat-solver. Discrete Appl. Math. **155** (2007) 1549–1561
32. Alviano, M.: The Maze Generation Problem is NP-complete. In: Proc. of ICTCS '09. (2009)
33. Eiter, T., Gottlob, G.: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. AMAI **15** (1995) 289–323
34. Knuth, D.E.: The Stanford GraphBase : A Platform for Combinatorial Computing. ACM Press, New York (1994)
35. Pfeifer, G.: Improving the Model Generation/Checking Interplay to Enhance the Evaluation of Disjunctive Programs. In: LPNMR-7. LNCS 2923, (2004) 220–233
36. Dechter, R., Frost, D.: Backjump-based backtracking for constraint satisfaction problems. AI **136** (2002) 147–188
37. Bayardo, R., Schrag, R.: Using CSP Look-back Techniques to Solve Real-world SAT Instances. In: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-97). (1997) 203–208
38. Silva, J.P.M., Sakallah, K.A.: GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Transaction on Computers **48** (1999) 506–521
39. Zhang, L., Malik, S.: Conflict Driven Learning in a Quantified Boolean Satisfiability Solver. In: Proc. of ICCAD 2002. (2002) 442–449
40. Zhang, L., Malik, S.: Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In: CP 2002. NY, USA, (2002) 200–215
41. Giunchiglia, E., Narizzano, M., Tacchella, A.: Backjumping for Quantified Boolean Logic Satisfiability. AI **145** (2003) 99–120
42. Letz, R.: Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In: TABLEAUX 2002. Denmark, (2002) 160–175
43. Ward, J., Schlipf, J.S.: Answer Set Programming with Clause Learning. In: LPNMR-7. LNCS 2923, (2004) 302–313
44. Ward, J.: Answer Set Programming with Clause Learning. PhD thesis, Ohio State University, Cincinnati, Ohio, USA (2004)
45. Janhunen, T., Niemelä, I.: Gnt - a solver for disjunctive logic programs. In: LPNMR-7. LNCS 2923, Fort Lauderdale, Florida, USA, (2004) 331–335
46. Ricca, F., Faber, W., Leone, N.: A Backjumping Technique for Disjunctive Logic Programming. AI Communications **19** (2006) 155–172
47. Maratea, M., Ricca, F., Faber, W., Leone, N.: Look-back techniques and heuristics in dlv: Implementation, evaluation and comparison to qbf solvers. Journal of Algorithms in Cognition, Informatics and Logics **63** (2008) 70–89
48. Pelov, N., Denecker, M., Bruynooghe, M.: Well-founded and Stable Semantics of Logic Programs with Aggregates. TPLP **7** (2007) 301–353
49. Faber, W., Leone, N., Pfeifer, G.: Semantics and complexity of recursive aggregates in answer set programming. AI **175** (2011) 278–298 Special Issue: John McCarthy's Legacy.
50. Buccafurri, F., Leone, N., Rullo, P.: Enhancing Disjunctive Datalog by Constraints. IEEE TKDE **12** (2000) 845–860

Testing ASP programs in *ASPIDE*

Onofrio Febbraro[1], Kristian Reale[2], and Francesco Ricca[2]

[1] DLVSystem s.r.l. - P.zza Vermicelli, Polo Tecnologico, 87036 Rende, Italy
febbraro@dlvsystem.com
[2]Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
{reale,ricca}@mat.unical.it

Abstract. Answer Set Programming (ASP) is a declarative logic programming formalism, which nowadays counts several advanced real-world applications, and has stimulated some interest also in industry. Although some environments for ASP-program development have been proposed in the last few years, the crucial task of *testing* ASP programs received less attention, and is an Achilles' heel of the available programming environments.
In this paper we present a new language for specifying and running *unit tests* on ASP programs. The testing language has been implemented in *ASPIDE*, a comprehensive IDE for ASP, which supports the entire life-cycle of ASP development with a collection of user-friendly graphical tools for program composition, *testing*, debugging, profiling, solver execution configuration, and output-handling.

1 Introduction

Answer Set Programming (ASP) [1] is an expressive [2] logic programming paradigm proposed in the area of non-monotonic reasoning. ASP allows one to declaratively specify a complex computational problem by a logic program whose answer sets correspond to solutions and then use a solver to find such a solution [3]. The high expressive power of ASP has been profitably exploited for developing advanced applications belonging to several fields, from Artificial Intelligence [4–10] to Information Integration [11], and Knowledge Management [12–14]. Interestingly, these applications of ASP recently have stimulated some interest also in industry [15].

On the one hand, the effective application of ASP in real-world scenarios was made possible by the availability of efficient ASP systems [4, 16–26]. On the other hand, the adoption of ASP can be further boosted by offering effective programming tools capable of supporting the programmers in managing large and complex projects [27].

In the last few years, a number of tools for developing ASP programs have been proposed, including editors and debuggers [28–38]. Among them, *ASPIDE* [38] –which stands for Answer Set Programming Integrated Development Environment– is one of the most complete development tools[1] and it integrates a cutting-edge editing tool (featuring dynamic syntax highlighting, on-line syntax correction, autocompletion, code-templates, quick-fixes, refactoring, etc.) with a collection of user-friendly graphical

[1] For an exaustive feature-wise comparison with existing environments for developing logic programs we refer the reader to [38].

tools for program composition, debugging, profiling, DBMS access, solver execution configuration and output-handling.

Although so many tools for developing ASP programs have been proposed up to now, the crucial task of *testing* ASP programs received less attention [39], and is an Achilles' heel of the available programming environments. Indeed, the majority of available graphic programming environments for ASP does not provide the user with a testing tool (see [38]), and also the one present in the first versions of *ASPIDE* is far from being effective.

In this paper we present a pragmatic solution for testing ASP programs. In particular, we present a new language for specifying and running *unit tests* on ASP programs. The testing language presented in this paper is inspired to the JUnit framework [40]: the developer can specify the rules composing one or several units, specify one or more inputs and assert a number of conditions on the expected outputs. The obtained test case specification can be run by exploiting an ASP solver, and the assertions are automatically verified by analyzing the output of the chosen ASP solver. Note that test case specification is applicable independently of the used ASP solver. The testing language was implemented in *ASPIDE*, which also provides the user with some graphic tools that make the development of test cases simpler. The testing tool described in this work extends significantly the one formerly available in *ASPIDE*, and enriches its collection of user-friendly graphical tools for program composition, debugging, profiling, database management, solver execution configuration, and output-handling.

As far as related work is concerned, the task of testing ASP programs was approached for the first time, to the best of our knowledge, in [39] where the notion of structural testing for ground normal ASP programs is defined and a method for automatically generating tests is introduced. The results presented in [39] are, somehow, orthogonal to the contribution of this paper. Indeed, no language/implementation is proposed in [39] for specifying/automatically-running the produced test cases; whereas, the language presented in this paper can be used for encoding the output of a test case generator based on the methods proposed in [39]. Finally, it is worth noting that, testing approaches developed for other logic languages, like prolog [41–43], cannot be straightforwardly ported to ASP because of the differences between the languages.

The remainder of this paper is organized as follows: in Section 2 we overview *ASPIDE*; in section 3 we introduce a language for specifying unit tests for ASP programs; in Section 4 we describe the user interface components of *ASPIDE* conceived for creating and running tests; finally, in Section 5 we draw the conclusion.

2 *ASPIDE*: Integrated Development Environment for ASP

ASPIDE is an Integrated Development Environment (IDE) for ASP, which features a rich *editing tool* with a collection of user-friendly *graphical tools* for ASP program development. In this section we first summarize the main features of the system and then we overview the main components of the *ASPIDE* user interface. For a more detailed description of *ASPIDE*, as well as for a complete comparison with competing tools, we refer the reader to [38] and to the online manual published in the system web site `http://www.mat.unical.it/ricca/aspide`.

System Features. *ASPIDE* is inspired to Eclipse, one of the most diffused programming environments. The main features of *ASPIDE* are the following:

- *Workspace management*. The system allows one to organize ASP programs in projects, which are collected in a special directory (called workspace).
- *Advanced text editor*. The editing of ASP files is simplified by an advanced text editor. Currently, the system is able to load and store ASP programs in the syntax of the ASP system DLV [16], and supports the `ASPCore` language profile employed in the ASP System Competition 2011 [44]. *ASPIDE* can also manage *TYP files* specifying a mapping between program predicates and database tables in the DLV^{DB} syntax [45]. Besides the core functionality that basic text editors offer (like code line numbering, find/replace, undo/redo, copy/paste, etc.), *ASPIDE* offers others advanced functionalities, like: *Automatic completion*, *Dynamic code templates*, *Quick fix*, and *Refactoring*. Indeed, the system is able to complete (on request) predicate names, as well as variable names. Predicate names are both learned while writing, and extracted from the files belonging to the same project; variables are suggested by taking into account the rule we are currently writing. When several possible alternatives for completion are available the system shows a pop-up dialog. Moreover, the writing of repeated programming patterns (like transitive closure or disjunctive rules for guessing the search space) is assisted by advanced auto-completion with code templates, which can generate several rules at once according to a known pattern. Note that code templates can be also user defined by writing DLT [46] files. The refactoring tool allows one to modify in a guided way, among others, predicate names and variables (e.g., variable renaming in a rule is done by considering bindings of variables, so that variables/predicates/strings occurring in other expressions remain unchanged). Reported errors or warnings can be automatically fixed by selecting (on request) one of the system's suggested quick fixes, which automatically change the affected part of code.
- *Outline navigation*. *ASPIDE* creates an outline view which graphically represents program elements. Each item in the outline can be used to quickly access the corresponding line of code (a very useful feature when dealing with long files), and also provides a graphical support for building rules in the visual editor (see below).
- *Dynamic code checking and errors highlighting*. Syntax errors and relevant conditions (like safety) are checked *while typing programs*: portions of code containing errors or warnings are immediately highlighted. Note that the checker considers the entire project, and warns the user by indicating e.g., that atoms with the same predicate name have different arity in several files. This condition is usually revealed only when programs divided in multiple files are run together.
- *Dependency graph*. The system is able to display several variants of the dependency graph associated to a program (e.g., depending on whether both positive and negative dependencies are considered).
- *Debugger and Profiler*. Semantic errors detection as well as code optimization can be done by exploiting graphic tools. In particular, we developed a graphical user interface for embedding in *ASPIDE* the debugging tool *spock* [30] (we have also adapted spock for dealing with the syntax of the DLV system). Regarding the profiler, we have fully embedded the graphical interface presented in [47].

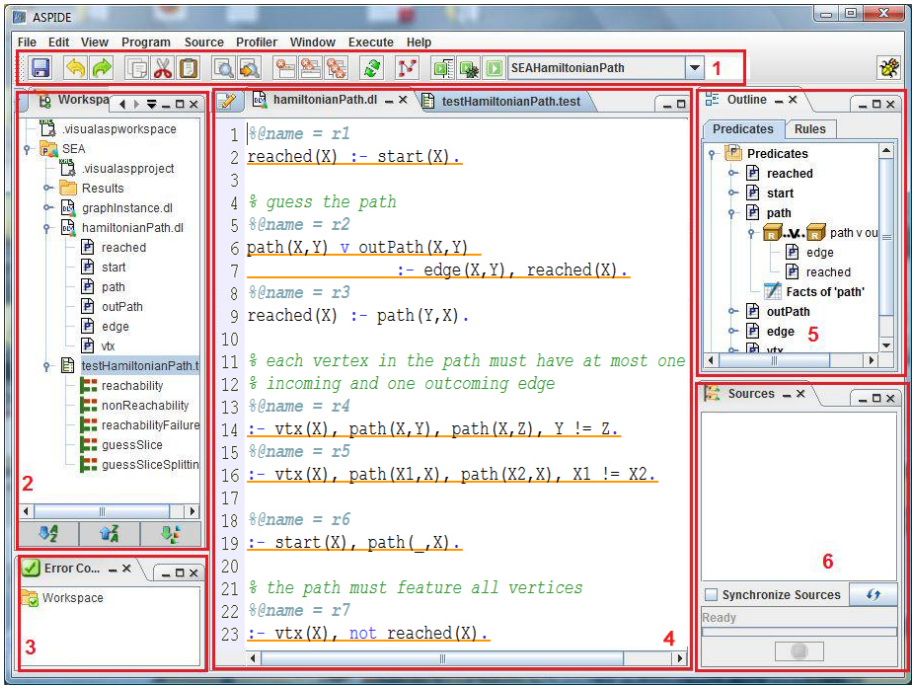


Fig. 1. The *ASPIDE* graphical user interface.

- *Unit Testing*. The user can define unit tests and verify the behavior of programs units. The language for specifying unit tests, as well as the graphical tools of *ASPIDE* assisting the development of tests, are described in detail in the following sections.
- *Configuration of the execution*. This feature allows one to configure and manage input programs and execution options (called *run configurations*).
- *Presentation of results*. The outputs of the program (either answer sets, or query results) are visualized in a tabular representation or in a text-based console. The result of the execution can be also saved in text files for subsequent analysis.
- *Visual Editor*. The users can *draw* logic programs by exploiting a full graphical environment that offers a QBE-like tool for building logic rules [48]. The user can switch, every time he needs, from the text editor to the visual one (and vice-versa) thanks to a reverse-rengineering mechanism from text to graphical format.
- *Interaction with databases.* Interaction with external databases is useful in several applications (e.g., [11, 15, 8]). *ASPIDE* provides a fully graphical import/export tool that automatically generates mappings by following the DLV^{DB} Typ files specifications [45]. Text editing of Typ mappings is also assisted by syntax coloring and auto-completion. Database oriented applications can be run by setting DLV^{DB} as solver in a run configuration.

Interface Overview The system interface of *ASPIDE* is depicted in Figure 1. The most common operations can be quickly executed through a toolbar present in the upper part

of the *ASPIDE* interface (zone 1). From left to right there are buttons allowing one to: save files, undo/redo, copy & paste, find & replace, switch between visual to text editor, run the solver/profiler/debugger. The main editing area (zone 4) is organized in a multi-tabbed panel possibly collecting several open files. On the left there is the explorer panel (zone 2) which allows one to browse the workspace; and the error console (zone 3). The explorer panel lists projects and files included in the workspace, while the error console organizes errors and warnings according to the project and files where they are localized. On the right, there are the outline panel (zone 5) and the sources panel (zone 6). The first shows an outline of the currently edited file, while the latter reports a list of the database sources connected with the current project. Note that the layout of the system can be customized by the user, indeed panels can be moved and rearranged as the user likes.

ASPIDE is written in Java and runs on the most diffused operating systems (Microsoft Windows, Linux, and Mac OS) and can connect to any database supporting Java DataBase Connectivity (JDBC).

3 A language for testing ASP programs

Software testing [49] is an activity aimed at evaluating the behavior of a program by verifying whether it produces the required output for a particular input. The goal of testing is not to provide a mean for establishing whether the program is totally correct; conversely testing is a pragmatic and cheap way of finding errors by executing some test. A test case is the specification of some input I and corresponding expected outputs O. A test case fails when the outputs produced by running the program do not correspond to O, it passes otherwise.

One of the most diffused white-box[2] testing techniques is *unit testing*. The idea of unit testing is to asses an entire software by testing its subparts called *units* (and corresponding to small testable parts of a program). In a software implemented by using imperative object-oriented languages, unit testing corresponds to assessing separately portions of the code like class methods. The same idea can be applied to ASP, once the notion of unit is given. We intend as unit of an ASP program P any subset of the rules of P corresponding to a splitting set [50] (actually the system exploits a generalization of the splitting theorem by Lifschitz and Turner [50] to the non-ground case [51]). In this way, the behavior of units can be verified (by avoiding unwanted behavioral changes due to cycles) both when they run isolated from the original program as well as when they are left immersed in (part of) the original program.

In the following, we present a pragmatic solution for testing ASP programs, which is a new language, inspired to the JUnit framework [40], for specifying and running *unit tests*. The developer, given an ASP program, can select the rules composing an unit, specify one or more inputs, and assert a number of conditions on the expected output. The obtained test case specification can be run, and the assertions automatically

[2] A test conceived for verifying some functionality of an application without knowing the code internals is said to be a black-box test. A test conceived for verifying the behavior of a specific part of a program is called white-box test. White box testing is an activity usually carried out by developers and is a key component of agile software development [49].

verified by calling an ASP solver and checking its output. In particular, we allow three test execution modes:

- *Execution of selected rules*. The selected rules will be executed separated from the original program on the specified inputs.
- *Execution of split program*. The program corresponding to the splitting set containing the atoms of the selected rules is run and tested. In this way, the "interface" between two splitting sets can be tested (e.g. one can assert some expected properties on the candidates produced by the guessing part of a program by excluding the effect of some constraints in the checking part).
- *Execution in the whole program*. The original program is run and specific assertions regarding predicates contained in the unit are checked. This corresponds to filter test results on the atoms contained in the selected rules.

Testing Language. A test file can be written according to the following grammar:[3]

```
1 : invocation("invocationName" [ ,"solverPath", "options" ]?);
2 : [ [ input("program"); ] | [ inputFile("file"); ] ]*
3 : [
4 : testCaseName([ SELECTED_RULES | SPLIT_PROGRAM | PROGRAM ]?)
5 : {
6 : [newOptions("options");]?
7 : [ [ input("program"); ] | [ inputFile("file"); ] ]*
8 : [ [ excludeInput("program"); ]
9 : | [ excludeInputFile("file"); ] ]*
10 : [
11 : [ filter | pfilter | nfilter ]
12 : [ [ (predicateName [ ,predicateName ]* ) ]
13 : | [SELECTED_RULES] ] ;
14 : ]?
15 : [ selectRule(ruleName); ]*
16 : [ assertName( [ intnumber, ]? "program" ); ]*
17 : }
18 : ]*
19 : [ assertName( [ intnumber, ]? "program" ); ]*
```

A test file might contain a single test or a test suite (a set of tests) including several test cases. Each test case includes one or more assertions on the execution results.

The *invocation* statement (line 1) sets the global invocation settings, that are applied to all tests specified in the same file (name, solver, and execution options). In the implementation, the invocation name might correspond to an *ASPIDE* run configuration, and the solver path and options are not mandatory.

The user can specify one or more global inputs by writing some *input* and *inputFile* statements (line 2). The first kind of statement allows for writing the input of the test in the form of ASP rules or simply facts; the second statement indicates a file that contains some input in ASP format.

[3] Non-terminals are in bold face; token specifications are omitted for simplicity.

A test case declaration (line 4) is composed by a name and an optional parameter that allows one to choose if the execution will be done on the entire program, on a subset of rules, or considering program corresponding to the splitting set containing the selected rules.

The user can specify specific solver options (line 6), as well as specific inputs (line 7) which are valid in a given test case. Moreover, global inputs of the test suite can be excluded by exploiting *excludeInput* and *excludeInputFile* statements (lines 8 and 9).

The optional statements *filter*, *pfilter* and *nfilter* (lines 11, 12 and 13) are used to filter out output predicates from the test results predicates, specified as parameter, on the execution result when assertions will be executed.[4]

The statement *selectRule* (line 15) allows one for selecting rules among the ones composing the global input program. A rule r to be selected must be identified by a name, which is expected to be specified in the input program in a comment appearing in the row immediately preceding r (see Figure 1). *ASPIDE* adds automatically the comments specifying rule names. If a set of selected rules does not belong to the same splitting set, the system has to print a warning indicating the problem.

The expected output of a test case is expressed in term of assertions statements (lines 16/19). The possible assertions are:

- *assertTrue({"atomList."})/assertCautiouslyTrue({"atomList."})*. Asserts that all atoms of the atom list must be true in any answer sets;
- *assertBravelyTrue({"atomList."})*. Asserts that all atoms of the atom list must be true in at least one answer set;
- *assertTrueIn(number, {"atomList."})*. Asserts that all atoms of the atom list must be true in exactly *number* answer sets;
- *assertTrueInAtLeast(number, {"atomList."})*. Asserts that all atoms of the atom list must be true in at least *number* answer sets;
- *assertTrueInAtMost(number, {"atomList."})*. Asserts that all atoms of the atom list must be true in at most *number* answer sets;

together with the corresponding negative assertions: *assertFalse*, *assertCautiouslyFalse*, *assertBravelyFalse*, *assertFalseIn*, *assertFalseInAtLeast*, *assertFalseInAtMost*. Assertions can be global (line 19) or local to a single test (line 16).

In the following we report an example of test case file.

Test case example. The Hamiltonian Path problem is a classical *NP-complete* problem in graph theory. *Given a finite directed graph $G = (V, A)$ and a node $a \in V$ of this graph, does there exist a path in G starting at a and passing through each node in V exactly once?* Suppose that the graph G is specified by using facts over predicates vtx (unary) and $edge$ (binary), and the starting node a is specified by the predicate $start$ (unary). The program in Figure 1 solves the problem.

The disjunctive rule (r_2) guesses a subset S of the arcs to be in the path, while the rest of the program checks whether S constitutes a Hamiltonian Path. Here, an auxiliary

[4] *pfilter* selects only positive literals and excludes the strong negated ones, while *nfilter* has opposite behavior.

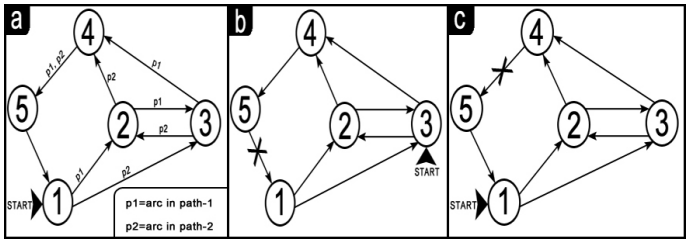


Fig. 2. Input graphs.

predicate *reached* is defined, which specifies the set of nodes which are reached from the starting node. In the checking part, the two constraints r_4 and r_5 ensure that the set of arcs S selected by *path* meets the following requirements, which any Hamiltonian Path must satisfy: (i) a vertex must have at most one incoming edge, and (ii) a vertex must have at most one outgoing edge. The constraints r_6 and r_7 enforces that all nodes in the graph are reached from the starting node in the subgraph induced by S and the start node must be the first node of the path.

In order to test this encoding we define a test suite file. Suppose that the encoding is stored in a file named *hamiltonianPath.dl*. Suppose also that the graph instance of Figure 2a is stored in a file named *graphInstance.dl*, and is composed by the following facts: *vtx(1). vtx(2). vtx(3). vtx(4). vtx(5). edge(1, 2). edge(2, 3). edge(3, 4). edge(4, 5). edge(3, 2). edge(1, 3). edge(2, 4). edge(5, 1).*.

The following is a simple test suite specification for the above-reported ASP program:

```
invocation("SEAHamiltonianPath", "/usr/bin/dlv", "");
inputFile("hamiltonianPath.dl");
reachability()
{
inputFile("graphInstance.dl");
input("start(1).");
assertTrue("reached(1).reached(2).reached(3).reached(4).reached(5).");
}
guessSlice(SELECTED_RULES)
{
inputFile("graphInstance.dl");
input("reached(1).");
selectRule("r2");
selectRule("r3");
assertBravelyFalse("path(1, 2).");
assertBravelyTrue("path(1, 2).");
}
guessSliceNonReachability(SPLIT_PROGRAM)
{
inputFile("graphInstance.dl");
input("start(3).");
```

```
excludeInput( "edge(5, 1)." );
selectRule("r2");
selectRule("r3");
assertCautiouslyFalse( "reached(1)." );
}
assertFalse("path(1, 2).");
```

Here, we first setup the invocation parameters by indicating DLV as solver, then we specify the file to be tested *hamiltonianPath.dl* by exploiting a global input statement; then, we add the test case *reachability*, in which we verify that if node 1 is the starting node than nodes $\{1,2,3,4,5\}$ are reached (see Figure 2a). To this end we specify *graphInstance.dl* as local input file and *start(1)* as local input and write some assertion requiring that atoms $\{$*reached(1), reached(2), reached(3), reached(4), reached(5)*$\}$ are (cautiously) true.

In the second test case, named *guessSlice*, we select rules r_2 and r_3 and we require to test selected rules in isolation. The (local) inputs in this case are: the file *graphInstance.dl* and the fact $\{$*reached(1).*$\}$. In this case we are testing only the part of the program that guesses the paths, and we specify a couple of additional assertions (i.e., *path(1, 2)* has to be true in some answer set and false in some other).

Test case *guessSliceNonReachability* is run in *SPLIT_PROGRAM* modality, which requires to test the subprogram containing all the rules belonging to the splitting set corresponding to the selection (i.e., $\{$*path, outPath, edge, reached*$\}$). With this test case the sub-program that we are testing is composed by all the rules of the Hamiltonian Path example without the constraints. Also in this case we include the graph instance file *graphInstance.dl*. Additionally, we remove edge from 5 to 1 (see Fig. 2b), so that node 1 is not reached, then we add the assertion corresponding to this observation. Finally, we add a global assertion (*assertFalse("path(1,2).")* to check if *path(1,2)* is not contained in any answer set (note that the graph instance and the starting node are missing in the global inputs).

The test file described above can be created graphically and executed in *ASPIDE* as described in the following section.

4 Unit Testing in *ASPIDE*

In this section we describe the graphic tools implemented in *ASPIDE* conceived for developing and running test cases. Space constraints prevent us from providing a complete description of all the usage scenarios and available commands. However, in order to have an idea about the capabilities of the testing interface of *ASPIDE*, we describe step by step how to implement the example illustrated in the previous section.

Suppose that we have created in *ASPIDE* a project named SEA, which contains the files *hamiltonianPath.dl* and *graphInstance.dl* (see Fig. 1) storing the encoding of the hamiltonian path problem and the graph instance, respectively. Since the file that we want to test in our example is *hamiltonianPath.dl*, we select it in the *workspace explorer*, then we click the right button of the mouse and select *New Test* from the popup menu (Fig. 3a). The system shows the test creation dialog (Fig. 3b), which allows for both

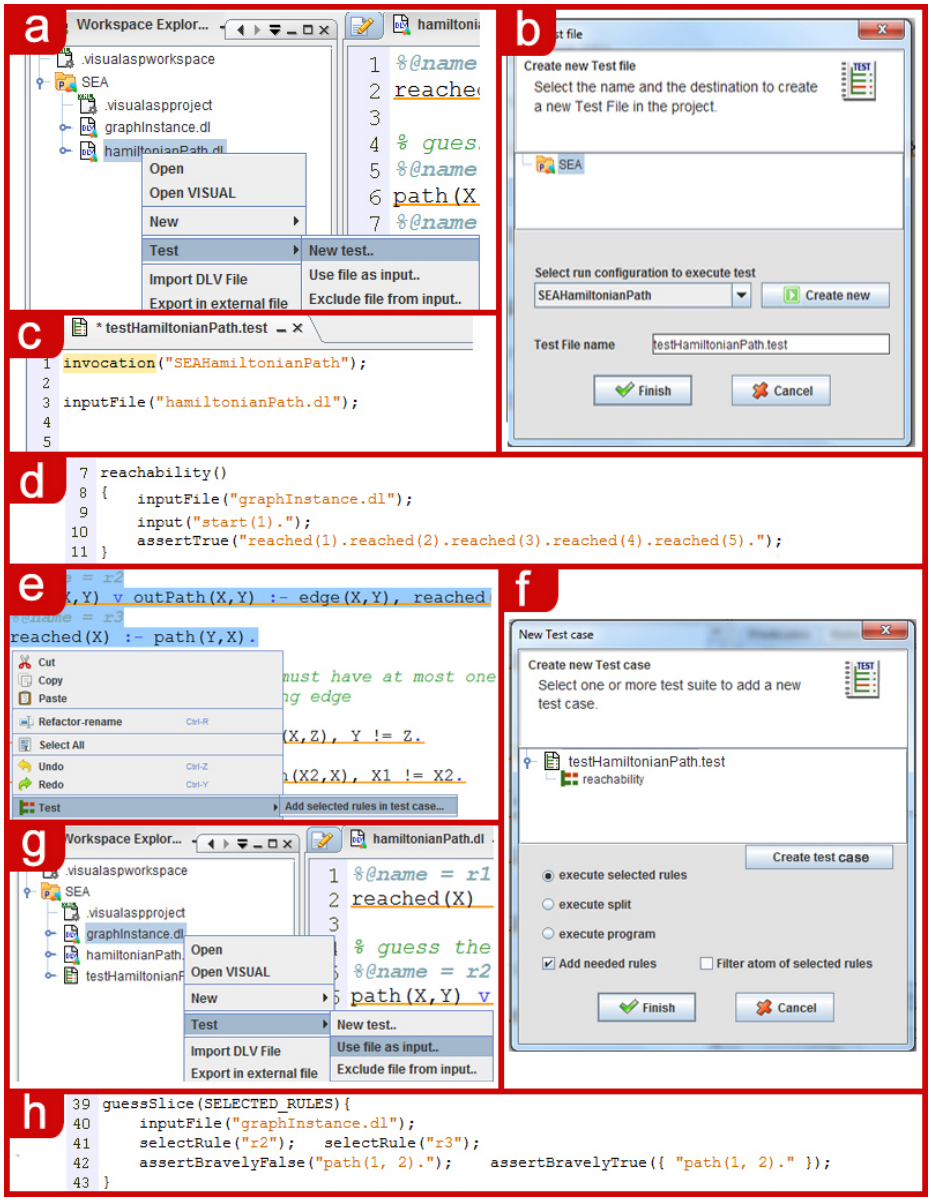


Fig. 3. Test case creation.

setting the name of the test file and selecting a previously-defined run configuration (storing execution options). In this case we select the one named *SEAHamiltonianPath* explicitly referring to the DLV system (*ASPIDE* will get the solver path directly from the selected run configuration). By clicking on the *Finish* button, the new test file is

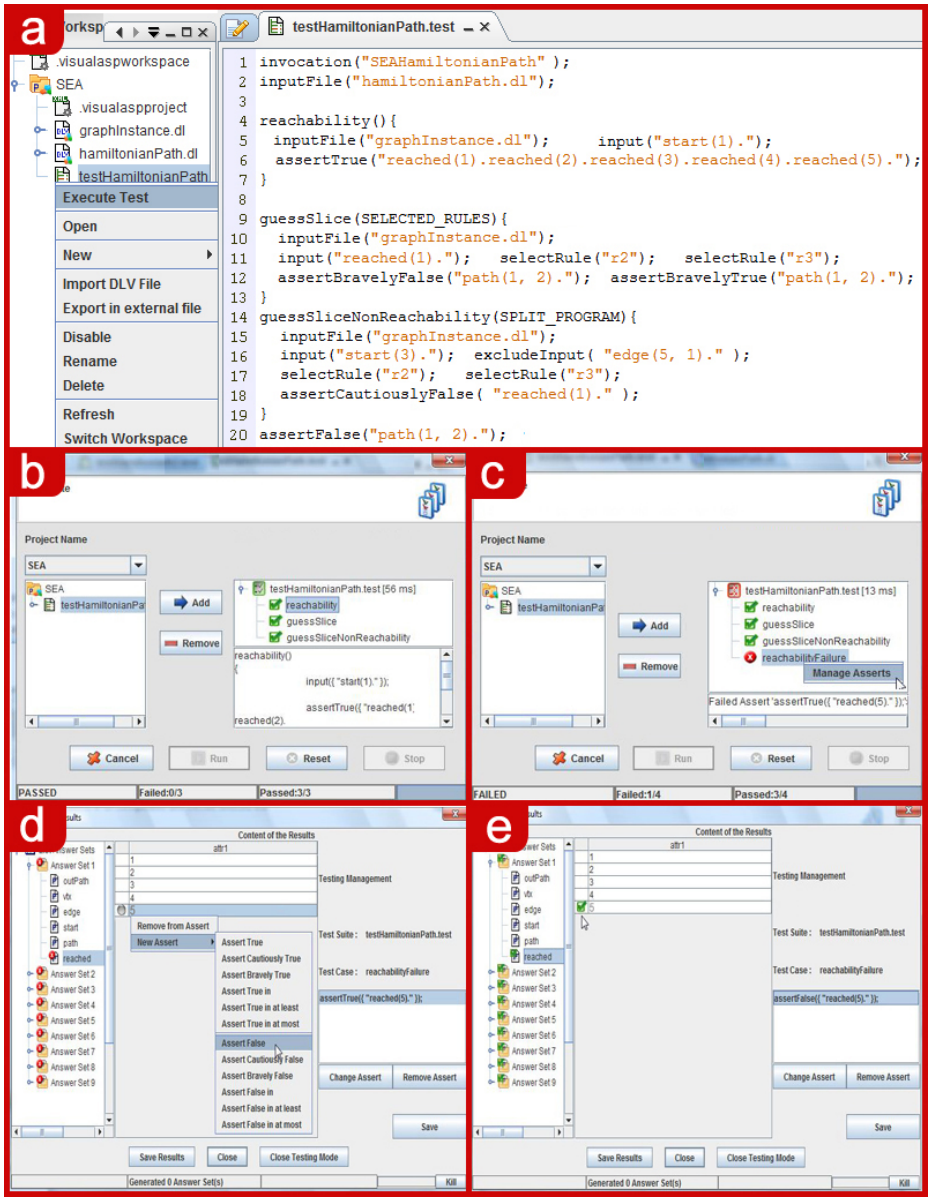


Fig. 4. Test case execution and assertion management.

created (see Fig. 3c) where the statements regarding input files and invocation are added automatically. We add the first unit test (called *reachability*) by exploiting the text editor (see Fig. 3d), whereas we build the remaining ones (working on some selected rules) by exploiting the logic program editor. After opening the *hamiltonianPath.dl* file, we select

rules r_2 and r_3 inside the text editor, we right-click on them and we select *Add selected rules in test case* from the menu item *Test* of the popup menu (fig. 3e). The system opens a dialog window where we indicate the test file in which we want to add the new test case (fig. 3f). We click on the *Create test case*; the system will ask for the name of the new test case and we write *guessSlice*; after that, on the window, we select the option *execute selected rules* and click on the *Finish* button. The system will add the test case *guessSlice* filled with the *selectRule* statements indicating the two selected rules. To add project files as input of the test case, we select them from the *workspace explorer* and click on *Use file as input* in the menu item *Test* (fig. 3g). The test created up to now is shown in figure 3h. Following an analogous procedure we create the remaining test cases (see Fig. 4a). To execute our tests, we right-click on the test file and select *Execute Test*. The *Test Execution Dialog* appears and the results are shown to the programmer (see Fig. 4b). Failing tests are indicated by a red icon, while green icons indicate passing tests. At this point, in order to show how to modify graphically a test case, we add the following additional test that purposely fails:

```
reachabilityFailure()
{
inputFile("graphInstance.dl");
input("start(1).");
excludeInput("edge(4, 5).");
excludeInput(":- vtx(X), not reached(X).");
assertTrue("reached(5).");
}
```

As shown in Figure 2c this additional test (as expected) fails, and the reason for this failure is indicated (see Fig. 4c) in the test execution dialog. In order to know which literals of the solution do not satisfy the assertion, we right-click on the failed test and select *Manage Asserts* from the menu. A dialog showing the outputs of the test appears where, in particular, predicates and literals matching correctly the assertions are marked in green, whereas the ones violating the assertion are marked in red (gray icons may appear to indicate missing literals which are expected to be in the solution). In our example, the assertion is *assertTrue("reached(5).")*, so we expect that all solutions contain the literal *reached(5)*; however, in our instance, node 5 is never reached, this is because the test case purposely contains an error. We modify this test case by adding the right assertion. This can be obtained by acting directly on the result window (fig. 4d). We remove the old assertion by selecting it and clicking on the *Remove Assert* button. Finally, we save the modifications, and we execute the test suite again (see Fig. 4e).

5 Conclusion

This paper presents a pragmatic environment for testing ASP programs. In particular, we present a new language, inspired to the JUnit framework [40], for specifying and running *unit tests* on ASP programs. The testing language has been implemented in *ASPIDE* together with some graphic tools for easing both the development of tests and the analysis of test execution.

As far as future work is concerned, we plan to extend *ASPIDE* by improving/introducing additional dynamic editing instruments, and graphic tools. In particular, we plan

to further improve the testing tool by supporting (semi)automatic test case generation based on the structural testing techniques proposed in [39].

Acknowledgments. This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009.

References

1. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC **9** (1991) 365–385
2. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS **22**(3) (1997) 364–418
3. Lifschitz, V.: Answer Set Planning. In: ICLP'99) 23–37
4. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: LPNMR'07. LNCS 4483, (2007) 3–17
5. Balduccini, M., Gelfond, M., Watson, R., Nogeira, M.: The USA-Advisor: A Case Study in Answer Set Planning. In: LPNMR 2001 (LPNMR-01). LNCS 2173, (2001) 439–442
6. Baral, C., Gelfond, M.: Reasoning Agents in Dynamic Domains. In: Logic-Based Artificial Intelligence. Kluwer (2000) 257–279
7. Baral, C., Uyan, C.: Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In: LPNMR 2001 (LPNMR-01). LNCS 2173, (2001) 186–199
8. Friedrich, G., Ivanchenko, V.: Diagnosis from first principles for workflow executions. Tech. Rep., http://proserver3-iwas.uni-klu.ac.at/download_area/Technical-Reports/technical_report_2008_02.pdf.
9. Franconi, E., Palma, A.L., Leone, N., Perri, S., Scarcello, F.: Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In: LPAR 2001. LNCS 2250, (2001) 561–578
10. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-Prolog Decision Support System for the Space Shuttle. In: Practical Aspects of Declarative Languages, Third International Symposium (PADL 2001). LNCS 1990, (2001) 169–183
11. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: SIGMOD 2005, Baltimore, Maryland, USA, ACM Press (2005) 915–917
12. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. CUP (2003)
13. Bardadym, V.A.: Computer-Aided School and University Timetabling: The New Wave. In: Practice and Theory of Automated Timetabling, First International Conference 1995. LNCS 1153, (1996) 22–45
14. Grasso, G., Iiritano, S., Leone, N., Ricca, F.: Some DLV Applications for Knowledge Management. In: Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009). LNCS 5753, (2009) 591–597
15. Grasso, G., Leone, N., Manna, M., Ricca, F.: Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of M. Gelfond. LNCS 6565 (2010)
16. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL **7**(3) (2006) 499–562
17. Simons, P.: Smodels Homepage (since 1996) `http://www.tcs.hut.fi/Software/smodels/`.

18. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. AI **138** (2002) 181–234
19. Zhao, Y.: ASSAT homepage (since 2002) http://assat.cs.ust.hk/.
20. Lin, F., Zhao, Y.: ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In: AAAI-2002, Edmonton, Alberta, Canada, AAAI Press / MIT Press (2002)
21. Babovich, Y., Maratea, M.: Cmodels-2: Sat-based answer sets solver enhanced to non-tight programs. http://www.cs.utexas.edu/users/tag/cmodels.html (2003)
22. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007,(2007) 386–392
23. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL **7**(1) (2006) 1–37
24. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: LPNMR'05. LNCS 3662, (2005) 447–451
25. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia, AAAI Press (2008) 422–432
26. Denecher, M., Vennekens, J., Bond, S., Gebser, M., M., M.T.: The second answer set programming system competition. In: Logic Programming and Nonmonotonic Reasoning — 10th International Conference, LPNMR'09. LNCS 5753, Potsdam, Germany, Berlin // Heidelberg (2009) 637–654
27. Dovier, A., Erdem, E.: Report on application session @lpnmr09 (2009) http://www.cs.nmsu.edu/ALP/2010/03/report-on-application-session-lpnmr09/.
28. Perri, S., Ricca, F., Terracina, G., Cianni, D., Veltri, P.: An integrated graphic tool for developing and testing DLV programs. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07). (2007) 86–100
29. Sureshkumar, A., Vos, M.D., Brain, M., Fitch, J.: APE: An AnsProlog* Environment. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07). (2007) 101–115
30. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: That is Illogical Captain! The Debugging Support Tool spock for Answer-Set Programs: System Description. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA'07). (2007) 71–85
31. Brain, M., De Vos, M.: Debugging Logic Programs under the Answer Set Semantics. In: Proceedings ASP05 - Answer Set Programming: Advances in Theory and Implementation, Bath, UK (2005)
32. El-Khatib, O., Pontelli, E., Son, T.C.: Justification and debugging of answer set programs in ASP. In: Proceedings of the Sixth International Workshop on Automated Debugging, California, USA, ACM (2005)
33. Oetsch, J., Pührer, J., Tompits, H.: Catching the ouroboros: On debugging non-ground answer-set programs. In: Proc. of the ICLP'10. (2010)
34. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: LPNMR'07. LNCS 4483, (2007) 31–43
35. De Vos, M., Schaub, T., eds.: SEA'07: Software Engineering for Answer Set Programming. In: . Volume 281., CEUR (2007) Online at http://CEUR-WS.org/Vol-281/.
36. De Vos, M., Schaub, T., eds.: SEA'09: Software Engineering for Answer Set Programming. In: . Volume 546., CEUR (2009) Online at http://CEUR-WS.org/Vol-546/.
37. Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., Leone, N.: OntoDLV: an ASP-based system for enterprise ontologies. Journal of Logic and Computation (2009)

38. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: Integrated Development Environment for Answer Set Programming. In: Logic Programming and Nonmonotonic Reasoning — 11th International Conference, LPNMR'11, Vancouver, Canada, 2011, Proceedings. LNCS 6645, (May 2011) 317–330
39. Janhunen, T., Niemelä, I., Oetsch, J., Pührer, J., Tompits, H.: On testing answer-set programs. In: Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, The Netherlands, IOS Press (2010) 951–956
40. JUnit.org community: JUnit, Resources for Test Driven Development `http://www.junit.org/`.
41. Jack, O.: Software Testing for Conventional and Logic Programming. Walter de Gruyter & Co., Hawthorne, NJ, USA (1996)
42. Wielemaker, J.: Prolog Unit Tests `http://www.swi-prolog.org/pldoc/package/plunit.html`.
43. Cancinos, C.: Prolog Development Tools - ProDT `http://prodevtools.sourceforge.net`.
44. Calimeri, F., Ianni, G., Ricca, F.: The third answer set programming system competition (since 2011) `https://www.mat.unical.it/aspcomp2011/`.
45. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. TPLP **8** (2008) 129–165
46. Ianni, G., Ielpa, G., Pietramala, A., Santoro, M.C.: Answer Set Programming with Templates. In: ASP'03, Messina, Italy (2003) 239–252 Online at `http://CEUR-WS.org/Vol-78/`.
47. Calimeri, F., Leone, N., Ricca, F., Veltri, P.: A Visual Tracer for DLV. In: Proc. of SEA'09, Potsdam, Germany (2009)
48. Febbraro, O., Reale, K., Ricca, F.: A Visual Interface for Drawing ASP Programs. In: Proc. of CILC2010, Rende(CS), Italy (2010)
49. Sommerville, I.: Software Engineering. Addison-Wesley (2004)
50. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: ICLP'94, MIT Press (1994) 23–37
51. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. **172** (2008) 1495–1539

Complexity of Super-Coherence Problems in ASP *

Mario Alviano[1], Wolfgang Faber[1], and Stefan Woltran[2]

[1] University of Calabria, Italy
{alviano,faber}@mat.unical.it
[2] Vienna University of Technology, Austria
woltran@dbai.tuwien.ac.at

Abstract. Adapting techniques from database theory in order to optimize Answer Set Programming (ASP) systems, and in particular the grounding components of ASP systems, is an important topic in ASP. In recent years, the Magic Set method has received some interest in this setting, and a variant of it, called DMS, has been proposed for ASP. However, this technique has a caveat, because it is not correct (in the sense of being query-equivalent) for all ASP programs. In recent work, a large fragment of ASP programs, referred to as *super-coherent programs*, has been identified, for which DMS is correct. An open question remained: How complex is it to determine whether a given program is super-coherent? This question turned out to be quite difficult to answer precisely. In this paper, we formally prove that deciding whether a propositional program is super-coherent is Π_3^P-complete in the disjunctive case, while it is Π_2^P-complete for normal programs. The hardness proofs are the difficult part in this endeavor: We proceed by characterizing the reductions by the models and reduct models which the ASP programs should have, and then provide instantiations that meet the given specifications.

1 Introduction

Answer Set Programming (ASP) is a powerful formalism for knowledge representation and common sense reasoning [5]. Allowing disjunction in rule heads and nonmonotonic negation in bodies, ASP can express every query belonging to the complexity class Σ_2^{P} ($\mathrm{NP}^{\mathrm{NP}}$). Encouraged by the availability of efficient inference engines, such as DLV [17], GnT [15], Cmodels [18], or ClaspD [8], ASP has found several practical applications in various domains, including data integration [16], semantic-based information extraction [20, 21], e-tourism [24], workforce management [25], and many more. As a matter of fact, these ASP systems are continuously enhanced to support novel optimization strategies, enabling them to be effective over increasingly larger application domains.

Frequently, optimization techniques are inspired by methods that had been proposed in other fields, for example database theory, satisfiability solving, or constraint satisfaction. Among techniques adapted to ASP from database theory, Magic Sets [26, 4, 6]

* This work will also be presented at the ASPOCP 2011 workshop. Partly supported by Regione Calabria and EU under POR Calabria FESR 2007-2013 and within the PIA project of DLVSYSTEM s.r.l., and by MIUR under the PRIN project LoDeN. We also thank the anonymous reviewers for their valuable comments.

have recently achieved a lot of attention. Following some earlier work [14, 7], recently an adapted method called *DMS* has been proposed for ASP in [3]. However, this technique has a caveat, because it is not correct (in the sense of being query-equivalent) for all ASP programs. In recent work [2, 1], a large fragment of ASP programs, referred to as *super-coherent programs* (ASP^{sc}), has been identified, for which DMS can be proved to be correct.

While our main motivation for studying ASP^{sc} stemmed from the applicability of DMS, this class actually has many more important motivations. Indeed, it can be viewed as the class of *non-constraining programs*: Adding extensional information to these programs will always result in answer sets. One important implication of this property is for modular evaluation. For instance, when using the splitting set theorem of [19], if a top part of a split program is an ASP^{sc} program, then any answer set of the bottom part will give rise to at least one answer set of the full program—so for determining answer set existence, there would be no need to evaluate the top part.

On a more abstract level, one of the main criticisms of ASP (being voiced especially in database theory) is that there are programs which do not admit any answer set (traditionally this has been considered a more serious problem than the related nondeterminism in the form of multiple answer sets, cf. [23]). From this perspective, programs which guarantee coherence (existence of an answer set) have been of interest for quite some time. In particular, if one considers a fixed program and a variable "database," one arrives naturally at the class ASP^{sc} when requiring existence of an answer set. This also indicates that deciding super-coherence of programs is related to some problems from the area of equivalence checking in ASP [13, 10, 22]. For instance, when deciding whether, for a given arbitrary program P, there is a uniformly equivalent definite positive (or definite Horn) program, super-coherence of P is a necessary condition—this is straightforward to see because definite Horn programs have exactly one answer set, so a non-super-coherent program cannot be uniformly equivalent to any definite Horn program.

Since the property of being super-coherent is a semantic one, a natural question arises: How difficult is it to decide whether a given program belongs to ASP^{sc}? It turns out that the precise complexity is rather difficult to establish. Some bounds have been given in [2], in particular showing decidability, but especially hardness results seemed quite hard to obtain.

In order to focus on the essentials of this problem, in this paper we deal with propositional programs and show the precise complexity (in terms of completeness) for deciding whether a given propositional ASP program belongs to ASP^{sc}. In Section 2 we first define some terminology needed later on. In Section 3 we formulate the problem that we analyze and state the results. The remainder of the paper contains the proofs — in Section 4 for disjunctive programs and in Section 5 for normal programs — and in Section 6 we briefly discuss the relation to equivalence problems before concluding the work in Section 7.

2 Preliminaries

In this paper we consider propositional programs, so an atom p is a member of a countable set $\mathcal{U}$. A *literal* is either an atom p (a positive literal), or an atom preceded by the *negation as failure* symbol not (a negative literal). A *rule* r is of the form

$$p_1 \vee \cdots \vee p_n \leftarrow q_1, \ldots, q_j, not\ q_{j+1}, \ldots, not\ q_m$$

where $p_1, \ldots, p_n, q_1, \ldots, q_m$ are atoms and $n \geq 0, m \geq j \geq 0$. The disjunction $p_1 \vee \cdots \vee p_n$ is the *head* of r, while the conjunction $q_1, \ldots, q_j, not\ q_{j+1}, \ldots, not\ q_m$ is the *body* of r. Moreover, $H(r)$ denotes the set of head atoms, while $B(r)$ denotes the set of body literals. We also use $B^+(r)$ and $B^-(r)$ for denoting the set of atoms appearing in positive and negative body literals, respectively, and $At(r)$ for the set $H(r) \cup B^+(r) \cup B^-(r)$. A rule r is normal (or disjunction-free) if $|H(r)| = 1$ or $|H(r)| = 0$ (in this case r is also referred to as a *constraint*), positive (or negation-free) if $B^-(r) = \emptyset$, a *fact* if both $B(r) = \emptyset$ and $|H(r)| = 1$.

A *program* P is a finite set of rules; if all rules in it are positive (resp. normal), then P is a positive (resp. normal) program. Odd-cycle-free and stratified programs constitute two other interesting classes of programs. An atom p appearing in the head of a rule r *depends* on each atom q that belongs to $B(r)$; if q belongs to $B^+(r)$, p depends positively on q, otherwise negatively. A program without constraints is *odd-cycle-free* if there is no cycle of dependencies involving an odd number of negative dependencies, while it is *stratified* if each cycle of dependencies involves only positive dependencies. Programs containing constraints have been excluded by the definition of odd-cycle-free and stratified programs. In fact, constraints intrinsically introduce odd-cycles in programs as a constraint of the form

$$\leftarrow q_1, \ldots, q_j, not\ q_{j+1}, \ldots, not\ q_m$$

can be replaced by the following equivalent rule:

$$co \leftarrow q_1, \ldots, q_j, not\ q_{j+1}, \ldots, not\ q_m, not\ co,$$

where co is a fresh atom (i.e., an atom that does not occur elsewhere in the program).

Given a program P, let $At(P)$ denote the set of atoms that occur in it, that is, let $At(P) = \bigcup_{r \in P} At(r)$. An *interpretation* I for a program P is a subset of $At(P)$. An atom p is true w.r.t. an interpretation I if $p \in I$; otherwise, it is false. A negative literal $not\ p$ is true w.r.t. I if and only if p is false w.r.t. I. The body of a rule r is true w.r.t. I if and only if all the body literals of r are true w.r.t. I, that is, if and only if $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. An interpretation I *satisfies* a rule $r \in P$ if at least one atom in $H(r)$ is true w.r.t. I whenever the body of r is true w.r.t. I. An interpretation I is a *model* of a program P if I satisfies all the rules in P.

Given an interpretation I for a program P, the reduct of P w.r.t. I, denoted by P^I, is obtained by deleting from P all the rules r with $B^-(r) \cap I \neq \emptyset$, and then by removing all the negative literals from the remaining rules. The semantics of a program P is given by the set $AS(P)$ of the answer sets of P, where an interpretation M is an answer set for P if and only if M is a subset-minimal model of P^M.

In the subsequent sections, we will use the following properties that the models and models of reducts of programs satisfy (see, e.g. [9, 13]):

(P1) for any disjunctive program P and interpretations $I \subseteq J \subseteq K$, if I satisfies P^J, then I also satisfies P^K;
(P2) for any normal program P and interpretations $I, J \subseteq K$, if I and J both satisfy P^K, then also $(I \cap J)$ satisfies P^K.

We now introduce super-coherent ASP programs ($\mathrm{ASP}^{\mathrm{sc}}$ programs), the main class of programs studied in this paper.

Definition 1 ($\mathrm{ASP}^{\mathrm{sc}}$ programs [1, 2]). *A program P is* super-coherent *if, for every set of facts F, $AS(P \cup F) \neq \emptyset$. Let $\mathrm{ASP}^{\mathrm{sc}}$ denote the set of all super-coherent programs.*

Note that $\mathrm{ASP}^{\mathrm{sc}}$ programs include all odd-cycle-free programs (and therefore also all stratified programs). Indeed, every odd-cycle-free program admits at least one answer set and remains odd-cycle-free even if an arbitrary set of facts is added to its rules. On the other hand, there are programs having odd-cycles that are in $\mathrm{ASP}^{\mathrm{sc}}$, cf. [2].

An important question regards whether $\mathrm{ASP}^{\mathrm{sc}}$ programs are as expressive as ASP programs. Of course, checking coherence (existence of answer sets) is a trivial task for $\mathrm{ASP}^{\mathrm{sc}}$ programs. But when considering query answering, it turns out that expressivity is not lowered. Indeed, all expressivity results of [12] hold for disjunctive programs with stratified negation (examining the proofs, actually for disjunctive programs with input negation, that is, having at most two strata), which guarantee super-coherence and are a proper subset of $\mathrm{ASP}^{\mathrm{sc}}$. It therefore follows that all properties in Σ_2^P or Π_2^P are expressible by $\mathrm{ASP}^{\mathrm{sc}}$ programs using a query under brave or cautious reasoning, respectively. The picture is less clear for nondisjunctive $\mathrm{ASP}^{\mathrm{sc}}$ programs.

However, we should point out that many (probably most) existing ASP programs follow a "Guess&Check" or "Generate&Test" methodology, which usually relies on integrity constraints, the presence of which usually contradicts super-coherence. As an alternative, violated integrity constraints can derive a special atom, on which the query atom should depend negatively. If the Guess/Generate part involves non-stratified negation, it depends on how this construct is used in the encoding. If it just encodes a choice, this can often be easily converted to a disjunction, while for encodings that entangle guess and check using unstratified negation, an automated conversion to an $\mathrm{ASP}^{\mathrm{sc}}$ program seems less straightforward. In general, however, we feel that for obtaining computationally efficient $\mathrm{ASP}^{\mathrm{sc}}$ encodings, different encoding methodologies should be developed.

3 Problem Statement and Main Theorems

In this paper, we study the complexity of the following natural problem.

– Given a program P, is P super-coherent, i.e. does $AS(P \cup F) \neq \emptyset$ hold for any set F of facts.

We will study the complexity for this problem for the case of disjunctive logic programs and non-disjunctive (normal) logic programs. We first have a look at a similar problem, which turns out to be rather trivial to decide.

Proposition 1. *The problem of deciding whether, for a given disjunctive program P, there is a set F of facts such that $AS(P \cup F) \neq \emptyset$ is* NP*-complete;* NP*-hardness holds already for normal programs.*

Proof. We start by observing that there is F such that $AS(P \cup F) \neq \emptyset$ if and only if P has at least one classical model. Indeed, if M is a model of P, then $P \cup M$ has M as its answer set. On the other hand, if P has no model, then no addition of facts F will yield an answer set for $P \cup F$. It is well known that deciding whether a program has at least one (classical) model is NP-complete for both disjunctive and normal programs. □

In contrast, the complexity for deciding super-coherence is surprisingly high, which we shall show next. To start, we give a straight-forward observation.

Proposition 2. *A program P is super-coherent if and only if for each set $F \subseteq At(P)$, $AS(P \cup F) \neq \emptyset$.*

Proof. The only-if direction is by definition. For the if-direction, let F be any set of facts. F can be partitioned into $F' = F \cap At(P)$ and $F'' = F \setminus F'$. By assumption, $P \cup F'$ is coherent. Let M be an answer set of $P \cup F'$. We shall show that $M \cup F''$ is an answer set of $P \cup F = P \cup F' \cup F''$. This is in fact a consequence of the splitting set theorem [19], as the atoms in F'' are only defined by facts not occurring in $P \cup F'$. □

Our main results are as follows. The proofs are contained in the subsequent sections.

Theorem 1. *The problem of deciding super-coherence for disjunctive programs is Π_3^P-complete.*

Theorem 2. *The problem of deciding super-coherence for normal programs is Π_2^P-complete.*

4 Proof of Theorem 1

Membership follows by the following straight-forward nondeterministic algorithm for the complementary problem, i.e. given a program P, does there exist a set F of facts such that $AS(P \cup F) = \emptyset$: we guess a set $F \subseteq At(P)$ and check $AS(P \cup F) = \emptyset$ via an oracle-call. Restricting the guess to $At(P)$ can be done by Proposition 2. Checking $AS(P \cup F) = \emptyset$ is known to be in Π_2^P [11]. This shows Π_3^P-membership.

For the hardness we reduce the Π_3^P-complete problem of deciding whether QBFs of the form $\forall X \exists Y \forall Z \phi$ are true to the problem of super-coherence. Without loss of generality, we can consider ϕ to be in DNF and, indeed, $X \neq \emptyset$, $Y \neq \emptyset$, and $Z \neq \emptyset$. We also assume that each disjunct of Φ contains at least one variable from X, one from Y and one from Z. More precisely, we shall construct for each such QBF Φ a program P_Φ such that Φ is true iff P_Φ is super-coherent. Before showing how to actually construct P_Φ from Φ in polynomial time, we give the required properties for P_Φ. We then show that for programs P_Φ satisfying these properties, the desired relation (Φ is true iff P_Φ is super-coherent) holds, and finally we provide the construction of P_Φ.

Definition 2. *Let $\Phi = \forall X \exists Y \forall Z \phi$ be a QBF with ϕ in DNF. We call any program P satisfying the following properties a* Φ-reduction*:*

1. *P is given over atoms $U = X \cup Y \cup Z \cup \overline{X} \cup \overline{Y} \cup \overline{Z} \cup \{u, v, w\}$, where all atoms in sets $\overline{S} = \{\overline{s} \mid s \in S\}$ and $\{u, v, w\}$ are fresh and mutually disjoint;*
2. *P has the following models:*
   - *U*
   - *for each $I \subseteq X$, $J \subseteq Y$,*
   $$M[I, J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{u, v\}$$
   and
   $$M'[I, J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup Z \cup \overline{Z} \cup \{v, w\};$$
3. *for each $I \subseteq X$, $J \subseteq Y$, the models[3] of the reduct $P^{M[I,J]}$ are $M[I, J]$ and*
   $$O[I] = I \cup \overline{(X \setminus I)};$$
4. *for each $I \subseteq X$, $J \subseteq Y$, the models of the reduct $P^{M'[I,J]}$ are $M'[I, J]$ and*
   - *for each $K \subseteq Z$ such that $I \cup J \cup K \not\models \phi$,*
   $$N[I, J, K] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup K \cup \overline{(Z \setminus K)} \cup \{v\};$$
5. *the models of the reduct P^U are given only by the models already mentioned above, i.e. U itself, $M[I, J]$, $M'[I, J]$, and $O[I]$, for each $I \subseteq X$, $J \subseteq Y$, and $N[I, J, K]$ for each $I \subseteq X$, $J \subseteq Y$, $K \subseteq Z$, such that $I \cup J \cup K \not\models \phi$.*

The structure of models of Φ-reductions and the "countermodels" (see below what we mean by this term) of the relevant reducts is sketched in Figure 1. The center of the diagram contains the models of the Φ-reduction and their subset relationship. For each of the model the respective box lists the "countermodels," by which we mean those reduct models which can serve as counterexamples for the original model being an answer set, that is, those reduct models which are proper subsets of the original model.

We just note at this point that the models of the reduct P^U given in Item 5 are not specified for particular purposes, but are required to allow for a realization via disjunctive programs. In fact, these models are just an effect of property (P1) mentioned in Section 2. However, before showing a program satisfying the properties of a *Φ-reduction*, we first show the rationale behind the concept of Φ-reductions.

Lemma 1. *For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with ϕ in DNF, a Φ-reduction is super-coherent iff Φ is true.*

Proof. Suppose that Φ is false. Hence, there exists an $\mathcal{I} \subseteq X$ such that, for all $J \subseteq Y$, there is a $K_Y \subseteq Z$ with $\mathcal{I} \cup J \cup K_Y \not\models \phi$. Now let P be any Φ-reduction and $F_\mathcal{I} = \mathcal{I} \cup \overline{(X \setminus \mathcal{I})}$. We show that $AS(P \cup F_\mathcal{I}) = \emptyset$, thus P is not super-coherent. Let $\mathcal{M}$ be a model of $P \cup F_\mathcal{I}$. Since P is a Φ-reduction, the only candidates for $\mathcal{M}$ are U, $M[\mathcal{I}, J]$, and $M'[\mathcal{I}, J]$, where $J \subseteq Y$. Indeed, for each $I \neq \mathcal{I}$, $M[I, J]$ and $M'[I, J]$ cannot be models of $P \cup F_\mathcal{I}$ because $F_\mathcal{I} \not\subseteq M[I, J]$, resp. $F_\mathcal{I} \not\subseteq M'[I, J]$. We now analyze these three types of potential candidates:

[3] Here and below, for a reduct P^M we only list models of the form $N \subseteq M$, since those are the relevant ones for our purposes. Recall that $N = M$ is always a model of P^M in case M is a model of P.

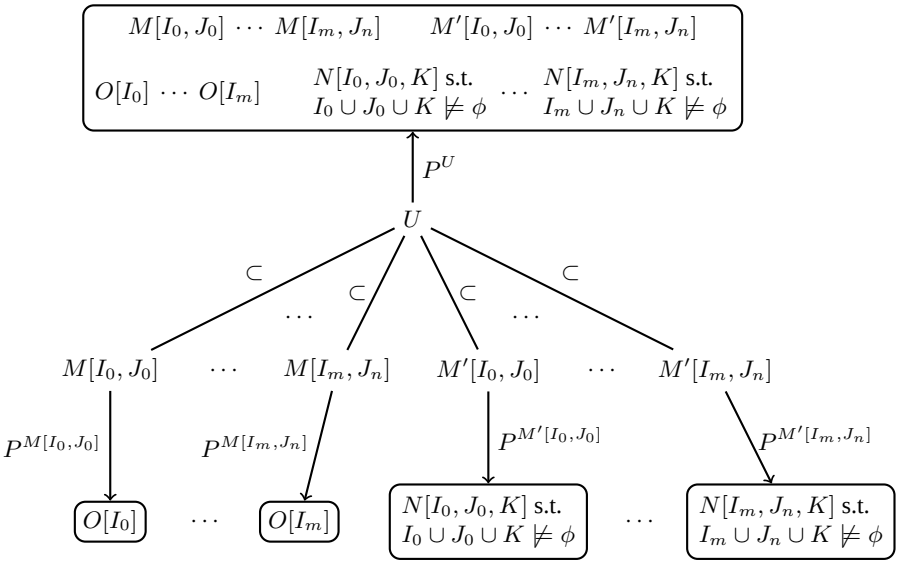


Fig. 1. Models and reduct "countermodels" of Φ-reductions

- $\mathcal{M} = U$: Then, for instance, $M[\mathcal{I}, J] \subset U$ is a model of $(P \cup F_{\mathcal{I}})^{\mathcal{M}} = P^{\mathcal{M}} \cup F_{\mathcal{I}}$ for any $J \subseteq Y$. Thus, $\mathcal{M} \notin AS(P \cup F_{\mathcal{I}})$.
- $\mathcal{M} = M[\mathcal{I}, J]$ for some $J \subseteq Y$. Then, by the properties of Φ-reductions, $O[\mathcal{I}] \subset \mathcal{M}$ is a model of $(P \cup F_{\mathcal{I}})^{\mathcal{M}} = P^{\mathcal{M}} \cup F_{\mathcal{I}}$. Thus, $\mathcal{M} \notin AS(P \cup F_{\mathcal{I}})$.
- $\mathcal{M} = M'[\mathcal{I}, J]$ for some $J \subseteq Y$. By the initial assumption, there exists a $K_Y \subseteq Z$ with $\mathcal{I} \cup J \cup K_Y \not\models \phi$. Then, by the properties of Φ-reductions, $N[\mathcal{I}, J, K] \subset \mathcal{M}$ is a model of $P^{\mathcal{M}}$. Thus, $\mathcal{M} \notin AS(P \cup F_{\mathcal{I}})$.

In each of the cases we have obtained $\mathcal{M} \notin AS(P \cup F_{\mathcal{I}})$, hence $AS(P \cup F_{\mathcal{I}}) = \emptyset$ and P is not super-coherent.

Suppose that Φ is true. It is sufficient to show that for each $F \subseteq U$, $AS(P \cup F) \neq \emptyset$. We have the following cases:

If $\{s, \overline{s}\} \subseteq F$ for some $s \in X \cup Y$ or $\{u, w\} \subseteq F$. Then $U \in AS(P \cup F)$ since U is a model of $P \cup F$ and each potential model $M \subset U$ of the reduct P^U (see the properties of Φ-reductions) does not satisfy $F \subseteq M$; thus each such M is not a model of $P^U \cup F = (P \cup F)^U$.

Otherwise, we have $F \subseteq M[I, J]$ or $F \subseteq M'[I, J]$ for some $I \subseteq X$, $J \subseteq Y$. In case $F \subseteq M[I, J]$ and $F \not\subseteq O[I]$, we observe that $M[I, J] \in AS(P \cup F)$ since $O[I]$ is the only model of the reduct $P^{M[I,J]}$. Thus for each such F there cannot be a model $M \subset M[I, J]$ of $P^{M[I,J]} \cup F = (P \cup F)^{M[I,J]}$. As well, in case $F \subseteq M'[I, J]$, where $w \in F$, $M'[I, J]$ can be shown to be an answer set of $P \cup F$. Indeed, in this case no $M \subset M'[I, J]$ is a model of $P^{M'[I,J]}$ because Φ is true.

It remains to consider the case $F \subseteq O[I]$ for each $I \subseteq X$. We show that $M'[I, J]$ is an answer set of $P \cup F$, for some $J \subseteq Y$. Since Φ is true, we know that, for each

$I \subseteq X$, there exists a $J_I \subseteq Y$ such that, for all $K \subseteq Z$, $I \cup J_I \cup K \models \phi$. As can be verified by the properties of Φ-reductions, then there is no model $M \subset M'[I, J_I]$ of $P^{M'[I,J_I]}$. Consequently, there is also no such model of $(P \cup F)^{M'[I,J_I]}$, and thus $M'[I, J_I] \in AS(P \cup F)$.

So in each of these cases $AS(P \cup F) \neq \emptyset$ and since these cases cover all possible $F \subseteq U$, we obtain that P is supercoherent.

In total we have shown that Φ being false implies that any Φ-reduction P is not super-coherent, while Φ being true implies that any Φ-reduction is super-coherent, which proves the lemma.

□

It remains to show that for any QBF of the desired form, a Φ-reduction can be obtained in polynomial time (w.r.t. the size of Φ). For the construction below, let us denote a negated atom a in the propositional part of the QBF Φ as $\overline{a}$.

Definition 3. *For any QBF $\Phi = \forall X \exists Y \forall Z \phi$ with $\phi = \bigvee_{i=1}^{n} l_{i,1} \wedge \cdots \wedge l_{i,m_i}$ a DNF (i.e., a disjunction of conjunctions over literals), we define*

$$
\begin{aligned}
P_\Phi = \{&x \vee \overline{x} \leftarrow;\ u \leftarrow x, \overline{x};\ w \leftarrow x, \overline{x};\ x \leftarrow u, w;\ \overline{x} \leftarrow u, w \mid x \in X\} \cup && (1)\\
&\{y \vee \overline{y} \leftarrow v;\ u \leftarrow y, \overline{y};\ w \leftarrow y, \overline{y};\ y \leftarrow u, w;\\
&\ \overline{y} \leftarrow u, w;\ v \leftarrow y;\ v \leftarrow \overline{y} \mid y \in Y\} \cup && (2)\\
&\{z \vee \overline{z} \leftarrow v;\ u \leftarrow z, not\ w;\ u \leftarrow \overline{z}, not\ w;\ v \leftarrow z;\ v \leftarrow \overline{z};\\
&\ z \leftarrow w;\ \overline{z} \leftarrow w;\ z \leftarrow u;\ \overline{z} \leftarrow u;\ w \vee u \leftarrow z, \overline{z} \mid z \in Z\} \cup && (3)\\
&\{w \vee u \leftarrow l_{i,1}, \ldots, l_{i,m_i} \mid 1 \leq i \leq n\} && (4)\\
&\{v \leftarrow w;\ v \leftarrow u;\ v \leftarrow not\ u\}. && (5)
\end{aligned}
$$

Obviously, the program from above definition can be constructed in polynomial time in the size of the reduced QBF. To conclude the proof of Theorem 1 it is thus sufficient to show the following relation.

Lemma 2. *For any QBF $\Phi = \forall X \exists Y \forall Z \phi$, the program P_Φ is a Φ-reduction.*

Proof. Obviously, $At(P_\Phi)$ contains the atoms as required in 1) of Definition 2. We continue to show 2). To see that U is a model of P_Φ is obvious. We next show that the remaining models M are all of the form $M[I, J]$ or $M'[I, J]$. First we have $v \in M$ because of the rules $v \leftarrow u$ and $v \leftarrow not\ u$ in (5). In case $w \in M$, $Z \cup \overline{Z} \subseteq M$ by the rules in (3). In case $w \notin M$, we have $K \cup \overline{(Z \setminus K)} \subseteq M$ for some $K \subseteq Z$, since $v \in M$ and by (3). But then, since $w \notin M$, $u \in M$ holds (rules $u \leftarrow z, not\ w$ resp. $u \leftarrow \overline{z}, not\ w$). Hence, also here $Z \cup \overline{Z} \subseteq M$. In both cases, we observe that by (1) and (2), $I \cup \overline{(X \setminus U)} \cup J \cup \overline{(Y \setminus J)} \subseteq M$, for some $I \subseteq X$ and $J \subseteq Y$. This yields the desired models, $M[I, J]$, $M'[I, J]$. It can be checked that no other model exists by showing that for $N \not\subseteq M[I, J]$, resp. $N \not\subseteq M'[I, J]$, $N = U$ follows.

We next show that, for each $I \subseteq X$ and $J \subseteq Y$, $P^{M[I,J]}$ and $P^{M'[I,J]}$ possess the required models. Let us start by showing that $O[I]$ is a model of $P^{M[I,J]}$. In fact, it can be observed that all of the rules of the form $x \vee \overline{x} \leftarrow$ in (1) are satisfied because either

x or $\overline{x}$ belong to $O[I]$, while all of the other rules in $P^{M[I,J]}$ are satisfied because of a false body literal. We also note that each strict subset of $O[I]$ does not satisfy some rule of the form $x \vee \overline{x} \leftarrow$, and thus it is not a model of $P^{M[I,J]}$. Similarly, any interpretation W such that $O[I] \subset W \subset M[I,J]$ does not satisfy some rule in $P^{M[I,J]}$ (note that rules of the form $u \leftarrow z$ and $u \leftarrow \overline{z}$ occur in $P^{M[I,J]}$ because $w \notin M[I,J]$; such rules are obtained by rules in (3)).

Let us now consider $P^{M'[I,J]}$ and let $W \subseteq M'[I,J]$ be one of its models. We shall show that either $W = M'[I,J]$, or $W = N[I,J,K]$ for some $K \subseteq Z$ such that $I \cup J \cup Z \not\models \phi$. Note that v is a fact in $P^{M'[I,J]}$, hence v must belong to W. By (1) and (2), since $v \in W$ and $W \subseteq M'[I,J]$, we can conclude that all of the atoms in $I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)}$ belong to W. Consider now the atom w. If w belongs to W, by the rules in (3) we conclude that all of the atoms in $Z \cup \overline{Z}$ belong to W, and thus $W = M'[I,J]$. Otherwise, if $w \notin W$, by the rules of the form $z \vee \overline{z} \leftarrow v$ in (3), there must be a set $K \subseteq Z$ such that $K \cup \overline{(Z \setminus K)}$ is contained in W. Note that no other atoms in $Z \cup \overline{Z}$ can belong to W because of the last rule in (3). Hence, $W = N[I,J,K]$. Moreover, $w \notin W$ and $u \notin W$ imply that $I \cup J \cup K \not\models \phi$ holds because of (4).

Finally, one can show that P^U does not yield additional models as those which are already present by other models. Let $W \subseteq U$ be a model of P^U. By (1), $O[I] \subseteq W$ must hold for some $I \subseteq X$. Consider now the atom v. If $v \notin W$, we conclude that the model W is actually $O[I]$. We can thus consider the other case, i.e. $v \in W$. By (2), $J \cup \overline{(Y \setminus J)} \subseteq W$ must hold for some $J \subseteq Y$. Consider now the atom u. If $u \in W$, we have $Z \cup \overline{Z} \subseteq W$ because of (3). If no other atom belongs to W, then $W = M[I,J]$ holds. Otherwise, if any other atom belongs to W, it can be checked that W must be equal to U. We can then consider the case in which $u \notin W$, and the atom w. Again, we have two possibilities. If w belongs to W, by (3) we conclude that all of the atoms in $Z \cup \overline{Z}$ belong to W, and thus either $W = M'[I,J]$ or $W = U$. Otherwise, if $w \notin W$, by the rules of the form $z \vee \overline{z} \leftarrow v$ in (3), there must be a set $K \subseteq Z$ such that $K \cup \overline{(Z \setminus K)}$ is contained in W. Note that no other atoms in $Z \cup \overline{Z}$ can belong to W because of the last rule in (3). Hence, $W = N[I,J,K]$. Moreover, because of (4), $w \notin W$ and $u \notin W$ imply that $I \cup J \cup K \not\models \phi$ holds. □

Note that the program from Definition 3 does not contain constraints. As a consequence, the Π_3^P-hardness result presented in this section also holds if we only consider disjunctive ASP programs without constraints.

5 Proof of Theorem 2

Membership follows by the straight-forward nondeterministic algorithm for the complementary problem presented in the previous section. We have just to note that a $co-\mathrm{NP}$ oracle can be used for checking the consistency of a normal program. Thus, Π_2^P-membership is established.

For the hardness we reduce the Π_2^P-complete problem of deciding whether QBFs of the form $\forall X \exists Y \phi$ are true to the problem of super-coherence. Without loss of generality, we can consider ϕ to be in CNF and, indeed, $X \neq \emptyset$, and $Y \neq \emptyset$. We also assume that

each clause of Φ contains at least one variable from X and one from Y. More precisely, we shall adapt the notion of Φ-reduction to normal programs. In particular, we have to take into account a fundamental difference between disjunctive and normal programs: while disjunctive programs allow for using disjunctive rules for guessing a subset of atoms, such a guess can be achieved only by means of unstratified negation within a normal program. For example, one atom in a set $\{x, y\}$ can be guessed by means of the following disjunctive rule: $x \vee y \leftarrow$. Within a normal program, the same result can be obtained by means of the following rules: $x \leftarrow not\ y$ and $y \leftarrow not\ x$. However, these last rules would be deleted in the reduced program associated with a model containing both x and y, which would allow for an arbitrary subset of $\{x, y\}$ to be part of a model of the reduct. More generally speaking, we have to take Property (P2), as introduced in Section 2, into account. This makes the following definition a bit more cumbersome compared to Definition 2.

Definition 4. *Let $\Phi = \forall X \exists Y \phi$ be a QBF with ϕ in CNF. We call any program P satisfying the following properties a Φ-*norm-reduction*:*

1. *P is given over atoms $U = X \cup Y \cup \overline{X} \cup \overline{Y} \cup \{v, w\}$, where all atoms in sets $\overline{S} = \{\overline{s} \mid s \in S\}$ and $\{v, w\}$ are fresh and mutually disjoint;*
2. *P has the following models:*
   - *for each $J \subseteq Y$, and for each J^* such that $J \cup \overline{(Y \setminus J)} \subseteq J^* \subseteq Y \cup \overline{Y}$*
   $$O[J^*] = X \cup \overline{X} \cup J^* \cup \{v, w\};$$
   - *for each $I \subseteq X$,*
   $$M[I] = I \cup \overline{(X \setminus I)} \cup \{v\};$$
   - *for each $I \subseteq X$, $J \subseteq Y$, such that $I \cup J \models \phi$,*
   $$N[I, J] = I \cup \overline{(X \setminus I)} \cup J \cup \overline{(Y \setminus J)} \cup \{w\};$$
3. *the only models of a reduct $P^{M[I]}$ are $M[I]$ and $M[I] \setminus \{v\}$; the only model of a reduct $P^{N[I,J]}$ is $N[I, J]$;*
4. *each model M of a reduct $P^{O[J^*]}$ satisfies the following properties:*
   (a) *for each $y \in Y$ such that $y \in O[J^*]$ and $\overline{y} \notin O[J^*]$, if $w \in M$, then $y \in M$;*
   (b) *for each $y \in Y$ such that $\overline{y} \in O[J^*]$ and $y \notin O[J^*]$, if $w \in M$, then $\overline{y} \in M$;*
   (c) *if $(Y \cup \overline{Y}) \cap M \neq \emptyset$, then $w \in M$;*
   (d) *if there is a clause $l_{i,1} \vee \cdots \vee l_{i,m_i}$ of ϕ such that $\{\bar{l}_{i,1}, \ldots, \bar{l}_{i,m_i}\} \subseteq M$, then $v \in M$;*
   (e) *if there is an $x \in X$ such that $\{x, \overline{x}\} \subseteq M$, or there is a $y \in Y$ such that $\{y, \overline{y}\} \subseteq M$, or $\{v, w\} \subseteq M$, then it must hold that $X \cup \overline{X} \cup \{v, w\} \subseteq M$.*

Lemma 3. *For any QBF $\Phi = \forall X \exists Y \phi$ with ϕ in CNF, a Φ-norm-reduction is super-coherent iff Φ is true.*

Proof. Suppose that Φ is false. Hence, there exists an $\mathcal{I} \subseteq X$ such that, for all $J \subseteq Y$, $\mathcal{I} \cup J \not\models \phi$. Now, let P be any Φ-norm-reduction and $F_{\mathcal{I}} = \mathcal{I} \cup \overline{(X \setminus \mathcal{I})}$. We show that $AS(P \cup F_{\mathcal{I}}) = \emptyset$, thus P is not super-coherent. Let $\mathcal{M}$ be a model of $P \cup F_{\mathcal{I}}$. Since

P is a Φ-norm-reduction, the only candidates for $\mathcal{M}$ are $O[J^*]$ for some $J \subseteq Y$ and J^* such that $J \cup \overline{(Y \setminus J)} \subseteq J^* \subseteq Y \cup \overline{Y}$, $M[\mathcal{I}]$, and $N[\mathcal{I}, J']$, where $J' \subseteq Y$ satisfies $\mathcal{I} \cup J' \models \phi$. However, from our assumption (for all $J \subseteq Y$, $\mathcal{I} \cup J \not\models \phi$), no such $N[\mathcal{I}, J']$ exists. Thus, it remains to consider $O[J^*]$ and $M[\mathcal{I}]$. By the properties of Φ-norm-reductions, $M[\mathcal{I}] \setminus \{v\}$ is a model of $P^{M[\mathcal{I}]}$, and hence $M[\mathcal{I}] \setminus \{v\}$ is also a model of $P^{M[\mathcal{I}]} \cup F_{\mathcal{I}} = (P \cup F_{\mathcal{I}})^{M[\mathcal{I}]}$. Thus, $M[\mathcal{I}]$ is not an answer set of $P \cup F_{\mathcal{I}}$. On the other hand, it can be checked that $M[\mathcal{I}] \setminus \{v\}$ is a model of $P^{O[J^*]} \cup F_{\mathcal{I}} = (P \cup F_{\mathcal{I}})^{O[J^*]}$, for any $O[J^*]$, and so none of these $O[J^*]$ are answer sets of $P \cup F_{\mathcal{I}}$ either. Since this means that no model of $P \cup F_{\mathcal{I}}$ is an answer set, we conclude $AS(P \cup F_{\mathcal{I}}) = \emptyset$, and hence P is not super-coherent.

Suppose that Φ is true. It is sufficient to show that, for each $F \subseteq U$, $AS(P \cup F) \neq \emptyset$. We distinguish the following cases for $F \subseteq U$:

$F \subseteq I \cup \overline{(X \setminus I)} \cup \{v\}$ for some $I \subseteq X$: If $v \in F$, then $M[I]$ is the unique model of $P^{M[I]} \cup F = (P \cup F)^{M[I]}$, and thus $M[I] \in AS(P \cup F)$. Otherwise, if $v \notin F$, since Φ is true, there exists a $J \subseteq Y$ such that $I \cup J \models \phi$. Thus, $N[I, J]$ is a model of $P \cup F$, and since no subset of $N[I, J]$ is a model of $(P \cup F)^{N[I,J]}$ (by property 3 of Φ-norm-reductions), $N[I, J] \in AS(P \cup F)$.

$I \cup \overline{(X \setminus I)} \subset F \subseteq N[I, J]$ for some $I \subseteq X$ and $J \subseteq Y$ such that $I \cup J \models \phi$: In this case $N[I, J]$ is a model of $P \cup F$ and, by property 3 of Φ-norm-reductions, $N[I, J]$ is also the unique model of $P^{N[I,J]} \cup F = (P \cup F)^{N[I,J]}$.

$I \cup \overline{(X \setminus I)} \subset F \subseteq N[I, J]$ for some $I \subseteq X$ and $J \subseteq Y$ such that $I \cup J \not\models \phi$: We shall show that $O[J]$ is an answer set of $P \cup F$ in this case. Let M be a model of $P^{O[J]} \cup F = (P \cup F)^{O[J]}$. Since $I \cup \overline{(X \setminus I)} \subset F \subseteq N[I, J]$, either $w \in F$ or $(Y \cup \overline{Y}) \cap F \neq \emptyset$. Clearly, $F \subseteq M$ and so $w \in M$ in the first case. Note that $w \in M$ holds also in the second case because of property 4(c) of Φ-norm-reductions. Thus, as a consequence of properties 4(a) and 4(b) of Φ-norm-reductions, $J \cup \overline{(Y \setminus J)} \subseteq M$ holds. Since $I \cup J \not\models \phi$ and because of property 4(d) of Φ-norm-reductions, $v \in M$ holds. Finally, since $\{v, w\} \subseteq M$ and because of property 4(e) of Φ-norm-reductions, $X \cup \overline{X} \subseteq M$ holds and, thus, $M = O[I]$.

In all other cases, either $\{v, w\} \subseteq F$, or there is an $x \in X$ such that $\{x, \overline{x}\} \subseteq F$, or there is a $y \in Y$ such that $\{y, \overline{y}\} \subseteq F$. We shall show that in such cases there is an $O[J^*]$ which is an answer set of $P \cup F$. Let $O[J^*]$ be such that $J^* = F \cap (Y \cup \overline{Y})$ and let M be a model of $P^{O[J^*]} \cup F = (P \cup F)^{O[J^*]}$ such that $M \subseteq O[J^*]$. We shall show that $O[J^*] \subseteq M$ holds, which would imply that $O[J^*] = M$ is an answer set of $P \cup F$. Clearly, $F \subseteq M$ holds. By property 4(e) of Φ-norm-reductions, $X \cup \overline{X} \cup \{v, w\} \subseteq M$ holds. Thus, by property 4(a) of Φ-norm-reductions and because $w \in M$, it holds that $y \in M$ for each $y \in Y$ such that $y \in O[J^*]$ and $\overline{y} \notin O[J^*]$. Similarly, by property 4(b) of Φ-norm-reductions and because $w \in M$, it holds that $\overline{y} \in M$ for each $y \in Y$ such that $\overline{y} \in O[J^*]$ and $y \notin O[J^*]$. Moreover, for all $y \in Y$ such that $\{y, \overline{y}\} \subseteq O[J^*]$, it holds that $\{y, \overline{y}\} \subseteq F \subseteq M$. Therefore, $O[J^*] \subseteq M$ holds and, consequently, $O[J^*] \in AS(P \cup F)$.

So in each of these cases $AS(P \cup F) \neq \emptyset$ and since these cases cover all possible $F \subseteq U$, we obtain that P is supercoherent.

Summarizing, we have shown that Φ being false implies that any Φ-norm-reduction P is not super-coherent, while Φ being true implies that any Φ-norm-reduction is super-coherent, which proves the lemma.

□

Definition 5. *For any QBF $\Phi = \forall X \exists Y \phi$ with $\phi = \bigwedge_{i=1}^{n} l_{i,1} \vee \cdots \vee l_{i,m_i}$ in CNF, we define*

$$N_\Phi = \{x \leftarrow not\ \overline{x};\ \overline{x} \leftarrow not\ x \mid x \in X\} \cup \tag{6}$$

$$\{y \leftarrow not\ \overline{y}, w;\ \overline{y} \leftarrow not\ y, w;\ w \leftarrow y;\ w \leftarrow \overline{y} \mid y \in Y\} \cup \tag{7}$$

$$\{z \leftarrow v, w;\ z \leftarrow x, \overline{x};\ z \leftarrow y, \overline{y} \mid z \in X \cup \overline{X} \cup \{v, w\}, x \in X, y \in Y\} \cup \tag{8}$$

$$\{v \leftarrow \bar{l}_{i,1}, \ldots, \bar{l}_{i,m_i} \mid 1 \leq i \leq n\} \cup \tag{9}$$

$$\{w \leftarrow not\ v\}. \tag{10}$$

Again, the program from the above definition can be constructed in polynomial time in the size of the reduced QBF. To conclude the proof, it is thus sufficient to show the following relation.

Lemma 4. *For any QBF $\Phi = \forall X \exists Y \phi$ with ϕ in CNF, the program N_Φ is a Φ-norm-reduction.*

Proof. We shall first show that N_Φ has the requested models. Let M be a model of N_Φ. Let us consider the atoms v and w. Because of the rule $w \leftarrow not\ v$ in (10), three cases are possible:

1. $\{v, w\} \subseteq M$. Thus, $X \cup \overline{X} \subseteq M$ holds because of (8). Moreover, there exists $J \subseteq Y$ such that $J \cup \overline{(Y \setminus J)} \subseteq M$ because of (7). Note that any other atom in U could belong to M. These are the models $O[J^*]$.
2. $v \in M$ and $w \notin M$. Thus, there exists $I \subseteq X$ such that $I \cup \overline{(X \setminus I)} \subseteq M$ because of (6). Moreover, no atoms in $Y \cup \overline{Y}$ belong to M because of (7) and $w \notin M$ by assumption. Thus, $M = M[I]$ in this case.
3. $v \notin M$ and $w \in M$. Thus, there exist $I \subseteq X$ and $J \subseteq Y$ such that $I \cup \overline{(X \setminus I)} \subseteq M$ and $J \cup \overline{(Y \setminus J)} \subseteq M$ because of (6) and (7). Hence, in this case $M = N[I, J]$ and, because of (9), it holds that $I \cup J \models \phi$.

Let us consider a reduct $P^{M[I]}$ and one of its models $M \subseteq M[I]$. First of all, note that $P^{M[I]}$ contains a fact for each atom in $I \cup \overline{(X \setminus I)}$. Thus, $I \cup \overline{(X \setminus I)} \subseteq M$ holds. Note also that, since each clause of ϕ contains at least one variable from Y, all of the rules of (9) have at least one false body literal. Hence, either $M = M[I]$ or $M = M[I] \setminus \{v\}$, as required by Φ-norm-reductions.

For a reduct $P^{N[I,J]}$ such that $I \cup J \models \phi$ it is enough to note that $P^{N[I,J]}$ contains a fact for each atom of $N[I, J]$.

Let us consider a reduct $P^{O[J^*]}$ and one of its models $M \subseteq O[J^*]$. The first observation is that for each $y \in Y$ such that $y \in O[J^*]$ and $\overline{y} \notin O[J^*]$, the reduct $P^{O[J^*]}$ contains a rule of the form $y \leftarrow w$ (obtained by some rule in (7)). Similarly, for each

$y \in Y$ such that $\overline{y} \in O[J^*]$ and $y \notin O[J^*]$, the reduct $P^{O[J^*]}$ contains a rule of the form $\overline{y} \leftarrow w$ (obtained by some rule in (7)). Hence, M must satisfy properties 4(a) and 4(b) of Φ-norm-reductions. Property 4(c) is a consequence of (7), property 4(d) follows from (9) and, finally, property 4(e) must hold because of (8). □

Note that the program from Definition 5 does not contain constraints. As a consequence, the Π_2^P-hardness result presented in this section also holds if we only consider normal ASP programs without constraints.

6 Some Implications

In [22] the following problem has been studied under the name "uniform equivalence with projection:"

> Given two programs P and Q, and two sets A, B of atoms, $P \equiv_B^A Q$ iff for each set $F \subseteq A$ of facts, $\{I \cap B \mid I \in AS(P \cup F)\} = \{I \cap B \mid I \in AS(Q \cup F)\}$.

Let us call A the context alphabet and B the projection alphabet. As is easily verified the following relation holds.

Proposition 3. *A program P over atoms U is super-coherent iff $P \equiv_\emptyset^U Q$, where Q is an arbitrary definite Horn program.*

Note that $P \equiv_\emptyset^U Q$ means $\{I \cap \emptyset \mid I \in AS(P \cup F)\} = \{I \cap \emptyset \mid I \in AS(Q \cup F)\}$ for all sets $F \subseteq U$. Now observe that for any $F \subseteq U$, both of these sets are either empty or contain the empty set, depending on whether the programs (extended by F) have answer sets.

$$\{I \cap \emptyset \mid I \in AS(P \cup F)\} = \begin{cases} \emptyset & \text{iff } AS(P \cup F) = \emptyset \\ \{\emptyset\} & \text{iff } AS(P \cup F) \neq \emptyset \end{cases}$$

$$\{I \cap \emptyset \mid I \in AS(Q \cup F)\} = \begin{cases} \emptyset & \text{iff } AS(Q \cup F) = \emptyset \\ \{\emptyset\} & \text{iff } AS(Q \cup F) \neq \emptyset \end{cases}$$

If Q is a definite Horn program, then $AS(Q \cup F) \neq \emptyset$ for all $F \subseteq U$, and therefore the statement of Proposition 3 becomes equivalent to checking whether $AS(P \cup F) \neq \emptyset$ for all $F \subseteq U$, and thus whether P is super-coherent.

In [22], the complexity of the problem of deciding uniform equivalence with projection has also been investigated, reporting Π_3^P-completeness for disjunctive programs and Π_2^P-completeness for normal programs. However, these hardness results use bound context alphabets $A \subset U$ (where U are all atoms from the compared programs). Our results thus strengthen the observations in [22]. Using Proposition 3 and the main results in this paper, we obtain Π_3^P-hardness (resp. Π_2^P-hardness in the case of normal programs) for uniform equivalence with projection even for the particular parameterization where the context alphabet is unrestricted, the projection set is empty, and one of the compared programs are of a very simple structure (in fact, even the empty program is sufficient for Q in Proposition 3).

7 Conclusion

Many recent advances in ASP rely on the adaptions of technologies from other areas. One important example is the Magic Set method, which stems from the area of databases and is used in state-of-the-art ASP grounders. Recent work showed that a particular variant of this technique only applies to a certain class of programs called super-coherent [2]. Super-coherent programs are those which possess at least one answer set, no matter which set of facts is added to them. We believe that this class of programs is interesting of its own (for instance, since there is a certain relation to some problems in equivalence checking) and thus studied here the exact complexity of recognizing the property of super-coherence for disjunctive and normal programs. Our results show that the problems are surprisingly hard, viz. complete for Π_3^P and resp. Π_2^P. One direction of future work is to search for methods to turn arbitrary programs into super-coherent ones with minimal changes. Our proofs might provide valuable foundations for such methods. That said, for using super-coherent programs efficiently for applications, we believe that an approach that uses a methodology different from the currently prevailing "Guess&Check" or "Generate&Test" should be developed.

References

1. Alviano, M., Faber, W.: Dynamic magic sets for super-consistent answer set programs. In: Balduccini, M., Woltran, S. (eds.) Proceedings of the 3rd Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP 2010) (Jul 2010)
2. Alviano, M., Faber, W.: Dynamic magic sets and super-coherent answer set programs. AI Communications 24(2), 125–145 (2011)
3. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Tech. Rep. 09/2009, Dipartimento di Matematica, Università della Calabria, Italy (2009), `http://www.wfaber.com/research/papers/TRMAT092009.pdf`
4. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic Sets and Other Strange Ways to Implement Logic Programs. In: Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems. pp. 1–15. Cambridge, Massachusetts (1986)
5. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
6. Beeri, C., Ramakrishnan, R.: On the power of magic. Journal of Logic Programming 10(1–4), 255–259 (1991)
7. Cumbo, C., Faber, W., Greco, G., Leone, N.: Enhancing the magic-set method for disjunctive datalog programs. In: Proceedings of the the 20th International Conference on Logic Programming – ICLP'04. LNCS, vol. 3132, pp. 371–385 (2004)
8. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In: Brewka, G., Lang, J. (eds.) Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008). pp. 422–432. AAAI Press, Sydney, Australia (2008)
9. Eiter, T., Fink, M., Tompits, H., Woltran, S.: On eliminating disjunctions in stable logic programming. In: Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR 2004). pp. 447–458. AAAI Press (2004)
10. Eiter, T., Fink, M., Woltran, S.: Semantical Characterizations and Complexity of Equivalences in Stable Logic Programming. ACM Transactions on Computational Logic 8(3), 1–53 (2007)

11. Eiter, T., Gottlob, G.: On the computational cost of disjunctive logic programming: Propositional case. Annals of Mathematics and Artificial Intelligence 15(3-4), 289–323 (1995)
12. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Transactions on Database Systems 22(3), 364–418 (Sep 1997)
13. Eiter, T., Tompits, H., Woltran, S.: On Solution Correspondences in Answer Set Programming. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05). pp. 97–102. Professional Book Center (2005)
14. Greco, S.: Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. IEEE Transactions on Knowledge and Data Engineering 15(2), 368–385 (March/April 2003)
15. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM Transactions on Computational Logic 7(1), 1–37 (Jan 2006)
16. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005). pp. 915–917. ACM Press, Baltimore, Maryland, USA (Jun 2005)
17. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (Jul 2006)
18. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) Logic Programming and Nonmonotonic Reasoning — 8th International Conference, LPNMR'05, Diamante, Italy, September 2005, Proceedings. LNCS, vol. 3662, pp. 447–451. Springer Verlag (Sep 2005)
19. Lifschitz, V., Turner, H.: Splitting a Logic Program. In: Van Hentenryck, P. (ed.) Proceedings of the 11th International Conference on Logic Programming (ICLP'94). pp. 23–37. MIT Press, Santa Margherita Ligure, Italy (Jun 1994)
20. Manna, M., Ruffolo, M., Oro, E., Alviano, M., Leone, N.: The HiLeX System for Semantic Information Extraction. Transactions on Large-Scale Data and Knowledge-Centered Systems (2011), to appear
21. Manna, M., Scarcello, F., Leone, N.: On the complexity of regular-grammars with integer attributes. Journal of Computer and System Sciences (JCSS) 77(2), 393–421 (2011)
22. Oetsch, J., Tompits, H., Woltran, S.: Facts do not cease to exist because they are ignored: Relativised uniform equivalence with answer-set projection. In: Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07). pp. 458–464. AAAI Press (2007)
23. Papadimitriou, C.H., Yannakakis, M.: Tie-breaking semantics and structural totality. Journal of Computer and System Sciences 54(1), 48–60 (1997)
24. Ricca, F., Alviano, M., Dimasi, A., Grasso, G., Ielpa, S.M., Iiritano, S., Manna, M., Leone, N.: A Logic-Based System for e-Tourism. Fundamenta Informaticae 105(1–2), 35–55 (2010)
25. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the Gioia-Tauro seaport. Theory and Practice of Logic Programming (2011), to appear, doi:10.1017/S147106841100007X
26. Ullman, J.D.: Principles of Database and Knowledge Base Systems. Computer Science Press (1989)

Verifying Compliance of Business Processes with Temporal Answer Sets

Davide D'Aprile[1], Laura Giordano[1], Valentina Gliozzi[2], Alberto Martelli[2], Gian Luca Pozzato[2], and Daniele Theseider Dupré[1]

[1] Dipartimento di Informatica, Università del Piemonte Orientale
{davide.daprile,laura.giordano,dtd}@mfn.unipmn.it
[2] Dipartimento di Informatica, Università di Torino
{gliozzi,mrt,pozzato}@di.unito.it

Abstract. In this paper we provide a framework for the specification and the verification of business processes, which is based on a temporal extension of answer set programming (ASP) and we address the problem of verifying the compliance of business processes to norms. The logical formalism we use, is a combination of Answer Set Programming and Dynamic Linear Time Temporal Logic (DLTL), and allows for a declarative specification of a business process, as well as the specification of norms, by means of a set of temporal rules and a set of temporal constraints. A notion of commitment, borrowed from the social approach to agent communication, is used to capture obligations within norms. Besides allowing for a declarative specification of business processes, the proposed framework can be used for encoding business processes specified in conventional workflow languages. The verification of temporal properties of a business process, expressed by temporal formulas, can be done by encoding bounded model checking techniques in ASP. Verifying compliance of a business process to norms consists, in particular, in verifying that there are no executions of the business process which leave commitments unfulfilled.

1 Introduction

The problem of verifying the compliance of business processes has attracted a lot of interest in recent years. Many organizations (banks, hospitals, public administrations, etc.), whose activities are subject to regulations, are required to justify their behaviors with respect to the norms and to show that the business procedures they adopt conform to such norms. For instance, in the financial domain, the Sarbanes-Oxley Act (commonly named SOX), enacted in 2002 in the USA, describes mandates and requirements for financial reporting, and was proposed in order to restore investors' confidence in capital markets after major accounting scandals. MiFID (Markets in Financial Instruments Directive) is a EU law, effective from 2007, with similar goals, including transparency.

In this paper, we address the problem of automatic verification of business process compliance and, to this purpose, we propose a framework for the specification and the verification of business processes, which is based on a temporal

extension of answer set programming (ASP [12]). The choice of a logical formalism for the specification of business processes has the advantage of allowing a declarative specification of business processes and web services, which has been advocated in recent literature [29, 27, 24], as opposed to the more rigid transition based approach. A declarative specification of a process is, generally, more concise than transition based specification as it abstracts away form rigid control-flow details and does not require the order among the actions in the process to be rigidly defined. A further advantage of logical formalisms is that the computational mechanisms of the underlying logic can then be exploited in the verification of business process properties.

The framework for the specification and the verification of business processes proposed in this paper, is based on a temporal extension of Answer Set Programming (ASP [12]) defined in [17], which combines Answer Set Programming with a temporal logic, namely Dynamic Linear Time Temporal Logic (DLTL) [22]. DLTL extends propositional temporal logic of linear time (LTL) with regular programs of propositional dynamic logic, that are used for indexing temporal modalities. The language in [17] is a temporal action language, which allows for the specification of atomic actions by means of their effects and preconditions, as well as for the specification of complex actions and general temporal constraints to specify the wanted interactions among the tasks (and, under this respect, our approach to business process specification has similarities with that of *DecSerFlow* [29]). Besides allowing for a declarative specification of business processes, this language is well suited for encoding processes specified workflow languages.

As concerns compliance verification, to represent the obligations which can be enforced by the application of norms, we make use of the notion of *commitment*, which is borrowed from the area of multi-agent communication [27, 11, 20, 16, 5]. We show that the temporal language is well suited to model norms as directional rules which generate commitments, and that defeasible negation of ASP can be exploited to capture exceptions to the norms, by allowing norms to be defeasible. Given the specification of a business process in temporal ASP and the specification of norms as a set of (defeasible) causal laws generating obligations (commitments), the problem of compliance verification can be given a logical characterization. It consists in verifying that there are no executions of the business process which leave some commitment unfulfilled.

For the verification of the business process properties and, in particular, for compliance verification, we exploit Bounded Model Checking [6]. In particular, we exploit an approach developed in [17] for DLTL bounded model checking in ASP, which extends the approach for Bounded LTL Model Checking with Stable Models that has been developed in [21].

2 A temporal extension of ASP

We shortly recall the temporal action language introduced in [17], which is based on a temporal extension of Answer Set Programming (ASP).

2.1 Temporal action language

The action language is based on the Dynamic Linear Time Temporal Logic (DLTL) [22]. DLTL extends LTL by allowing the *until* operator $\mathcal{U}^\pi$ to be indexed by a program π, an expression of Propositional Dynamic Logic: π can be any regular expression built from atomic actions using sequence (;), non-deterministic choice (+) and finite iteration ($*$). The usual LTL modalities $\Box$ (always), $\Diamond$ (eventually), $\bigcirc$ (next), and $\mathcal{U}$ (until) can be defined from $\mathcal{U}^\pi$ as well as the new temporal modalities $[\pi]$ and $\langle\pi\rangle$.

Informally, a formula $[\pi]\alpha$ is true in a world w of a linear temporal model if α holds in all the worlds of the model which are reachable from w through any execution of the program π. A formula $\langle\pi\rangle\alpha$ is true in a world w of a linear temporal model if there exists a world of the model reachable from w through an execution of the program π, in which α holds. A formula $\alpha\,\mathcal{U}^\pi\,\beta$ is true at a world w if "α until β" is true at w on a finite stretch of behavior which is in the linear time behavior of the program π.

A domain description is defined as a set of laws describing the effects of actions as well as their executability preconditions. Atomic propositions describing the state of the domain are called *fluents*. Actions may have direct effects, that are described by action laws, and indirect effects, that capture the causal dependencies among fluents and are described by causal laws. The execution of an action a in a state s leads to a new state s' in which the effect of the action holds. The fluents which hold in s and are not affected by the action a, still hold in s'.

Let $\mathcal{P}$ be a set of atomic propositions, the *fluents*, including the inconsistency, $\bot$. A *simple fluent literal* l is a fluent name f or its negation $\neg f$. We will denote by Lit_S the set of all simple fluent literals. In the language, we also make use of *temporal literals*, that is, literals that are prefixed by the temporal modalities $[a]$, with $a \in \Sigma$, and $\bigcirc$. Their intended meaning is: $[a]l$ holds in a state s if l holds in the state obtained after the execution of action a in s; $\bigcirc l$ holds in a state s if l holds in the state next to s. Lit_T is the set of *temporal fluent literals*: if $l \in Lit_S$, then $[a]l, \bigcirc l \in Lit_T$, where a is an action name (an atomic proposition, possibly containing variables), and $[a]$ and $\bigcirc$ are the temporal operators introduced in the previous section. Let $Lit = Lit_S \cup Lit_T \cup \{\bot\}$. Given a (simple or temporal) fluent literal l, *not* l represents the default negation of l. A (simple or temporal) fluent literal possibly preceded by a default negation, will be called an *extended fluent literal*.

The laws are formulated as rules of a temporally extended logic programming language. Rules have the form

$$\Box(t'_0 \; or \ldots or \; t'_h \leftarrow t_1, \ldots, t_m, not \; t_{m+1}, \ldots, not \; t_n) \tag{1}$$

or the form

$$t'_0 \; or \ldots or \; t'_h \leftarrow t_1, \ldots, t_m, not \; t_{m+1}, \ldots, not \; t_n \tag{2}$$

where the t'_i's and the t_i's are either simple fluent literals or temporal fluent literals. While laws of the form (1) can be applied in all states, laws of the form

(2) can only be applied in the initial state. Action laws, causal laws, precondition laws, persistency laws, initial state laws, etc., which are normally used in action theories, can all be defined as instances of (1) and (2).

A *domain description* D is defined as a tuple $(\Pi, \mathcal{C})$, where Π is a set of laws of the form 1 and 2 and $\mathcal{C}$ is a set of *temporal constraints*, i.e. general DLTL formulas.

2.2 Temporal Answer Sets

To define the the semantics of a domain description, we extend the notion of *answer set* [12] to capture the linear structure of temporal models. In the following, we consider the ground instantiation of the domain description Π, and we denote by Σ the set of all the ground instances of the action names in Π.

We define a *(partial) temporal interpretation* as a pair (σ, S), where $\sigma \in \Sigma^\omega$ is a sequence of actions and S is a consistent set of literals of the form $[a_1; \ldots; a_k]l$, where $a_1 \ldots a_k$ is a prefix of σ, meaning that l holds in the state obtained by executing $a_1 \ldots a_k$. S is *consistent* iff it is not the case that both $[a_1; \ldots; a_k]l \in S$ and $[a_1; \ldots; a_k]\neg l \in S$, for some l, or $[a_1; \ldots; a_k]\bot \in S$. A temporal interpretation (σ, S) is said to be *total* if either $[a_1; \ldots; a_k]p \in S$ or $[a_1; \ldots; a_k]\neg p \in S$, for each $a_1 \ldots a_k$ prefix of σ and for each fluent name p.

We define the *satisfiability of a simple, temporal or extended literal t in a partial temporal interpretation (σ, S) in the state $a_1 \ldots a_k$*, (written $S, a_1 \ldots a_k \models t$) as follows:

$(\sigma, S), a_1 \ldots a_k \models \top, \qquad (\sigma, S), a_1 \ldots a_k \not\models \bot$

$(\sigma, S), a_1 \ldots a_k \models l$ *iff* $[a_1; \ldots; a_k]l \in S$, for a literal l

$(\sigma, S), a_1 \ldots a_k \models [a]l$ *iff* $[a_1; \ldots; a_k; a]l \in S$ or $a_1 \ldots a_k, a$ is not a prefix of σ

$(\sigma, S), a_1 \ldots a_k \models \bigcirc l$ *iff* $[a_1; \ldots; a_k; b]l \in S$, where $a_1 \ldots a_k b$ is a prefix of σ

$(\sigma, S), a_1 \ldots a_k \models \mathit{not}\ l$ *iff* $(\sigma, S), a_1 \ldots a_k \not\models l$

Observe that $[a]l$ is true in any state of a linear model in which a is not the next action to be executed. The satisfiability of rule bodies and rule heads in a temporal interpretation are defined as usual. A rule $\Box(H \leftarrow \mathit{Body})$ is satisfied in a temporal interpretation (σ, S) if, for all action sequences $a_1 \ldots a_k$ (including the empty one), $(\sigma, S), a_1 \ldots a_k \models \mathit{Body}$ implies $(\sigma, S), a_1 \ldots a_k \models H$.

A rule $H \leftarrow \mathit{Body}$ is satisfied in a partial temporal interpretation (σ, S) if, $(\sigma, S), \varepsilon \models \mathit{Body}$ implies $(\sigma, S), \varepsilon \models H$, where ε is the empty action sequence.

To define the answer sets of Π, we introduce the notion of *reduct* of Π, containing rules of the form: $[a_1; \ldots; a_h](H \leftarrow \mathit{Body})$. Such rules are evaluated in the state $a_1 \ldots a_h$.

Let Π be a set of rules over an action alphabet Σ, not containing default negation, and let $\sigma \in \Sigma^\omega$.

Definition 1. *A temporal interpretation (σ, S) is a* temporal answer set of Π *if S is minimal (in the sense of set inclusion) among the S' such that (σ, S') is a partial interpretation satisfying the rules in Π.*

To define answer sets of a program Π containing negation, given a temporal interpretation (σ, S) over $\sigma \in \Sigma^\omega$, we define the *reduct, $\Pi^{(\sigma,S)}$, of Π relative to* (σ, S) extending Gelfond and Lifschitz' transform [13] to compute a different reduct of Π for each prefix $a_1, \ldots, a_h$ of σ.

Definition 2. *The* reduct, $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$, of Π relative to (σ, S) and to the prefix $a_1, \ldots, a_h$ of σ *, is the set of all the rules*

$$[a_1; \ldots; a_h](H \leftarrow l_1, \ldots, l_m)$$

such that $H \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n$ is in Π and $(\sigma, S), a_1, \ldots, a_h \not\models l_i$, for all $i = m+1, \ldots, n$.

The reduct $\Pi^{(\sigma,S)}$ of Π relative to (σ, S) *is the union of all reducts $\Pi^{(\sigma,S)}_{a_1,\ldots,a_h}$ for all prefixes $a_1, \ldots, a_h$ of σ.*

Definition 3. *A temporal interpretation (σ, S) is an* answer set of Π *if (σ, S) is an answer set of the reduct $\Pi^{(\sigma,S)}$.*

Although the answer sets of a domain description Π are partial interpretations, in some cases, e.g., when the initial state is complete and all fluents are inertial, it is possible to guarantee that the temporal answer sets of Π are total.

In case the initial state is not complete,we consider all the possible ways to complete the initial state by introducing in Π, for each fluent name f, the rules: $f \leftarrow not\ \neg f$ and $\neg f \leftarrow not\ f$. The case of total temporal answer sets is of special interest as a total temporal answer set (σ, S) can be regarded as temporal model (σ, V), where, for each finite prefix $a_1 \ldots a_k$ of σ, $V(a_1, \ldots, a_k) = \{p : [a_1; \ldots; a_k]p \in S\}$. In the following, we restrict our consideration to domain descriptions Π, such that all the answer sets of Π are total.

The notion of *extension* of a domain description $D = (\Pi, \mathcal{C})$ over Σ is defined in two steps: first, the answer sets of Π are computed; second, all the answer sets which do not satisfy the temporal constraints in $\mathcal{C}$ are filtered out. For the second step, we need to define when a temporal formula α is satisfied in a total temporal interpretation S. Observe that a total answer set S over σ can be regarded as a linear temporal (DLTL) model [22]. Given a total answer set S over σ we define the corresponding temporal model as $M_S = (\sigma, V_S)$, where $p \in V_S(a_1, \ldots, a_h)$ if and only if $[a_1; \ldots; a_h]p \in S$, for all atomic propositions p. We say that a total answer set S over σ satisfies a DLTL formula α if $M_S, \varepsilon \models \alpha$.

Definition 4. *An* extension of a domain description $D = (\Pi, \mathcal{C})$ over Σ*, is any (total) answer set S of Π satisfying the constraints in $\mathcal{C}$.*

3 Specifying the business process and the norms

Let us consider a business process of an investment firm, where the firm offers financial instruments to an investor. The description of the business processes given in Figure 1 in the language YAWL (Yet AnotherWorkflow Language) [28].

Also, let us consider a regulation containing the following norms:

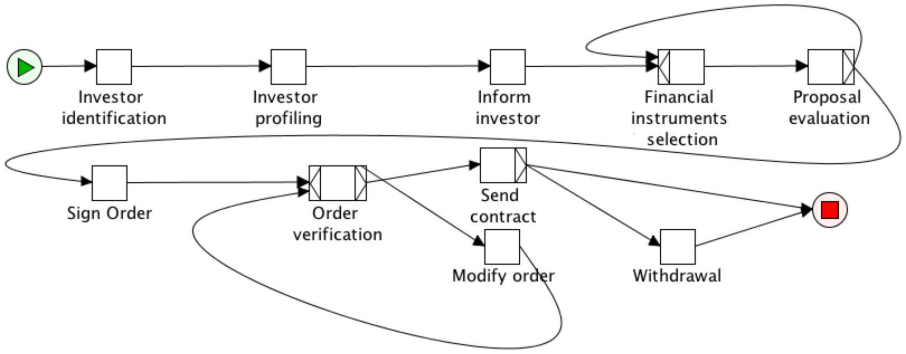


Fig. 1. Example business process in YAWL

(1) the firm shall provide to the investor adequate information on its services and policies before any contract is signed;
(2) if the investor signs an order, the firm is obliged to provide him a copy of the contract.

The execution of each task in the process has some preconditions and effects. Due to the presence of norms, the execution of a task in the process above should generate obligations to be fulfilled. For instance, according to the second norm, signing an order generates for the firm the obligation to provide a copy of the contract to the investor. Verifying the compliance of a business process to a regulation requires to check that, in all the executions of the business process, the obligations triggered by the norms are fulfilled.

In the following, we provide the specification of the business process and of the related norms in an action theory. The problem of verifying compliance of the business process to the norms is then defined as a reasoning problem in the action theory. Concerning the specification of a business process, we distinguish two parts in this specification: the specification of the workflow itself, for which we will exploit the capability of the action language to represent complex actions, and the specification of the atomic tasks occurring in it. The description of the atomic actions occurring in the business process provides the background knowledge, which is common both to the business process and to the norms. The effects and, possibly, the preconditions of the atomic tasks are defined by introducing propositions representing the properties of the world that are affected by the execution of the tasks and that are subject to the norms. They are the properties whose value is to be checked, for verifying the compliance of the process to the norms themselves. Such properties are sometimes used in the literature [14, 19, 30] as annotations that decorate the business process. Here, we exploit the action language to provide a more expressive formalism for formulating properties annotations. In the next subsections, we address the problems of specifying the atomic tasks, specifying the business process control and specifying the norms.

3.1 Semantic annotations: the specification of atomic tasks

The atomic tasks occurring in the business process are those represented by boxes in the YAWL specification. In the action theory, they are modeled as atomic actions with the same name as the atomic tasks.

To introduce the propositions needed to describe the effects and, possibly, the preconditions of atomic tasks, we introduce the following fluent names in $\mathcal{P}$:

$investor(C)$: C has been identified as a client (an investor);
$investor\_classified(C)$: the investor C has been classified;
$risk\_averse(C)$, $risk\_seeking(C)$ represent the profile of the client;
$informed(C)$: the client has been informed about the bank services and policies;
$selected(T, C)$: client C has selected financial instrument T;
$accepted(T, C)$: client C has accepted financial instrument T;
$order\_signed(T, C)$: client C has signed the order of financial instrument T;
$order\_confirmed(T, C)$: the order has been confirmed by the bank;
$sent\_contract(T, C)$: the contract of the order has been sent to the client;
$order\_deleted(T, C)$: the order has been canceled.

In the following, *profiling* stands for *investor profiling*, *inform* stands for *inform investor*, *fi_selection* stands for *financial instrument selection*, *p_eval* stands for *proposal evaluation* and, finally, *order_verif* stands for order verification.

The following action and causal laws describe the effect of the actions in the business process. Although the action language is propositional, in the following, we use variables in fluent names and in atomic action, so that each action/causal law stands for a finite number of ground action/causal laws:

$\Box([investor\_identification(C)]investor\_identified(C)) \leftarrow client(C))$
$\Box([profiling(C)]investor\_classified(C) \leftarrow client(C))$
$\Box([profiling(C)]risk\_averse(C)$ *or* $[profiling(C)]risk\_seeking(C)$
$\leftarrow client(C))$
$\Box([inform(C)]informed(C) \leftarrow client(C))$
$\Box([fi\_selection(C)]selected(t_1, C)$ *or* $\ldots$ *or* $[fi\_selection(C)]selected(t_n, C)$
$\leftarrow financial\_instr(t_1) \wedge \ldots \wedge financial\_instr(t_n) \wedge risk\_averse(C))$
$\Box([p\_eval(T, C)]accepted(T, C)$ *or* $[p\_eval(T, C)]\neg accepted(T, C))$
$\Box([sign\_order(T, C)]order\_signed(T, C) \leftarrow order(T), client(C))$
$\Box([order\_verif(T, C)]confirmed(T, C)$ *or* $[order\_verif(T, C)]\neg confirmed(T, C)$
$\leftarrow order(T), client(C))$
$\Box([send\_contract(T, C)]sent\_contract(T, C) \leftarrow order(T), client(C))$
$\Box([withdraw(T, C)]order\_deleted(T, C)) \leftarrow order(T), client(C))$
$\Box(\neg confirmed(T, C) \leftarrow order\_deleted(T, C))$
$\Box([end\_procedure]end)$

Laws defining executability conditions for atomic tasks can also be given. For instance,

$\Box([send\_contract(T, C)]\bot \leftarrow not\ confirmed(T, C))$

states that it is possible to send a contract to the investor only if the contract has been confirmed. Such temporal formulae can be seen as "good properties", that the modeler would like to verify on the business process and they can be verified to hold in all possible executions of the business process with the same technique that will be introduced for verifying norm compliance. Indeed, some of the norms will be formalized as precondition laws.

3.2 The specification of the business process workflow

When we are faced with the problem of specifying a business process, even in a given language as the one introduced in section 2.2, many options are available.

In [9], we have shown that the control flow of a business process can be modeled in a rigid way by means of a program expression π, i.e. by defining a complex action using composition operators like sequence, non deterministic choice and finite iteration, as well as test actions $p?$ which can be suitably introduced in the language. Then, a temporal constraint $\langle\pi\rangle\top$ is introduced in the set of constraints $\mathcal{C}$ to select those extensions of the domain description, corresponding to the possible executions of the program π.

Although this is a very simple solution, in the general case, the workflow of a business process may be non-structured or, even, we may want to provide a declarative specification of the business process, as done, for instance, in the declarative flow language ConDec [25]. It must be observed that the logical nature of our action language makes it well suited for a declarative specification. Indeed, the presence of general DLTL constraints in action domains allows for a simple way to constrain activities in a business process. As DLTL is an extension of LTL, it is possible to provide an encoding of ConDec constraints our action language.

Besides allowing for a declarative specification of business processes, the language introduced in Section 2 is well suited for encoding processes specified in conventional workflow languages, through the specification of action effects, preconditions and constraints. Consider, for instance, the atomic task *Investor profiling*. We can model the fact that this task can be executed only if the atomic task *Investor identification* has been executed, by introducing the precondition law: $\Box([profiling(C)]\bot \leftarrow not\ investor\_identified(C))$. Moreover, the fact that *Investor profiling* has to be executed after *Investor identification* is executed can be modeled by the temporal constraint:

$$\Box[investor\_identification(C)]\Diamond\langle profiling(C)\rangle\top$$

Consider, in addition, the atomic task *Order verification*. After its execution *Send contract* is to be executed if the order has been confirmed, otherwise, the task *Modify order* has to be executed. This can be modeled introducing the precondition laws

$$\Box([send\_contract(T,C)]\bot \leftarrow \neg confirmed(T,C))$$
$$\Box([modify\_order(T,C)]\bot \leftarrow confirmed(T,C))$$

and temporal constraints

$$\Box[order\_verif(C)](confirmed(T,C) \rightarrow \Diamond\langle send\_contract(T,C)\rangle\top)$$
$$\Box[order\_verif(C)](\neg confirmed(T,C) \rightarrow \Diamond\langle modify\_order(T,C)\rangle\top)$$

This approach can be generalized for translating YAWL processes into a domain description by action effects and preconditions as well as constraints. The translation, in general, would require to introduce new fluent names as, for instance, for each atomic task a, a fluent $executable(a)$, which is made true when the execution of task a is enabled. Some technicalities (that we do not address here) are needed to model AND/OR splits and AND/OR joins.

The approach we adopt in this paper for reasoning about actions is well suited for reasoning about systems with infinite computations (see [17]). To deal with finite computations we introduce a dummy action, which can be repeated infinitely many times after the termination of the process (thus giving rise to an infinite computation). In practice, however, as an optimization of the ASP translation, we can avoid looking for arbitrary models with loops during model checking and restrict to ad hoc computations corresponding to finite traces.

3.3 Normative specification

According to the normative specification, the execution of each task in the business process can trigger some normative positions (obligations, permissions, prohibitions). For instance, the *identification* task in the business process above, which introduces a new investor C, also generates the obligation to inform the investor. This obligation must be fulfilled during the course of execution of the business process, if the process is compliant with the norm stating that the firm has the obligation to inform customers.

In the following we make use of causal laws to represent norms in the action theory, and we introduce a notion of commitment to model obligations. The use of commitments has long been recognized as a "key notion" to allow coordination and communication in multi-agent systems [27, 20, 11]. A notion of commitment for reasoning about agent protocols in a temporal action logic has been adopted in [16]. Following [16], we introduce two kinds of commitments (which are regarded as special fluent propositions): *base-level commitments* having the form $\mathrm{C}(i,j,A)$ and meaning that agent i is committed to agent j to bring about A (where A is an arbitrary propositional formula not containing commitment fluents); *conditional commitments* having the form $\mathrm{CC}(i,j,B,A)$ and meaning that agent i is committed to agent j to bring about A, if condition B is brought about.

A base level commitment $\mathrm{C}(i,j,A)$ can be naturally regarded as an obligation (namely, OA, "A is obligatory"), in which the debtor and the creditor are made explicit. The two kinds of base-level and conditional commitments we use here are essentially those introduced in [31]. Our present choice is different from the one in [20], where agents are committed to execute an action rather than to achieve a condition.

The idea is that commitments (or obligations) are created as effects of the execution of some basic tasks in the business process and they are "discharged" when they have been fulfilled. A commitment $\mathrm{C}(i, j, A)$, created at a given state of a run of the process, is regarded to be *fulfilled* in the run if there is a later state of the run in which A holds. As soon as a commitment is fulfilled in a run, it is considered to be satisfied and no longer active: it can be discharged.

Given the notion of commitment introduced above, the norms which generate obligations to be fulfilled can be modeled as causal laws which trigger new commitments/obligations. Other norms which define preconditions on the executability of some actions or, in general, ordering constraints on the executions of atomic tasks can be encoded by general temporal formulas. For instance, we can encode the norms (1) and (2) above by the following precondition and causal laws:

$$\Box([sign\_order(T, C)]\bot \leftarrow not\ informed(C))$$
$$\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$$

The first one is a precondition for $sign\_order(T, C)$, stating that, if the client has not been informed, he cannot sign an order. The second one, a causal law, states that when an order is signed by C, the firm is committed to C to send her the information required.

Causal laws are needed for modeling the interplay of commitments and fluent changes. In particular, for each commitment $\mathrm{C}(i, j, \alpha)$, we introduce the following dynamic causal laws in the domain description:

(i) $\Box(\bigcirc\neg\mathrm{C}(i, j, \alpha) \leftarrow \mathrm{C}(i, j, \alpha), \bigcirc\alpha)$
(ii) $\Box(\bigcirc\mathrm{C}(i, j, \alpha) \leftarrow \mathrm{CC}(i, j, \beta, \alpha), \bigcirc\beta)$
(iii) $\Box(\bigcirc\neg\mathrm{CC}(i, j, \beta, \alpha) \leftarrow \mathrm{CC}(i, j, \beta, \alpha), \bigcirc\beta)$

A commitment to bring about α is considered fulfilled and is discharged as soon as α holds (i). A conditional commitment $\mathrm{CC}(i, j, \beta, \alpha)$ becomes a base-level commitment $\mathrm{C}(i, j, \alpha)$ when β has been brought about (ii) and, in that case, the conditional commitment is discharged (iii).

One of the central issues in the representation of norms comes from the defeasible nature of norms. Norms may have exceptions: recent norms may cancel older ones; more specific norms override more general norms and, in other cases, explicit priority information (not necessarily related to recency or specificity) is needed for eliminating conflicts. Consider the following example from [19]:

r_1: $\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), premium\_customer(M)$
r_2: $\neg\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), special\_order(S, M, O)$

Rule r_1 states that a seller has the obligation to apply a discount to premium customers. Rule r_2 states that customers are not entitled for a discount in case the order (O) is a special order. Observe that, if two rules are regarded as being strict, a state in which the fluents $premium\_customer(M)$, $special\_order(S, M, O)$ and $sells(S, M, O)$ hold results to be inconsistent.

To avoid conflicting situations as the one above, priorities among rules can be incorporated. Suppose the two rules above are regarded as defeasible and assume that rule r_2 has preference over rule r_1 (we write $r_2 > r_1$). The priority between the conflicting norms r_1 and r_2, with $r_2 > r_1$, can be modeled using default negation. For instance, we can transform the rules r_1 and r_2 as follows:

$\Box(\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), premium(M), not\ bl(r_1(S, M, O)))$
$\Box(\neg\mathrm{C}(S, M, O, discount) \leftarrow sells(S, M, O), special\_order(C), not\ bl(r_2(S, M, O)))$
$\Box(bl(r_1(S, M, O)) \leftarrow sells(S, M, O) \wedge special\_order(C), not\ bl(r_2(S, M, O)))$

where $bl(r_i(S, M, O))$ means that rule r_i is blocked. In this way, rule r_2, when applicable, blocks the application of r_1, but not vice-versa.

This treatment of priorities among conflicting rules, in essence, relies on the idea of using abnormality predicates for capturing exceptions. It is not intended to provide a general solution to the problem of modeling priorities among rules, as, in the general case, priorities may be also allowed between non conflicting rules. The problem of dealing with prioritized programs under the answer set semantics has been addressed, for instance, in [7] and in [10] in a more general setting. We believe that the approach proposed in [10] can be exploited in this setting to model defeasible norms as prioritized defeasible causal laws.

A further issue to be addressed when modeling norms is that of formalizing violations and reparation obligations, and we refer to [9] for a possible encoding of reparation chains in our language.

4 Compliance verification by model checking

In this section we provide a characterization of the problem of compliance, as a problem of reasoning about action in the action theory defined above. In Section 3.3, we have devised two different typologies of norms which we may want to verify compliance with: norms which can be encoded as a temporal formula (in the example, a precondition formula) and norms whose application generates obligations to be fulfilled, which can be modeled as causal laws generating commitments. Concerning the first kind of norms, the temporal formula encoding the norm has to be verified to be true in all the extensions of the domain description. Concerning the second kind of norms, verifying the compliance of the business process with such norms amounts to check that, in all the possible extensions of the domain description D, all the commitments generated will be eventually fulfilled, unless they have been cancelled: $\Box(\mathrm{C}(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$. Action *withdraw*, for instance, might have the indirect effect of canceling the commitment to send the contract, if it has not yet been sent. Observe that canceling a commitment would not be possible if the commitment were encoded by the temporal formula $\Diamond\alpha$.

Let $D_B = (\Pi_B, \mathcal{C}_B)$ be a domain description defined as the specification of a given business process B, including the specification of the atomic tasks involved in the process (semantic annotations). Let N be a set of norms, which have been encoded by a set of causal laws Π_N and a set of temporal formulas P_N. The

domain description resulting from the encoding of the business process and the norms can then be defined as $D = (\Pi_B \cup \Pi_N, \mathcal{C}_B)$. We can define the problem of verifying the compliance of a business process to a set of norms as follows:

Definition 5. *The business process B is compliant with the set of norms $N = (\Pi_N, P_N)$ if, for each extension (σ, S) of the domain description $D = (\Pi_B \cup \Pi_N, \mathcal{C}_B)$, the following conditions hold:*

- *for each temporal formula α in P_N, (σ, S) satisfies α;*
- *for each commitment $\mathrm{C}(i, j, \alpha)$ occurring in Π_N, (σ, S) satisfies the formula $\Box(\mathrm{C}(i, j, \alpha) \rightarrow \Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$.*

Dually, the problem of identifying a *violation* to the norms can be regarded as a satisfiability problem: the problem of finding an execution of the business process which violates some of the norms, that is, the problem of finding an extension (σ, S) of D such that either (σ, S) contains unfulfilled commitments, i.e., it satisfies $\Diamond(\mathrm{C}(i, j, \alpha) \wedge \neg\Diamond(\alpha \vee \neg\mathrm{C}(i, j, \alpha)))$, or it falsifies a formula in Π_N.

Consider the domain description D, including the specification D_B of the business problem example and the causal law $\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \leftarrow order\_signed(T, C))$. Each extension S of the domain description satisfies the temporal formulas

$$\Box(\mathrm{C}(firm, C, sent\_contract(T, C)) \rightarrow \Diamond sent\_contract(T, C))$$
$$\Box([sign\_order(T, C)]\bot \leftarrow informed(C))$$

Hence, the business process is compliant with the norms. In fact, in all the execution of the business process, the commitment to send the contract is eventually fulfilled by the execution of the action *send_contract*, which has to be eventually executed in the business process; and, for the second formula, the execution of *sign_order* is always after $inform$ which makes the client informed.

In [17] we exploit bounded model checking (BMC) techniques [6] for computing the extensions of a temporal domain description and for verifying its temporal properties. More precisely, we describe a translation of a temporal domain description into standard ASP, so that the temporal answer sets of the domain description can then be computed as the standard answer sets of its translation. Extensions of the domain description satisfying the temporal constraints or given temporal properties are computed by bounded model checking, following the approach proposed in [17] for the verfication of DLTL formulas, which extends the one developed in [21] for bounded LTL model checking with Stable Models.

As an alternative to encoding the business process control flow in the logical formalisms (as done in section 3.2), a direct encoding of the workflow computations in the ASP program is also feasible, and makes the verification more efficient. In this case, the action language is used only for the specification of the semantic annotations and of the norms. Based on these ideas, we have used bounded model checking in ASP verify business process compliance. The implementation we have developed is based on the DLV system [23].

5 Conclusions and related work

The paper presents an approach to the verification of the compliance of business processes with norms. The approach is based on a temporal extension of ASP. Both the business process, their semantic annotation and the norms are encoded using temporal ASP rules as well as temporal constraints. In particular, defeasible causal laws are used for modeling norms and commitments are introduced for representing obligations. The verification of compliance can be performed by using BMC techniques. In particular, we exploit an approach developed in [17] for DLTL bounded model checking in ASP, which extends the approach for bounded LTL model checking with Stable Models in [21]. We are currently testing our implementation on several workflow examples to verify the scalability of the approach, and to compare with other approaches to compliance verification, including the traditional Petri net approach.

Several proposals in the literature introduce annotations on business processes for dealing with compliance verification [14, 19, 30]. In particular, [19] proposes a logical approach to the problem of business process compliance based on the idea of annotating the business process. Process annotations and normative specifications are provided in the same logical language, namely, the Formal Contract Language (FCL), which combines defeasible logic [3] and deontic logic of violations [18]. Compliance is verified by traversing the graph describing the process and identifying the effects of tasks and the obligations triggered by task execution. Ad hoc algorithms for propagating obligations through the process graph are defined.

In [30] a formal execution semantics for annotated business processes is introduced. The proposed semantics combines a Petri-net like (token passing) semantics for BPMN process execution, coming from the workflow community, with a declarative specification of actions preconditions and effects in clausal form, coming from the AI literature of actions and state changes. Several verification tasks are defined to check whether the business process control flow interacts correctly with the behaviour of the individual activities. However, [30] does not address the problem of verifying compliance of the business process with norms.

An approach to compliance based on a commitment semantics in the context of multi-agent systems is proposed in [8]. The authors formalize notions of conformance, coverage, and interoperability, proving that they are orthogonal to each other. Another approach to the verification of agents compliance with protocols, based on a temporal action theory, has been proposed in [16]. These papers do not address the problem of compliance with norms.

[4] presents an approach to compliance checking for BPMN process models using BPMN-Q, a visual language based on BPMN. Compliance rules are given a declarative representation as BPMN-Q queries. Then, BPMN-Q queries are translated into temporal formulas for verification.

In [24] the Abductive Logic Programming framework SHIFF [2] is exploited in the declarative specification of business processes as well as in the (static and runtime) verification of their properties. In particular, in [1] expectations are

used for modelling obligations and prohibitions and norms are formalized by abductive integrity constraints.

In [26] Concurrent Transaction Logic (CTR) is used to model and reason about general service choreographies. Service choreographies and contract requirements are represented in CTR. The paper addresses the problem of deciding if there is an execution of the service choreography that complies both with the service policies and the client contract requirements.

Temporal rule patterns for regulatory policies are introduced in [15], where regulatory requirements are formalized as sets of compliance rules in a real-time temporal object logic. The approach is used essentially for event monitoring.

Acknowledgments

We want to thank the anonymous referees for their helpful comments. This work has been partially supported by Regione Piemonte, Project "ICT4LAW: *ICT Converging on Law: Next Generation Services for Citizens, Enterprises, Public Administration and Policymakers*"

References

1. M. Alberti, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, and G. Sartor. Mapping of Deontic Operators to Abductive Expectations. *NORMAS*, pages 126–136, 2005.
2. Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Log.*, 9(4), 2008.
3. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Representation results for defeasible logic. *ACM Trans. on Computational Logic*, 2:255–287, 2001.
4. Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using bpmn-q and temporal logic, lncs 5240. In *BPM*, pages 326–341. Springer, 2008.
5. Matteo Baldoni, Cristina Baroglio, and Elisa Marengo. Behavior-oriented commitment-based protocols. In *Proceedings ECAI 2010*, pages 137–142, 2010.
6. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
7. Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
8. A.K. Chopra and M.P. Sing. Producing compliant interactions: Conformance, coverage and interoperability. *DALT IV, LNCS(LNAI) 4327*, pages 1–15, 2006.
9. D. D'Aprile, L. Giordano, V. Gliozzi, A. Martelli, G. L. Pozzato, and D. Theseider Dupré. Verifying business process compliance by reasoning about actions. In *CLIMA XI*, pages 99–116, 2010.
10. James P. Delgrande, Torsten Schaub, and Hans Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 3(2):129–187, 2003.
11. N. Fornara and M. Colombetti. Defining Interaction Protocols using a Commitment-based Agent Communication Language. *AAMAS03*, pages 520–527.
12. M. Gelfond. Answer Sets. *Handbook of Knowledge Representation, chapter 7, Elsevier*, 2007.

13. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proc. of the 5th Int. Conf. and Symposium*, pages 1070–1080, 1988.
14. A. Ghose and G. Koliadis. Auditing business process compliance. *ICSOC, LNCS 4749*, pages 169–180, 2007.
15. C. Giblin, S. Müller, and B. Pfitzmann. From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation. *IBM Reasearch Report*, 2007.
16. L. Giordano, A. Martelli, and C. Schwind. Specifying and Verifying Interaction Protocols in a Temporal Action Logic. *Journal of Applied Logic (Special issue on Logic Based Agent Verification)*, 5:214–234, 2007.
17. L. Giordano, A. Martelli, and D. Theseider Dupré. Reasoning about Actions with Temporal Answer Sets. *Proc. CILC 2010, 25th Italian Conference on Computational Logic*, 2010.
18. G. Governatori and A. Rotolo. Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. *Australasian Journal of Logic*, 4:193–215, 2006.
19. G. Governatori and S. Sadiq. The journey to business process compliance. *Handbook of Research on BPM, IGI Global*, pages 426–454, 2009.
20. F. Guerin and J. Pitt. Verification and Compliance Testing. *Communications in Multiagent Systems, Springer LNAI 2650*, 2003.
21. K. Heljanko and I. Niemelä. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming*, 3(4-5):519–550, 2003.
22. J.G. Henriksen and P.S. Thiagarajan. Dynamic Linear Time Temporal Logic. *Annals of Pure and Applied logic*, 96(1-3):187–207, 1999.
23. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.
24. Marco Montali, Paolo Torroni, Federico Chesani, Paola Mello, Marco Alberti, and Evelina Lamma. Abductive logic programming as an effective technology for the static verification of declarative business processes. *Fundam. Inform.*, 102(3-4):325–361, 2010.
25. Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops, LNCS 4103*, pages 169–180. Springer, 2006.
26. Dumitru Roman and Michael Kifer. Semantic web service choreography: Contracting and enactment. In *International Semantic Web Conference, LNCS 5318*, pages 550–566, 2008.
27. M. P. Singh. A social semantics for Agent Communication Languages. *Issues in Agent Communication, LNCS(LNAI) 1916*, pages 31–45, 2000.
28. W. van der Aalst and A. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
29. Wil M. P. van der Aalst and Maja Pesic. Decserflow: Towards a truly declarative service flow language. In *The Role of Business Processes in Service Oriented Architectures*, volume 06291 of *Dagstuhl Seminar Proceedings*, 2006.
30. Ingo Weber, Jörg Hoffmann, and Jan Mendling. Beyond soundness: On the verification of semantic business process models. *Distributed and Parallel Databases (DAPD)*, 2010.
31. P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. *AAMAS'02*, pages 527–534, 2002.

The CHR-based Implementation of the $\mathcal{S}$CIFF Abductive System

Marco Alberti[1], Marco Gavanelli[2], and
Evelina Lamma[2]

[1] CENTRIA - DI/FCT - Universidade Nova de Lisboa
Quinta da Torre - 2829-516 Caparica, Portugal
[2] ENDIF - Università di Ferrara
Via Saragat, 1 - 44100 Ferrara, Italy.
m.alberti@fct.unl.pt,{marco.gavanelli|evelina.lamma}@unife.it

Abstract. Abduction is a form of inference that supports hypothetical reasoning and has been applied to a number of domains, such as diagnosis, planning, protocol verification. Abductive Logic Programming (ALP) is the integration of abduction in logic programming. Usually, the operational semantics of an ALP language is defined as a proof procedure.
The first implementations of ALP proof-procedures were based on the meta-interpretation technique, which is flexible but limits the use of the built-in predicates of logic programming systems. Another, more recent, approach exploits theoretical results on the similarity between abducibles and constraints. With this approach, which bears the advantage of an easy integration with built-in predicates and constraints, Constraint Handling Rules has been the language of choice for the implementation of abductive proof procedures. The first CHR-based implementation mapped integrity constraints directly to CHR rules, which is an efficient solution, but prevents defined predicates from being in the body of integrity constraints and does not allow a sound treatment of negation by default.
In this paper, we describe the CHR-based implementation of the $\mathcal{S}$CIFF abductive proof-procedure, which follows a different approach. The $\mathcal{S}$CIFF implementation maps integrity constraints to CHR constraints, and the transitions of the proof-procedure to CHR rules, making it possible to treat default negation, while retaining the other advantages of CHR-based implementations of ALP proof-procedures.

1 Introduction

According to the philosopher Peirce [1], abductive reasoning is one of the basic inferences a reasoning agent (and a human in particular) uses. It is a type of hypothetical reasoning associated with finding explanations for a given evidence. Its most classical application is diagnosis: we are given a symptom of a patient, or a wrong behaviour of a machine, plus a set of rules explaining which illnesses might cause the symptom/misbehaviour, and we have to guess the right

cause. Besides diagnosis, abductive reasoning has been applied to a number of applications, like planning [2], protocol verification [3], etc.

Abductive Logic Programming (ALP) [4] is a language that embeds abductive reasoning into logic programming. In ALP, we have a set of predicates that have no definition, and are called *abducibles*. The truth of such predicates cannot be proven, but it can be assumed: the abductive derivation will provide in the computed answer the set of abduced hypotheses, together with the binding (the classical answer of Logic Programming languages). However, in typical applications, not all combinations of assumptions make sense: some illnesses are to be excluded beforehand, depending e.g. on the sex of the patient. For this reason, in ALP the user can typically define a set of rules, called *Integrity Constraints*, that must be satisfied by the set of hypotheses. The operational semantics of an ALP is typically defined as a proof-procedure. A number of proof-procedures have been proposed in the past for performing abductive reasoning; they are typically implemented as Prolog meta-interpreters [5–8].

A number of researchers have become interested in abductive reasoning because it deals in a simple and sound way with negation [9]. Literal $not(a)$ is rewritten as an integrity constraint $a \rightarrow \mathit{false}$, and then handled appropriately by the proof procedure. This type of negation is also called *negation by default*.

ALP has also been integrated with Constraint Logic Programming [6, 8, 10], in order to use both abductive reasoning and constraint propagation.

Kowalski et al. [11] studied the theoretical similarities between constraints and abducibles. Such similarity was later exploited for the implementation of abductive proof-procedures where abducibles are mapped to CLP constraints. For this purpose, a promising is Constraint Handling Rules (CHR) [12] a language designed to implement new constraints and constraint solvers in a simple and efficient way.

The first works on the implementation of abductive reasoning in CHR [13–16] implemented directly the integrity constraints into CHR rules: in a sense, CHR becomes also the language for writing integrity constraints. Thus, the user can write rules such as

$$\mathbf{p} \wedge \mathbf{q} \rightarrow r,$$

where $\mathbf{p}$ and $\mathbf{q}$ are abducible predicates and r can be either an abducible or a defined predicate. The interest of a CHR implementation is not only theoretical: thanks to the tight integration of CHR in the host language (which is often Prolog), those proof-procedures can seamlessly access built-in constructs and constraint solvers. This means that they have access to the innumerable libraries written in Prolog, and they can even *recurse* through external predicates: the abductive program can invoke Prolog predicates, and also meta-predicates (e.g., `findall`, `minimize`, ...), which can in their turn request the abduction of atoms, etc. A proof-procedure written in *CHR* benefits immediately from all the improvements of *CHR* engines, as recently happened with the Leuven CHR implementation [17]. Finally, those ALP which do not exploit abduction (or use abduction only in a limited subset of the application) do not suffer from the meta-interpretation overhead, but run at full speed.

However, a rule with a defined predicate in the antecedent is not allowed: these languages sacrifice negation by default on the altar of efficiency, which is a sensible thing to do in some applications, but it is not in others.

The $\mathcal{S}$CIFF proof-procedure [18] was developed in 2003 with an alternative CHR implementation, in which integrity constraints are first-class objects, and the proof-procedure can actively reason about them. In particular, we map abducibles into *CHR* constraints and implement the transitions of the operational semantics as *CHR* rules; in this way, the implementation follows very closely the operational semantics. Thanks to the sound operational semantics, $\mathcal{S}$CIFF has a sound treatment of default (and also explicit) negation. Thanks to the *CHR* implementation, $\mathcal{S}$CIFF is smoothly integrated with a constraint solver. From a language viewpoint, $\mathcal{S}$CIFF has unique features that do not appear in other abductive proof-procedures: it handles universally quantified variables both in the abducibles and in the integrity constraints; CLP constraints (treated as quantifier restrictions [19]) can be imposed both on existentially and on universally quantified variables.

$\mathcal{S}$CIFF has been continuously developed and improved in the past few years, and now it is smoothly integrated in graphical interfaces, semantic web applications; it is considerably faster, more robust, and provides more features.

In this paper, we show the implementation in CHR of the abductive proof-procedure $\mathcal{S}$CIFF, and we report about its recent improvements.

The rest of the paper is organised as follows. We first describe the $\mathcal{S}$CIFF abductive framework in Section 2. After some preliminaries on *CHR* (Section 3), we present the implementation of $\mathcal{S}$CIFF in *CHR* (Section 4). Discussion of related work (Section 5) and conclusions (Section 6) follow.

2 An abductive framework

Abductive Logic Programming is a family of programming languages that integrate abductive reasoning into logic programming. An ALP is a logic program, consisting of a set of clauses, that can contain in the body some distinguished predicates, belonging to a set $\mathcal{A}$ and called *abducibles*, (that will be shown in the following in **boldface**). The aim is finding a set of abducibles $\Delta \subseteq \mathcal{A}$ that, together with the knowledge base, is an explanation for a given known effect (also called *goal* $\mathcal{G}$):

$$KB \cup \Delta \models \mathcal{G}.$$

Also, Δ should satisfy a set of logic formulae, called *Integrity Constraints* IC:

$$KB \cup \Delta \models IC.$$

E.g., if a patient has a headache, a physician may consult a knowledge base

$$\begin{array}{l} \mathit{headache} \leftarrow \mathbf{flu}. \\ \mathit{headache} \leftarrow \mathbf{migraine}. \\ \mathit{headache} \leftarrow \mathbf{meningitis}. \end{array}$$

and the abductive system will return one of the three explanations.

$\mathcal{S}$CIFF [18] is a language in the ALP class. CLP [20] constraints can be imposed on variables (which allows, for instance, to express that an event is expected to happen in a given time interval). For example, we might have an integrity constraint

$$\mathbf{flu} \rightarrow \mathbf{temp}(T), T < 39$$

saying that the explanation **flu** is acceptable only if the temperature of the patient is less than 39^oC. The computed answer includes in general three elements: a substitution for the variables in the goal (as usual in Prolog), the constraint store (as in CLP), and the set Δ of abduced literals.

$\mathcal{S}$CIFF was originally developed for the verification of interaction in multiagent systems [21, 22] and it is an extension of the IFF proof-procedure [7].

3 A brief introduction to Constraint Handling Rules

Constraint Handling Rules [12] (*CHR* for brevity hereafter) are essentially a committed-choice language consisting of guarded rules that rewrite constraints in a store into simpler ones until they are solved. *CHR* define both *simplification* (replacing constraints by simpler constraints while preserving logical equivalence) and *propagation* (adding new, logically redundant but computationally useful, constraints) over user-defined constraints.

The main intended use for *CHR* is to write constraint solvers, or to extend existing ones. However, the computational model of *CHR* presents features that make it a useful tool for the implementation of the $\mathcal{S}$CIFF proof-procedure.

There are three types of *CHR*s: *simplification*, *propagation* and *simpagation*.

Simplification CHRs. Simplification rules are of the form

$$H_1, \ldots, H_i \iff G_1, \ldots, G_j | B_1, \ldots, B_k \tag{1}$$

with $i > 0$, $j \geq 0$, $k \geq 0$ and where the multi-head $H_1, \ldots, H_i$ is a nonempty sequence of *CHR* constraints, the guard $G_1, \ldots, G_j$ is a sequence of built-in constraints, and the body $B_1, \ldots, B_k$ is a sequence of built-in and *CHR* constraints.

Declaratively, a simplification rule states that, if the guard is true, then the left-hand-side and the right-hand-side are equivalent.

Operationally, when constraint instances $H_1, \ldots, H_i$ in the head are in the store and the guard $G_1, \ldots, G_j$ is true, they are replaced by constraints $B_1, \ldots, B_k$ in the body.

Propagation CHRs. Propagation rules have the form

$$H_1, \ldots, H_i \implies G_1, \ldots, G_j | B_1, \ldots, B_k \tag{2}$$

where the symbols have the same meaning of those in the simplification rules (1).

Declaratively, a propagation rule is an implication, provided that the guard is true. Operationally, when the constraints in the head are in the store, and the guard is true, the constraints in the body are added to the store.

Simpagation CHRs. Simpagation rules have the form

$$H_1, \ldots, H_l \backslash H_{l+1}, \ldots, H_i \Longleftrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k \tag{3}$$

where $l > 0$ and the other symbols have the same meaning and constraints of those of simplification *CHR*s (1).

Declaratively, the rule of Eq. (3) is equivalent to

$$H_1, \ldots, H_l, H_{l+1}, \ldots, H_i \Longleftrightarrow G_1, \ldots, G_j | B_1, \ldots, B_k, H_1, \ldots, H_l \tag{4}$$

Operationally, when the constraints in the head are in the store and the guard is true, $H_1, \ldots, H_l$ remain in the store, and $H_{l+1}, \ldots, H_i$ are replaced by $B_1, \ldots, B_k$.

For example, the constraint $\leq$ can be implemented in CHR by giving its base properties, namely the following rules:

$$A \leq A \Leftrightarrow true \tag{5}$$

$$A \leq B, B \leq A \Leftrightarrow A = B \tag{6}$$

$$A \leq B, B \leq C \Rightarrow A \leq C \tag{7}$$

where the symbol '=' stands for unification. The CHR engine rewrites the constraints in the store occurring as in the left-hand-side of the rules; for example, if the constraints $X \leq Y$, $Y \leq X$ are in the store, they are removed from the store and the variables X and Y are unified, as prescribed by rule 6. Note that on the left-hand-side of a CHR rule only constraints defined with CHR can appear: while the right-hand-side can contain any Prolog predicate (including CLP(FD) constraints, unifications, etc.), these elements cannot appear on the left-hand-side.

4 Implementation of the $\mathcal{S}$CIFF proof-procedure

One of the features obtained through a *CHR* implementation (avoiding meta-interpretation) is that the resolvent of the proof is directly represented as the Prolog resolvent. This allows us to exploit the Prolog stack for depth-first exploration of the tree of states. More importantly, this means that we extensively reuse the Prolog machinery, and that built-in predicates in the host Prolog system can be called from the user's Abductive Logic Programs. We remark again that this feature comes for free together with the CHR implementation, and is not easily available in metainterpreter-based implementations of abductive proof-procedures.

In the same way, the constraint store of the constrained abductive proof-procedure[3] is represented as the union of the CLP constraint stores. For the implementation of the proof-procedure, we used the CLP(FD) and CLP(B) libraries, available both on SICStus [23] and SWI Prolog [24] We also have a

[3] This constraint store, which contains CLP constraints over variables, should not be confused with the *CHR* constraint store, which is used for the implementation of the other data structures.

CHR-based solver on finite and infinite domains, and we defined an *ad-hoc* solver for reified unification. Recently, the interface between $\mathcal{S}$CIFF and the constraint solver has been re-engineered, and now it allows the developer to adopt any constraint solver that implements a given interface. In this way, the user can choose for each application which solver he/she wants to use; moreover, new solvers can be added with very limited effort. For example, the constraint solver on the reals, CLP($\mathcal{R}$) [25] has been integrated into $\mathcal{S}$CIFF: the new solver is based on the simplex algorithm (plus branch-and-bound), which is very efficient for linear constraints.

To the best of our knowledge, the other abductive proof-procedures implemented in CHR map abducibles to CHR constraints. Integrity constraints, instead, are often represented as CHR rules (typically, propagation rules) [13, 15]. Since a propagation CHR can have only CHR constraints in the multiple heads, the corresponding abductive proof-procedure can contain only abducibles in the precondition. This limitation forbids in the proof-procedure the implementation of default negation, that was one of the main motivations behind Abductive Logic Programming [9]. The operational semantics is then an extension of the operational semantics of CHR.

$\mathcal{S}$CIFF was developed following a different idea: we wanted increased flexibility in our language, while retaining the features that come for free with the CHR implementation. We first defined the declarative and operational semantics of $\mathcal{S}$CIFF as extensions of the IFF [7]. The operational semantics is given through a set of transitions that transform a state into another. The implementation, which maps integrity constraints, as well as the other relevant data strutures, to CHR constraints (rather than CHR rules) and transitions to CHR rules, follows the operational semantics very closely.

In the following, we first show some examples of transitions; the interested reader can find the complete list of transitions in a previous publication [18], together with the proofs of soundness and completeness of the $\mathcal{S}$CIFF proof-procedure. We then describe the implementation of some transitions in Section 4.2.

4.1 Examples of transitions

Given an abducible $\mathbf{a}(X)$ and an integrity constraint

$$\mathbf{a}(Y) \wedge p(Z) \wedge Y > Z \rightarrow r(Z)$$

transition *propagation* generates the following implication (that we call *Partially Solved Integrity Constraint* or PSIC for short):

$$X = Y \wedge p(Z) \wedge Y > Z \rightarrow r(Z) \tag{8}$$

Now, a transition *case analysis* generates two nodes of an OR-tree: in the first we consider the case $X = Y$, so the previous implication is transformed into

$$p(Z) \wedge X > Z \rightarrow r(Z),$$

in the second node, we consider the case that $X \neq Y$, and in this case the implication (8) is already satisfied.

Suppose we choose the first node, and that the knowledge base contains the definition of predicate $p(Z)$, e.g., as a fact $p(1)$. Transition *unfolding* generates the following implication:

$$X > 1 \rightarrow r(1)$$

Now, *case analysis* is again applied to the implication: in the first node we consider the case $X > 1$, while in the second $X \leq 1$. In the first case, the goal $r(1)$ is invoked.

These are just some examples of the transitions. $\mathcal{S}$CIFF contains transitions for handling correctly the various items (abducibles, expectations, happened events, CLP constraints, negation by default, explicit negation, etc.) in the $\mathcal{S}$CIFF language.

4.2 CHR implementation

The implementation of the transitions in CHR follows very closely the operational semantics. The various types of data are mapped to CHR constraints, while the transitions are mapped into CHR rules. For example, abducibles are represented as $abd(X)$; this means that abducibles can be directly used in the knowledge base, and CHR will take care of all the machinery necessary to abduce a new literal and propagate its consequences. For example, the clause

$$g(X) : -\mathbf{a}(X), b.$$

can be written as

```
g(X) :- abd(a(X)), b.
```

A (partially solved) integrity constraint

$$\mathbf{a}(X) \wedge p(Y) \rightarrow r(Z) \vee q(Z)$$

is mapped to the CHR constraint

```
psic([abd(a(X)),p(Y)],(r(Z);q(Z))).
```

As a first attempt, the propagation transition (together with case analysis) can be implemented via the CHR rule

```
abd(X), psic([abd(Y)|Rest],Head)
 ==>
copy(psic([abd(Y)|Rest],Head),psic([abd(Y1)|Rest1],Head1)),   (9)
 reif_unify(X,Y1,B),
 (B#=1, psic(Rest1,Head1) ; B#=0).
```

where `copy` performs a renaming of an atom (which also considers the various types of quantification in the $\mathcal{S}$CIFF [18], as well as CLP constraints attached

to the variables), `#=` is the finite domain equality constraint and `reif_unify` is a CHR implementation of *reified unification* [26].

`reif_unify` is a CHR constraint that declaratively imposes that either `B` $= 1$ and the first two arguments unify, or `B` $= 0$ and the two atoms do not unify; in logics, `reif_unify(X,Y,B)` is true iff

$$X = Y \leftrightarrow B = 1.$$

Note that some of the details are taken care of directly by CHR: if we have a set of abducibles and a set of PSICs we do not have to remember explicitly which PSICs have been tried with which abducibles (in order to avoid loops), as CHR itself does this work.

Note also that propagation is attempted only with the first element of the partially solved integrity constraint's antecedent, but this does not impact on what integrity constraints will be completely solved. For instance, given the integrity constraint $\mathbf{a}, \mathbf{b} \rightarrow \mathbf{c}$, if $\mathbf{b}$ and $\mathbf{a}$ are abduced in sequence, $\mathbf{b}$ will not be propagated as soon as it is abduced, but only after $\mathbf{a}$ has been abduced and propagated, and the partially solved integrity constraint $\mathbf{b} \rightarrow \mathbf{c}$ has been added to the constraint store; in the end, $\mathbf{c}$ will be abduced anyway. In this way, we ensure that each atom is propagated only once with each integrity constraint, without a need to keep track of previous propagations.

A number of improvements can be done to rule (9). First of all, CHR uses efficient indexing and hash tables to avoid checking all the possible pairs of CHR constraints. Sadly, rule (9) does not exploit such features of CHR. Note that the constraints in the antecedent of the propagation *CHR* do not share any variable, thus the CHR engine has to try each possible pair of constraints of types `abd` and `psic`, while, intuitively, one should try only those pairs whose arguments may unify. A first idea would be to rewrite the transition as:

```
abd(X), psic([abd(X)|Rest],Head)
        ==> ...
```

which would use CHR hash tables much more efficiently, but it would propagate only when the arguments are already ground or bound to the same term. This would be a very lazy propagation, that does not exploit the reified unification algorithm.

However, since abducibles are atoms, they always have a main functor, thus the argument of `abd` is always a term, which can contain variables, but it cannot be a variable itself. It is sensible to exploit the main functor for improving the selection of candidates. We represent each abducible as a CHR constraint with two arguments, where the first argument contains a ground term used to improve the hashing: in the current implementation, it is a list containing the main functor and its arity. The code for abducing an atom X is then:

```
abd(X) :- functor(X,F,A), abd([F,A],X).
```

Now, the *propagation transition* can be implemented with the *CHR* propagation rule:

```
abd(F,X), psic([abd(F,Y)|Rest],Head)
 ==>
 fn_ok(X,Y) |
 copy(psic([abd(Y)|Rest],Head),psic([abd(Y1)|Rest1],Head1)),
 reif_unify(X,Y1,B),
 (B#=1, psic(Rest1,Head1) ; B#=0).
```
(10)

i.e., only those pairs with identical first argument (i.e., abducibles that share the same functor and arity) are tried. `fn_ok` is a predicate that checks if the two arguments can possibly unify, and is also used for improving efficiency.

Many of the transitions of $\mathcal{S}$CIFF open a choice point, as we can see from the example of Eq. (10). However, in case `reif_unify` immediately yields 0 or 1, there is no point in opening a choice point. Otherwise, one could delay the disjunction, in order to open choice points as late as possible, hoping that other transitions might constrain the value of the `B` variable, possibly making it ground. In other words, it would be more desirable to delay as much as possible the non-deterministic transitions (those opening choice points), while expediting the deterministic ones (those that do not open choice points). One reason is that the deterministic may fail, and in this case the choice points opened by nondeterministic choices would be useless.

In order to implement the delay mechanism, we defined a CHR constraint '`nondeterministic`' that holds, as argument, a non-deterministic goal. In the previous example, the propagation transition is actually rewritten as

```
abd(F,X), psic([abd(F,Y)|Rest],Head)
==>
fn_ok(X,Y) |
copy(psic([abd(Y)|Rest],Head),psic([abd(RenY)|RenRest],RenHead)),
reif_unify(X,RenY,B),
(B == 1 -> psic(RenRest,RenHead) ;
 B == 0 -> true ;
 nondeterministic((B#=1, psic(RenRest,RenHead)) ; B#=0)).
```

i.e., we check if reified unification imposed a value on the boolean variable `B`, and we open a choice point only in case it did not. The choice point is not actually opened immediately, but it is declared in a CHR constraint.

Then, we defined a set of CHRs for dealing with `nondeterministic` constraints. We alternate a deterministic and a non-deterministic phase: initially, in the derivation, only deterministic transitions can be activated. Later, when the fixed point of the deterministic ones is reached, *one* non-deterministic transition can be applied, and we return to the deterministic phase. In CHR:

```
switch2det @ phase(nondeterministic), nondeterministic(G) <=>
    call(G),
```

```
    phase(deterministic).
switch2nondet @ phase(deterministic) <=> phase(nondeterministic).
```

where rule `switch2nondet` should be one of the last rules to be tried.

Transition Unfolding. Differently from HYPROLOG [15], integrity constraints can involve literals built on defined predicates. This allows for a sound treatment of default negation: a negative literal $not(a)$ is converted into an implication $a \rightarrow false$. Given a PSIC whose body contains a literal of a predicate defined in the KB, transition *unfolding* unfolds the literal:

```
psic([Atom|Rest],Head) <=>
is_defined_literal(Atom) |
findall(clause(Atom,Body),clause(Atom,Body),Clauses),
unfold(Clauses,psic([Atom|Rest],Head)).

unfold([],_).
unfold([clause(Atom,Body)|Clauses],psic([Atom1|Rest1],Head1)):-
  ccopy(psic([Atom1,Rest1],Head1),psic([Atom2|Rest2],Head2)),
  Atom = Atom2,
  append(Body,Rest2,NewBody),
  psic(NewBody,Head2),
  unfold(Clauses,psic([Atom1|Rest1],Head1)).
```

This might pose problems of termination: if the unfolded predicate is recursive, there exists an infinite branch in the derivation. For example, consider the IC:

$$\mathbf{a}(List), member(Term, List) \rightarrow \mathbf{b}(Term) \tag{11}$$

with the knowledge base:

$$member(X, [X|T]).$$
$$member(X, [Y|T]) : -member(X, T).$$
$$g : -\mathbf{a}([1, 2, 3]).$$

Intuitively, the goal g is true provided that we abduce $\mathbf{a}([1, 2, 3])$ and $\mathbf{b}(1)$, $\mathbf{b}(2)$, $\mathbf{b}(3)$. However, if we unfold predicate *member* in the IC (11) before the atom $\mathbf{a}([1, 2, 3])$ was abduced, the unfolding generates an infinite number of implications. For this reason, early versions of $\mathcal{S}$CIFF delay the unfolding after the other transitions, in the hope of binding some of the variables. In this particular example, if `member` is unfolded only after $\mathbf{a}([1, 2, 3])$ is abduced, the number of implications generated is equal to the number of elements in the list L, which is finite.

However, in other cases defined predicates provide just the value of a parameter, in this example, a deadline:

$$\mathbf{start}(a, T_a) \wedge deadline(D) \rightarrow \mathbf{end}(b, T_b) \wedge T_b \leq T_a + D$$

The knowledge base contains a simple fact *deadline*(5) stating that the deadline is 5 time units. In this case, if the unfolding of *deadline* is postponed after propagation of the **start**(a, T_a) event, it is repeated as many times as the number of **start** atoms that will be abduced, which might be a big number. For this reason, recent versions of $\mathcal{S}$CIFF unfold eagerly the predicates defined only by facts, and lazily the other predicates.

Results. The efficiency of $\mathcal{S}$CIFF has greatly improved with respect to earlier versions [27]. The following experiments were run on a 1.5GHz Pentium M 715 processor, 512MB RAM computer running SICStus 4.0.7.

Experiment	$\mathcal{S}$CIFF 2005	$\mathcal{S}$CIFF 2011
Auction Protocol	2.27s	0.37s
Block World	45.0s	15.7s
A^{l}LoWS Feeble conformance	84.4s	36.8s
A^{l}LoWS non-conformant	3.7s	3.3s

The aim of these experiments is not to compete with other abductive proof-procedures, but to show the improvements obtained taking into consideration the features of *CHR*. The version 2011 features improved hashing, eager unfolding, and other minor improvements. The experiments are real-life applications that we developed in $\mathcal{S}$CIFF: the proof of conformance of agents to an auction protocol, planning in the abductive event calculus, and A^{l}LoWS [28], a system based on $\mathcal{S}$CIFF for the conformance verification of web services to choreographies.

4.3 $\mathcal{S}$CIFF as a System

From a software engineering perspective, since its first prototypical implementation [27] $\mathcal{S}$CIFF has been greatly improved, and it is now a fully fledged development system (see Fig. 1). An integrated development environment for $\mathcal{S}$CIFF ALPs, implemented as an Eclipse plugin, is now available. Through a RuleML parser, ALPs can be obtained dynamically from the web. Animations of the output are possible through Scalable Vector Graphics (SVG), the W3C standard for vector graphics and animations. A Graphical User Interface displays relevant information to the user [3]. Facts can be added dynamically from a number of sources, including Linda blackboards, Apache log files, Jade Sniffer Agent output.

5 Related work

The $\mathcal{S}$CIFF abductive framework is derived from the IFF proof procedure [7], which it extends in several directions: dynamic update of the knowledge base by

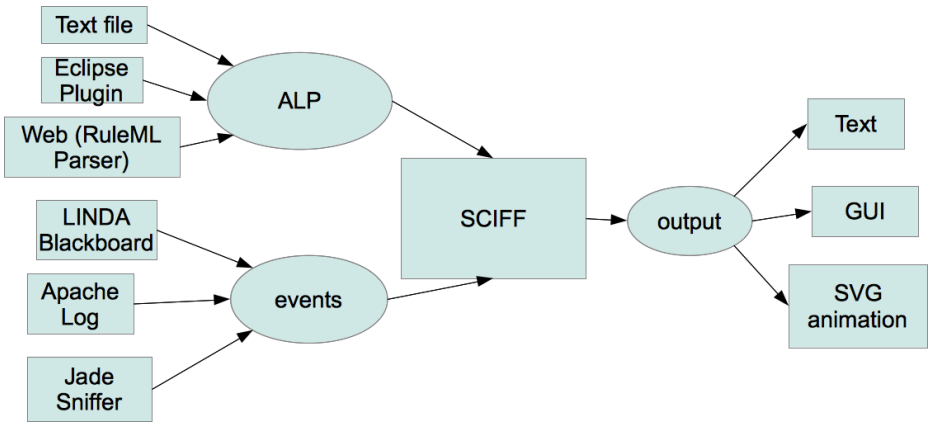


Fig. 1. Architecture of the $\mathcal{S}$CIFF system, illustrating some of the available inputs and outputs.

happening events, confirmation and disconfirmation of hypotheses, hypotheses with universally quantified variables, CLP constraints. Many other abductive proof-procedures have been proposed in the past; the interested reader can refer to the exhaustive survey by Kakas et al. [29].

Other proof-procedures deal with constraints; in particular we mention ACLP [6] and the $\mathcal{A}$-system [8], which are deeply focussed on efficiency issues.

Some conspicuous work has been done with the integration of the IFF proof-procedure with constraints [11]. Endriss et al. [10] present an implementation of an abductive proof-procedure that extends IFF [7] in two ways: by dealing with constraint predicates and with non-allowed abductive logic programs. The cited work, however, does not deal with confirmation and disconfirmation of hypotheses and universally quantified variables in abducibles, as ours does.

All of these proof-procedures are implemented as Prolog meta-interpreters. However, we believe that a *CHR* implementation has features that a meta-interpreted version cannot have, as we explained in the introduction.

Abdennadher and Christiansen [13] and Christiansen and Dahl [30] propose to exploit the *CHR* language to extend SICStus Prolog to support abduction more efficiently than with metainterpretation-based solutions. They represent abducibles as *CHR* constraints as we do, but they represent integrity constraints directly as *CHR* propagation rules, using the built-in *CHR* matching mechanism for propagation: this does not seem possible in our framework, in which we pose no limitations on the type of literals that occur in the conditions of integrity constraints. We also experimented with a similar implementation [14, 16], but it proved insufficient for our needs, as we needed a sound treatment of default negation and more flexibility in the quantification of variables.

6 Conclusions and future work

In this paper, we have presented the implementation of an abductive proof-procedure in *CHR*. We believe that the use of *CHR* in writing abductive proof-procedures has several advantages, compared to traditional approaches based on meta-interpretation. The first advantage is that $\mathcal{S}$CIFF benefits immediately from new implementations and improvements of *CHR* engines [31, 32, 17]. Another advantage is that the proof-procedure does not require meta-interpretation, which lets the user invoke built-in Prolog (meta)predicates within an Abductive Logic Program, without the need of contemplating explicitly their occurrence in the meta-interpreter. Also, Prolog is an instance of ALP (that does not use abduction): in $\mathcal{S}$CIFF, a Prolog program that does not use abduction runs at full speed, without the overhead of meta-interpretation.

An interesting extension of this work would be to integrate the two main ideas for implementing abduction in *CHR* in a unique framework. Each of the ideas have their own pros and cons: HYPROLOG, that implements integrity constraints as *CHR* rules, has less overhead, while $\mathcal{S}$CIFF, that maps integrity constraints into *CHR* constraints, is able to deal with default negation and is provably sound and complete. We are currently studying the idea of selecting syntactically the integrity constraints in an ALP in a preprocessing phase, and implementing each in the most efficient possible way, i.e., as *CHR* rules, whenever possible, or as *CHR* constraints when they contain defined predicates or CLP(FD) constraints.

Concerning confirmation, there are many possible extensions of this work, which we intend to pursue in the future. For instance, it would be worthwhile to let the user impose the failure of a branch of the reasoning tree, regardless of the confirmation or disconfirmation of the hypotheses made in the branch, in order to explore branches that the user finds more promising. We also intend to support a breadth-first exploration of the computation tree, as an alternative to the depth-first exploration of the current implementation. Besides, we believe that the formal framework would benefit from the introduction of a formalism to express priorities among the possible alternative hypotheses, in a given state of the computation.

Another direction of improvement could be towards better computational performance, possibly exploiting alternative efficient *CHR* implementations, such as the one proposed by Wolf [32].

Acknowledgments

This work has been supported by the European Commission within the e-Policy project (n. 288147).

References

1. Hartshorne, C., Weiss, P., eds.: Collected Papers of Charles Sanders Peirce, 1931–1958. Volume 2. Harvards University Press (1965)

2. Eshghi, K.: Abductive planning with the event calculus. In: Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, Cambridge, MA, MIT Press (1988)
3. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Compliance verification of agent interaction: a logic-based tool. Applied Artificial Intelligence **20** (2006) 133–157
4. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive Logic Programming. Journal of Logic and Computation **2** (1993) 719–770
5. Denecker, M., Schreye, D.D.: SLDNFA: an abductive procedure for abductive logic programs. Journal of Logic Programming **34** (1998) 111–167
6. Kakas, A.C., Michael, A., Mourlas, C.: ACLP: Abductive Constraint Logic Programming. Journal of Logic Programming **44** (2000) 129–177
7. Fung, T.H., Kowalski, R.A.: The IFF proof procedure for abductive logic programming. Journal of Logic Programming **33** (1997) 151–165
8. Kakas, A.C., van Nuffelen, B., Denecker, M.: $\mathcal{A}$-System: Problem solving through abduction. In Nebel, B., ed.: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA (IJCAI-01), Seattle, Washington, USA, Morgan Kaufmann Publishers (2001) 591–596
9. Eshghi, K., Kowalski, R.A.: Abduction compared with negation by failure. In Levi, G., Martelli, M., eds.: Proceedings of the 6th International Conference on Logic Programming, Cambridge, MA, MIT Press (1989) 234–255
10. Endriss, U., Mancarella, P., Sadri, F., Terreni, G., Toni, F.: The CIFF proof procedure for abductive logic programming with constraints. In Alferes, J.J., Leite, J.A., eds.: Proc. JELIA 2004. Volume 3229 of LNAI., Springer-Verlag (2004) 31–43
11. Kowalski, R., Toni, F., Wetzel, G.: Executing suspended logic programs. Fundamenta Informaticae **34** (1998) 203–224
12. Frühwirth, T.: Theory and practice of constraint handling rules. Journal of Logic Programming **37** (1998) 95–138
13. Abdennadher, S., Christiansen, H.: An experimental CLP platform for integrity constraints and abduction. In Larsen, H., Kacprzyk, J., Zadrozny, S., Andreasen, T., Christiansen, H., eds.: FQAS, Flexible Query Answering Systems. LNCS, Warsaw, Poland, Springer-Verlag (2000) 141–152
14. Gavanelli, M., Lamma, E., Mello, P., Milano, M., Torroni, P.: Interpreting abduction in CLP. In Buccafurri, F., ed.: APPIA-GULP-PRODE Joint Conference on Declarative Programming, Reggio Calabria, Italy, Università Mediterranea di Reggio Calabria (2003) 25–35
15. Christiansen, H., Dahl, V.: HYPROLOG: A new logic programming language with assumptions and abduction. In Gabbrielli, M., Gupta, G., eds.: Proc. ICLP 2005. Volume 3668 of LNCS., Springer (2005) 159–173
16. Alberti, M., Chesani, F., Daolio, D., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interaction protocols in a logic-based system. Scalable Computing: Practice and Experience **8** (2007) 1–13
17. Schrijvers, T., Demoen, B.: The K.U. Leuven CHR system: implementation and application. In Frhwirth, T., Meister, M., eds.: Proc. CHR'04, Ulm, Germany (2004)
18. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Verifiable agent interaction in abductive logic programming: the SCIFF framework. ACM Transactions on Computational Logic **9** (2008)
19. Bürckert, H.: A resolution principle for constrained logics. Artificial Intelligence **66** (1994) 235–271

20. Jaffar, J., Maher, M.: Constraint logic programming: a survey. Journal of Logic Programming **19-20** (1994) 503–582
21. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: Specification and verification of agent interactions using social integrity constraints. Electronic Notes in Theoretical Computer Science **85** (2003)
22. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: An Abductive Interpretation for Open Agent Societies. In Cappelli, A., Turini, F., eds.: AI*IA 2003: Advances in Artificial Intelligence, Proceedings of the 8th Congress of the Italian Association for Artificial Intelligence, Pisa. Volume 2829 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2003) 287–299
23. Carlsson, M., et al.: SICStus Prolog user's manual. Swedish Institute of Computer Science, Kista, Sweden. 4.0.7 edn. (2009) `www.sics.se/sicstus/`.
24. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming (2011) `http://arxiv.org/abs/1011.5332`.
25. Holzbaur, C.: OFAI clp(q,r) Manual. Austrian Research Institute for Artificial Intelligence, Vienna. 1.3.3 edn. (1995) TR-95-09.
26. Nuffelen, B.V.: Abductive Constraint Logic Programming: Implementation and Applications. PhD thesis, Katholieke Universiteit Leuven (2004)
27. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E.: The CHR-based Implementation of a System for Generation and Confirmation of Hypotheses. Number 2005-01 in Ulmer Informatik-Berichte (2005) 111–122
28. Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., Montali, M.: An abductive framework for a-priori verification of web services. In Maher, M., ed.: Proceedings of the Eighth Symposium on Principles and Practice of Declarative Programming, July 10-12, 2006, Venice, Italy, New York, USA, Association for Computing Machinery (ACM), Special Interest Group on Programming Languages (SIGPLAN), ACM Press (2006) 39–50
29. Kakas, A.C., Kowalski, R.A., Toni, F.: The role of abduction in logic programming. In Gabbay, D.M., Hogger, C.J., Robinson, J.A., eds.: Handbook of Logic in Artificial Intelligence and Logic Programming. Volume 5., Oxford University Press (1998) 235–324
30. Christiansen, H., Dahl, V.: Assumptions and abduction in Prolog. In Muñoz-Hernández, S., Gómez-Perez, J.M., Hofstedt, P., eds.: Workshop on Multiparadigm Constraint Programming Languages (MultiCPL'04), Saint-Malo, France (2004) Workshop notes.
31. Holzbaur, C., Frühwirth, T.: Compiling constraint handling rules into Prolog with attributed variables. In Nadathur, G., ed.: PPDP. (1999)
32. Wolf, A.: Adaptive constraint handling with CHR in Java. In Walsh, T., ed.: Principles and Practice of Constraint Programming - CP 2001. Volume 2239 of Lecture Notes in Computer Science., Paphos, Cyprus, Springer Verlag (2001) 256–270

Controlling Polyvariance for Specialization-Based Verification

Fabio Fioravanti[1], Alberto Pettorossi[2], Maurizio Proietti[3], and Valerio Senni[2,4]

[1] Dipartimento di Scienze, University 'G. D'Annunzio', Pescara, Italy
fioravanti@sci.unich.it
[2] DISP, University of Rome Tor Vergata, Rome, Italy
{pettorossi,senni}@disp.uniroma2.it
[3] CNR-IASI, Rome, Italy
maurizio.proietti@iasi.cnr.it
[4] LORIA-INRIA, Villers-les-Nancy, France
valerio.senni@loria.fr

Abstract. We present some extensions of a method for verifying safety properties of infinite state reactive systems. Safety properties are specified by constraint logic programs encoding (backward or forward) reachability algorithms. These programs are transformed, before their use for checking safety, by specializing them with respect to the initial states (in the case of backward reachability) or with respect to the unsafe states (in the case of forward reachability). In particular, we present a specialization strategy which is more general than previous proposals and we show, through some experiments performed on several infinite state reactive systems, that by using the specialized reachability programs obtained by our new strategy, we considerably increase the number of successful verifications. Then we show that the specialization time, the size of the specialized program, and the number of successful verifications may vary, depending on the *polyvariance* introduced by the specialization, that is, the set of specialized predicates which have been introduced. Finally, we propose a general framework for controlling polyvariance and we use our set of examples of infinite state reactive systems to compare in an experimental way various control strategies one may apply in practice.

1 Introduction

Program specialization is a program transformation technique that, given a program and a specific context of use, derives a specialized program that is more effective in the given context [19]. Program specialization techniques have been proposed for several programming languages and, in particular, for (constraint) logic languages (see, for instance [7,11,16,17,21,22,24,27]).

Program specialization may generate *polyvariant procedures*, that is, it may derive, starting from a single procedure, multiple specialized versions of that procedure. In the case of (constraint) logic programming, program specialization may introduce several new predicates corresponding to specialized versions of a predicate occurring in the original program. The application of specialized

procedures to specific input values often results in a very efficient computation. However, if the number of new predicate definitions and, hence, the size of the specialized program, is overly large, we may have difficulties during program compilation and execution.

In order to find an optimal balance between the degree of specialization and the size of the specialized program, several papers have addressed the issue of *controlling* polyvariance (see [22,26], in the case of logic programming). This issue is related to the one of controlling *generalization* during program specialization, because a way of reducing unnecessary polyvariance is to replace several specialized procedures by a single, more general one.

In this paper we address the issue of controlling polyvariance in the context of specialization-based techniques for the automatic verification of properties of reactive systems [12,13,23].

One of the present challenges in the verification field is the extension of model checking techniques [5] to systems with an infinite number of states. For these systems exhaustive state exploration is impossible and, even for restricted classes, simple properties such as *safety* (or *reachability*) properties are undecidable (see [9] for a survey of relevant results).

In order to overcome this limitation, several authors have advocated the use of *constraints* to represent infinite sets of states and constraint logic programs to encode temporal properties (see, for instance, [8,15]). By using constraint-based methods, a temporal property can be verified by computing the least or the greatest models of programs, represented as finite sets of constraints. Since, in general, the computation of these models may not terminate, various techniques have been proposed based on *abstract interpretation* [2,3,6,8] and *program specialization* [12,13,23].

The techniques based on abstract interpretation compute a conservative approximation of the program model, which is sometimes sufficient to prove that the property of interest actually holds. However, in the case where the property does not hold in the approximated model, one cannot conclude that the property does not hold.

The techniques based on program specialization transform the program that encodes the property of interest by taking into account the property to be proved and the initial states of the system, so that the construction of the model of the transformed program may terminate more often than the one of the original program, that is, the so-called *verification precision* is improved.

In this paper we show that the control of polyvariance plays a very relevant role in verification techniques based on program specialization. Indeed, the specialization time, the size of the specialized program, and the precision of verification may vary depending on the set of specialized predicates introduced by different specialization strategies. We also propose a general framework for controlling polyvariance during specialization and, through several examples of infinite state reactive systems taken from the verification literature, we compare in an experimental way various control strategies that may be applied in practice.

Our paper is structured as follows. In Section 2 we present a method based on constraint logic programming for specifying and verifying safety properties of infinite state reactive systems. In Sections 3 and 4 we present a general framework for specializing constraint logic programs that encode safety properties of infinite state reactive systems and, in particular, for controlling polyvariance during specialization. In Section 5 we present some experimental results. Finally, in Section 6 we compare our method with related approaches in the field of program specialization and verification.

2 Specialization-Based Reachability Analysis of Infinite State Reactive Systems

An infinite state reactive system is specified as follows. A *state* is an n-tuple $\langle a_1, \ldots, a_n \rangle$ where each a_i is either an element of a finite domain $\mathbb{D}$ or an element of the set $\mathbb{R}$ of the real numbers. By X we denote a variable ranging over states, that is, an n-tuple of variables $\langle X_1, \ldots, X_n \rangle$ where each X_i ranges over either $\mathbb{D}$ or $\mathbb{R}$. Every constraint c is a (possibly empty) conjunction $fd(c)$ of equations on a finite domain $\mathbb{D}$ and a (possibly empty) conjunction $re(c)$ of linear inequations on $\mathbb{R}$. An equation on $\mathbb{R}$ is considered as a conjunction of two inequations. Given a constraint c, every equation in $fd(c)$ and every linear inequation in $re(c)$ is said to be an *atomic constraint*.

The set I of the *initial states* is represented by a disjunction $init_1(X) \vee \ldots \vee init_k(X)$ of constraints on X. The *transition relation* is a disjunction $t_1(X, X') \vee \ldots \vee t_m(X, X')$ of constraints on X and X', where X' is the n-tuple $\langle X'_1, \ldots, X'_n \rangle$ of primed variables.

A constraint c is also denoted by $c(X)$, when we want indicate that the variable X occurs in it. Similarly, for constraints denoted by $c(X')$ or $c(X, X')$. Given a constraint c and a tuple V of variables, we define the *projection* $c|_V$ to be the constraint d such that: (i) the variables of d are among the variables in V, and (ii) $\mathbb{D} \cup \mathbb{R} \models d \leftrightarrow \exists Z\ c$, where Z is the tuple of the variables occurring in c and not in V. We assume that the set of constraints is closed under projection.

Given a clause C of the form $H \leftarrow c \wedge G$, by $con(C)$ we denote the constraint c. A clause of the form $H \leftarrow c$, where c is a constraint, is said to be a *constrained fact*. We say that a constrained fact $H \leftarrow c$ *subsumes* a clause $H \leftarrow d \wedge G$, where d is a constraint and G is a goal, iff d *entails* c, written $d \sqsubseteq c$, that is, $\mathbb{D} \cup \mathbb{R} \models \forall (d \rightarrow c)$.

In this paper we will focus on the verification of *safety* properties. A safety property holds iff *an unsafe state cannot be reached from an initial state of the system*. The set U of the unsafe states is represented by a disjunction $u_1(X) \vee \ldots \vee u_n(X)$ of constraints.

One can verify a safety property by one of the following two strategies:
(i) the *Backward Strategy*: one applies a *backward reachability* algorithm, thereby computing the set BR of states from which it is possible to reach an *unsafe* state, and then one checks whether or not BR has an empty intersection with the set I of the initial states;

(ii) the *Forward Strategy*: one applies a *forward reachability* algorithm, thereby computing the set FR of states reachable from an initial state, and then one checks whether or not FR has an empty intersection with the set U of the unsafe states.

Variants of these two strategies have been proposed and implemented in various automatic verification tools [1,4,14,20,28].

The Backward and Forward Strategies can easily be encoded into constraint logic programming. In particular, we can encode the backward reachability algorithm by means of the following constraint logic program Bw:

$$
\begin{array}{rl}
I_1\colon & \mathit{unsafe} \leftarrow \mathit{init}_1(X) \wedge \mathit{bwReach}(X) \\
& \cdots \\
I_k\colon & \mathit{unsafe} \leftarrow \mathit{init}_k(X) \wedge \mathit{bwReach}(X) \\
T_1\colon & \mathit{bwReach}(X) \leftarrow t_1(X, X') \wedge \mathit{bwReach}(X') \\
& \cdots \\
T_m\colon & \mathit{bwReach}(X) \leftarrow t_m(X, X') \wedge \mathit{bwReach}(X') \\
U_1\colon & \mathit{bwReach}(X) \leftarrow u_1(X) \\
& \cdots \\
U_n\colon & \mathit{bwReach}(X) \leftarrow u_n(X)
\end{array}
$$

We have that: (i) $\mathit{bwReach}(X)$ holds iff an unsafe state can be reached from the state X in zero or more applications of the transition relation, and (ii) *unsafe* holds iff there exists an initial state of the system from which an unsafe state can be reached.

The semantics of program Bw is given by the *least model*, denoted $M(Bw)$, that is, the set of ground atoms derived by using: (i) the theory of equations over the finite domain $\mathbb{D}$ and the theory of linear inequations over the reals $\mathbb{R}$ for the evaluation of the constraints, and (ii) the usual least model construction (see [18] for more details).

The system is *safe* if and only if $\mathit{unsafe} \notin M(Bw)$.

Example 1. Let us consider an infinite state reactive system where each state is a pair of real numbers and the following holds:
(i) the set of initial states is the set of pairs $\langle X_1, X_2\rangle$ such that the constraint $X_1 \geq 1 \wedge X_2 = 0$ holds;
(ii) the transition relation is the set of pairs of states $\langle\langle X_1, X_2\rangle, \langle X_1', X_2'\rangle\rangle$ such that the constraint $X_1' = X_1 + X_2 \wedge X_2' = X_2 + 1$ holds; and
(iii) the set of unsafe states is the set of pairs $\langle X_1, X_2\rangle$ such that the constraint $X_2 > X_1$ holds.
For the above system the predicate *unsafe* is defined by the following CLP program $Bw1$:

1. $\mathit{unsafe} \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge \mathit{bwReach}(X_1, X_2)$
2. $\mathit{bwReach}(X_1, X_2) \leftarrow X_1' = X_1 + X_2 \wedge X_2' = X_2 + 1 \wedge \mathit{bwReach}(X_1', X_2')$
3. $\mathit{bwReach}(X_1, X_2) \leftarrow X_2 > X_1$ □

The Backward Strategy can be implemented by the bottom-up construction of the least fixpoint of the *immediate consequence operator* S_{Bw}, that is, by computing $S_{Bw}\uparrow\omega$ [18]. The operator S_{Bw} is analogous to the usual immediate consequence operator associated with logic programs, but constructs a set of

constrained facts, instead of a set of ground atoms. We have that $M(Bw)$ is the set of ground atoms of the form $A\vartheta$ such that there exists a constrained fact $A \leftarrow c$ in $S_{Bw}\uparrow\omega$ and the constraint $c\vartheta$ is satisfiable. BR is the set of all states s such that there exists a constrained fact of the form $bwReach(X) \leftarrow c(X)$ in $S_{Bw}\uparrow\omega$ and $c(s)$ holds. Thus, by using clauses $I_1, \ldots, I_k$, we have that the atom *unsafe* holds iff $BR \cap I \neq \emptyset$.

One weakness of the Backward Strategy is that, when computing BR, it does not take into account the constraints holding on the initial states. This may lead to a failure of the verification process, even if the system is safe, because the computation of $S_{Bw}\uparrow\omega$ may not terminate. A similar weakness is also present in the Forward Strategy as it does not take into account the properties holding on the unsafe states when computing FR.

In this paper we present a method, based upon the program specialization technique introduced in [13], for overcoming these weaknesses. For reasons of space we will present the details of our method for the Backward Strategy only. The application of our method in the case of the Forward Strategy is similar, and we will briefly describe it when presenting our experimental results in Section 5.

The objective of program specialization is to transform the constraint logic program Bw into a new program $SpBw$ such that: (i) $unsafe \in M(Bw)$ iff $unsafe \in M(SpBw)$, and (ii) the computation of $S_{SpBw}\uparrow\omega$ terminates more often than $S_{Bw}\uparrow\omega$ because it exploits the constraints holding on the initial states.

Let us show how our method based program specialization works on the infinite state reactive system of Example 1.

Example 2. Let us consider the program $Bw1$ of Example 1. The computation of $S_{Bw1}\uparrow\omega$ does not terminate, because it does not take into account the information about the set of initial states, represented by the constraint $X_1 \geq 1 \wedge X_2 = 0$. (One can also check that the top-down evaluation of the query *unsafe* does not terminate either.)

This difficulty can be overcome by specializing the program $Bw1$ with respect to the constraint $X_1 \geq 1 \wedge X_2 = 0$. Similarly to [13], we apply a specialization technique based on the *unfolding* and *folding* transformation rules for constraint logic programs (see, for instance, [10]). We introduce a new predicate $new1$ defined as follows:

4. $new1(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge bwReach(X_1, X_2)$

We fold clause 1 using clause 4, that is, we replace the atom $bwReach(X_1, X_2)$ by $new1(X_1, X_2)$ in the body of clause 1, and we get:

5. $unsafe \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge new1(X_1, X_2)$

Now we continue the transformation from the definition of the newly introduced predicate $new1$. We unfold clause 4, that is, we replace the occurrence of $bwReach(X_1, X_2)$ by the bodies of the clauses 2 and 3 defining $bwReach(X_1, X_2)$ in $Bw1$, and we derive:

6. $new1(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge X_1' = X_1 \wedge X_2' = 1 \wedge bwReach(X_1', X_2')$

In order to fold clause 6 we may use the following definition, whose body consists (modulo variable renaming) of the atom $bwReach(X_1', X_2')$ and the constraint $X_1 \geq 1 \wedge X_2 = 0 \wedge X_1' = X_1 \wedge X_2' = 1$ projected w.r.t. the variables $\langle X_1', X_2' \rangle$:

7. $newp(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 = 1 \wedge bwReach(X_1, X_2)$

However, if we repeat the process of unfolding and, in order to fold, we introduce new predicate definitions whose bodies consist of the atom $bwReach(X_1', X_2')$ and projected constraints w.r.t. $\langle X_1', X_2' \rangle$, then we will introduce, in fact, an infinite sequence of new predicate definitions of the form:

$$newq(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 = k \wedge bwReach(X_1, X_2)$$

where k gets the values $1, 2, \ldots$ In order to terminate the specialization process we apply a *generalization strategy* and we introduce the following predicate definition which is a generalization of both clauses 4 and 7:

8. $new2(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 \geq 0 \wedge bwReach(X_1, X_2)$

We fold clause 6 using clause 8 and we get:

9. $new1(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge X_1' = X_1 \wedge X_2' = 1 \wedge new2(X_1', X_2')$

Now we continue the transformation from the definition of the newly introduced predicate $new2$. By unfolding clause 8 and then folding using again clause 8 we derive:

10. $new2(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 \geq 0 \wedge X_1' = X_1 + X_2 \wedge X_2' = X_2 + 1 \wedge new2(X_1', X_2')$
11. $new2(X_1, X_2) \leftarrow X_1 \geq 1 \wedge X_2 > X_1$

The final specialized program, called $SpBw1$, is made out of clauses 5, 9, 10, and 11. Now the computation of $S_{SpBw1} \uparrow \omega$ terminates due to the presence of the constraint $X_1 \geq 1$ which holds on the initial states and occurs in all clauses of $SpBw1$. □

The form of the specialized program strongly depends on the strategy used for introduction of new predicates corresponding to the specialized versions of the predicate *bwReach*. For instance, in Example 1 we have introduced the two new predicates $new1$ and $new2$, and then we have obtained the specialized program by deriving mutually recursive clauses defining those predicates. Note, however, that the definition of $new2$ is *more general than* the definition of $new1$, because the constraint occurring in the body of the clause defining $new1$ implies the constraint occurring in the body of the clause defining $new1$. Thus, by applying an alternative strategy we could introduce $new2$ only and derive a program $SpBw2$ where clauses 5 and 9 are replaced by the following clause:

12. $unsafe \leftarrow X_1 \geq 1 \wedge X_2 = 0 \wedge new2(X_1, X_2)$

Program $SpBw2$ is smaller than $SpBw1$ and the computation of $S_{SpBw2} \uparrow \omega$ terminates in fewer steps than the one of $S_{SpBw1} \uparrow \omega$.

In general, when applying our specialization-based verification method there is an issue of *controlling polyvariance*, that is, of introducing a set of new predicate definitions that perform well with respect to the following objectives:
(i) ensuring the termination and the efficiency of the specialization strategy,
(ii) minimizing the size of the specialized program, and

(iii) ensuring the termination and the efficiency of the fixpoint computation of the least models.

The objective of ensuring the termination of the fixpoint computation (and, thus, the *precision* of the verification) can be in contrast with the other objectives, because it may need the introduction of less general predicates, while the achievement of other objectives is favoured by the introduction of more general predicates. In the next section we will present a framework for controlling polyvariance and achieving a good balance between the requirements we have listed above.

3 A Generic Algorithm for Controlling Polyvariance During Specialization

The core of our technique for controlling polyvariance is an algorithm for specializing the CLP program *Bw* with respect to the constraints characterizing the set of initial states. Our algorithm is *generic*, in the sense that it depends on three unspecified procedures: (1) *Partition*, (2) *Generalize*, and (3) *Fold*. Various definitions of the *Partition*, *Generalize*, and *Fold* procedures will be given in the next section, thereby providing concrete specialization algorithms. These definitions encode techniques already proposed in the specialization and verification fields (see, for instance, [6,13,22,27]) and also new techniques proposed in this paper.

Our generic specialization algorithm (see Figure 1) constructs a tree, called *DefsTree*, where: (i) each node is labelled by a clause of the form $newp(X) \leftarrow d(X) \wedge bwReach(X)$, called a *definition*, defining a new predicate introduced during specialization, and (ii) each arc from node D_i to node D_j is labelled by a subset of the clauses obtained by unfolding the definition of node D_i. When no confusion arises, we will identify a node with its labelling definition. An arc from definition D_i to definition D_j labelled by the set Cs of clauses is denoted by $D_i \xrightarrow{Cs} D_j$.

The definition at the root of *DefsTree* is denoted by the special symbol T. Initially, *DefsTree* is $\{\mathsf{T} \xrightarrow{\{I_1\}} D_1, \ldots, \mathsf{T} \xrightarrow{\{I_k\}} D_k\}$, where (i) $I_1, \ldots, I_k$ are the clauses defining the predicate *unsafe* in program *Bw* (see Section 2), and (ii) for $j = 1, \ldots, k$, D_j is the clause $new_j(X) \leftarrow init_j(X) \wedge bwReach(X)$, such that new_j is a new predicate symbol and the body of D_j is equal to the body of I_j.

A definition D in *DefsTree* is said to be *recurrent* iff D labels both a leaf node and a non-leaf node of *DefsTree*.

We construct the children of a non-recurrent definition D in the definition tree *DefsTree* constructed so far, as follows. We unfold D with respect to the atom $bwReach(X)$ occurring in its body, that is, we replace $bwReach(X)$ by the bodies of the clauses $T_1, \ldots, T_m, U_1, \ldots, U_n$ that define *bwReach* in *Bw*, thereby deriving a set *UnfD* of $m+n$ clauses. Then, from *UnfD* we remove all clauses whose body contains an unsatisfiable constraint and all clauses that are *subsumed* by a (distinct) constrained fact in *UnfD*.

Next we apply the *Partition* procedure and we compute a set $\{B_1, \ldots, B_h\}$ of pairwise disjoint sets of clauses, called *blocks*, such that $\mathit{UnfD} = B_1 \cup \ldots \cup B_h$.

Finally, we apply the *Generalize* procedure to each block of the partition. This generalization step is often useful because, as it has been argued in [27], it allows us to derive more efficient programs. Our *Generalize* procedure takes as input the clause D, a block B_i of the partition of UnfD, and the whole definition tree constructed so far. As we will indicate below, this third argument of the *Generalize* procedure allows us to express the various techniques presented in [6,13,22,27] for controlling generalization and polyvariance.

The output of the *Generalize* procedure is, for each block B_i, a definition G_i such that the constraint occurring in the body of G_i is entailed by every constraint occurring in the body of a *non-unit* clause (that is, a clause different from a constrained fact) in B_i and, hence, every non-unit clause in B_i can be folded using G_i. If all clauses in B_i are constrained facts (and thus, no folding step is required), then G_i is the special definition denoted by the symbol $\Box$. If a clause in B_i has the form $h(X) \leftarrow c(X, X') \wedge \mathit{bwReach}(X')$, then G_i has the form $\mathit{newp}(X) \leftarrow d(X) \wedge \mathit{bwReach}(X)$ and $c(X, X') \sqsubseteq d(X')$. However, we postpone the folding steps until the end of the construction of the whole tree *DefsTree*. For $i = 1, \ldots, h$, we add to *DefsTree* the arc $D \xrightarrow{B_i} G_i$.

The construction of *DefsTree* terminates when all leaf clauses of the current *DefsTree* are recurrent. In general, termination of this construction is not guaranteed and it depends on the particular *Generalize* procedure one considers. All *Generalize* procedures presented in the next section guarantee termination (see also [13,22,27]).

When the construction of *DefsTree* terminates we construct the specialized program *SpBw* by applying the *Fold* procedure which consists in: (i) collecting all clauses occurring in the blocks that label the arcs of *DefsTree*, and then (ii) folding every non-unit clause by using a definition that labels a node of *DefsTree*. Recall that, by construction, every non-unit clause occurring in a block that labels an arc of *DefsTree* can be folded by a definition that labels a node of *DefsTree*.

In the following Section, we will show how the specialization technique of Example 2 can be regarded as an instance of our generic specialization algorithm.

By using the correctness results for the unfolding, folding, and clause removal rules (see, for instance, [10]), we can prove the correctness of our generic specialization algorithm, as stated by the following theorem.

Theorem 1 (Correctness of the Specialization Algorithm). *Let programs Bw and SpBw be the input and the output programs, respectively, of the specialization algorithm that uses any given Partition, Generalize, and Fold procedures. Then* $\mathit{unsafe} \in M(\mathit{Bw})$ *iff* $\mathit{unsafe} \in M(\mathit{SpBw})$.

Input: Program Bw.
Output: Program $SpBw$ such that $unsafe \in M(Bw)$ iff $unsafe \in M(SpBw)$.

INITIALIZATION:
$DefsTree := \{\mathsf{T} \xrightarrow{\{I_1\}} D_1, \ldots, \mathsf{T} \xrightarrow{\{I_k\}} D_k\}$;

while there exists a non-recurrent definition D: $newp(X) \leftarrow c(X) \wedge bwReach(X)$ in $DefsTree$ *do*

UNFOLDING: $UnfD := \{newp(X) \leftarrow c(X) \wedge t_1(X, X') \wedge bwReach(X'),\ \ldots,$
$newp(X) \leftarrow c(X) \wedge t_m(X, X') \wedge bwReach(X'),$
$newp(X) \leftarrow c(X) \wedge u_1(X),\ \ldots,$
$newp(X) \leftarrow c(X) \wedge u_n(X)\}$;

CLAUSE REMOVAL:
while in $UnfD$ there exist two distinct clauses E and F such that E is a constrained fact that subsumes F or there exists a clause F whose body has a constraint which is not satisfiable *do* $UnfD := UnfD - \{F\}$ *end-while*;

DEFINITION INTRODUCTION:
$Partition(UnfD, \{B_1, \ldots, B_h\})$;
for $i = 1, \ldots, h$ *do*
$Generalize(D, B_i, DefsTree, G_i)$;
$DefsTree := DefsTree \cup \{D \xrightarrow{B_i} G_i\}$
end-for;
end-while;

FOLDING: $Fold(DefsTree, SpBw)$

Fig. 1. The generic specialization algorithm.

4 Partition, Generalize, and Fold Procedures

In this section we provide several definitions of the *Partition*, *Generalize*, and *Fold* procedures used by the generic specialization algorithm. Let us start by introducing the following notions.

First, note that the set of all conjunctions of equations on $\mathbb{D}$ can be viewed as a finite lattice whose partial order is defined by the entailment relation $\sqsubseteq$. Given the constraints $c_1, \ldots, c_n$, we define their *most specific generalization*, denoted $\gamma(c_1, \ldots, c_n)$, the conjunction of: (i) the least upper bound of the conjunctions $fd(c_1), \ldots, fd(c_n)$ of equations on $\mathbb{D}$, and (ii) the *convex hull* [6] of the constraints $re(c_1), \ldots, re(c_n)$ on $\mathbb{R}$, which is the least (w.r.t. the $\sqsubseteq$ ordering) constraint h in $\mathbb{R}$ such that $re(c_i) \sqsubseteq h$, for $i = 1, \ldots, n$. (Note that this notion of generalization is different from the one that is commonly used in logic programming.)

Note that, for $i = 1, \ldots, n$, $c_i \sqsubseteq \gamma(c_1, \ldots, c_n)$. Given a set of constraints $Cs = \{c_1, \ldots, c_n\}$, we define the equivalence relation $\simeq_{fd}$ on Cs such that, for every $c_1, c_2 \in Cs$, $c_1 \simeq_{fd} c_2$ iff $fd(c_1)$ is equivalent to $fd(c_2)$ in $\mathbb{D}$. We also define the equivalence relation $\simeq_{re}$ on Cs as the reflexive, transitive closure of the relation $\downarrow_{\mathbb{R}}$ on Cs such that, for every $c_1, c_2 \in Cs$, $c_1 \downarrow_{\mathbb{R}} c_2$ iff $re(c_1) \wedge re(c_2)$ is satisfiable in $\mathbb{R}$.

For example, let us consider an element $a \in \mathbb{D}$. Let c_1 be the constraint $X_1 > 0 \wedge X_2 = a$ and c_2 be the constraint $X_1 < 0 \wedge X_2 = a$. Then we have that

$c_1 \simeq_{fd} c_2$ on $\{c_1, c_2\}$. Now, let c_3 be the constraint $X_1 > 0 \wedge X_1 < 2$, c_4 be the constraint $X_1 > 1 \wedge X_1 < 3$, and c_5 be the constraint $X_1 > 2 \wedge X_1 < 4$. Since $c_3 \downarrow_{\mathbb{R}} c_4$ and $c_4 \downarrow_{\mathbb{R}} c_5$, we have $c_3 \simeq_{re} c_5$ on $\{c_3, c_4, c_5\}$. Note that $c_3 \not\simeq_{re} c_5$ on $\{c_3, c_5\}$ because $c_3 \wedge c_5$ is *not* satisfiable in $\mathbb{R}$.

Partition. The *Partition* procedure takes as input the following set of n (≥ 1) clauses:

$$\begin{array}{lll} \mathit{UnfD} := \{ C_1: & \mathit{newp}(X) \leftarrow c_1(X, X') \wedge \mathit{bwReach}(X'), \\ & \cdots \\ C_m: & \mathit{newp}(X) \leftarrow c_m(X, X') \wedge \mathit{bwReach}(X'), \\ C_{m+1}: & \mathit{newp}(X) \leftarrow c_{m+1}(X, X'), \\ & \cdots \\ C_n: & \mathit{newp}(X) \leftarrow c_n(X, X') \, \} \end{array}$$

where, for some m, with $0 \leq m \leq n$, $C_1, \ldots, C_m$ are not constrained facts, and $C_{m+1}, \ldots, C_n$ are constrained facts. The *Partition* procedure returns as output a partition $\{B_1, \ldots, B_h\}$ of *UnfD*, such that $B_h = \{C_{m+1}, \ldots, C_n\}$. The integer h and the blocks $B_1, \ldots, B_{h-1}$ are computed by using one of the following *partition operators*. For the operators *FiniteDomain*, *Constraint*, and *FDC*, the integer h to be computed is obtained as a result of the computation of the blocks B_i's.

(i) *Singleton*: $h = m+1$ and, for $1 \leq i \leq h-1$, $B_i = \{C_i\}$, which means that every non-constrained fact is in a distinct block;
(ii) *FiniteDomain*: for $1 \leq i \leq h-1$, for $j, k = 1, \ldots, m$, two clauses C_j and C_k belong to the same block B_i iff their finite domain constraints on the primed variables are equivalent, that is, iff $c_j|_{X'} \simeq_{fd} c_k|_{X'}$ on $\{c_1|_{X'}, \ldots, c_m|_{X'}\}$;
(iii) *Constraint*: for $1 \leq i \leq h-1$, for $i, j = 1, \ldots, m$, two clauses C_j and C_k belong to the same block B_i iff there exists a sequence of clauses in *UnfD* starting with C_j and ending with C_k such that for any two consecutive clauses in the sequence, the conjunction of the real constraints on the primed variables is satisfiable, that is, iff $c_j|_{X'} \simeq_{re} c_k|_{X'}$ on $\{c_1|_{X'}, \ldots, c_m|_{X'}\}$;
(iv) *FDC*: for $1 \leq i \leq h-1$, for $i, j = 1, \ldots, m$, two clauses C_j and C_k belong to the same block B_i iff they belong to the same block according to both the *FiniteDomain* and the *Constraint* partition operator, that is, iff $c_j|_{X'} \simeq_{fd} c_k|_{X'}$ and $c_j|_{X'} \simeq_{re} c_k|_{X'}$ on $\{c_1|_{X'}, \ldots, c_m|_{X'}\}$;
(v) *All*: $h = 2$ and $B_1 = \{C_1, \ldots, C_m\}$, which means that all non-constrained facts are in a single block.

Generalize. The *Generalize* procedure takes as input a definition D, a block B of clauses computed by the *Partition* procedure, and the tree *DefsTree* of definitions introduced so far, and returns a definition clause G. If B is a set of constrained facts then G is the special definition denoted by the symbol $\Box$. Otherwise, if B is the set $\{E_1, \ldots, E_k\}$ of clauses and none of which is a constrained fact, then clause G is obtained as follows.

Step 1. Let $b(X')$ denote the most specific generalization $\gamma(\mathit{con}(E_1)|_{X'}, \ldots, \mathit{con}(E_k)|_{X'})$.
if there exists a nearest ancestor A_1 of D (possibly D itself) in *DefsTree* such that A_1 is of the form: $\mathit{newq}(X') \leftarrow a_1(X') \wedge \mathit{bwReach}(X')$ (modulo

variable renaming) and $a_1(X') \simeq_{fd} con(D)$
then $b_{anc}(X') = \gamma(a_1(X'), b(X'))$ *else* $b_{anc}(X') = b(X')$;

Step 2. Let us consider a *generalization operator* $\ominus$ (see [13] and the operators *Widen* and *WidenSum* defined below).
if in *DefsTree* there exists a clause H: $newt(X') \leftarrow d(X') \wedge bwReach(X')$ (modulo variable renaming) such that $b_{anc}(X') \sqsubseteq d(X')$
then G is H
else let *newu* be a new predicate symbol
if there exists a nearest ancestor A_2 of D (possibly D itself) in *DefsTree* such that A_2 is a definition of the form:
$newr(X') \leftarrow a_2(X'), bwReach(X')$ and $a_2(X') \simeq_{fd} b_{anc}(X')$
then G is $newu(X') \leftarrow (a_2(X') \ominus b_{anc}(X')) \wedge bwReach(X')$
else G is $newu(X') \leftarrow b_{anc}(X') \wedge bwReach(X')$.

In [13] we have defined and compared several generalization operators. Among those, now we consider the following two operators which we have used in the experiments we have reported in the next section. Indeed, as indicated in [13], these two operators perform better than all other operators.

Widen. Given any two constraints c and d such that c is $a_1 \wedge \ldots \wedge a_m$, where the a_i's are atomic constraints, the operator *Widen*, denoted $\ominus_W$, returns the constraint $c \ominus_W d$ which is the conjunction of the atomic constraints of c which are entailed by d, that is, which are in the set $\{a_h \mid 1 \leq h \leq m \text{ and } d \sqsubseteq a_h\}$ (see [6] for a similar widening operator used in static analysis). Note that, in the case of our Generalize procedure, we have that $fd(d)$ is a subconjunction of $c \ominus_W d$.

WidenSum. Let us first define the thin well-quasi ordering $\precsim_S$. For any atomic constraint a on $\mathbb{R}$ of the form $q_0 + q_1 X_1 + \ldots + q_k X_k \lessdot 0$, where $\lessdot$ is either $<$ or $\leq$, we define $sumcoeff(a)$ to be $\sum_{j=0}^{k} |q_j|$. Given the two atomic constraints a_1 of the form $p_1 < 0$ and a_2 of the form $p_2 < 0$, we have that $a_1 \precsim_S a_2$ iff $sumcoeff(a_1) \leq sumcoeff(a_2)$. Similarly, if we are given the atomic constraints a_1 of the form $p_1 \leq 0$ and a_2 of the form $p_2 \leq 0$. Given any two constraints $c = a_1 \wedge \ldots \wedge a_m$ and $d = b_1 \wedge \ldots \wedge b_n$, where the a_i's and the b_i's are atomic constraints, the operator *WidenSum*, denoted $\ominus_{WS}$, returns the constraint $c \ominus_{WS} d$ which is the conjunction of the constraints in the set $\{a_h \mid 1 \leq h \leq m \text{ and } d \sqsubseteq a_h\} \cup \{b_k \mid b_k \text{ occurs in } re(d) \text{ and } \exists\, a_i \text{ occuring in } re(c), b_k \precsim_S a_i\}$, which is the set of atomic constraints which either occur in c and are entailed by d, or occur in d and are less than or equal to some atomic constraint in c, according to the thin well-quasi ordering $\precsim_S$. Note that, in the case of our Generalize procedure, we have that $fd(d)$ is a subconjunction of $c \ominus_{WS} d$.

Our generic Partition and Generalize procedures can be instantiated to get known specialization algorithms and abstract interpretation algorithms. In particular, (i) the technique proposed by Cousot and Halbwachs [6] can be obtained by using the operators *FiniteDomain* and *Widen*, (ii) the specialization algorithm by Peralta and Gallagher [27] can be obtained by using the operators *All* and *Widen*, and (iii) our technique presented in [13] can be obtained by using the

partition operator *Singleton* together with the generalization operators *Widen* or *WidenSum*.

Fold. Let us first introduce the following definition. Given the two clauses $C\colon newp(X) \leftarrow c(X) \wedge bwReach(X)$ and $D\colon newq(X) \leftarrow d(X) \wedge bwReach(X)$, we say that C is *more general than* D, and by abuse of language, we write $D \sqsubseteq C$, iff $d(X) \sqsubseteq c(X)$. A clause C is said to be *maximally general* in a set S of clauses iff for all clauses $D \in S$, if $C \sqsubseteq D$ then $D \sqsubseteq C$. (Recall that the relation $\sqsubseteq$ is not antisymmetric.) For the *Fold* procedure we have the following two options.

Immediate Fold (*Im*, for short): (Step 1) all clauses occurring in the labels of the arcs of *DefsTree* are collected in a set F, and then (Step 2) for every non-unit clause E in F such that E occurs in the block B_i labelling an arc of the form $D \xrightarrow{B_i} D_i$, for some clause D, E is folded using D_i.

Maximally General Fold (*MG*, for short): (Step 1) is equal to that of *Immediate Fold* procedure, and (Step 2) every non-unit clause in F is folded using a maximally general clause in *DefsTree*.

Immediate Fold is simpler than *Maximally General Fold*, because it does not require any comparison between definitions in *DefsTree* to compute a maximally general one. Note also that a unique, most general definition for folding a clause may not exist, that is, there exist clauses that can be folded by using two definitions which are incomparable with respect to the $\sqsubseteq$ ordering. However, the *Maximally General Fold* procedure allows us to use a subset of the definitions introduced by the specialization algorithm, thereby reducing polyvariance and deriving specialized programs of smaller size.

As already mentioned in the previous section, the specialization technique which we have applied in Example 2 can be obtained by instantiating our generic specialization algorithm using the following operators: *Singleton* for partitioning, *Widen* for generalization, and *Immediate Fold* for folding.

5 Experimental Evaluation

We have implemented the generic specialization algorithm presented in Section 3 using MAP [25], an experimental system for transforming constraint logic programs. The MAP system is implemented in SICStus Prolog 3.12.8 and uses the `clpr` library to operate on constraints. All experiments have been performed on an Intel Core 2 Duo E7300 2.66 GHz under the Linux operating system.

We have performed the backward and forward reachability analyses of several infinite state reactive systems taken from the literature [1,2,4,8,20,28], encoding, among others, mutual exclusion protocols, cache coherence protocols, client-server systems, producer-consumer systems, array bound checking, and Reset Petri nets.

For backward reachability we have applied the method presented in Section 2. For forward reachability we have applied a variant of that method and in particular, first, (i) we have encoded the forward reachability algorithm by a constraint logic program *Fw* and we have specialized *Fw* with respect to the set

of the unsafe states, thereby deriving a new program *SpFw*, and then, (ii) we have computed the least fixpoint of the immediate consequence operator S_{SpFw} (associated with program *SpFw*).

In Tables 1 and 2 we have reported the results of our verification experiments for backward reachability (that is, program *Bw*) and forward reachability (that is, program *Fw*), respectively. For each example of infinite state reactive system, we have indicated the total verification time (in milliseconds) of the non-specialized system and of the various specialized systems obtained by applying our strategy.

The symbol '∞' means that either the program specialization or the least fixpoint construction did not terminate within 200 seconds. If the time taken is less than 10 milliseconds, we have written the value '0'. Between parentheses we have also indicated the number of predicate symbols occurring in the specialized program. This number is a measure of the degree of polyvariance determined by our specialization algorithm.

In the column named *Input*, we have indicated the time taken for computing the least fixpoint of the immediate consequence operator of the input, non-specialized program (whose definition is based on program *Bw* for backward reachability, and program *Fw* for forward reachability). In the six rightmost columns, we have shown the sum of the specialization time and the time taken for computing the least fixpoint of the immediate consequence operator of the specialized programs obtained by using the following six pairs of partition operators and generalization operators: (i) $\langle$*All*, *Widen*$\rangle$, called *All_W*, (ii) $\langle$*FDC*, *Widen*$\rangle$, called *FDC_W*, (iii) $\langle$*Singleton*, *Widen*$\rangle$, called *Single_W*, (iv) $\langle$*All*, *WidenSum*$\rangle$, called *All_WS*, (v) $\langle$*FDC*, *WidenSum*$\rangle$, called *FDC_WS*, and (vi) $\langle$*Singleton*, *WidenSum*$\rangle$, called *Single_WS*. For each example the tables have two rows corresponding, respectively, to the *Immediate Fold* procedure (*Im*) and *Maximally General Fold* procedure (*MG*).

If we consider *precision*, that is, the number of successful verifications, we have that the best combinations of the partition procedure and the generalization operators are: (i) *FDC_WS* and *Single_WS* for backward reachability, each of which verified 54 properties out of 58 (in particular, 27 with *Im* and 27 with *MG*), and (ii) *Single_WS* for forward reachability, which verified 36 properties out of 58 (in particular, 18 with *Im* and 18 with *MG*).

If we compare the *Generalize* procedures we have that *WidenSum* is strictly more precise than *Widen* (up to 50%). Moreover, except for a few cases (backward reachability of CSM, forward reachability of Kanban), if a property cannot be proved by using *WidenSum* then it cannot be proved using *Widen*. *WidenSum* is usually more polyvariant than *Widen*. If we consider the verification times, they are generally favourable to *WidenSum* with respect to *Widen*, with some exceptions.

If we compare the partition operators we have that *All* is strictly less precise than the other operators: it successfully terminates in 138 cases out of 232 tests obtained by varying: (i) the given example-program, (ii) the property to be proved (either forward reachability or backward reachability), (iii) the generalization operator, and (iv) the *Fold* procedure. However, *All* is the only partition

operator which allows us to verify the McCarty91 examples. By using the *Singleton* operator, the verification terminates in 158 cases out of 232, and by using the *FDC* operator, the verification successfully terminates in 159 cases out of 232. However, there are some properties (forward reachability of Peterson, InsertionSort and SelectionSort) which can only be proved using *Singleton*.

Note also that, if a property can be verified by using the *FDC* partition operator, then it can be verified by using either the operator *All* or the operator *Singleton*.

The two operators *Singleton* and *FDC* have similar polyvariance and verification times, while the operator *All* yields a specialized program with lower polyvariance and requires shorter verification times than *Singleton* and *FDC*.

If we compare the two *Fold* procedures, we have that *Maximally General Fold* for most of the examples has lower polyvariance and shorter verification times than *Immediate Fold*, while the precision of the two procedures is almost identical, except for a few cases where *Maximally General Fold* verifies the property, while *Immediate Fold* does not (backward reachability of Bakery4, Peterson and CSM).

6 Related Work and Conclusions

We have proposed a framework for controlling polyvariance during the specialization of constraint logic programs in the context of verification of infinite state reactive systems. In our framework we can combine several techniques for introducing a set of specialized predicate definitions to be used when constructing the specialized programs. In particular, we have considered new combinations of techniques introduced in the area of constraint-based program analysis and program specialization such as convex hull, widening, most specific generalization, and well-quasi orderings (see, for instance, [6,13,22,27]).

We have performed an extensive experimentation by applying our specialization framework to the reachability analysis of infinite state systems. We have considered constraint logic programs that encode both backward and forward reachability algorithms and we have shown that program specialization improves the termination of the computation of the least fixpoint needed for the analysis. However, by applying different instances of our framework, we may get different termination results and different verification times. In particular, we have provided an experimental evidence that the degree of polyvariance has an influence on the effectiveness of our specialization-based verification method.

Our experiments confirm that, on one hand, a high degree of polyvariance often corresponds to high precision of analysis (that is, high number of terminating verifications) and, on the other hand, a low degree of polyvariance often corresponds to short verification times. We have also determined a specific combination of techniques for controlling polyvariance and provides, with respect to our set of examples, a good balance between precision and verification time.

Other techniques for controlling polyvariance during the specialization of logic programs have been proposed in the literature [7,13,22,26,27]. As already

	Input	*Fold*	*All_W*	*FDC_W*	*Single_W*	*All_WS*	*FDC_WS*	*Single_WS*
Bakery2	60	*Im*	140 (5)	130 (36)	130 (36)	80 (6)	20 (23)	20 (23)
		MG	100 (3)	110 (14)	100 (14)	80 (6)	20 (15)	20 (15)
Bakery3	2710	*Im*	7240 (5)	3790 (144)	3870 (144)	1100 (6)	200 (77)	150 (77)
		MG	3380 (3)	2620 (64)	2190 (61)	1110 (6)	200 (60)	190 (60)
Bakery4	129900	*Im*	∞	112340 (535)	111540 (539)	19340 (6)	102140 (1745)	101300 (1745)
		MG	129940 (3)	37760 (292)	37010 (296)	19340 (6)	78190 (1172)	76940 (1172)
MutAst	1220	*Im*	4370 (6)	350 (173)	330 (173)	7850 (7)	170 (112)	150 (112)
		MG	1400 (3)	350 (59)	330 (59)	1980 (3)	190 (86)	150 (86)
Peterson N	166520	*Im*	∞	∞	∞	620 (9)	260 (22)	220 (22)
		MG	∞	∞	167650 (3)	650 (9)	260 (22)	230 (22)
Ticket	∞	*Im*	∞	30 (11)	10 (11)	∞	20 (11)	20 (11)
		MG	∞	20 (11)	20 (11)	∞	20 (11)	20 (11)
Berke-RISC	20	*Im*	80 (5)	70 (6)	30 (6)	70 (5)	50 (8)	40 (8)
		MG	80 (3)	70 (3)	30 (3)	70 (5)	50 (8)	30 (8)
DEC Firefly	50	*Im*	140 (5)	160 (7)	100 (7)	320 (7)	30 (6)	20 (6)
		MG	140 (3)	160 (3)	90 (3)	300 (5)	20 (6)	10 (6)
Futurebus+	14890	*Im*	16900 (6)	45240 (14)	44340 (14)	16910 (6)	2580 (19)	2410 (19)
		MG	15150 (3)	15590 (3)	14990 (3)	15140 (3)	2560 (15)	2220 (15)
Illinois Univ	70	*Im*	210 (5)	150 (7)	60 (7)	110 (5)	30 (6)	20 (6)
		MG	190 (3)	150 (5)	70 (5)	100 (3)	30 (6)	20 (6)
MESI	60	*Im*	120 (5)	50 (6)	50 (6)	90 (5)	40 (7)	20 (7)
		MG	90 (3)	60 (4)	20 (4)	90 (5)	40 (7)	30 (7)
MOESI	50	*Im*	220 (6)	190 (7)	130 (7)	250 (6)	90 (7)	50 (7)
		MG	200 (3)	140 (3)	90 (3)	210 (3)	90 (5)	50 (5)
Synapse N+1	10	*Im*	30 (4)	20 (5)	10 (5)	30 (4)	20 (5)	20 (5)
		MG	20 (3)	20 (4)	20 (4)	20 (3)	30 (4)	10 (4)
Xerox Dragon	80	*Im*	230 (5)	180 (7)	80 (7)	470 (7)	60 (8)	30 (8)
		MG	240 (3)	170 (5)	60 (5)	470 (5)	60 (8)	20 (8)
Barber	420	*Im*	290 (5)	5170 (31)	3210 (35)	750 (6)	900 (44)	300 (43)
		MG	270 (3)	3080 (6)	690 (6)	750 (6)	930 (44)	290 (43)
B-Buffer	20	*Im*	170 (5)	400 (11)	280 (11)	210 (6)	4490 (75)	3230 (75)
		MG	150 (3)	300 (3)	170 (3)	210 (6)	4550 (75)	3310 (75)
U-Buffer	20	*Im*	100 (6)	200 (12)	150 (12)	70 (6)	210 (12)	130 (12)
		MG	100 (3)	150 (4)	100 (4)	60 (3)	140 (4)	110 (4)
CSM	188110	*Im*	∞	∞	∞	∞	9870 (146)	6920 (154)
		MG	195700 (3)	203290 (3)	186980 (3)	∞	10310 (146)	7010 (154)
Insert Sort	40	*Im*	90 (7)	60 (23)	60 (23)	130 (8)	90 (28)	80 (28)
		MG	110 (7)	60 (9)	50 (9)	150 (8)	100 (14)	100 (14)
Select Sort	∞	*Im*	∞	∞	∞	∞	220 (35)	170 (32)
		MG	∞	∞	∞	∞	250 (19)	200 (19)
Light Control	20	*Im*	60 (5)	20 (9)	10 (9)	50 (5)	20 (9)	20 (9)
		MG	50 (3)	20 (7)	10 (7)	50 (3)	20 (7)	10 (7)
R-Petri Nets	∞	*Im*	∞	∞	∞	20 (5)	10 (5)	20 (5)
		MG	∞	∞	∞	0 (3)	0 (3)	10 (3)
GB	1750	*Im*	4780 (6)	3300 (10)	3300 (10)	6520 (6)	2190 (10)	2190 (10)
		MG	1870 (3)	1840 (4)	1840 (4)	1870 (3)	2070 (5)	2070 (5)
Kanban	∞	*Im*	∞	∞	∞	∞	8310 (162)	8170 (162)
		MG	∞	∞	∞	∞	8390 (162)	8320 (162)
McCarthy 91	∞	*Im*	∞	∞	∞	4130 (104)	∞	∞
		MG	∞	∞	∞	4120 (3)	∞	∞
Scheduler	∞	*Im*	4020 (5)	5770 (20)	5700 (20)	17530 (7)	3220 (91)	3120 (91)
		MG	2230 (3)	4730 (15)	4610 (15)	12420 (3)	3320 (83)	3220 (83)
Train	∞	*Im*	1710 (6)	1340 (14)	1330 (14)	3030 (8)	20250 (299)	19850 (299)
		MG	1700 (5)	970 (6)	940 (6)	3020 (7)	15730 (166)	15270 (166)
TTP	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
Consistency	∞	*Im*	∞	∞	∞	350 (13)	160 (20)	160 (21)
		MG	∞	∞	∞	370 (13)	160 (20)	140 (21)
no. of successes	20	*Im*	19	21	21	24	27	27
		MG	21	22	23	24	27	27

Table 1. Verification Results using Backward Reachability.

	Input		*All_W*	*FDC_W*	*Single_W*	*All_WS*	*FDC_WS*	*Single_WS*
Bakery2	∞	*Im*	20 (5)	∞	∞	30 (5)	20 (20)	20 (20)
		MG	20 (5)	∞	∞	30 (5)	30 (16)	20 (16)
Bakery3	∞	*Im*	∞	∞	∞	∞	1380 (223)	1190 (240)
		MG	∞	∞	∞	∞	1450 (200)	1270 (213)
Bakery4	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
MutAst	370	*Im*	420 (4)	1790 (190)	1720 (190)	410 (4)	280 (141)	280 (141)
		MG	400 (3)	780 (51)	730 (51)	390 (3)	310 (135)	270 (135)
Peterson N	630	*Im*	∞	∞	1220 (6)	∞	∞	8000 (80)
		MG	∞	∞	730 (3)	∞	∞	8040 (80)
Ticket	50	*Im*	60 (4)	240 (30)	210 (30)	60 (4)	210 (26)	180 (26)
		MG	50 (3)	210 (11)	180 (11)	50 (3)	230 (17)	200 (17)
Berke-RISC	∞	*Im*	40 (3)	50 (3)	10 (4)	40 (3)	40 (3)	20 (4)
		MG	40 (3)	40 (3)	10 (4)	40 (3)	40 (3)	10 (4)
DEC Firefly	∞	*Im*	110 (3)	130 (3)	∞	110 (3)	100 (3)	60 (9)
		MG	100 (3)	120 (3)	∞	120 (3)	120 (3)	70 (9)
Futurebus+	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
Illinois Univ	∞	*Im*	150 (3)	150 (3)	∞	140 (3)	150 (3)	70 (8)
		MG	140 (3)	140 (3)	∞	140 (3)	140 (3)	60 (8)
MESI	∞	*Im*	90 (3)	90 (3)	∞	90 (3)	90 (3)	∞
		MG	90 (3)	100 (3)	∞	90 (3)	90 (3)	∞
MOESI	∞	*Im*	130 (3)	130 (3)	∞	130 (3)	130 (3)	∞
		MG	130 (3)	130 (3)	∞	120 (3)	150 (3)	∞
Synapse N+1	∞	*Im*	10 (3)	20 (3)	0 (4)	20 (3)	20 (3)	10 (4)
		MG	20 (3)	20 (3)	0 (4)	20 (3)	20 (3)	10 (4)
Xerox Dragon	∞	*Im*	180 (3)	190 (3)	∞	190 (3)	210 (3)	80 (8)
		MG	180 (3)	190 (3)	∞	180 (3)	190 (3)	70 (8)
Barber	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
B-Buffer	∞	*Im*	∞	50 (4)	20 (4)	∞	50 (4)	20 (4)
		MG	∞	50 (4)	20 (4)	∞	50 (4)	20 (4)
U-Buffer	∞	*Im*	∞	210 (8)	70 (8)	∞	190 (8)	70 (8)
		MG	∞	230 (8)	80 (8)	∞	230 (8)	80 (8)
CSM	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
Insert Sort	∞	*Im*	∞	∞	10 (14)	∞	∞	20 (14)
		MG	∞	∞	30 (14)	∞	∞	30 (14)
Select Sort	∞	*Im*	∞	∞	180 (37)	∞	∞	310 (47)
		MG	∞	∞	180 (37)	∞	∞	320 (45)
Light Control	∞	*Im*	∞	30 (18)	20 (18)	∞	30 (18)	20 (18)
		MG	∞	30 (18)	30 (18)	∞	30 (18)	20 (18)
R-Petri Nets	∞	*Im*	∞	∞	∞	0 (6)	10 (6)	0 (6)
		MG	∞	∞	∞	0 (6)	0 (6)	0 (6)
GB	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
Kanban	44860	*Im*	46840 (4)	46860 (4)	56100 (13)	∞	∞	∞
		MG	45060 (3)	45210 (3)	44130 (3)	∞	∞	∞
McCarthy 91	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
Scheduler	840	*Im*	910 (3)	910 (4)	1750 (32)	930 (3)	920 (4)	127370 (530)
		MG	940 (3)	910 (4)	1110 (4)	940 (3)	900 (4)	127400 (530)
Train	∞	*Im*	∞	∞	∞	∞	∞	410 (51)
		MG	∞	∞	∞	∞	∞	400 (51)
TTP	∞	*Im*	∞	∞	∞	650 (4)	1140 (15)	∞
		MG	∞	∞	∞	660 (4)	1180 (14)	∞
Consistency	∞	*Im*	∞	∞	∞	∞	∞	∞
		MG	∞	∞	∞	∞	∞	∞
no. of successes	5	*Im*	12	14	12	13	17	18
		MG	12	14	12	13	17	18

Table 2. Verification Results using Forward Reachability.

mentioned, the techniques presented in [13,27] can be considered as instances of our framework, while [22,26] do not consider constraints, which are of primary concern in this paper. Our framework generalizes and improves the framework of [13], by introducing partitioning and folding operators which, as shown in Section 5, increase the precision and the performance of the verification process. The *offline specialization* approach followed by [7] is based on a preliminary *binding time analysis* to decide when to unfold a call and when to introduce a new predicate definition. In the context of verification of infinite state reactive systems considered here, due to the very simple structure of the program to be specialized, deciding whether or not to unfold a call is not a relevant issue, and in our approach the binding time analysis is not performed.

As a future work we plan to continue our experiments on a larger set of infinite state reactive systems so as to enhance and better evaluate the specialization framework presented here. We also plan to extend our approach to a framework for the specialization of constraint logic programs outside the context of verification of infinite state reactive systems.

Acknowledgements

This work has been partially supported by PRIN-MIUR and by a joint project between CNR (Italy) and CNRS (France). The last author has been supported by an ERCIM grant during his stay at LORIA-INRIA. Thanks to Laurent Fribourg and John Gallagher for many stimulating conversations.

References

1. A. Annichini, A. Bouajjani, and M. Sighireanu. TReX: A tool for reachability analysis of complex systems. In *Proceedings of CAV 2001*, Lecture Notes in Computer Science 2102, pages 368–372. Springer, 2001.
2. G. Banda and J. P. Gallagher. Analysis of linear hybrid systems in CLP. In *Proceedings of LOPSTR 2008*, Lecture Notes in Computer Science 5438, pages 55–70. Springer, 2009.
3. G. Banda and J. P. Gallagher. Constraint-based abstract semantics for temporal logic: A direct approach to design and implementation. In *Proceedings of LPAR 2010*, Lecture Notes in Artificial Intelligence 6355, pages 27–45. Springer, 2010.
4. S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Acceleration from theory to practice. *International Journal on Software Tools for Technology Transfer*, 10(5):401–424, 2008.
5. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth ACM Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96. ACM Press, 1978.
7. S.-J. Craig and M. Leuschel. A compiler generator for constraint logic programs. In M. Broy and A. V. Zamulin, editors, *5th Ershov Memorial Conference on Perspectives of Systems Informatics, PSI 2003*, Lecture Notes in Computer Science 2890, pages 148–161. Springer, 2003.
8. G. Delzanno and A. Podelski. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer*, 3(3):250–270, 2001.

9. J. Esparza. Decidability of model checking for infinite-state concurrent systems. *Acta Informatica*, 34(2):85–107, 1997.
10. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
11. F. Fioravanti, A. Pettorossi, and M. Proietti. Automated strategies for specializing constraint logic programs. In *Proceedings of LOPSTR '00*, Lecture Notes in Computer Science 2042, pages 125–146. Springer-Verlag, 2001.
12. F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite state systems by specializing constraint logic programs. In *Proceedings of VCL'01*, Technical Report DSSE-TR-2001-3, pages 85–96. University of Southampton, UK, 2001.
13. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Program specialization for verifying infinite state systems: An experimental evaluation. In *Proceedings of LOPSTR 2010*, Lecture Notes in Computer Science Vol. 6564, pages 164–183. Springer, 2011.
14. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005*, Lecture Notes in Computer Science 3414, pages 258–273. Springer, 2005.
15. L. Fribourg. Constraint logic programming applied to model checking. In A. Bossi, editor, *Proceedings of the 9th International Workshop on Logic-based Program Synthesis and Transformation (LOPSTR '99), Venezia, Italy*, Lecture Notes in Computer Science 1817, pages 31–42. Springer-Verlag, 2000.
16. J. P. Gallagher. Tutorial on specialisation of logic programs. In *Proceedings of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '93, Copenhagen, Denmark*, pages 88–98. ACM Press, 1993.
17. T. J. Hickey and D. A. Smith. Towards the partial evaluation of CLP languages. In *Proceedings of the 1991 ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation, PEPM '91, New Haven, CT, USA*, SIGPLAN Notices, 26, 9, pages 43–51. ACM Press, 1991.
18. J. Jaffar, M. Maher, K. Marriott, and P. Stuckey. The semantics of constraint logic programming. *Journal of Logic Programming*, 37:1–46, 1998.
19. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
20. LASH. homepage: http://www.montefiore.ulg.ac.be/∼boigelot/research/lash.
21. M. Leuschel and M. Bruynooghe. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4&5):461–515, 2002.
22. M. Leuschel, B. Martens, and D. De Schreye. Controlling generalization and polyvariance in partial deduction of normal logic programs. *ACM Transactions on Programming Languages and Systems*, 20(1):208–258, 1998.
23. M. Leuschel and T. Massart. Infinite state model checking by abstract interpretation and program specialization. In *Proceedings of LOPSTR '99*, Lecture Notes in Computer Science 1817, pages 63–82. Springer, 2000.
24. J. W. Lloyd and J. C. Shepherdson. Partial evaluation in logic programming. *Journal of Logic Programming*, 11:217–242, 1991.
25. MAP. The MAP transformation system.
Available from http://www.iasi.cnr.it/∼proietti/system.html, 1995–2010.

26. C. Ochoa, G. Puebla, and M. V. Hermenegildo. Removing superfluous versions in polyvariant specialization of prolog programs. In *Proceedings of LOPSTR '05*, Lecture Notes in Computer Science 3961, pages 80–97. Springer, 2006.
27. J. C. Peralta and J. P. Gallagher. Convex hull abstractions in specialization of CLP programs. In *Proceedings of LOPSTR '02*, Lecture Notes in Computer Science 2664, pages 90–108. Springer, 2003.
28. T. Yavuz-Kahveci and T. Bultan. Action Language Verifier: An infinite-state model checker for reactive software specifications. *Formal Methods in System Design*, 35(3):325–367, 2009.

Finding Partitions of Arguments with Dung's Properties via SCSPs

Stefano Bistarelli[1,2], Paola Campli[3], and Francesco Santini[1]

[1] Dipartimento di Matematica e Informatica, Università di Perugia, Italy
[bista,francesco.santini]@dmi.unipg.it
[2] Istituto di Informatica e Telematica (CNR), Pisa, Italy
[stefano.bistarelli]@iit.cnr.it
[3] Dipartimento di Scienze, Università G.d'Annunzio di Chieti-Pescara, Italy
campli@sci.unich.it

Abstract. Forming coalition structures allows agents to join their forces to achieve a common task. We suggest it would be interesting to look for homogeneous groups which follow distinct *lines of thought*. For this reason, we extend the Dung Argumentation Framework in order to deal with coalitions of arguments. The initial set of arguments is partitioned into subsets (or coalitions). Each coalition represents a different line of thought, but all the found coalitions show the same property inherited by Dung, e.g. all the coalitions in the partition are admissible (or conflict-free, complete, stable). Some problems in weighted argumentation are NP complete; we use (soft) constraints as a formal approach to reason about coalitions and to model all these problems in the same framework. Semiring algebraic structures can be used to model different optimization criteria for the obtained coalitions. To implement this mapping and practically find its solutions we use JaCoP, a Java constraint solver, and we test the code over a small-world network.

1 Introduction and Motivations

A coalition structure is a temporary alliance or partnering of groups in order to achieve a common purpose. Forming coalitions with other members of similar values, interests and goals, allow agents to combine their resources and become more powerful than when they each acted alone [12]. To form a successful coalition, the recognition of compatible interests and common *lines of thought* is needed, since the goal of different agents can be shared by multiple parties.

The abstract nature of Dung's seminal theory [9] of argumentation accounts for its widespread application for various species of non-monotonic reasoning. A Dung argumentation framework (see Sect. 2) is classically instantiated by arguments and a binary conflict based attack relation, defined by some underlying logical theory. The justified arguments under different extensional semantics (e.g. conflict-free ones) are then evaluated, and the claims of these arguments define the inferences of the underlying theory. The aim of this paper is to partition a set of arguments into coalition structures of arguments [8, 1, 6]. A classical scenario

could be represented by the need to aggregate a set of distinct arguments into different lines of thought. Suppose, for example, to have some statements belonging to candidates of different political parties; it would be interesting to check how consistent their ideas are. For example, "We do not want immigrants with the right to vote" is clearly closer to "Immigration must be stopped", than to "We need a multicultural and open society in order to enrich the life of everyone and boost our economy". In general, cooperating groups, referred to as coalition structures [16], have been thoroughly investigated in AI and Game Theory and have proved to be useful in both real-world economic scenarios and Multi-agent Systems [16, 19, 2]. The basic idea behind this work is to start from a single set of arguments and partition them to several agents, with the condition that each subset has to show the same properties defined by Dung, e.g. admissibility [9]. Some applications might be task allocation problem (let tasks be the agents), sensor network problems (agents must form groups), distributed winner determination in combinatorial auctions, agents grouping to handle work-flows (just-in-time incorporation) [16, 19, 2]. In order to model and solve the proposed extended problems we use *(Soft) Constraint Programming* (*(S)CP*) [18] (see Sect. 3), which is a powerful paradigm for solving combinatorial problems that draws on a wide range of techniques from AI, Databases, Programming Languages, and Operations Research [18]. The idea of the semiring-based constraint formalism presented in [4, 3] was to further extend the classical constraint notion by adding the concept of a structure representing the levels of satisfiability of the constraints. Such a structure is similar to a semiring (see Sec. 3). Problems defined according to the semiring-based framework are called *Soft Constraint Satisfaction Problems* (SCSPs) [4, 3, 18]. There already exist many efficient techniques, as constraint propagation [18], to solve such complex problems. The solution of the obtained SCSP represents the partition of the arguments (see Sec. 4) where each subset (i.e. coalition) of arguments has the same property originally defined by Dung in [9], e.g. each coalition in the partition is admissible. Semirings can be used to relax conflict-free partitions, by allowing a certain degree of conflicts inside the coalitions, by representing a weight (or preference) associated with each attack between arguments (see Sec. 5 - 6). At last (in Sec. 7), we show an implementation of a crisp CSP (equivalent to use a *Boolean* semiring in SCSPs) with the *Java Constraint Programming* solver (*JaCoP*) [15] and we test it over a small-world network randomly generated with the *Java Universal Network/Graph Framework* (*JUNG*) [17].

2 Dung Argumentation

Dung proposed an abstract framework for argumentation in which he focuses on the definition of the status (*attacked* / *defended*) of arguments [9]. It can be assumed that a set of arguments and the different conflicts among them are given.

Definition 1 ([9]). *An Argumentation Framework (AF) is a pair* $\langle \mathcal{A}, R \rangle$ *of a set* $\mathcal{A}$ *of arguments and a binary relation* R *on* $\mathcal{A}$ *called the attack relation.*

$\forall a_i, a_j \in \mathcal{A}$, $a_i \, R \, a_j$ means that a_i attacks a_j. An AF may be represented by a directed graph (the interaction graph) whose nodes are arguments and edges represent the attack relation. A set of arguments $\mathcal{B}$ attacks an argument a if a is attacked by an argument of $\mathcal{B}$. A set of arguments $\mathcal{B}$ attacks a set of arguments $\mathcal{C}$ if there is an argument $b \in \mathcal{B}$ which attacks an argument $c \in \mathcal{C}$.

Fig. 1: A classical AF using weather forecast; e.g. b attacks c and viceversa.

In Fig. 1 we show an example of AF represented as an *interaction graph*. Dung [9] gave several semantics of "acceptability", which produce none, one or several acceptable sets of arguments, called extensions. The stable semantics is only defined via the notion of attacks:

Definition 2 ([9]). *A set $\mathcal{B} \subseteq \mathcal{A}$ is conflict-free iff for no two arguments a and b in $\mathcal{B}$, a attacks b. A conflict-free set $\mathcal{B} \subseteq \mathcal{A}$ is a stable extension iff each argument not in $\mathcal{B}$ is attacked by an argument in $\mathcal{B}$.*

The other semantics for "acceptability" rely upon the concept of defense. An admissible set of arguments according to Dung must be a conflict-free set which defends all its elements. Formally:

Definition 3 ([9]). *An argument b is defended by a set $\mathcal{B} \subseteq \mathcal{A}$ (or $\mathcal{B}$ defends b) iff for any argument $a \in \mathcal{A}$, if a attacks b then $\mathcal{B}$ attacks a. A conflict-free set $\mathcal{B} \subseteq \mathcal{A}$ is admissible iff each argument in $\mathcal{B}$ is defended by $\mathcal{B}$*

Besides the stable semantics, one semantics refining admissibility has been introduced by Dung [9].

Definition 4 ([9]). *An admissible $\mathcal{B} \subseteq \mathcal{A}$ is a complete extension iff each argument which is defended by $\mathcal{B}$ is in $\mathcal{B}$.*

In Fig. 2 we show an example of a stable (A), admissible (B) but not complete (due to x_6) and complete (C) extension.

3 Semirings and Soft Constraints

A semiring [4, 3] S is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ where A is a set with two special elements $\mathbf{0}, \mathbf{1} \in A$ (respectively the bottom and top elements of A) and with two operations $+$ and $\times$ that satisfy certain properties: $+$ is defined over (possibly infinite) sets of elements of A and is commutative, associative and idempotent;

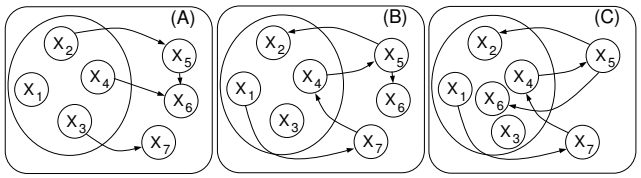


Fig. 2: A stable (A), an admissible (B) and a complete (C) extension (clearly also conflict-free).

it is closed, **0** is its unit element and **1** is its absorbing element; $\times$ is closed, associative, commutative and distributes over $+$, **1** is its unit element and **0** is its absorbing element (for the exhaustive definition, please refer to [4]). The $+$ operation defines a partial order $\leq_S$ over A such that $a \leq_S b$ iff $a+b=b$; we say that $a \leq_S b$ if b represents a value *better* than a. Moreover, $+$ and $\times$ are monotone on $\leq_S$, **0** is its min and **1** its max, $\langle A, \leq_S \rangle$ is a complete lattice and $+$ is its lub. A *soft constraint* [4, 3] may be seen as a constraint where each instantiation of its variables has an associated preference. Given $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a finite domain D, a soft constraint is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. Using this notation $\mathcal{C} = \eta \rightarrow A$ is the set of all possible constraints that can be built starting from S, D and V. Any function in $\mathcal{C}$ depends on the assignment of only a finite subset of V. For instance, a binary constraint $c_{x,y}$ over variables x and y, is a function $c_{x,y} : V \rightarrow D \rightarrow A$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$ (the *support*, or *scope*, of the constraint). Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$. Notice that $c\eta$ is the application of a constraint function $c : V \rightarrow D \rightarrow A$ to a function $\eta : V \rightarrow D$; what we obtain is a semiring value $c\eta = a$. $\bar{a}$ represents the constraint functions associating a to all assignments of domain values. Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times c_2\eta$ [4, 3]. The $\otimes$ builds a new constraint which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples. Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* [4, 3] of c over $V - \{v\}$, written $c \Downarrow_{(V \setminus \{v\})}$ is the constraint c' such that $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support.

An SCSP [3] is defined as $P = \langle C \rangle$ where C is the set of constraints. The *best level of consistency* notion defined as $blevel(P) = Sol(P) \Downarrow_{\emptyset}$, where $Sol(P) = \bigotimes C$ [3]. A problem P is α-consistent if $blevel(P) = \alpha$ [3]; P is instead simply "consistent" iff there exists $\alpha >_S \mathbf{0}$ such that P is α-consistent [3]. P is inconsistent if it is not consistent.

An SCSP Example. Figure 3 shows a weighted SCSP as a graph: the *Weighted* semiring is used, i.e. $\langle \mathbb{R}^+ \cup \infty, \min, \hat{+}, \infty, 0 \rangle$ ($\hat{+}$ is the arithmetic plus operation). Variables and constraints are represented respectively by nodes and arcs (unary for c_1 and c_3, and binary for c_2), and semiring values are written to the right of each tuple, $D = \{a, b\}$. The solution of the CSP in Fig. 3 associates

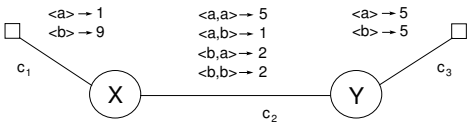


Fig. 3: An SCSP based on a Weighted semiring.

a semiring element to every domain value of variables X and Y by combining all the constraints together, i.e. $Sol(P) = \bigotimes C$. For instance, for the tuple $\langle a, a\rangle$ (that is, $X = Y = a$), we have to compute the sum of 1 (which is the value assigned to $X = a$ in constraint c_1), 5 (which is the value assigned to $\langle X = a, Y = a\rangle$ in c_2) and 5 (which is the value for $Y = a$ in c_3). Hence, the resulting value for this tuple is 11. For the other tuples, $\langle a, b\rangle \rightarrow 7$, $\langle b, a\rangle \rightarrow 16$ and $\langle b, b\rangle \rightarrow 16$. The *blevel* for the example in Fig. 3 is 7, related to the solution $X = a$, $Y = b$.

4 Extending Dung Argumentation to Coalitions

Given the set of arguments $\mathcal{A}$, the problem of coalition formation consists in selecting an appropriate partition of $\mathcal{A}$, $\mathcal{G} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$ ($|\mathcal{G}| = |\mathcal{A}|$ if each argument forms a coalition on its own), such that $\bigcup_{B_i \in G} B_i = \mathcal{A}$ and $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$, if $i \neq j$; clearly, $\forall i.\mathcal{B}_i \neq \emptyset$. In this section we extend Dung's semantics (see Sec. 2) in order to deal with a partition of arguments, that is, we cluster the arguments into different subsets representing *distinct lines of thought*. An example representing the difference between the original framework [9] and our extension is illustrated in Fig. 4: Fig. 4 (A) represents a conflict-free extension as described in Def. 3, while Fig. 4 (B) represents a conflict-free partition of coalitions, since each coalition is conflict-free (see Def. 5). Thus, while in Dung it is sufficient to find only one set with the conflict-free property, we want to find a set of conflict-free sets that represents a partition of the given arguments; we can compute partitions by considering the other properties as well, i.e. admissible, complete and stable semantics. Notice that, in general, we can have a combinatorial number of partitions for a given set of arguments [7, 16]. For example, instead of $P_1 = \{\{x_1, x_2, x_3\}, \{x_4, x_5, \}, \{x_6, x_7, x_8, x_9\}\}$ we can have $P_2 = \{\{x_1, x_2, x_3, x_4\}, \{x_5\}, \{x_6, x_7, x_8, x_9\}\}$. We can have 21147 different partitions for the 9 elements in Fig. 4 (B): this number is called *Bell Number* and is recursively computed as $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$ [7] (with $B_0 = B_1 = 1$). Clearly, not all of these partition are (e.g.) conflict-free.

In the following, we extend the definitions given in Sec. 2 to deal with coalitions.

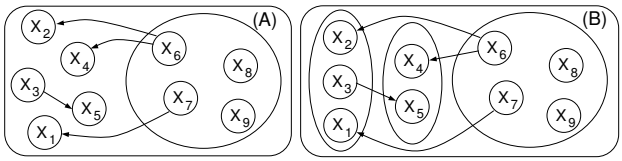


Fig. 4: Differences between classical Dung AF (A) and the extended partitioned framework (B).

Definition 5. *A partition of coalitions* $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ *is* ***conflict-free*** *iff for each* $\mathcal{B}_i \in \mathcal{G}$*,* $\mathcal{B}_i$ *is conflict-free, i.e.* $\forall a, b \in \mathcal{B}_i.(a, b) \notin R$*: no attacking arguments inside the same coalition.*

From the argumentation theory point of view, finding a conflict-free partition of coalitions corresponds to partitioning the arguments into coherent subsets, in order to find feasible lines of thought which do not internally attack themselves. Now we revise the concept of attack/defence among coalitions and arguments and the notion of stable partitions of coalitions:

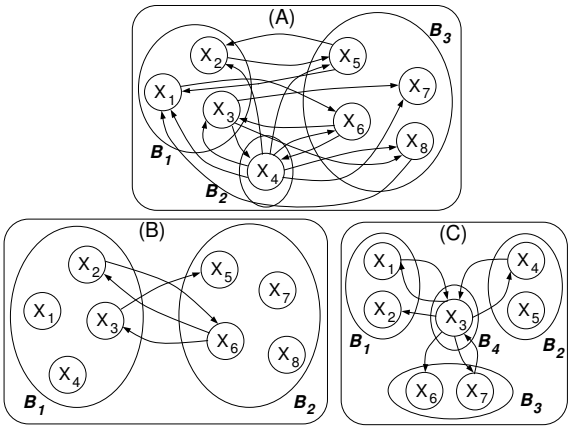


Fig. 5: A stable (A), an admissible and complete (B) and an admissible but not complete (C) partitions of coalitions.

Definition 6. *A coalition* $\mathcal{B}_i$ ***attacks*** *another coalition* $\mathcal{B}_j$ *if one of its elements attacks at least one element in* $\mathcal{B}_j$*, i.e.* $\exists a \in \mathcal{B}_i, b \in \mathcal{B}_j$ *s.t.* $a\,R\,b$*.* $\mathcal{B}_i$ ***defends*** *an attacked argument* a*, e.g.* $b\,R\,a$*, if* $\exists c \in \mathcal{B}_i$ *s.t.* $c\,R\,b$*.*

Definition 7. *A* conflict-free *partition* $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ *is* ***stable*** *iff for each coalition* $\mathcal{B}_i \in \mathcal{G}$*, all its elements* $a \in \mathcal{B}_i$ *are attacked by all the other coalitions* $\mathcal{B}_j$ *with* $j \neq i$*, i.e.* $\forall a \in \mathcal{B}_i, \exists b \in \mathcal{B}_j . b\,R\,a$ $(\forall j \neq i)$*.*

Fig. 5 (A) represents a stable partition: each argument in $\mathcal{B}_2$ (i.e. x_4) is attacked by at least one argument in $\mathcal{B}_1$ (i.e. x_3) and one argument in $\mathcal{B}_3$ (i.e. x_6), and the same also holds for the arguments in $\mathcal{B}_2$ and $\mathcal{B}_3$. To have a stable partition means that each of the arguments cannot be moved from one coalition to another without inducing a conflict in the new coalition. In the next two definitions we respectively extend the concept of admissible and complete extensions.

However different definitions for stable partitions can also be defined, for instance one could require that each argument has to be attacked by some (rather than all of the) other coalitions.

Definition 8. *A* conflict-free *partition* $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ *of coalitions is* ***admissible*** *iff for each argument* $a \in \mathcal{B}_i$ *attacked by* $b \in \mathcal{B}_j$ *(i.e.* $b\,R\,a$*),* $\exists c \in \mathcal{B}_i$ *that attacks* $b \in \mathcal{B}_j$ *(i.e.* $c\,R\,b$*), that is each* $\mathcal{B}_i$ *defends all its arguments.*

According to Dung's definition of admissible extension, *"the set of all arguments accepted by a rational agent is a set of arguments which can defend itself against all attacks on it"* [9]. Notice that if only one argument a in the interaction graph has no grandparents, it is not possible to obtain even one admissible partition: no argument in $\mathcal{A}$ is able to defend a. In Def. 8, we have naturally extended the definition of admissible extension [9] to coalitions: since each coalition represents the line of thought of an agent, each rational agent is able to defend its line of thought because it counter-attacks all its attacking lines.

Fig. 5 (B) represents an admissible partition as it is conflict-free and both $\mathcal{B}_1$ and $\mathcal{B}_2$ defend themselves: x_5 is defended by x_6 and for the attack performed by $x_6 \in \mathcal{B}_2$, x_2 and x_3 are defended by x_2.

Definition 9. *An admissible partition* $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ *is a* ***complete*** *partition of coalitions iff each argument* a *which is defended by* $\mathcal{B}_i$ *is in* $\mathcal{B}_i$ *(i.e.* $a \in \mathcal{B}_i$*).*

Fig. 5 (B) is a complete partition because all the elements defended by $\mathcal{B}_2$ (i.e. x_5, x_8) belong to $\mathcal{B}_2$ and all elements defended by $\mathcal{B}_1$ (x_2, x_3) belong to $\mathcal{B}_1$. Figure 5 (C) represents an admissible but not complete partition because x_6 is defended also by coalitions $\mathcal{B}_1$ (via x_1) and $\mathcal{B}_2$ (via x_4) but belongs to $\mathcal{B}_3$ (defending it via x_7). Intuitively, the notion of complete partition captures the rational agents who believe in every argument they can defend [9].

In Th. 1 we prove that each of the coalitions in every possible conflict-free partition is a conflict-free extension as defined by Dung [9]. Respectively, we can prove the same property for admissible, complete and stable partitions.

Theorem 1. *Given an AF* $\langle \mathcal{A}, R \rangle$ *as in Def. 1 and*

- *given the set of all* CFE *conflict-free extensions which can be obtained over an interaction graph by using Dung's semantics [9] (see also Sec. 2), each* CFP *conflict-free partition as defined in Def. 5 is a subset of them, i.e.* $CFP \subseteq CFE$*.*
- *given the set of all* AE *admissible extensions [9], each* AP *admissible partition as defined in Def. 8 is a subset of them, i.e.* $AP \subseteq AE$*.*

- *given the set of all CE complete extensions [9], each CP complete partition as defined in Def. 9 is a subset of them, i.e. $CP \subseteq CE$.*
- *given the set of all SE stable extensions [9], each SP stable partition as defined in Def. 7 is a subset of them, i.e. $SP \subseteq SE$.*

We can now define the hierarchy of the set inclusions among the proposed partitions like Dung has shown for set inclusions among classical extensions [9]:

Theorem 2. *Given the $CFPS$, APS, CPS and SPS respectively the set of all conflict-free, admissible, complete and stable partitions, we have that $SPS \subseteq CPS \subseteq AS \subseteq CFPS$.*

These two theorems can be proved by reasoning on the sets of classical extensions defined in [9]: the partitions, as defined in this paper, directly inherit their properties. Notice that since our aim is to find partitions and not classical extensions, it is possible that, given the same set of arguments, a stable (for example) extension exists, but a stable partition may not be possible. Let us consider the following example: $A = \{a, b, c, d, e\}$ and $R = \{(b, c), (c, d), (d, e), (e, b)\}$. According to Dung's stable semantics, this framework has two stable extensions: $\{a, b, d\}$ and $\{a, c, e\}$; however, it has no stable partition since the argument a is not attacked and it cannot be in two sets. Even if the situation in which more agents agree about an argument might be possible in several scenarios, we want an argument to be held by exactly one agent, that is, the one who first declared it. This is not a limitation, because our goal is to simultaneously form distinct stable extensions within the same set of arguments, which represent different lines of thought to be assigned to different agents. An application in the real world corresponds to the partitioning of arguments to find the difference among political parties. Indeed, even if an argument might be put forward by several political parties, it is necessary that this argument belongs only to one coalition.

5 Weighted Partitions

Weighted AFs extend Dung's AFs by adding weight values to every edge in the attack graph, intuitively corresponding to the strength of the attack, or equivalently, how reluctant we would be to disregard it [5, 10]. In this section we define a quantitative framework where attacks have an associated preference/weight and, consequently, also the computation of the coalitions as presented in this paper has an associated weight representing the level of inconsistency we tolerate in the solution: more specifically, "how much conflict" we tolerate inside a conflict-free partition, which can now include attacking arguments in the same coalition. Modeling this kind of problems as SCSPs (see Sec. 3) leads to a partition that optimizes the criteria defined by the chosen semiring, which is used to mathematically represent the attack weights.

Fig. 6 represents weighted attack relationships among arguments; in this example $\mathcal{A} = \{a, b, c\}$, $a\,R\,b$ and $c\,R\,b$, moreover, each of these two attack relationships is associated with a fuzzy weight (in $[0, 1]$) representing the strength

of the attack: a attacks b with more strength (i.e. 0.5) than c attacks b (i.e. 0.9). In this case 0 represents the strongest possible attack and 1 the weakest one.

Many other classical weighted AFs in literature can be modeled with semirings [5]. An argument can be seen as a chain of possible events that makes the hypothesis true. The credibility of a hypothesis can then be measured by the total probability that it is supported by arguments. The proper semiring to solve this problem consists in the *Probabilistic* semiring [3]: $\langle [0..1], max, \hat{\times}, 0, 1\rangle$, where the arithmetic multiplication (i.e. $\hat{\times}$) is used to compose the probability values together (assuming that the probabilities being composed are independent). The Fuzzy Argumentation [5] approach enriches the expressive power of the classical argumentation model by allowing to represent the relative strength of the attack relationships between arguments, as well as the degree to which arguments are accepted. In this case, the *Fuzzy* semiring $\langle [0..1], max, min, 0, 1\rangle$ can be used (e.g. in Fig. 6). In addition, the *Weighted* semiring $\langle \mathbb{R}^+ \cup \infty, min, \hat{+}, \infty, 0\rangle$, where $\hat{+}$ is the arithmetic plus ($\mathbf{0} = \infty$ and $\mathbf{1} = 0$), can model the (e.g. money) cost of the attack: for example, the number of votes in support of the attack [10]. By using the *Boolean* semiring $\langle \{true, false\}, \vee, \wedge,\ false, true\rangle$ we can cast the classic AF originally defined by Dung [9] in the same semiring-based framework ($\mathbf{0} = false, \mathbf{1} = true$). The implementation in Sec. 7 models the use of a *Boolean* semiring, since it adopts crisp constraints. Definition 10 rephrases the notion of AF given by Dung (see Sec. 2) into *semiring-based AF*, i.e. an AF_S:

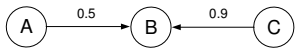


Fig. 6: A fuzzy Argumentation Framework with fuzzy scores modeling the attack strength.

Definition 10 ([5]). *A semiring-based Argumentation Framework (AF_S) is a quadruple $\langle \mathcal{A}, R, W, S\rangle$, where S is a semiring $\langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$, $\mathcal{A}$ is a set of arguments, R the attack binary relation on $\mathcal{A}$, and $W : \mathcal{A} \times \mathcal{A} \longrightarrow A$ a binary function called the* weight *function. Given $a, b \in \mathcal{A}$, $\forall (a, b) \in R$, $W(a, b) = s$ means that a attacks b with a strength level $s \in A$.*

In Def. 11 we define the notion of α-conflict-free partition: conflicts inside the same coalition can be now part of the solution until a cost threshold α is met, and not worse:

Definition 11. *Given a semiring-based AF_S, a partition of coalitions $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ is α-conflict-free for AF_S iff* $\prod_{\forall \mathcal{B}_i \in \mathcal{G}. b,c \in \mathcal{B}_i} W(b, c) \geq_S \alpha$ *(the $\prod$ uses the $\times$ of the semiring).*

In Fig. 7 there is an example of a 0.5-conflict-free partition using a *Fuzzy* semiring, i.e. the $\times$ used to compose the weights corresponds to min. Notice that only the attacks within the same coalition are considered: $min(0.6, 0.7, 0.5) = 0.5$.

Proposition 1. *If a partition is α_1-conflict-free, then the same partition is also α_2-conflict-free if $\alpha_1 <_S \alpha_2$.*

For instance, in *Weighted* semirings a 3-conflict-free partitions is also 4-conflict-free. In Def. 12 we extend with weights also the other kinds of partitions.

Definition 12. *Given an AF_S, a partition of coalitions $\mathcal{G} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n\}$ can be defined as α-stable (or α-admissible or α-complete) by only replacing conflict-free partitions with α-conflict-free partitions in Def. 7 (or Def. 8 or Def. 9).*

In Prop. 2 we relate the weighted partitions with those not weighted presented in Sec. 4.

Proposition 2. *Iff a partition is* **1***-conflict-free (or* **1***-stable,* **1***-admissible,* **1***-complete), then the same partition is also conflict-free (or stable, admissible, complete) as shown in Sec. 4.*

As a proof sketch, no attacks are present in the same coalition since **1** means "no attack", being the top element of the semiring.

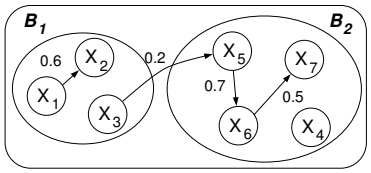


Fig. 7: A 0.5-conflict-free partition by using the *Fuzzy* semiring, i.e. $min(0.6, 0.7, 0.5) = 0.5$. The attack between x_3 and x_5 is not considered since they belong to different coalitions.

6 Mapping Partition Problems to SCSPs

In this section we show a mapping from the AF_S extended to coalitions (see Sec. 5) to SCSPs (see Sec. 3), i.e. $\mathcal{M} : AF_S \rightarrow SCSP$. $\mathcal{M}$ is described as follows: given an AF_S as described in Sec. 5, we define a variable for each argument $a_i \in \mathcal{A}$, i.e. $V = \{a_1, a_2, \ldots, a_n\}$. The value of a variable represents the coalition to which argument a_i belongs: i.e. each variable domain is $D = \{1, n\}$. For example if $a_1 = 2$ it means that the first argument belongs to the second coalition. We can have a maximum of n coalitions, that is all singletons.

In the following explanation, "b attacks a" means that b is a parent of a in the corresponding interaction graph, and "c attacks b attacks a" means that c is a grandparent of a. For the following constraint classes we consider a $AF_S = \langle \mathcal{A}, R, W, S \rangle$ where $S = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and $s \in A$:

1. **Conflict-free constraints.** Since we want to find an α-conflict-free partition, if $a_i \, R \, a_j$ and $W(a_i, a_j) = s$ we need to assign a s preference to the solution that includes both a_i and a_j in the same coalition of the partition: $c_{a_i,a_j}(a_i = k, a_j = k) = s$. Otherwise $c_{a_i,a_j}(a_i = k, a_j = l) = \mathbf{1}$ (with $l \neq k$).

2. **Admissible constraints.** For the admissibility of a partition, if a_i has several grandparents $a_{g1}, a_{g2}, \ldots, a_{gk}$ the parent a_f, we need to add a $k+1$-ary constraint $c_{a_i,a_{g1},\ldots,a_{gk}}(a_i = h, a_{g1} = j_1, \ldots, a_{gk} = j_k) = \mathbf{0}$ if $\forall j_i, j_i \neq h$ ($\mathbf{1}$ otherwise). This is because at least a grandparent must be taken in the same coalition, in order to defend a_i from his parent a_f. Notice that, if an argument is not attacked (i.e. has no parents), it can be taken or not in any admissible set. Moreover, if a_i has a parent but no grandparents, it is not possible to find any admissible partition, that is the SCSP is inconsistent (see Sec. 3).
3. **Complete constraints.** If we have an argument a_i with multiple grandchildren $a_{s1}, a_{s2}, \ldots, a_{sk}$, we need to add the constraint $c_{a_i,a_{s1},\ldots,a_{sk}}(a_i = j, a_{s1} = j, \ldots, a_{sk} = j) = \mathbf{1}$ ($\mathbf{0}$ otherwise). In words, if a_i is taken in a coalition j, all of its grandchildren must be included in the same coalition because j has to include all the defended arguments.
4. **Stable constraints.** They can be represented with a constraint such that for each pair of arguments a_i, a_j belonging to two different coalitions, respectively k and z, at least one of the attacks to a_j has to come from an argument in coalition k: if $b_1, b_2, \ldots b_n$ are all the arguments that attack a_j, $c_{a_i=k,a_j\neq k,b_1,b_2,\ldots,b_n}((b_1 = k) \vee (b_2 = k) \vee \cdots \vee (b_n = k)) = \mathbf{1}$ ($\mathbf{0}$ otherwise). Therefore, we model stable constraints with disjunctive constraints, which are difficult to solve.

Notice that in $\mathcal{M}$ only conflict-free constraints are soft in the strict sense, while the other constraints are associated with $\mathbf{0}$ (not admitted) or $\mathbf{1}$ (admitted) values of the semiring set.

Theorem 3 (Solution equivalence). *Given an $AF_S = \langle \mathcal{A}, R, W, S \rangle$, the solutions of the related SCSP obtained with the mapping $\mathcal{M}$ correspond to:*
- *all the α-conflict-free partitions of coalitions by using conflict-free constraints;*
- *all the α-stable partitions by using stable and conflict-free constraints;*
- *all the α-admissible partitions by using admissible, and conflict-free constraints;*
- *all the α-complete partitions by using complete and conflict-free constraints.*

Conflict-free, stable, admissible and complete partitions can be found by searching for $\mathbf{1}$-consistent solutions in the respective problems defined in Th. 3, as defined in Prop. 2. Notice that finding $\mathbf{1}$-conflict-free partitions is equivalent to well-known graph coloring problems which have been deeply studied also in constraint programming [18], where no two adjacent vertices share the same color:

Proposition 3. *The problem of finding a conflict-free partition of coalitions corresponds to finding a vertex-coloring partition of a graph [18], where each node of the same color belongs to the same coalition in a $\mathbf{1}$-conflict-free partition. The minimum number of colors needed to solve the problem corresponds to the minimum number of coalitions in a possible partition.*

In Fig. 8 we can see an example of classical (i.e. the attacks are not weighted) interaction graph. Only for this example we have 15 conflict-free partitions reported in Tab. 1. Among these conflict-free partitions, P_1, P_2, P_3, P_4, P_5 are also admissible partitions and P_1 is also the only one complete and stable partition; these partitions have been obtained with the implementation in Sec. 7.

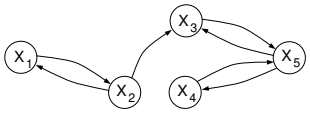


Fig. 8: An interaction graph.

$P_1 = \{\{x_1, x_3, x_4\}, \{x_2, x_5\}\}$	$P_2 = \{\{x_1, x_3, x_4\}, \{x_2\}, \{x_5\}\}$	$P_3 = \{\{x_1, x_3\}, \{x_2, x_4\}, \{x_5\}\}$
$P_4 = \{\{x_1, x_3\}, \{x_2, x_5\}, \{x_4\}\}$	$P_5 = \{\{x_1, x_3\}, \{x_2\}, \{x_4\}, \{x_5\}\}$	$P_6 = \{\{x_1, x_4\}, \{x_2, x_5\}, \{x_3\}\}$
$P_7 = \{\{x_1, x_4\}, \{x_2\}, \{x_3\}, \{x_5\}\}$	$P_8 = \{\{x_1, x_5\}, \{x_2, x_4\}, \{x_3\}\}$	$P_9 = \{\{x_1\}, \{x_2, x_4\}, \{x_3\}, \{x_5\}\}$
$P_{10} = \{\{x_1, x_5\}, \{x_2\}, \{x_3, x_4\}\}$	$P_{11} = \{\{x_1, \}, \{x_2, x_5\}, \{x_3, x_4\}\}$	$P_{12} = \{\{x_1\}, \{x_2\}, \{x_3, x_4\}, \{x_5\}\}$
$P_{13} = \{\{x_1, x_5\}, \{x_2\}, \{x_3\}, \{x_4\}\}$	$P_{14} = \{\{x_1\}, \{x_2, x_5\}, \{x_3\}, \{x_4\}\}$	$P_{15} = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}$

Table 1: The list of all the conflict-free partitions of coalitions for the example in Fig. 8.

7 Implementation in JaCoP

The *Java Constraint Programming* solver [15] (JaCoP) is a Java library which provides a *Finite Domain Constraint Programming* paradigm [18].

To practically develop and test our model, we adopted the *Java Universal Network/Graph Framework* (*JUNG*) [17], a software library for the modeling, generation, analysis and visualization of graphs. Interaction graphs, where nodes are arguments and edges are attacks (see Sec. 2), clearly represent a kind of social network and consequently show the related properties [6]. Therefore, for the following tests we used the *KleinbergSmallWorldGenerator* class [17, 14] in JUNG, which randomly generates a $m \times n$ lattice with small-world properties [14]; each node has 4 local connections and 1 long range connection chosen randomly. An example of such random graphs with 25 nodes is shown in Fig. 9.

In this first implementation we decided to only implement **1**-conflict-free partitions, i.e. we do not consider weights on the attacks, and therefore we only need the crisp constraints of JaCoP. With this tool we can immediately check if a given partition is conflict-free, admissible, complete or stable. Moreover, we can exhaustively generate the partitions with such given properties: since the problem is $O(n^n)$ [16] (where n is the number of arguments) we limit the implementation to a partial search. In particular, we used the *Limited Discrepancy Search* (*LDS*), which is a kind of *Depth First Search* procedure adopting the method proposed in [11]. If a given number of different decisions along a search path is exhausted, then backtracking is initiated [15, 11]. Each time during the search, we select the variable which has most constraints assigned to it and we try the median from its current domain. Moreover, we set a timeout of 60 sec. to

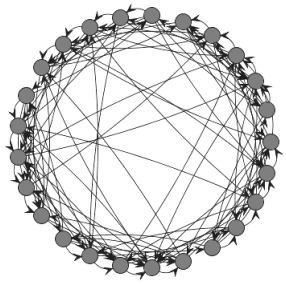


Fig. 9: A small-world network with 25 nodes generated with JUNG by using the *KleinbergSmallWorldGenerator* class [17, 14].

interrupt the search procedure and to report the number of solutions found only in that interval; we ran our experiments over 3 different random graphs with 9, 25 and 100 nodes. The results are shown in Tab. 2: it reports the number of found conflict-free and stable partitions (which limit the number of the other admissible and complete partitions as defined in Th. 2), the number of constraints used to represent the problem and the measured max depth of the search tree. Notice that, within the 60 sec. timeout, the proposed partial search is able to find only one stable partition for 100 nodes; also the reported number of conflict-free solutions in Tab. 2 is less for 100 than for 25 nodes. Therefore, further constraint solving techniques need to be used to improve these performance (left to future work in Sec. 9).

Nodes	Attacks	CFPS	SPS	#constr.	Max Depth
9	45	123	8	∼220	11
25	125	495984	119543	∼1440	61
100	500	92562	1	∼20600	218

Table 2: The test and the related statistics on three different small-world graphs : CFPS and SPS respectively are all the found conflict-free and stable partitions.

Notice that, in order to prevent symmetrically equivalent solutions we have also implemented symmetry breaking constraints for graph coloring as explained in [13] (see Prop. 3 for the analogies): any value permutation is a value symmetry in the coalition assignment of arguments.

8 Related Work and Comparison

The framework of Dung for argumentation is extended by Amgoud in [1] with a preference relation between elements; more in detail, Amgoud [1] provides the semantics (conflict-free, stable and preferred ones) of a coalition structure and a proof theory for testing whether a coalition is in the set of acceptable coalitions. An application of the model is also provided for the problem of task allocation

among partitions of autonomous agents. With respect to the work in this paper, the view in [1] is not focused on generating partitions of arguments, but on directly checking the property of already given coalition structures. Furthermore, [1] has no implementation to practically find solutions, as we instead do in Sec. 7. Moreover, the method to compute the weights of coalitions is not quantitative (but it is only qualitative) and parametric, as we are alternatively able to represent with semirings. In [8] an extension of the *Alternating-time Temporal Logic* (*ATL*) for modeling coalitions through argumentation is presented: a merge between ATL and the coalitional framework is obtained in order to express that agents are able to form a coalition which can successfully achieve a given property; the notions of defence and conflict-free are defined in terms of defeat rather than attack and preferences of arguments are given in a qualitative way (instead of quantitative as in our paper); to compute the desired classes of coalitions a model checker can be used; however, with such techniques, exponential complexity can be hardly faced while constraint programming provides a lot of techniques to tackle combinatorial problems [18]. In [6], social viewpoints (a model for goal based reasoning) are used to argue about coalitions in argumentation theory. The attack relation is based on the goal that agents have to achieve, that is, a coalition attacks another coalition if they share the same goal; this work does not provide a computational framework and only qualitative preferences over arguments are considered. In [5] a common computational and quantitative framework is presented, where attacks (and consequently, also the computation of the classical Dung's semantics) have an associated weight to represent how much inconsistency we tolerate in the solution. Our work extends [5] by considering partitions of arguments and showing an implementation in JaCoP (no implementation is given in [5]) with related tests on small-world graphs. Partitions of arguments implies redefining the whole (argumentation) theory concepts w.r.t [5], e.g. stability.

9 Summary and Future Work

We extended classical argumentation frameworks of [9] to the problem of forming coalitions of arguments, partitioning all the arguments of a given starting set. We redefined the classical definitions of Dung's extensions (conflict-free, admissible, stable and complete ones) in order to consider a partition of all the arguments into multiple coalitions, and modeled the problem of finding such coalitions with SCSPs [4, 3, 18]: this semiring-based formalism can be used to relax the concept of conflict-free partitions in order to allow some inconsistency (i.e. attacks) within the same coalition. The proposed quantitative framework can be used also to solve classical (i.e. crisp) CSPs. We have also solved a problem example considering only $\mathbf{1}$-solutions with JaCoP [15] and then we performed tests on a small-world network randomly generated with [17]. Starting from a single set of arguments, the goal has been to partition it into multiple coalitions with the same features (e.g. stability or admissibility) without discarding any argument. In the future we want to implement α-conflict-free, α-stable, α-admissible and

α-complete partitions in JaCoP, for $\alpha <_S \mathbf{1}$. Moreover, we want to improve the performance obtained in Sec. 7 by testing different solvers and constraint techniques (e.g. by taking the inspiration from [16]).

References

1. L. Amgoud. An argumentation-based model for reasoning about coalition structures. In *ArgMAS05*, volume 4049 of *LNCS*, pages 217–228. Springer, 2005.
2. K. R. Apt and A. Witzel. A generic approach to coalition formation. *CoRR*, abs/0709.0435, 2007.
3. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *LNCS*. Springer, 2004.
4. S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, March 1997.
5. S. Bistarelli and F. Santini. A common computational framework for semiring-based argumentation systems. In *ECAI'10*, volume 215, pages 131–136. IOS Press, 2010.
6. G. Boella, L. van der Torre, and S. Villata. Social viewpoints for arguing about coalitions. In *PRIMA*, volume 5357 of *LNCS*, pages 66–77. Springer, 2008.
7. K. P. Bogart. *Introductory Combinatorics*. Academic Press, Inc., Orlando, FL, USA, 2000.
8. N. Bulling, J. Dix, and C. I. Chesñevar. Modelling coalitions: Atl + argumentation. pages 681–688. IFAAMAS, 2008.
9. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–357, 1995.
10. P. E. Dunne, A. Hunter, P. McBurney, S. Parsons, and M. Wooldridge. Inconsistency tolerance in weighted argument systems. pages 851–858. IFAAMS, 2009.
11. W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995.
12. B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *Knowl. Eng. Rev.*, 19(4):281–316, 2004.
13. G. Katsirelos and T. Walsh. Dynamic symmetry breaking constraints. In *Workshop on Modeling and Solving Problems with Constraints (at ECAI08)*, pages 39–44. Informal Proc., 2008.
14. J. Kleinberg. Navigation in a small world. *Nature*, 406:845, 2000.
15. K. Kuchcinski and R. Szymanek. Jacop - java constraint programming solver, 2001. http://jacop.osolpro.com/.
16. N. Ohta, V. Conitzer, R. Ichimura, Y. Sakurai, A. Iwasaki, and M. Yokoo. Coalition structure generation utilizing compact characteristic function representations. In *CP*, volume 5732 of *LNCS*, pages 623–638. Springer, 2009.
17. J. O'Madadhain, D. Fisher, S. White, and Y. Boey. The JUNG (Java Universal Network/Graph) framework. Technical report, UC Irvine, 2003.
18. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Elsevier Science Inc., NY, USA, 2006.
19. O. Shehory and S. Kraus. Task allocation via coalition formation among autonomous agents. In *IJCAI (1)*, pages 655–661, 1995.

A Tabled Prolog Program for Solving Sokoban

Neng-Fa Zhou[1] and Agostino Dovier[2]

[1] Department of Computer and Information Science,
CUNY Brooklyn College & Graduate Center, USA,
zhou@sci.brooklyn.cuny.edu
[2] Dipartimento di Matematica e Informatica,
Università di Udine, Udine, Italy,
agostino.dovier@uniud.it

Abstract. This paper presents our program in B-Prolog submitted to the third ASP solver competition for the Sokoban problem. This program, based on dynamic programming, treats Sokoban as a generalized shortest path problem. It divides a problem into independent subproblems and uses mode-directed tabling to store subproblems and their answers. This program is very simple but quite efficient. Without use of any sophisticated domain knowledge, it easily solved 11 of the 15 instances used in the competition.

1 Introduction

Sokoban is a type of transport puzzle, in which the player finds a plan for the Sokoban (means warehouse-keeper in Japanese) to push all the boxes into the designated areas. This problem has been shown to be NP-hard and has raised great interest because of its relation to robot motion planning [6]. This problem has been used as a benchmark in the Answer Set Programming competition [5, 4] and International Planning competition[3], and solutions for ASP solvers and PDDL are available. In [15], an IDA*-based program is presented with several domain-dependent enhancements.

This paper presents the program in B-Prolog, called the *BPSolver program* below, submitted to the third ASP solver competition [4]. The BPSolver program is based on the dynamic programming approach and uses mode-directed tabling [18] to store subproblems and their answers. The program was built after failed attempts to use CLP(FD) and the planning languages B [12] and BMV [8] for the problem (see Section 6). The BPSolver program is very simple (only a few lines of code) but quite efficient. In the competition, the BPSolver program solved 11 of the 15 instances of which the hardest instance took only 33 seconds, and failed to solve the remaining four instances due to lack of table space.

As far as we know, the BPSolver program is the first to apply the dynamic programming approach to the Sokoban problem. The BPSolver program treats Sokoban as a generalized shortest path problem where the locations of objects

[3] http://ipc.informatik.uni-freiburg.de/Domains

particular to a subproblem are tabled. The BPSolver program does not employ any sophisticated domain knowledge. It only checks for two simple deadlock cases: one is that a box is stuck in a corner and the other is that two boxes next to each other are stuck by a wall. With sophisticated domain knowledge, the BPSolver program is expected to perform much better.

The remainder of the paper is structured as follows: Section 2 gives a detailed description of the Sokoban problem; Section 3 introduces tabling, and in particular mode-directed tabling, as implemented in B-Prolog; Section 4 explains the BPSolver program line by line; Section 5 presents the competition results; Section 6 compares with related work and points out possible improvements; and Section 7 concludes the paper.

2 The Problem Description

The following is an adapted description of the Sokoban problem used in the ASP solver competition.[4]

Sokoban is a type of transport puzzle invented by *Hiroyuki Imabayashi* in 1980 and published by the Japanese company Thinking Rabbit, Inc. in 1982. "Sokoban" means "warehouse-keeper" in Japanese. The puzzle consists of a maze which has two types of squares: inaccessible *wall squares* and accessible *floor squares*. Several boxes are initially placed on some of the floor squares and the same number of floor squares are designated as storage squares. There is also a man (the Sokoban) whose duty is to move all the boxes to the designated storage squares. A floor square is *free* if it is not occupied by either a box or the man. The man can walk around by moving from his current position to any adjacent free floor square. He can also push a box into an adjacent free square, but in order to do so he needs to be able to get to the free square behind the box. The goal of the puzzle is to find a shortest plan to push all the boxes to the designated storage squares. To reduce the number of steps, the Sokoban moves and the successive sequence of pushes in the same direction are considered as an atomic action.

A problem instance is given by the following relations:

- `right`(L_1, L_2): location L_2 is immediately to the right of location L_1.
- `top`(L_1, L_2): location L_2 is immediately on the top of location L_1.
- `box`(L): location L initially holds a box.
- `sokoban`(L): the man is initially at location L.
- `storage`(L): location L is a storage square.

In this setting, the wall squares are completely ignored and the adjacency relation of the floor squares is given by the `right` and `top` predicates. This input is well suited for Prolog. According to the ASP competition requirements, the output should be represented by atoms of the form `push(`$L1$`,Dir,`L_2`,Time)`, where $L1$ and L_2 are two locations, L_2 is reachable from L_1 going through the direction

[4] https://www.mat.unical.it/aspcomp2011

`Dir` (`left`, `right`, `up`, or `down`), and `Time` is an integer greater than 0 (bounded the further input predicate `step`). For each admissible value of time exactly one `push` action must occur. We also allow a slight variation of this predicate where the time information is left implicit and a consecutive sequence of push is stored in a list that, in fact, represents a plan.

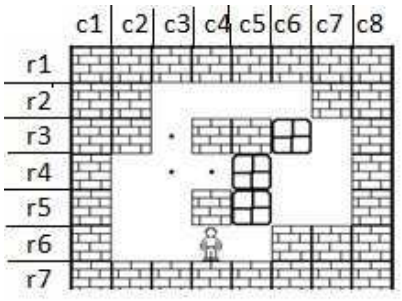


Fig. 1. A Sokoban problem.

Figure 1 shows an example problem where the storage squares have dots on them. This state is represented by the following facts:

```
top(c2r5,c2r4).          right(c3r2,c4r2).       right(c3r6,c4r6).
top(c2r6,c2r5).          right(c4r2,c5r2).       right(c4r6,c5r6).
top(c3r3,c3r2).          right(c5r2,c6r2).
top(c3r4,c3r3).          right(c6r3,c7r3).       box(c6r3).
top(c3r5,c3r4).          right(c2r4,c3r4).       box(c5r4).
top(c3r6,c3r5).          right(c3r4,c4r4).       box(c5r5).
top(c5r5,c5r4).          right(c4r4,c5r4).
top(c5r6,c5r5).          right(c5r4,c6r4).       storage(c3r3).
top(c6r3,c6r2).          right(c6r4,c7r4).       storage(c3r4).
top(c6r4,c6r3).          right(c2r5,c3r5).       storage(c4r4).
top(c6r5,c6r4).          right(c5r5,c6r5).
top(c7r4,c7r3).          right(c6r5,c7r5).       sokoban(c4r6).
top(c7r5,c7r4).          right(c2r6,c3r6).
```

The following gives a plan of 13 steps for the problem:

```
[push(c6r3,down,c6r5),  push(c5r4,left,c3r4),  push(c3r4,down,c3r5),
 push(c5r5,up,c5r4),    push(c6r5,left,c5r5),  push(c5r4,right,c6r4),
 push(c5r5,up,c5r4),    push(c6r4,up,c6r3),    push(c5r4,left,c4r4),
 push(c6r3,down,c6r4),  push(c3r5,up,c3r3),    push(c4r4,left,c3r4),
 push(c6r4,left,c4r4) ]
```

3 Tabling in B-Prolog

Tabling [17] has become a well-known and useful feature of many Prolog systems. The idea of tabling is to memorize answers to tabled subgoals and use the answers

to resolve subsequent variant or subsumed subgoals. This idea resembles the dynamic programming idea of reusing solutions to overlapping sub-problems and, naturally, tabling is amenable to dynamic programming problems.

B-Prolog is a tabled Prolog system that is based on linear tabling [19], allows variant subgoals to share answers, and uses the local strategy [9] (also called lazy strategy [19]) to return answers. In B-Prolog, tabled predicates are declared explicitly by declarations in the following form:

```
:-table P1/N1,...,Pk/Nk
```

where each `Pi` (`i` $\in \{1, \ldots, k\}$) is a predicate symbol and `Ni` is an integer that denotes the arity of `Pi`.

Consider, for example, the tabled predicate computing Fibonacci numbers:

```
:-table fib/2.
fib(0, 1).
fib(1, 1).
fib(N, F):-N>1,
    N1 is N-1,
    N2 is N-2,
    fib(N1, F1),
    fib(N2, F2),
    F is F1+F2.
```

Without tabling, the subgoal `fib(N,X)` would spawn 2^N subgoals, many of which are variants. With tabling, however, the time complexity drops to linear since the same variant subgoal is resolved only once.

For a tabled predicate, all the arguments of a tabled subgoal are used in variant checking and all answers are tabled. This table-all approach is problematic for many dynamic programming problems such as those that require computation of aggregates. Mode-directed tabling [13, 18] amounts to using table modes to instruct the system on how to table subgoals and their answers. In B-Prolog, a table mode declaration takes the following form:

```
:-table p(M1,...,Mn):C.
```

where `p/n` is a predicate symbol, `C`, called a *cardinality limit*, is an integer which limits the number of answers to be tabled for `p/n`, and each `Mi` (`i` $\in \{1, \ldots, k\}$) is a mode which can be `min`, `max`, `+`, or `-`. When `C` is 1, it can be omitted together with the preceding ':'. For each predicate, only one table mode declaration can be given. In the current implementation in B-Prolog, only one argument in a tabled predicate can have the mode `min` or `max`. Since an optimized argument can be a compound term and the built-in `@<`/2 is used to select better answers for compound terms, this restriction is not essential.

The mode `+` is called *input*, `-` *output*, `min` *minimized*, and `max` *maximized*. An argument with the mode `min` or `max` is called *optimized*. An optimized argument is assumed to be output. The system uses only input arguments in variant checking of tabled subgoals, ignoring all other arguments. Notice that a table mode

does not tell the instantiation state of an argument. Nevertheless, normally an input argument is ground and an output argument is a variable.

A mode declaration not only instructs the system on what arguments are used in variant checking, it also guides it in tabling answers. After an answer of a tabled subgoal is produced, the system tables it unconditionally if the cardinality limit is not reached yet. When the cardinality limit has been reached, however, the system tables the answer only if it is better than some existing answer in terms of the optimized argument. If no argument is optimized, all new answers are discarded once the cardinality limit has been reached.

Mode-directed tabling is very useful for declarative description of dynamic programming problems. The following predicate finds a path with the minimal weight between a pair of nodes in a directed graph.

```
:-table sp(+,+,-,min).
sp(X,Y,[(X,Y)],W) :-
    edge(X,Y,W).
sp(X,Y,[(X,Z)|Path],W) :-
    edge(X,Z,W1),
    sp(Z,Y,Path,W2),
    W is W1+W2.
```

The predicate `edge(X,Y,W)` defines a given weighted directed graph, where `W` is the weight of the edge from node `X` to node `Y`. The predicate `sp(X,Y,Path,W)` states that `Path` is a path from `X` to `Y` with the smallest weight `W`. Notice that whenever the predicate `sp/4` is called, the first two arguments are assumed to be instantiated. So for each pair of nodes, only one answer is tabled.

4 The Program

In this section, we explain the BPSolver program. The program treats the Sokoban problem as a generalized shortest path problem. For a state, if it is the goal state in which every box is in a storage location, it is done. Otherwise, the program chooses an intermediate state and splits the problem into two subproblems, one transforming the current state to the intermediate one and the other transforming the intermediate one to the goal state. All the states are tabled so that the same subproblem is solved only once.

4.1 Library and helper predicates

Before we show the program, we give the library and helper predicates used in the program. For each helper predicate written as part of the program, we give its definition.

- `member(`X, L`)`: succeeds when X is a member of the list L. It can be used to check if a given element is a member of a list and it can also be used to nondeterministically select an element from a list.

- `select(`X, L, R`)`: the same as `member(`X, L`)` except that it binds R to the rest of the list after X is selected from L.
- `neib(`$Loc1, Loc2, Dir$`)`: $Loc2$ is the next location of $Loc1$ along the direction Dir. It is defined as follows in terms of the given predicates `top/2` and `right/2`:

```
:-table neib/3.
neib(Loc1,Loc2,up):-top(Loc1,Loc2).
neib(Loc1,Loc2,down):-top(Loc2,Loc1).
neib(Loc1,Loc2,right):-right(Loc1,Loc2).
neib(Loc1,Loc2,left):-right(Loc2,Loc1).
```

  The predicate is tabled for better performance.
- `insert_ordered(X,L1,L2)`: inserts X into a sorted list $L1$, resulting in a new sorted list $L2$.

```
insert_ordered(X,[],[X]).
insert_ordered(X,[Y|Ys],[X,Y|Ys]):-
    X @=<Y,!.
insert_ordered(X,[Y|Ys],[Y|Ordered]):-
    insert_ordered(X,Ys,Ordered).
```

- `goal_reached(L)`: every location in L is a storage location.

```
goal_reached([]).
goal_reached([Loc|Locs]):-
    storage(Loc),
    goal_reached(Locs).
```

  This can be defined equivalently using the `foreach` construct of B-Prolog as follows:

```
goal_reached(Locs):-
    foreach(Loc in Locs, storage(Loc)).
```

- `corner(`Loc`)`: succeeds if Loc is a corner location. No box can be moved to a corner unless it is a storage square.

```
:-table corner/1.
corner(X) :- \+ noncorner(X).
noncorner(X) :- top(_,X),top(X,_).
noncorner(X) :- right(_,X),right(X,_).
```

  This predicate is tabled. For the problem instance shown in Figure 1, for example, the table will contain seven possible subgoals including `corner(c2r4)` and `corner(c3r2)`.
- `stuck(`$Loc1, Loc2$`)`: two boxes in $Loc1$ and $Loc2$ constitute a deadlock if they are next to each other by a wall, unless both locations are storage squares.

```
:-table stuck/2.
stuck(X,Y):-
    (right(X,Y);right(Y,X)),
    (\+ storage(X); \+ storage(Y)),
```

```
    (\+ top(X,_), \+ top(Y,_);
     \+ top(_,X), \+ top(_,Y)),!.
stuck(X,Y):-
    (top(X,Y);top(Y,X)),
    (\+ storage(X); \+ storage(Y)),
    (\+ right(X,_), \+ right(Y,_);
     \+ right(_,X), \+ right(_,Y)),!.
```

For example, for the problem instance shown in Figure 1, the subgoal `stuck(c3r2,c4r2)` succeeds and so does `stuck(c4r2,c3r2)`.

4.2 The main program

As already said, the main idea behind the main program reported in Figure 2 is to implement a tabled version of a generalization of the shortest path problem. The subgoal

```
plan_sokoban(SokobanLoc, BoxLocs, Plan, Len)
```

finds a plan `Plan` with the minimal length `Len` for the current state, where `SokobanLoc` is the location of the man and `BoxLocs` is a list of box locations. For example, for the problem instance shown in Figure 1, the subgoal would look like

```
plan_sokoban(c4r6,[c5r4,c5r5,c6r3],Plan,Len).
```

The predicate is tabled under control by the mode `plan_sokoban(+,+,-,min)`, which means that only one plan with the minimal length is tabled for each different state. The list `BoxLocs` is sorted in lexicographic order to make tabling more effective.

When the goal has been reached (`goal_reached(BoxLocs)` succeeds), an empty plan is returned. Otherwise, the second rule selects a box location `BoxLoc` from `BoxLocs` and a destination location `DestLoc` that can be reached from `BoxLoc` in the direction `Dir` (`up`, `down`, `left`, or `right`), and adds the action `push(BoxLoc,Dir,DestLoc)` into the plan. Only feasible actions are added. An action `push(BoxLoc,Dir,DestLoc)` is feasible if (1) the previous location `PrevNeibLoc` of `BoxLoc` in the direction `Dir` is free; (2) the man can walk to this location (`reachable_by_sokoban`); and (3) the location `DestLoc` is a good destination that does not result in a deadlock. The subgoal `choose_dest` non-deterministically chooses a destination `DestLoc` from the free squares ahead of `BoxLoc` in the direction `Dir`. After pushing the box at `BoxLoc` to `DestLoc`, the man moves to `NewSokobanLoc` which is the previous square of `DestLoc`.

The predicate `reachable_by_sokoban` checks if there is a path of free squares from one location to another. Again, tabling is used to prevent loops and avoid resolving the same subgoal more than once.

The predicate `good_dest` checks whether or not a location is a good destination for a box. A location `Loc` is a good destination if (1) it is not occupied

```
:-table plan_sokoban(+,+,-,min).
plan_sokoban(_SokobanLoc,BoxLocs,Plan,Len):-
    goal_reached(BoxLocs),!,
    Plan=[],Len=0.
plan_sokoban(SokobanLoc,BoxLocs,[push(BoxLoc,Dir,DestLoc)|Plan],Len):-
    select(BoxLoc,BoxLocs,BoxLocs1),
    neib(PrevNeibLoc,BoxLoc,Dir),
    \+ member(PrevNeibLoc,BoxLocs1),
    neib(BoxLoc,NextNeibLoc,Dir),
    good_dest(NextNeibLoc,BoxLocs1),
    reachable_by_sokoban(SokobanLoc,PrevNeibLoc,BoxLocs),
    choose_dest(BoxLoc,NextNeibLoc,Dir,DestLoc,NewSokobanLoc,BoxLocs1),
    insert_ordered(DestLoc,BoxLocs1,NewBoxLocs),
    plan_sokoban(NewSokobanLoc,NewBoxLocs,Plan,Len1),
    Len is Len1+1.

:-table reachable_by_sokoban/3.
reachable_by_sokoban(Loc,Loc,_BoxLocs).
reachable_by_sokoban(Loc1,Loc2,BoxLocs):-
    neib(Loc1,Loc3,_),
    \+ member(Loc3,BoxLocs),
    reachable_by_sokoban(Loc3,Loc2,BoxLocs).

good_dest(Loc,BoxLocs):-
    \+ member(Loc,BoxLocs),
    (corner(Loc)->storage(Loc);true),
    foreach(BoxLoc in BoxLocs, \+ stuck(BoxLoc,Loc)).

choose_dest(Loc,NextLoc,_Dir,Dest,NewSokobanLoc,_BoxLocs):-
    Dest=NextLoc, NewSokobanLoc=BoxLoc.
choose_dest(Loc,NextLoc,Dir,Dest,NewSokobanLoc,BoxLocs):-
    neib(NextLoc,NextNextLoc,Dir),
    good_dest(NextNextLoc,BoxLocs),
    choose_dest(NextLoc,NextNextLoc,Dir,Dest,NewSokobanLoc,BoxLocs).
```

Fig. 2. The main program

by any box; (2) it is not a corner unless it is a storage square; and (3) moving a box to `Loc` does not result in a deadlock. As mentioned above, two boxes next to each other by a wall constitute a deadlock unless the two locations are storage squares. There are more sophisticated deadlock cases that involve more than two locations [15], but these cases are not considered here.

5 The Competition Results

Sokoban was one of the benchmarks of the 2011 ASP competition [4]. The main scope of the competition is to challenge different solvers on declarative encodings. In particular, in the *System Track* different ASP solvers were required to run on a proposed encoding in Answer Set Programming (pure declarative code, no optimization). In the next Section we will briefly sketch this modeling. It is a decision version of the problem where a plan of a given length is looked for. The allowed actions are `push` of a block in the four directions. Moreover, the move of the Sokoban for reaching (if possible) a block is supposed to happen instantaneously immediately before the successive push move. Most of the ASP solvers behave quite well on the proposed instances. It must be observed, however, that the instances were not so large (the more difficult were of 6 boxes/20 moves). In the *Model and Solve* competition, competitors were allowed to encode directly the problem allowing some domain information. In this case a minimum length plan is looked for. Most of the submitted programs are variants of the one proposed for the System Track; for this approach, best performances have been obtained by the family of Clasp solvers [10] (`http://potassco.sourceforge.net/`) and by EZCSP [1] (`http://marcy.cjb.net/ezcsp/index.html`).

The BPSolver program is available at: `www.sci.brooklyn.cuny.edu/~zhou/asp11/` Table 1 gives the CPU times of the actual runs in the third ASP solver competition. In comparison, the result of Clasp, the solver that won this benchmark, is also shown. For the solved instances, BPSolver is actually a little faster than Clasp on average. BPSolver failed to solve four of the instances due to lack of table space.

6 Related Work

The Sokoban problem is a typical planning problem where a set of admissible *actions* might affect the value of some *fluents* that globally determine a *state*. This kind of problems are naturally encoded using Action Description Languages such as STRIPS, B, and PDDL. Before starting the encoding one needs to carefully choose the atomic actions allowed for the Sokoban and their duration.

The simplest choice (finest granularity) is that at each time step the Sokoban is allowed to do a single `move`, or a single `push` of a box, in one of the four direction `up`, `down`, `left`, and `right`; the duration of the move is 1. This is the basic encoding of the Sokoban problem (see, e.g. `http://ipc.informatik.uni-freiburg.de/Domains`); it is simple and elegant and the non deterministic branching in the search is limited to 4. Any state can be represented by 3ℓ

Table 1. Competition results (CPU time, seconds).

Instance	BPSolver	Clasp
1-sokoban-optimization-0-0.asp	0.58	0.06
13-sokoban-optimization-0-0.asp	0.06	0.74
18-sokoban-optimization-0-0.asp	0.00	9.80
20-sokoban-optimization-0-0.asp	33.57	13.24
24-sokoban-optimization-0-0.asp	2.66	3.52
27-sokoban-optimization-0-0.asp	0.78	1.16
29-sokoban-optimization-0-0.asp	0.78	2.92
33-sokoban-optimization-0-0.asp	1.96	26.74
37-sokoban-optimization-0-0.asp	0.38	8.52
4-sokoban-optimization-0-0.asp	Mem Out	0.62
43-sokoban-optimization-0-0.asp	Mem Out	35.67
45-sokoban-optimization-0-0.asp	Mem Out	9.30
47-sokoban-optimization-0-0.asp	Mem Out	18.66
5-sokoban-optimization-0-0.asp	0.00	0.16
9-sokoban-optimization-0-0.asp	0.00	2.12

fluents of the form `free(L)`, `box_in(L)`, `sokoban_in(L)`, where L is one of the ℓ admitted cells. The actions affect these fluents. However, with this encoding, a lot of steps are needed either to push a block without changing directions or to reach the next block to be pushed. This increases the number of steps necessary for the plan and, since the size of the search tree grows exponentially in this number, it is rather difficult to solve non-trivial instances.

As already said, the granularity chosen for the ASP competition is coarser. As soon as there is a path from the Sokoban position to the desired side of a block, the move action is left implicit (it takes zero time). Just a unique family of `push` actions are admitted, parametric on the starting `From` and arrival `To` points of the block (aligned in a given direction `D`). A push of any number of steps in the same direction is viewed as an atomic action. If, on the one side, this allows to dramatically decrease the number of (macro) actions needed for executing the plan, on the other side, it generates new problems. The first is that the branching is now increased. The second, more subtle, is that the modeling language needs to be able to deal with a dynamic notion of reachability.

Basically, for stating that the action `push(From,D,To)` be executable, we need to require that: `box_in(From)`, that all the cells `L` between `From` and `To` are `free`, and, morevoer, that the Sokoban can reach the cell adjacent to `From` in the direction `D`, external to the segment [`From`,`To`]. Let us call `S1` this cell, we need to require that `reachable(S1)` (namely that the cell is currently reachable from the Sokoban).

We need therefore to introduce ℓ additional Boolean fluents `reachable(L)` and to deal with them. This can be done using *static causal laws* (or rules) that are not allowed in all Action Description Languages. We should write two rules of the form (using the syntax of the language B):

```
sokoban_in(A) caused reachable(A).

reachable(B) and free(C) caused reachable(C) if adjacent(B,C).
```

The semantics of an Action Description Language in presence of static causal laws becomes complex [12, 8] and is related to the notion of Answer Set [11]. As shown in detail in [7] B programs can be either

- interpreted using constraint (logic) programming, or
- automatically translated in Answer Set Programming and then solved using an ASP solver.

The former encoding is also studied in a slight different context, by other authors (e.g., [2]); however, static laws are not considered. The encoding implemented in [7] deals with static causal laws, but the proposed implementation does not ensure correctness for some classes of static causal laws. Other encodings are viable, but they would introduce too many constraints. Intuitively, this happens when rules introduce loops of implications between literals. In the case of Sokoban, simultaneous un-justified changes of fluent values might satisfy the constraints, the Sokoban can reach unreachable cells, and not-allowed push moves can be executed. The same problem was pointed out in [16] where authors translated a ground ASP Program into a SAT encoding. In presence of such kind of loops, solutions of the SAT formula obtained are not admissible answer sets. The problem can be avoided introducing the so-called *loop-formulas* but their number can grow exponentially. Unfortunately, the above definition of reachability as static causal laws introduces these undesirable loops and therefore a CLP(FD) approach for solving it in this way (e.g., using B-Prolog) is not feasible.

As far as the latter approach is concerned, it works correctly on a modeling in B based on the ideas above.[5] It is well-known that the main problem of Answer Set Programming is the size of the ground version of the program that is computed in the first stage of the solution's search. We experimentally observe that this size is bigger (typically, twice) than the size of the ASP program written by the Clasp group that won the competition. A direct encoding of this problem in ASP, of course, produces clever code. Let us say some words about this program. First of all, it focuses on 2ℓ Boolean fluents `box` and `sokoban`, repeated for each time step (the fluent free is left implicit). The reachability relation (called `route`) is encoded directly in ASP (in a similar way as done above in B) and it is parametric on time steps. Push move is split into `push_from`, `push`, and `push_to`. This allows us to reduce the grounding. The `push_from` relation is defined to be a function w.r.t. the time step (and defined only if the goal has not yet been reached):

```
1 { push_from(L,D,S): loc(L): direction(D) } 1 :-
                                          step(S), not goal(S).
```

Namely, for each step `S` that in which the goal has not yet been reached, just one move is allowed (one location and one direction are selected). Constraints

[5] The encoding is available in `http://www.dimi.uniud.it/dovier/CLPASP/BBMVLAST`.

are added to ensure action executability. If `push_from` is enabled, then the length of the move is non deterministically chosen and the consequent effects on fluents are determined. Constraints are also added to eliminate useless push moves; this reduces either the size of the corresponding ground program or the search space. As already said, the size of the ground program and the time needed to compute it are strictly less than those needed for the program automatically obtained from B. However, the running time after grounding (both in Clasp) are comparable on the instances of the ASP competition.

In AI literature, Andreas Junghanns and Jonathan Schaeffer in [15] pointed out that the Sokoban problem is interesting for several reasons: in general it is difficult to find a tight lower bound for the number of moves, there is the problem of a deadlock (e.g. when a box is pushed to a corner), and, moreover, the branching factor is very high (considering macro moves). The same authors then published some improving solutions to the problem in the context of single agent planning, summarized in a paper with Adi Botea and Martin Müller [3]. In particular, they exploited an abstraction based on tunnels and rooms of the Sokoban warehouse that allowed to obtain good performances. In [14] the authors show how to develop a domain-independent heuristics for cost-optimal planning. They apply this idea to the Sokoban, and test a STRIPS encoding of the Sokoban on a collection, called "microban", developed by David W. Skinner and available from `http://users.bentonrea.com/~sasquatch/sokoban/`. The STRIPS encoding used is based on the finest granularity approach (simple move), but reachability and other techniques are used as heuristics for sequences of atomic moves. They choose a collection of moderate instances and they are able to solve the 70% of them. Interestingly, they are able to find plans of length 290 (atomic actions) on instance 140 in half of an hour of computation.

Apart from academic contributions to this challenging puzzle we would like to point out two working solvers available on the web:

- Sokoban Puzzle Solver `http://codecola.net/sps/sps.htm`, developed (in 2003–2005) by Faris Serdar Taşel is basically a generate and test solver for Sokoban. An executable file for Windows is downloadable, but extra details on the implementation are not available. In spite of its apparent simplicity, it solves in acceptable time most the instances available from that web page.
- Sokoban Automatic Solver `http://www.ic-net.or.jp/home/takaken/e/soko/index.html`, developed (in 2003–2008) by Ken'ichiro Takahashi is another solver for Windows. It finds solutions that are not ensured to be optimal. It allows two options: (1) brute force (generate and test) and (2) using analysis. The second options allows faster executions but the author gives no idea on how this analysis is performed.

7 Concluding Remarks

This paper has presented the BPSolver program for solving the Sokoban problem. This program has demonstrated for the first time that dynamic programming is

a viable approach to the problem and mode-directed tabling is effective. Without using sophisticated domain knowledge, this program is able to solve some interesting instances that our other program in B, interpreted using CLP(FD) have failed to solve. As shown in the competition results, this program is as competitive as the Clasp program for the instances that are not so memory demanding.

The BPSolver program basically explores all possible states including states that can never occur in an optimal solution: a way for improving it is to consider more deadlock states such as those involving multiple blocks [15] to be filtered out. Moreover, some domain knowledge such as the topological information should be exploited to reduce the graph. Lastly, heuristics should be employed to select a box to move and a destination to move the box to. Ideally, a path that leads to a goal state should be explored as early as possible.

We believe that reasonable sized planning problems can benefit of the same technique presented.

Acknowledgement

We really thank Andrea Formisano for his wise advice in the B encoding of the Sokoban. Neng-Fa Zhou was supported in part by NSF (No.1018006). Agostino Dovier is partially supported by INdAM-GNCS 2011 and PRIN 20089M932N.

References

1. Marcello Balduccini. Splitting a cr-prolog program. In *LPNMR*, pages 17–29, 2009.
2. R. Barták and D. Toropila. Reformulating constraint models for classical planning. In David Wilson and H. Chad Lane, editors, *FLAIRS'08: Twenty-First International Florida Artificial Intelligence Research Society Conference*, pages 525–530. AAAI Press, 2008.
3. Adi Botea, Martin Müller, and Jonathan Schaeffer. Using abstraction for planning in sokoban. In *Computers and Games*, pages 360–375, 2002.
4. Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The third answer set programming competition. In *LPNMR*, pages 388–403, 2011.
5. Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Miroslaw Truszczynski. The second answer set programming competition. In *LPNMR*, pages 637–654, 2009.
6. Dorit Dor and Uri Zwick. Sokoban and other motion planning problems. *Computational Geometry: Theory and Applications*, 13:215–228, 1995.
7. Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Perspectives on Logic-based Approaches for Reasoning About Actions and Change. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *LNCS*, pages 259–279. Springer, 2011.

8. Agostino Dovier, Andrea Formisano, and Enrico Pontelli. Multivalued action languages with constraints in CLP(FD). *TPLP*, 10(2):167–235, 2010.
9. Juliana Freire, Terrance Swift, and David Scott Warren. Beyond depth-first: Improving tabled logic programs through alternative scheduling strategies. *Journal of Functional and Logic Programming*, 1998.
10. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. Engineering an incremental asp solver. In *ICLP*, pages 190–205, 2008.
11. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP)*, pages 1070–1080, 1988.
12. Michael Gelfond and Vladimir Lifschitz. Action languages. *Electron. Trans. Artif. Intell.*, 2:193–210, 1998.
13. Hai-Feng Guo and Gopal Gupta. Simplifying dynamic programming via mode-directed tabling. *Softw., Pract. Exper.*, 38(1):75–94, 2008.
14. Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI*, pages 1007–1012, 2007.
15. Andreas Junghanns and Jonathan Schaeffer. Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artif. Intell.*, 129(1-2):219–251, 2001.
16. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1–2):115–137, 2004.
17. David Scott Warren. Memoing for logic programs. *Comm. of the ACM, Special Section on Logic Programming*, 35:93–111, 1992.
18. Neng-Fa Zhou, Yoshitaka Kameya, and Taisuke Sato. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *ICTAI*, pages 213–218, 2010.
19. Neng-Fa Zhou, Taisuke Sato, and Yi-Dong Shen. Linear tabling strategies and optimizations. *TPLP*, 8(1):81–109, 2008.

EM over Binary Decision Diagrams for Probabilistic Logic Programs

Elena Bellodi and Fabrizio Riguzzi

ENDIF – Università di Ferrara – Via Saragat, 1 – 44122 Ferrara, Italy.
{elena.bellodi,fabrizio.riguzzi}@unife.it

Abstract. Recently much work in Machine Learning has concentrated on representation languages able to combine aspects of logic and probability, leading to the birth of a whole field called Statistical Relational Learning. In this paper we present a technique for parameter learning targeted to a family of formalisms where uncertainty is represented using Logic Programming techniques - the so-called Probabilistic Logic Programs such as ICL, PRISM, ProbLog and LPAD. Since their equivalent Bayesian networks contain hidden variables, an EM algorithm is adopted. In order to speed the computation, expectations are computed directly on the Binary Decision Diagrams that are built for inference. The resulting system, called EMBLEM for "EM over Bdds for probabilistic Logic programs Efficient Mining", has been applied to a number of datasets and showed good performances both in terms of speed and memory usage.

Keywords: Statistical Relational Learning, Probabilistic Logic Programming, Distribution Semantics, Logic Programs with Annotated Disjunctions, Expectation Maximization

1 Introduction

Machine Learning has seen the development of the field of Statistical Relational Learning (SRL) where logical-statistical languages are used in order to effectively learn in complex domains involving relations and uncertainty. They have been successfully applied in social networks analysis, entity recognition, collective classification and information extraction, to name a few.

Similarly, a large number of works in Logic Programming have attempted to combine logic and probability, among which the *distribution semantics* [21] is a prominent approach. This semantics underlies for example PRISM [21], the Independent Choice Logic [14], Logic Programs with Annotated Disjunctions (LPADs) [29], ProbLog [4] and CP-logic [27]. The approach is particularly appealing because efficient inference algorithms appeared [4,17], which adopt Binary Decision Diagrams (BDDs).

In this paper we present the EMBLEM system for "EM over Bdds for probabilistic Logic programs Efficient Mining" [1] that learns parameters of probabilistic logic programs under the distribution semantics by using an Expectation

Maximization (EM) algorithm. Such an algorithm is a popular tool in statistical estimation problems involving incomplete data: it is an iterative method to estimate some unknown parameters Θ of a model, given a dataset where some of the data is missing. The aim is to find maximum likelihood or maximum a posteriori (MAP) estimates of Θ [13]. EM alternates between performing an expectation (E) step, where the missing data are estimated given the observed data and current estimate of the model parameters, and a maximization (M) step, which computes the parameters maximizing the likelihood of the data given the sufficient statistics on the data computed in the E step. The translation of the probabilistic programs into graphical models requires the use of hidden variables (see Section 3) and therefore of EM: the main characteristic of our system is the computation of the values of expectations using BDDs.

Since there are transformations with linear complexity that can convert a program in a language under the distribution semantics into the others [28], we will use LPADs for their general syntax. EMBLEM has been tested on the IMDB, Cora and UW-CSE datasets and compared with RIB [20], LeProbLog [4], Alchemy [15] and CEM, an implementation of EM based on [17].

The paper is organized as follows. Section 2 presents LPADs and Section 3 describes EMBLEM. Section 4 discusses related works. Section 5 shows the results of the experiments performed and Section 6 concludes the paper.

2 Logic Programs with Annotated Disjunctions

Formally a *Logic Program with Annotated Disjunctions* [29] consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form $h_{i1} : \Pi_{i1}; \ldots; h_{in_i} : \Pi_{in_i} : -b_{i1}, \ldots, b_{im_i}$. In such a clause $h_{i1}, \ldots h_{in_i}$ are logical atoms and $b_{i1}, \ldots, b_{im_i}$ are logical literals, $\Pi_{i1}, \ldots, \Pi_{in_i}$ are real numbers in the interval $[0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. $b_{i1}, \ldots, b_{im_i}$ is called the *body* and is indicated with $body(C_i)$. Note that if $n_i = 1$ and $\Pi_{i1} = 1$ the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$. We denote by $ground(T)$ the grounding of an LPAD T.

An *atomic choice* is a triple (C_i, θ_j, k) where $C_i \in T$, θ_j is a substitution that grounds C_i and $k \in \{1, \ldots, n_i\}$. (C_i, θ_j, k) means that, for the ground clause $C_i\theta_j$, the head h_{ik} was chosen. In practice $C_i\theta_j$ corresponds to a random variable X_{ij} and an atomic choice (C_i, θ_j, k) to an assignment $X_{ij} = k$. A set of atomic choices κ is *consistent* if $(C, \theta, i) \in \kappa, (C, \theta, j) \in \kappa \Rightarrow i = j$, i.e., only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$.

A *selection* σ is a composite choice that, for each clause $C_i\theta_j$ in $ground(T)$, contains an atomic choice (C_i, θ_j, k). We denote the set of all selections σ of a program T by $\mathcal{S}_T$. A selection σ identifies a normal logic program w_σ defined as $w_\sigma = \{(h_{ik} \leftarrow body(C_i))\theta_j | (C_i, \theta_j, k) \in \sigma\}$. w_σ is called a *world* of T. Since

selections are composite choices we can assign a probability to possible worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$.

We consider only *sound* LPADs, in which every possible world has a total well-founded model. In the following we write $w_\sigma \models Q$ to mean that the query Q is true in the well-founded model of the program w_σ.

The probability of a query Q according to an LPAD T is given by

$$P(Q) = \sum_{\sigma\in E(Q)} P(\sigma) \tag{1}$$

where we define $E(Q)$ as $\{\sigma \in \mathcal{S}_T, w_\sigma \models Q\}$, the set of selections corresponding to worlds where the query is true.

To reduce the computational cost of answering queries in our experiments, random variables can be directly associated to clauses rather than to their ground instantiations: atomic choices then take the form (C_i, k), meaning that head h_{ik} is selected from program clause C_i, i.e., that $X_i = k$.

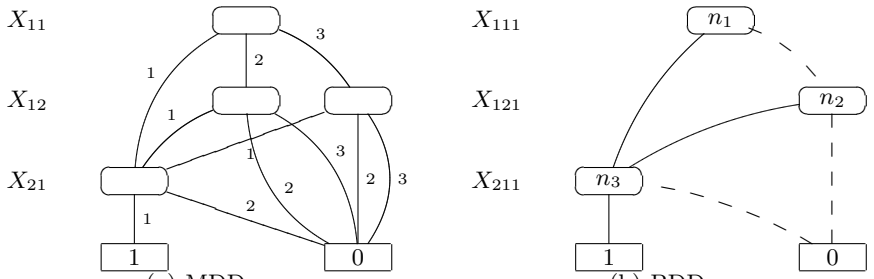
Example 1. The following LPAD T encodes a very simple model of the development of an epidemic or pandemic:

$C_1 = epidemic : 0.6; pandemic : 0.3 : -flu(X), cold.$
$C_2 = cold : 0.7.$
$C_3 = flu(david).$
$C_4 = flu(robert).$

Clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/david\}$ and $C_1\theta_2$ with $\theta_2 = \{X/robert\}$, so there are two random variables X_{11} and X_{12}; C_2 has only one grounding that is associated to the variable X_{21}. X_{11} and X_{12} have three values since C_1 has three head atoms (*epidemic*, *pandemic*, *null*); similarly X_{21} has two values since C_2 has two head atoms (*cold*, *null*).

The worlds in which a query is true can be represented using a Multivalued Decision Diagram (MDD) [25]. An MDD represents a function $f(\mathbf{X})$ taking Boolean values on a set of multivalued variables $\mathbf{X}$ by means of a rooted graph that has one level for each variable. Each node is associated to the variable of its level and has one child for each possible value of the variable. The leaves store either 0 or 1. Given values for all the variables $\mathbf{X}$, we can compute the value of $f(\mathbf{X})$ by traversing the graph starting from the root and returning the value associated to the leaf that is reached. A MDD can be used to represent the set $E(Q)$ by considering the multivalued variables X_{ij}s associated to the $C_i\theta_j$s of $ground(T)$. X_{ij} has values $\{1, \ldots, n_i\}$ and the atomic choice (C_i, θ_j, k) corresponds to the propositional equation $X_{ij} = k$. If we represent with an MDD the function $f(\mathbf{X}) = \bigvee_{\sigma\in E(Q)} \bigwedge_{(C_i,\theta_j,k)\in\sigma} X_{ij} = k$, then the MDD will have a path to a 1-leaf for each world where Q is true. While building MDDs, simplification operations can be applied that delete or merge nodes. Merging is performed when the diagram contains two identical sub-diagrams, while deletion is performed when all arcs from a node point to the same node. In this way a reduced MDD is obtained with respect to a Multivalued Decision Tree (MDT), i.e., a MDD in which every node has a single parent, all the children belong to the

level immediately below and all the variables have at least one node. For example, the reduced MDD corresponding to the query *epidemic* from Example 1 is shown in Figure 1(a). The labels on the edges represent the values of the variable associated to the node.



(a) MDD. (b) BDD.

Fig. 1. Decision diagrams for Example 1.

It is often unfeasible to find all the worlds where the query is true so inference algorithms find instead *explanations* for it, i.e. composite choices such that the query is true in all the worlds whose selections are a superset of them. Explanations however, differently from possible worlds, are not necessarily mutually exclusive with respect to each other, but exploiting the fact that MDDs split paths on the basis of the values of a variable and the branches are mutually disjoint so a dynamic programming algorithm can be applied for computing the probability.

Most packages for the manipulation of decision diagrams are however restricted to work on Binary Decision Diagrams, i.e., decision diagrams where all the variables are Boolean. A node n in a BDD has two children: the 1-child, indicated with $child_1(n)$, and the 0-child, indicated with $child_0(n)$. The 0-branch, the one going to the 0-child, is drawn with a dashed line.

To work on MDDs with a BDD package we must represent multivalued variables by means of binary variables. For a multivalued variable X_{ij}, corresponding to ground clause $C_i\theta_j$, having n_i values, we use $n_i - 1$ Boolean variables $X_{ij1}, \ldots, X_{ijn_i-1}$ and we represent the equation $X_{ij} = k$ for $k = 1, \ldots n_i - 1$ by means of the conjunction $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$, and the equation $X_{ij} = n_i$ by means of the conjunction $\overline{X_{ij1}} \wedge \overline{X_{ij2}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. Figure 1(b) shows the reduced BDD corresponding to the MDD in Figure 1(a). BDDs can be used for computing the probability of queries by associating to each Boolean variable X_{ijk} a parameter π_{ik} that represents $P(X_{ijk} = 1)$. If we define $g(i) = \{j|\theta_j \text{ is a substitution grounding } C_i\}$ then $P(X_{ijk} = 1) = \pi_{ik}$ for all $j \in g(i)$. The parameters are obtained from those of multivalued variables in this way:

$$\pi_{i1} = \Pi_{i1}$$

$$\ldots$$
$$\pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}$$
$$\ldots$$

up to $k = n_i - 1$.

3 EMBLEM

EMBLEM applies the algorithm for performing EM over BDDs, proposed in [26,9,10,8], to the problem of learning the parameters of an LPAD. EMBLEM takes as input a number of goals that represent the examples and for each one generates the BDD encoding its explanations. The examples are organized in a set of interpretations (sets of ground facts) each describing a portion of the domain of interest. The queries correspond to ground atoms in an interpretation whose predicate has been indicated as "target" by the user. The predicates can be treated as closed-world or open-world. In the first case the body of clauses is resolved only with facts in the interpretation, in the second case it is resolved both with facts in the interpretation and with clauses in the theory. If the last option is set and the theory is cyclic, we use a depth bound on SLD-derivations to avoid going into infinite loops, as proposed by [6]. Given the program containing only the clauses C_1 and C_2 from Example 1 and the interpretation $\{epidemic, flu(david), flu(robert)\}$, we obtain the BDD in Figure 1(b) that represents the query *epidemic*. A value of 1 for the Boolean variables X_{111} and X_{121} means that, for the ground clauses $C_1\theta_1$ and $C_1\theta_2$, the head $h_{11} =$ *epidemic* is chosen, regardless of the other variables for the clause (X_{112}, X_{122}) that are in fact omitted from the diagram.

Then EMBLEM enters the EM cycle, in which the steps of expectation and maximization are repeated until the log-likelihood of the examples reaches a local maximum. The necessity of exploiting EM depends on the fact that, to determine the parameters Π_{ik}, the number of times that a head h_{ik} has been chosen is required. The information about which selection was used in the derivation of a goal is unknown, so the random variables are hidden and we compute expected counts. For a single example Q:

- Expectation: computes $\mathbf{E}[c_{ik0}|Q]$ and $\mathbf{E}[c_{ik1}|Q]$ for all rules C_i and $k = 1, \ldots, n_i - 1$, where c_{ikx} is the number of times a variable X_{ijk} takes value x for $x \in \{0, 1\}$, with j in $g(i)$. $\mathbf{E}[c_{ikx}|Q]$ is given by $\sum_{j \in g(i)} P(X_{ijk} = x|Q)$.
- Maximization: computes π_{ik} for all rules C_i and $k = 1, \ldots, n_i - 1$.

$$\pi_{ik} = \frac{\mathbf{E}[c_{ik1}|Q]}{\mathbf{E}[c_{ik0}|Q] + \mathbf{E}[c_{ik1}|Q]} \tag{2}$$

If we have more than one example the contributions of each example simply sum up when computing $\mathbf{E}[c_{ijx}]$.

$P(X_{ijk} = x|Q)$ is given by $P(X_{ijk} = x|Q) = \frac{P(X_{ijk}=x,Q)}{P(Q)}$ with

$$\begin{aligned} P(X_{ijk} = x, Q) &= \sum_{\sigma \in \mathcal{S}_T} P(Q, X_{ijk} = x, \sigma) \\ &= \sum_{\sigma \in \mathcal{S}_T} P(Q|\sigma) P(X_{ijk} = x|\sigma) P(\sigma) \\ &= \sum_{\sigma \in E(Q)} P(X_{ijk} = x|\sigma) P(\sigma) \end{aligned}$$

where $P(X_{ijk} = 1|\sigma) = 1$ if $(C_i, \theta_j, k) \in \sigma$ for $k = 1, \ldots, n_i - 1$ and 0 otherwise.

Since there is a one to one correspondence between the possible worlds where Q is true and the paths to a 1 leaf in a Binary Decision Tree (a MDT with binary variables),

$$P(X_{ijk} = x, Q) = \sum_{\rho \in R(Q)} P(X_{ijk} = x|\rho) \prod_{d \in \rho} \pi(d)$$

where ρ is a path, and if σ corresponds to ρ, then $P(X_{ijk} = x|\sigma)$=$P(X_{ijk} = x|\rho)$. $R(Q)$ is the set of paths in the BDD for query Q that lead to a 1 leaf, d is an edge of ρ and $\pi(d)$ is the probability associated to the edge: if d is the 1-branch from a node associated to a variable X_{ijk}, then $\pi(d) = \pi_{ik}$, if d is the 0-branch from a node associated to a variable X_{ijk}, then $\pi(d) = 1 - \pi_{ik}$.

Now consider a BDT in which only the merge rule is applied, fusing together identical sub-diagrams. The resulting diagram, that we call Complete Binary Decision Diagram (CBDD), is such that every path contains a node for every level. For a CBDD, $P(X_{ijk} = x, Q)$ can be further expanded as

$$P(X_{ijk} = x, Q) = \sum_{\rho \in R(Q), (X_{ijk}=x) \in \rho} \prod_{d \in \rho} \pi(d)$$

where $(X_{ijk} = x) \in \rho$ means that ρ contains an x-edge from a node associated to X_{ijk}. We can then write

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n) = X_{ijk}, \rho_n \in R_n(Q), \rho^n \in R^n(Q,x)} \prod_{d \in \rho_n} \pi(d) \prod_{d \in \rho^n} \pi(d)$$

where $N(Q)$ is the set of nodes of the CBDD, $v(n)$ is the variable associated to node n, $R_n(Q)$ is the set containing the paths from the root to n and $R^n(Q, x)$ is the set of paths from n to the 1 leaf through its x-child.

$$\begin{aligned} P(X_{ijk} = x, Q) &= \sum_{n \in N(Q), v(n) = X_{ijk}} \sum_{\rho_n \in R_n(Q)} \sum_{\rho^n \in R^n(Q,x)} \prod_{d \in \rho_n} \pi(d) \prod_{d \in \rho^n} \pi(d) \\ &= \sum_{n \in N(Q), v(n) = X_{ijk}} \sum_{\rho_n \in R_n(Q)} \prod_{d \in \rho_n} \pi(d) \sum_{\rho^n \in R^n(Q,x)} \prod_{d \in \rho^n} \pi(d) \\ &= \sum_{n \in N(Q), v(n) = X_{ijk}} F(n) B(child_x(n)) \pi_{ikx} \end{aligned}$$

where π_{ikx} is π_{ik} if x=1 and $(1-\pi_{ik})$ if x=0. $F(n)$ is the *forward probability* [10], the probability mass of the paths from the root to n, while $B(n)$ is the *backward probability* [10], the probability mass of paths from n to the 1 leaf. If *root* is the root of a tree for a query Q then $B(root) = P(Q)$.

The expression $F(n)B(child_x(n))\pi_{ikx}$ represents the sum of the probabilities of all the paths passing through the x-edge of node n and is indicated with $e^x(n)$. Thus

$$P(X_{ijk} = x, Q) = \sum_{n \in N(Q), v(n)=X_{ijk}} e^x(n) \tag{3}$$

For the case of a BDD, i.e., a diagram obtained by applying also the deletion rule, Formula 3 is no longer valid since also paths where there is no node associated to X_{ijk} can contribute to $P(X_{ijk} = x, Q)$. These paths might have been obtained from a BDD having a node m associated to variable X_{ijk} that is a descendant of node n along the 0-branch and whose outgoing edges both point to $child_0(n)$. The correction of formula (3) to take into account this aspect is applied in the Expectation step.

EMBLEM's main procedure consists of a cycle in which the procedures Expectation and Maximization are repeatedly called. Procedure Expectation returns the log likelihood of the data that is used in the stopping criterion: EMBLEM stops when the difference between the log likelihood of the current iteration and the one of the previous iteration drops below a threshold ϵ or when this difference is below a fraction δ of the current log likelihood.

Procedure Expectation takes as input a list of BDDs, one for each example, and computes the expectations for each one, i.e. $P(X_{ijk} = x, Q)$ for all variables X_{ijk} in the BDD and values $x \in \{0, 1\}$. In the procedure we use $\eta^x(i, k)$ to indicate $\sum_{j \in g(i)} P(X_{ijk} = x, Q)$. Expectation first calls GetForward and GetBackward that compute the forward, the backward probability of nodes and $\eta^x(i, k)$ for non-deleted paths only. Then it updates $\eta^x(i, k)$ to take into account deleted paths. The expectations are updated in this way: for all rules i and for $k = 1$ to $n_i - 1$, $\mathbf{E}[c_{ikx}] = \mathbf{E}[c_{ikx}] + \eta^x(i, k)/P(Q)$.

Procedure Maximization computes the parameters values for the next EM iteration, as specified in (2).

Procedure GetForward traverses the diagram one level at a time starting from the root level, where F(root)=1, and for each node n it computes its contribution to the forward probabilities of its children. Then the forward probabilities of both children are updated in this way: $F(child_x(node)) = F(child_x(node)) + F(node) \cdot \pi_{ikx}$.

Function GetBackward computes the backward probability of nodes by traversing recursively the tree from the leaves to the root. When the calls of GetBackward for both children of a node n return, we have all the information that is needed to compute the e^x values and the value of $\eta^x(i, k)$ for non-deleted paths. An array ς is used here to store the contributions of the deleted paths by starting from the root level and accumulating $\varsigma(l)$ for the various levels l.

A fully detailed description of EMBLEM together with an example of its execution can be found in [1].

4 Related Works

Our work has close connection with various other works. [9,10] proposed an EM algorithm for learning the parameters of Boolean random variables given observations of the values of a Boolean function over them, represented by a BDD. EMBLEM is an application of that algorithm to probabilistic logic programs. Independently, also [26] proposed an EM algorithm over BDD to learn parameters for the CPT-L language.[7] presented the CoPrEM algorithm that performs EM over BDDs for the ProbLog language.

Approaches for learning probabilistic logic programs can be classified into three categories: those that employ constraint techniques (such as [16,18]), those that use EM and those that adopt gradient descent.

Among the approaches that use EM, [12] first proposed to use it to induce parameters and the Structural EM algorithm to induce ground LPADs structures. Their EM algorithm however works on the underlying Bayesian network. RIB [20] performs parameter learning using the information bottleneck approach, which is an extension of EM targeted especially towards hidden variables. The PRISM system [21,22] is one of the first learning algorithms based on EM.

Among the works that use a gradient descent technique, LeProbLog [5,6] finds the parameters of a ProbLog program that minimize the Mean Squared Error of the probability of queries and uses BDD to compute the gradient.

Alchemy [15] is a state of the art SRL system that offers various tools for inference, weight learning and structure learning of Markov Logic Networks (MLNs). MLNs differ significantly from the languages under the distribution semantics since they extend first-order logic by attaching weights to logical formulas, reflecting "how strong" they are, but do not allow to exploit logic programming techniques.

5 Experiments

EMBLEM has been tested over three real world datasets: IMDB[1], UW-CSE[2] [23] and Cora[3] [23]. We implemented EMBLEM in Yap Prolog[4] and we compared it with RIB [20]; CEM, an implementation of EM based on the `cplint` inference library [17,19]; LeProblog [5,6] and Alchemy [15]. All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM.

To compare our results with LeProbLog we exploited the translation of LPADs into ProbLog proposed in [3], for Alchemy we exploited the translation between LPADs and MLNs used in [20].

For the probabilistic logic programming systems (EMBLEM, RIB, CEM and LeProbLog) we consider various options. The first consists in choosing between

[1] `http://alchemy.cs.washington.edu/data/imdb`
[2] `http://alchemy.cs.washington.edu/data/uw-cse`
[3] `http://alchemy.cs.washington.edu/data/cora`
[4] `http://www.dcc.fc.up.pt/~vsc/Yap`

associating a distinct random variable to each grounding of a probabilistic clause or a single random variable to a non-ground probabilistic clause expressing whether the clause is used or not. The latter case makes the problem easier. The second option is concerned with imposing a limit on the depth of derivations as done in [6], thus eliminating explanations associated to derivations exceeding the depth limit. This is necessary for problems that contain cyclic clauses, such as transitive closure clauses. The third option involves setting the number of restarts for EM based algorithms. All experiments for probabilistic logic programming systems have been performed using open-world predicates.

IMDB regards movies, actors, directors and movie genres and it is divided into five mega-examples. We performed training on four mega-examples and testing on the remaining one. Then we drew a Precision-Recall curve and computed the Area Under the Curve (AUCPR) using the method reported in [2]. We defined 4 different LPADs, two for predicting the target predicate `sameperson/2`, and two for predicting `samemovie/2`. We had one positive example for each fact that is true in the data, while we sampled from the complete set of false facts three times the number of true instances in order to generate negative examples.

For predicting `sameperson/2` we used the same LPAD of [20]:

```
sameperson(X,Y):p:- movie(M,X),movie(M,Y).
sameperson(X,Y):p:- actor(X),actor(Y),workedunder(X,Z),
                    workedunder(Y,Z).
sameperson(X,Y):p:- gender(X,Z),gender(Y,Z).
sameperson(X,Y):p:- director(X),director(Y),genre(X,Z),genre(Y,Z).
```

where `p` is a placeholder meaning the parameter must be learned. We ran EMBLEM on it with the following settings: no depth bound, random variables associated to instantiations of clauses and a number of restarts chosen to match the execution time of EMBLEM with that of the fastest other algorithm.

The queries that LeProbLog takes as input are obtained by annotating with 1.0 each positive example for `sameperson/2` and with 0.0 each negative example for `sameperson/2`. We ran LeProbLog for a maximum of 100 iterations or until the difference in Mean Squared Error (MSE) between two iterations got smaller than 10^{-5}; this setting was used in all the following experiments as well. For Alchemy we always used the preconditioned rescaled conjugate gradient discriminative algorithm [11]. For this experiments we specified `sameperson/2` as the only non-evidence predicate.

A second LPAD has been created to evaluate the performance of the algorithms when some atoms are unseen:

```
sameperson_pos(X,Y):p:- movie(M,X),movie(M,Y).
sameperson_pos(X,Y):p:- actor(X),actor(Y),
                        workedunder(X,Z),workedunder(Y,Z).
sameperson_pos(X,Y):p:- director(X),director(Y),genre(X,Z),
                        genre(Y,Z).
sameperson_neg(X,Y):p:- movie(M,X),movie(M,Y).
sameperson_neg(X,Y):p:- actor(X),actor(Y),
```

```
                        workedunder(X,Z),workedunder(Y,Z).
sameperson_neg(X,Y):p:- director(X),director(Y),genre(X,Z),
                        genre(Y,Z).
sameperson(X,Y):p:- \+ sameperson_pos(X,Y), sameperson_neg(X,Y).
sameperson(X,Y):p:- \+ sameperson_pos(X,Y),\+ sameperson_neg(X,Y).
sameperson(X,Y):p:- sameperson_pos(X,Y), sameperson_neg(X,Y).
sameperson(X,Y):p:- sameperson_pos(X,Y), \+ sameperson_neg(X,Y).
```

The sameperson_pos/2 and sameperson_neg/2 predicates are unseen in the data. Settings are the same as the ones for the previous LPAD. In this experiment Alchemy was run with the −withEM option that turns on EM learning.

Table 1 shows the AUCPR averaged over the five folds for EMBLEM, RIB, LeProbLog, CEM and Alchemy. Results for the two LPADs are shown respectively in the IMDB-SP and IMDBu-SP rows. Table 2 shows the learning times in hours.

For predicting samemovie/2 we used the LPAD:

```
samemovie(X,Y):p:- movie(X,M),movie(Y,M),actor(M).
samemovie(X,Y):p:- movie(X,M),movie(Y,M),director(M).
samemovie(X,Y):p:- movie(X,A),movie(Y,B),actor(A),director(B),
                   workedunder(A,B).
samemovie(X,Y):p:- movie(X,A),movie(Y,B),director(A),director(B),
                   genre(A,G),genre(B,G).
```

To test the behaviour when unseen predicates are present, we transformed the program for samemovie/2 as we did for sameperson/2, thus introducing the unseen predicates samemovie_pos/2 and samemovie_neg/2. We ran EMBLEM on them with no depth bound, one variable for each instantiation of the rules and one random restart. With regard to LeProbLog and Alchemy, we ran them with the same settings as IMDB-SP and IMDBu-SP, by replacing sameperson with samemovie.

Table 1 shows, in the IMDB-SM and IMDBu-SM rows, the average AUCPR for EMBLEM, LeProblog and Alchemy. For RIB and CEM we obtained a lack of memory error (indicated with "me"); Table 2 shows the learning times in hours.

The Cora database contains citations to computer science research papers. For each citation we know the title, authors, venue and the words that appear in them. The task is to determine which citations are referring to the same paper, by predicting the predicate samebib(cit1,cit2).

From the MLN proposed in [24][5] we obtained two LPADs. The first contains 559 rules and differs from the direct translation of the MLN because rules involving words are instantiated with the different constants, only positive literals for the hasword predicates are used and transitive rules are not included:

```
samebib(B,C):p:- author(B,D),author(C,E),sameauthor(D,E).
samebib(B,C):p:- title(B,D),title(C,E),sametitle(D,E).
samebib(B,C):p:- venue(B,D),venue(C,E),samevenue(D,E).
```

[5] http://alchemy.cs.washington.edu/mlns/er

```
samevenue(B,C):p:-haswordvenue(B,word_06),haswordvenue(C,word_06).
...
sametitle(B,C):p:-haswordtitle(B,word_10),haswordtitle(C,word_10).
....
sameauthor(B,C):p:- haswordauthor(B,word_a),
                    haswordauthor(C,word_a).
.....
```

The dots stand for the rules for all the possible words. The Cora dataset comprises five mega-examples each containing facts for the four predicates `samebib/2`, `samevenue/2`, `sametitle/2` and `sameauthor/2`, which have been set as target predicates. We used as negative examples those contained in the Alchemy dataset. We ran EMBLEM on this LPAD with no depth bound, a single variable for each instantiation of the rules and a number of restarts chosen to match the execution time of EMBLEM with that of the fastest other algorithm.

The second LPAD adds to the previous one four transitive rules of the form

```
samebib(A,B):p :- samebib(A,C),samebib(C,B).
```

for every target predicate, for a total of 563 rules. In this case we had to run EMBLEM with a depth bound equal to two and a single variable for each non-ground rule; the number of restarts was one. As for LeProbLog, we separately learned the four predicates because learning the whole theory at once would give a lack of memory error. We annotated with 1.0 each positive example for `samebib/2`, `sameauthor/2`, `sametitle/2`, `samevenue/2` and with 0.0 the negative examples for the same predicates. As for Alchemy we learned weights with the four predicates as the non-evidence predicates. Table 1 shows in the Cora and CoraT (Cora transitive) rows the average AUCPR obtained by training on four mega-examples and testing on the remaining one. CEM and Alchemy on CoraT gave a lack of memory error while RIB was not applicable because it was not possible to split the input examples into smaller independent interpretations as required by RIB.

The UW-CSE dataset contains information about the Computer Science department of the University of Washington through 22 different predicates, such as `yearsInProgram/2`, `advisedBy/2`, `taughtBy/3` and is split into five mega-examples. The goal here is to predict the `advisedBy/2` predicate, namely the fact that a person is advised by another person: this was our target predicate. The negative examples have been generated by considering all couple of persons `(a,b)` where `a` and `b` appear in an `advisedby/2` fact in the data and by adding a negative example `advisedby(a,b)` if it is not in the data.

The theory used was obtained from the MLN of [23][6]. It contains 86 rules, such as for instance:

```
advisedby(S, P) :p :- courselevel(C,level_500),taughtby(C,P,Q),
                      ta(C, S, Q).
```

[6] `http://alchemy.cs.washington.edu/mlns/uw-cse`

We ran EMBLEM on it with a single variable for each instantiation of a rule, a depth bound of two and one random restart.

The annotated queries that LeProbLog takes as input have been created by annotating with 1.0 each positive example for `advisedby/2` and with 0.0 the negative examples. As for Alchemy, we learned weights with `advisedby/2` as the only non-evidence predicate. Table 1 shows the AUCPR averaged over the five mega-examples for all the algorithms.

Table 3 shows the p-value of a paired two-tailed t-test at the 5% significance level of the difference in AUCPR between EMBLEM and RIB/LeProbLog/CEM/ Alchemy (significant differences in bold).

Table 1. Results of the experiments on all datasets. IMDBu refers to the IMDB dataset with the theory containing unseen predicates. CoraT refers to the theory containing transitive rules. Numbers in parenthesis followed by r mean the number of random restarts (when different from one) to reach the area specified. "me" means memory error during learning. AUCPR is the area under the precision-recall curve averaged over the five folds. R is RIB, L is LeProbLog, C is CEM, A is Alchemy.

Dataset	AUCPR				
	EMBLEM	R	L	C	A
IMDB-SP	0.202(500r)	0.199	0.096	0.202	0.107
IMDBu-SP	0.175(40r)	0.166	0.134	0.120	0.020
IMDB-SM	1.000	me	0.933	0.537	0.820
IMDBu-SM	1.000	me	0.933	0.515	0.338
Cora	0.995(120r)	0.939	0.905	0.995	0.469
CoraT	0.991	no	0.970	me	me
UW-CSE	0.883	0.588	0.270	0.644	0.294

Table 2. Execution time in hours of the experiments on all datasets. R is RIB, L is LeProbLog, C is CEM and A is Alchemy.

Dataset	Time(h)				
	EMBLEM	R	L	C	A
IMDB-SP	0.01	0.016	0.35	0.01	1.54
IMDBu-SP	0.01	0.0098	0.23	0.012	1.54
IMDB-SM	0.00036	me	0.005	0.0051	0.0026
IMDBu-SM	3.22	me	0.0121	0.0467	0.0108
Cora	2.48	2.49	13.25	11.95	1.30
CoraT	0.38	no	4.61	me	me
UW-CSE	2.81	0.56	1.49	0.53	1.95

From the results we can observe that over IMDB EMBLEM has comparable performances with CEM for IMDB-SP, with similar execution time. On IMDBu-SP it has better performances than all other systems, with a learning time equal

Table 3. Results of t-test on all datasets. p is the p-value of a paired two-tailed t-test (significant differences at the 5% level in bold) between EMBLEM and all the others. R is RIB, L is LeProbLog, C is CEM, A is Alchemy.

Dataset	p			
	EMBLEM-R	EMBLEM-L	EMBLEM-C	EMBLEM-A
IMDB-SP	0.2167	**0.0126**	0.3739	**0.0134**
IMDBu-SP	0.1276	0.1995	**0.001**	**4.5234e-005**
IMDB-SM	me	0.3739	**0.0241**	0.1790
IMDBu-SM	me	0.3739	0.2780	**2.2270e-004**
Cora	**0.011**	0.0729	1	**0.0068**
CoraT	no	**0.0464**	me	me
UW-CSE	**0.0054**	**1.5017e-004**	**0.0088**	**4.9921e-004**

to the fastest other algorithm. On IMDB-SM it reaches the highest area value in less time (only one restart is needed). On IMDBu-SM it still reaches the highest area with one restart but with a longer execution time. Over Cora it has comparable performances with the best other system CEM but in a significantly lower time and over CoraT is one of the few systems to be able to complete learning, with better performances in terms of area and time. Over UW-CSE it has significant better performances with respect to all the algorithms.

Memory errors, that we encountered with some systems over certain datasets, have to be ascribed to the memory needs of the systems; for instance, some of them are not able to manage the LPAD for CoraT because its transitive rules generate large BDDs.

6 Conclusions

We have proposed a technique which applies an EM algorithm for learning the parameters of Logic Programs with Annotated Disjunctions. It can be applied to all languages that are based on the distribution semantics and exploits the BDDs that are built during inference to efficiently compute the expectations for hidden variables.

We executed the algorithm over the real datasets IMDB, UW-CSE and Cora, and evaluated its performances - together with those of four other probabilistic systems - through the AUCPR and AUCROC. These results show that EMBLEM uses less memory than RIB, CEM and Alchemy, allowing it to solve larger problems, as one can see from Table **??** where, for some datasets, not all the mentioned algorithms are able to terminate. Moreover its speed allows to perform a high number of restarts making it escape local maxima and achieve higher AUCPR.

EMBLEM is available in the `cplint` package in the source tree of Yap Prolog and information on its use can be found at `http://sites.google.com/a/unife.it/ml/emblem`.

In the future we plan to extend EMBLEM for learning the structure of LPADs by combining the standard Expectation Maximization algorithm, which optimizes parameters, with structure search for model selection.

References

1. Bellodi, E., Riguzzi, F.: EM over binary decision diagrams for probabilistic logic programs. Tech. Rep. CS-2011-01, ENDIF, Università di Ferrara, Italy (2011)
2. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: Cohen, W.W., Moore, A. (eds.) Proceedings of the 23rd International Conference on Machine Learning. ACM International Conference Proceeding Series, vol. 148, pp. 233–240. ACM (2006)
3. De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., Vennekens, J.: Towards digesting the alphabet-soup of statistical relational learning. In: Roy, D., Winn, J., McAllester, D., Mansinghka, V., Tenenbaum, J. (eds.) Proceedings of the 1st Workshop on Probabilistic Programming: Universal Languages, Systems and Applications, in NIPS (2008)
4. De Raedt, L., Kimmig, A., Toivonen, H.: ProbLog: A probabilistic prolog and its application in link discovery. In: Veloso, M.M. (ed.) Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 2462–2467. AAAI Press (2007)
5. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.D.: Parameter learning in probabilistic databases: A least squares approach. In: Daelemans, W., Goethals, B., Morik, K. (eds.) Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases. LNCS, vol. 5211, pp. 473–488. Springer (2008)
6. Gutmann, B., Kimmig, A., Kersting, K., Raedt, L.: Parameter estimation in ProbLog from annotated queries. Tech. Rep. CW 583, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2010)
7. Gutmann, B., Thon, I., De Raedt, L.: Learning the parameters of probabilistic logic programs from interpretations. Tech. Rep. CW 584, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (June 2010)
8. Inoue, K., Sato, T., Ishihata, M., Kameya, Y., Nabeshima, H.: Evaluating abductive hypotheses using an em algorithm on bdds. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). pp. 810–815. Morgan Kaufmann Publishers Inc. (2009)
9. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. In: Zelezn, F., Lavra, N. (eds.) Late Breaking Papers of the 18th International Conference on Inductive Logic Programming. pp. 44–49 (2008)
10. Ishihata, M., Kameya, Y., Sato, T., Minato, S.: Propositionalizing the em algorithm by bdds. Tech. Rep. TR08-0004, Dept. of Computer Science, Tokyo Institute of Technology (2008)
11. Lowd, D., Domingos, P.: Efficient weight learning for Markov logic networks. In: Kok, J.N., Koronacki, J., de Mántaras, R.L., Matwin, S., Mladenic, D., Skowron, A. (eds.) Proceedings of the 18th European Conference on Machine Learning. LNCS, vol. 4702, pp. 200–211. Springer (2007)
12. Meert, W., Struyf, J., Blockeel, H.: Learning ground CP-Logic theories by leveraging Bayesian network learning techniques. Fundamenta Informaticae 89(1), 131–160 (2008)

13. Neapolitan, R.: Learning Bayesian Networks. Prentice Hall, Upper Saddle River, NJ (2003)
14. Poole, D.: The Independent Choice Logic for modelling multiple agents under uncertainty. Artificial Intelligence 94(1-2), 7–56 (1997)
15. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1-2), 107–136 (2006)
16. Riguzzi, F.: ALLPAD: Approximate learning of logic programs with annotated disjunctions. In: Muggleton, S., Otero, R.P., Tamaddoni-Nezhad, A. (eds.) Proceedings of the 16th International Conference on Inductive Logic Programming. LNCS, vol. 4455, pp. 43–45. Springer (2007)
17. Riguzzi, F.: A top-down interpreter for LPAD and CP-Logic. In: Basili, R., Pazienza, M.T. (eds.) Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence. LNCS, vol. 4733, pp. 109–120. Springer (2007)
18. Riguzzi, F.: ALLPAD: approximate learning of logic programs with annotated disjunctions. Machine Learning 70(2-3), 207–223 (2008)
19. Riguzzi, F.: Extended semantics and inference for the Independent Choice Logic. Logic Journal of the IGPL 17(6), 589–629 (2009)
20. Riguzzi, F., Mauro, N.D.: Applying the information bottleneck to statistical relational learning. Machine Learning (2011), to appear
21. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: Sterling, L. (ed.) Proceedings of the 12th International Conference on Logic Programming. pp. 715–729. MIT Press (1995)
22. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research 15, 391–454 (2001)
23. Singla, P., Domingos, P.: Discriminative training of Markov logic networks. In: Veloso, M.M., Kambhampati, S. (eds.) Proceedings of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference. pp. 868–873. AAAI Press/The MIT Press (2005)
24. Singla, P., Domingos, P.: Entity resolution with Markov logic. In: Proceedings of the 6th IEEE International Conference on Data Mining. pp. 572–582. IEEE Computer Society (2006)
25. Thayse, A., Davio, M., Deschamps, J.P.: Optimization of multivalued decision algorithms. In: International Symposium on Multiple-Valued Logic. pp. 171–178. IEEE Computer Society Press (1978)
26. Thon, I., Landwehr, N., Raedt, L.D.: A simple model for sequences of relational state descriptions. In: Daelemans, W., Goethals, B., Morik, K. (eds.) Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2008)- Part II. Lecture Notes in Computer Science, vol. 5212, pp. 506–521. Springer-Verlag (2008)
27. Vennekens, J., Denecker, M., Bruynooghe, M.: Cp-logic: A language of causal probabilistic events and its relation to logic programming. Theory and Practice of Logic Programming 9(3), 245–308 (2009)
28. Vennekens, J., Verbaeten, S.: Logic programs with annotated disjunctions. Tech. Rep. CW386, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2003)
29. Vennekens, J., Verbaeten, S., Bruynooghe, M.: Logic programs with annotated disjunctions. In: Demoen, B., Lifschitz, V. (eds.) Proceedings of the 20th International Conference on Logic Programming. LNCS, vol. 3131, pp. 195–209. Springer (2004)

Synthesizing Concurrent Programs using Answer Set Programming

Emanuele De Angelis[1], Alberto Pettorossi[2], and Maurizio Proietti[3]

[1] Dipartimento di Scienze, University 'G. D'Annunzio', Pescara, Italy
deangelis@sci.unich.it
[2] DISP, University of Rome Tor Vergata, Italy
pettorossi@disp.uniroma2.it
[3] CNR-IASI, Rome, Italy
proietti@iasi.cnr.it

Abstract. We address the problem of the automatic synthesis of concurrent programs within a framework based on Answer Set Programming (ASP). The concurrent program to be synthesized is specified by providing both the behavioural and the structural properties it should satisfy. Behavioural properties, such as safety and liveness properties, are specified by using formulas of the Computation Tree Logic, which are encoded as a logic program. Structural properties, such as the symmetry of processes, are also encoded as a logic program. Then, the program which is the union of these two encodings, is given as input to an ASP system which returns as output a set of answer sets. Finally, each answer set is decoded into a synthesized program that, by construction, satisfies the desired behavioural and structural properties.

1 Introduction

We consider *concurrent programs* consisting of finite sets of *processes* which interact with each other through *communication protocols*. Such protocols are based on a set of instructions, called *synchronization instructions*, operating on shared variables ranging over finite domains. The communication protocols are realized in a distributed manner, that is, every process includes one or more regions of code consisting of synchronization instructions, responsible for the interaction between processes.

Even for a small number of processes, communication protocols which guarantee a desired behaviour of the concurrent programs may be hard to design. In this paper we propose a method for automatically synthesizing correct concurrent programs starting from the formal specification of their desired behaviour.

Methods for the automatic synthesis of concurrent programs from *temporal logic specifications* have been proposed in the past by Clarke and Emerson [6], Manna and Wolper [16], and Attie and Emerson [1,2]. All these authors reduce the task of synthesizing a concurrent program to the task of synthesizing the synchronization instructions of each process. We follow their approach and everything which is irrelevant to the synchronization among processes, is abstracted away and each process is considered to be a finite state automaton.

We introduce a framework, based on logic programming, for the automatic synthesis of concurrent programs. We assume that the *behavioural properties* of the concurrent programs, such as safety and liveness properties, are specified by using formulas of the Computation Tree Logic (CTL for short), which is a very popular propositional temporal logic over branching time structures (see, for instance, [5,6]). This temporal, behavioural specification φ is encoded as a set Π_φ of clauses. We also assume that the processes to be synthesized satisfy suitable *structural properties*, such as a *symmetry* property, and that those properties can be encoded as a set Π_Σ of clauses. Structural properties cannot be easily specified by using CTL formulas and we use, instead, a simple algebraic structure that we will present in the paper. Thus, the specification of a concurrent program to be synthesized consists of a logic program $\Pi = \Pi_\varphi \cup \Pi_\Sigma$ which encodes both the behavioural and the structural properties that the concurrent program should satisfy.

We show that every answer set (that is, every stable model) of the program Π represents a concurrent program satisfying the given specification. Thus, by using an Answer Set Programming (ASP) system, such as DLV [9] or smodels [20], which computes the answer sets of logic programs, we can synthesize concurrent programs which enjoy some desired properties.

We have performed some synthesis experiments and, in particular, we have synthesized some mutual exclusion protocols which are guaranteed to enjoy various properties, such as (i) bounded overtaking, (ii) absence of starvation, and (iii) maximal reactivity (their formal definition will be given in the paper). We finally compare our results with those presented in [1,2,12].

The paper is structured as follows. In Section 2 we recall some preliminary notions and terminology. In Section 3 we present our framework for synthesizing concurrent programs and we define the notion of a symmetric concurrent program. In Section 4 we describe our synthesis procedure and the logic program which we use for the synthesis. In Section 5 we present some examples of synthesis of symmetric concurrent programs. Finally, in Section 6 we discuss the related work and some topics that can be investigated in the future.

2 Preliminaries

Let us recall some basic notions and terminology we will use. We present: (i) the syntax of (a variant of) the *guarded commands* [7] which are used for defining concurrent programs, (ii) some basic notions of *group theory* which are required for defining symmetric concurrent programs, (iii) the syntax and the semantics of the Computation Tree Logic, and (iv) the syntax and the semantics of Answer Set Programming, which is the framework we use for our synthesis method.

Guarded commands. In our variant of the guarded commands we consider two basic sets: (i) variables, v in *Var*, each ranging over a finite domain D_v, and (ii) guards, g in *Guard*, of the form: $g ::= \mathit{true} \mid \mathit{false} \mid v = d \mid \neg g \mid g_1 \wedge g_2$, with $v \in \mathit{Var}$ and $d \in D_v$. We also have the following derived sets whose definitions are mutually recursive: (iii) commands, c in *Command*, of the form:

$c ::= \mathit{skip} \mid v := d \mid c_1 \,;\, c_2 \mid \texttt{if}\ gc\ \texttt{fi} \mid \texttt{do}\ gc\ \texttt{od}$, where ';' denotes the *sequential composition* of commands, and (iv) guarded commands, gc in *GCommand*, of the form: $gc ::= g \rightarrow c \mid gc_1 \,[]\, gc_2$, where '[]' denotes the *parallel composition* of guarded commands.

The execution of $\texttt{if}\ gc_1 \,[]\, \ldots \,[]\, gc_n\ \texttt{fi}$ is performed as follows: one of the guarded commands $g \rightarrow c$ in $\{gc_1, \ldots, gc_n\}$ whose guard g evaluates to *true* is chosen, then c is executed; otherwise, if no guard in $\{gc_1, \ldots, gc_n\}$ evaluates to *true* then the whole command $\texttt{if}\ \ldots\ \texttt{fi}$ terminates with failure.

The execution of $\texttt{do}\ gc_1 \,[]\, \ldots \,[]\, gc_n\ \texttt{od}$ is performed as follows: one of the guarded commands $g \rightarrow c$ in $\{gc_1, \ldots, gc_n\}$ whose guard g evaluates to *true* is chosen, then c is executed and the whole command $\texttt{do} \ldots \texttt{od}$ is executed again; otherwise, if no guard in $\{gc_1, \ldots, gc_n\}$ evaluates to *true* then the execution proceeds with the next command.

Symmetric Groups. A group G is a pair $\langle S, \circ \rangle$, where S is given a set and $\circ$ is a binary operation on S satisfying the following axioms: (i) $\forall x, y \in S.\, x \circ y \in S$ (closure), (ii) $\forall x, y, z \in S.\, (x \circ y) \circ z = x \circ (y \circ z)$ (associativity), (iii) $\exists e \in S. \forall x \in S. e \circ x = x \circ e = x$ (identity element), and (iv) $\forall x \in S. \exists y \in S.\, x \circ y = y \circ x = e$ (inverse element). The *order of a group* G is the cardinality of S. For any $x \in S$, for any $n \geq 0$, we write x^n to denote the term $x \circ \ldots \circ x$ with n occurrences of x. We stipulate that x^0 is e.

A group G is said to be *cyclic* iff there exists an element $x \in S$, called a *generator*, such that $S = \{x^n \mid n \geq 0\}$. We write G_x to denote the cyclic group generated by x.

We denote by $\mathit{Perm}(S)$ the set of all permutations (that is, bijections) on the set S. $\mathit{Perm}(S)$ is a group whose operation $\circ$ is function composition and the identity e is the identity permutation, denoted *id*. The *order of a permutation* p on a finite set S is the smallest natural number n such that $p^n = \mathit{id}$.

Computation Tree Logic. Computation Tree Logic (CTL) is a propositional branching time temporal logic [5].

Let *Elem* be a finite set of elementary propositions ranged over by b. The syntax of a CTL formula φ is as follows:

$$\varphi ::= b \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathsf{EX}\,\varphi \mid \mathsf{EG}\,\varphi \mid \mathsf{E}[\varphi_1\,\mathsf{U}\,\varphi_2]$$

Let us introduce the following abbreviations: (i) $\varphi_1 \vee \varphi_2$ for $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$, (ii) $\mathsf{EF}\varphi$ for $\mathsf{E}[\mathit{true}\,\mathsf{U}\,f]$ (iii) $\mathsf{AG}\,\varphi$ for $\neg\mathsf{EF}\,\neg\varphi$, (iv) $\mathsf{AF}\varphi$ for $\neg\mathsf{EG}\,\neg\varphi$, (v) $\mathsf{A}[\varphi_1\,\mathsf{U}\,\varphi_2]$ for $\neg\mathsf{E}[\neg\varphi_2\,\mathsf{U}\,(\neg\varphi_1 \wedge \neg\varphi_2)] \wedge \neg\mathsf{EG}\,\neg\varphi_2$, (vi) $\mathsf{AX}\,\varphi$ for $\neg\mathsf{EX}\,\neg\,\varphi$, (vii) $\mathsf{A}[\varphi_1\,\mathsf{R}\,\varphi_2]$ for $\neg\mathsf{E}[\neg\varphi_1\,\mathsf{U}\,\neg\varphi_2]$, and (viii) $\mathsf{E}[\varphi_1\,\mathsf{R}\,\varphi_2]$ for $\neg\mathsf{A}[\neg\varphi_1\,\mathsf{U}\,\neg\varphi_2]$.

We define the semantics of CTL by giving a Kripke structure $\mathcal{K} = \langle S, S_0, \lambda, R \rangle$, where: (i) S is a finite set of *states*, (ii) $S_0 \subseteq S$ is a set of *initial states*, (iii) $R \subseteq S \times S$ is a total *transition relation* (thus, $\forall u \in S,\ \exists v \in S,\ \langle u, v \rangle \in R$), and (iv) $\lambda : S \rightarrow \mathcal{P}(\mathit{Elem})$ is a total, *labelling function* that assigns to every state $s \in S$ a subset $\lambda(s)$ of the set *Elem*.

For reasons of simplicity, when the set of the initial states is a singleton $\{u\}$, we will feel free to identify $\{u\}$ with u.

A path π in $\mathcal{K}$ from a state is an infinite sequence $\langle s_0, s_1, \ldots \rangle$ of states such that, for all $i \geq 0$, $\langle s_i, s_{i+1} \rangle \in R$. For $i \geq 0$, we denote by π_i the i-th element

of π. The fact that a CTL formula φ holds in a state s of a Kripke structure $\mathcal{K}$ will be denoted by $\mathcal{K}, s \vDash \varphi$. For any CTL formula φ and state s, we define the relation $\mathcal{K}, s \vDash \varphi$ as follows:

$\mathcal{K}, s \vDash b$ iff $b \in \lambda(s)$
$\mathcal{K}, s \vDash \neg\, \varphi$ iff $\mathcal{K}, s \vDash \varphi$ does not hold
$\mathcal{K}, s \vDash \varphi_1 \wedge \varphi_2$ iff $\mathcal{K}, s \vDash \varphi_1$ and $\mathcal{K}, s \vDash \varphi_2$
$\mathcal{K}, s \vDash \mathsf{EX}\, \varphi$ iff there exists $\langle s, t\rangle \in R$ such that $\mathcal{K}, t \vDash \varphi$
$\mathcal{K}, s \vDash \mathsf{E}[\varphi_1\, \mathsf{U}\, \varphi_2]$ iff there exists a path $\pi = \langle s, s_1, \ldots\rangle$ in $\mathcal{K}$ and $i \geq 0$ such that $\mathcal{K}, \pi_i \vDash \varphi_2$ and for all $0 \leq j < i$, $\mathcal{K}, \pi_j \vDash \varphi_1$
$\mathcal{K}, s \vDash \mathsf{EG}\, \varphi$ iff there exists a path π such that $\pi_0 = s$ and for all $i \geq 0$, $\mathcal{K}, \pi_i \vDash \varphi$

2.1 Answer Set Programming

Answer set programming (ASP) is a declarative programming paradigm based on the answer set semantics of logic programs [10,14]. We assume the version of ASP with function symbols [3]. Now let us recall some basic definitions of ASP. For those not recalled here we refer to [3,10,14,15]. A *rule* r is an implication of the form:

$$a_1 \vee \ldots \vee a_k \leftarrow a_{k+1} \wedge \ldots \wedge a_m \wedge \mathtt{not}\, a_{m+1} \wedge \ldots \wedge \mathtt{not}\, a_n$$

where $a_1, \ldots, a_k, \ldots, a_n$ (for $k \geq 0, n \geq k$) are atoms and '`not`' denotes negation as failure [11]. Given a rule r, we define the following sets: $\mathit{head}(r) = \{a_1, \ldots, a_k\}$, $\mathit{pos}(r) = \{a_{k+1}, \ldots, a_m\}$, and $\mathit{neg}(r) = \{a_{m+1}, \ldots, a_n\}$. An *integrity constraint* is a rule r such that $\mathit{head}(r) = \emptyset$. A *logic program* is a set of rules. When we write a rule with variables, we actually mean all the ground instances of that rule.

An *interpretation* I of a program Π is a subset of the Herbrand base. The *Gelfond-Lifschitz transformation* of a program Π with respect to an interpretation I is the program $\Pi^I = \{\mathit{head}(r) \leftarrow \mathit{pos}(r) \mid r \in \Pi \wedge \mathit{neg}(r) \cap I = \emptyset\}$. An interpretation M is said to be an *answer set* of Π iff M is the least Herbrand model of Π^M. The answer set semantics of Π assigns to Π a set of answer sets, denoted $\mathit{ans}(\Pi)$. Given an answer set $M \in \mathit{ans}(\Pi)$ and an atom a, we write $M \models a$ to denote that $a \in M$.

3 Specifying Concurrent Programs

Let $\mathcal{P} = \{P_1, \ldots, P_k\}$ be a finite set of *processes*. With every process $P_i \in \mathcal{P}$ we associate a variable $\mathtt{s}_i$, called the *local state*, ranging over a finite domain L, which is the same for all processes. The variable $\mathtt{s}_i$ can be tested and modified by P_i only. All processes may test and modify also a *shared variable* `x`, which ranges over a finite domain D.

A *concurrent program* consists of a finite set $\mathcal{P}$ of processes that are executed in parallel and interact with each other through a communication protocol realized by a set of commands acting on the shared variable `x`. Here is the formal definition of a concurrent program.

Definition 1 (*k*-Process Concurrent Program). Let L be a set of local states and D be a domain of the shared variable $\mathtt{x}$. For any $k>1$, a *k-process concurrent program* C is a command of the form:

$$C: \quad \mathtt{s}_1 := l_1; \ldots; \mathtt{s}_k := l_k;\ \mathtt{x} := d_0;\ \mathtt{do}\ P_1 \,[]\, \ldots \,[]\, P_k\ \mathtt{od}$$

where $\mathtt{s}_1, \ldots, \mathtt{s}_k, \mathtt{x} \in \mathit{Var}$, $l_1, \ldots, l_k \in L$, and $d_0 \in D$.
Every process P_i in $P_1 \,[]\, \ldots \,[]\, P_k$ is a guarded command of the form:

$$P_i: \quad \mathit{true} \rightarrow \mathtt{if}\ gc_1\ \,[]\, \ldots \,[]\ gc_n\ \mathtt{fi}$$

Every guarded command gc in $gc_1 \,[]\, \ldots \,[]\, gc_n$ is of the form:

$$gc: \quad \mathtt{s}_i = l \wedge \mathtt{x} = d \ \rightarrow\ \mathtt{s}_i := l'; \mathtt{x} := d';$$

where $l, l' \in L$ and $d, d' \in D$. □

We shall use the guarded command $\mathtt{s}_i = l \wedge \mathtt{x} = d \rightarrow \mathit{skip}$ as a shorthand for $\mathtt{s}_i = l \wedge \mathtt{x} = d \rightarrow \mathtt{s}_i := l; \mathtt{x} := d$. The command $\mathtt{s}_1 := l_1; \ldots; \mathtt{s}_k := l_k;\ \mathtt{x} := d_0;$ is called *initialization* of C.

Example 1. Let L be the set $\{\mathtt{t}, \mathtt{u}\}$ and D be the set $\{\mathtt{0}, \mathtt{1}\}$. A 2-process concurrent program C is:

$$\mathtt{s}_1 := \mathtt{t};\ \mathtt{s}_2 := \mathtt{t};\ \mathtt{x} := \mathtt{0}\,;\ \mathtt{do}\ P_1 \,[]\, P_2\ \mathtt{od}$$

where P_1 and P_2 are defined as follows:

P_1 : *true* → **if**
  $\mathtt{s}_1 = \mathtt{t} \wedge \mathtt{x} = \mathtt{0} \rightarrow \mathtt{s}_1 := \mathtt{u};\ \mathtt{x} := \mathtt{0};$
[] $\mathtt{s}_1 = \mathtt{t} \wedge \mathtt{x} = \mathtt{1} \rightarrow \mathit{skip};$
[] $\mathtt{s}_1 = \mathtt{u} \wedge \mathtt{x} = \mathtt{0} \rightarrow \mathtt{s}_1 := \mathtt{t};\ \mathtt{x} := \mathtt{1};$
fi

P_2 : *true* → **if**
  $\mathtt{s}_2 = \mathtt{t} \wedge \mathtt{x} = \mathtt{1} \rightarrow \mathtt{s}_2 := \mathtt{u};\ \mathtt{x} := \mathtt{1};$
[] $\mathtt{s}_2 = \mathtt{t} \wedge \mathtt{x} = \mathtt{0} \rightarrow \mathit{skip};$
[] $\mathtt{s}_2 = \mathtt{u} \wedge \mathtt{x} = \mathtt{1} \rightarrow \mathtt{s}_2 := \mathtt{t};\ \mathtt{x} := \mathtt{0};$
fi

This program is the familiar program for two processes, each of which either 'thinks' in its noncritical section ($\mathtt{s}_i = \mathtt{t}$) or 'uses a resource' in its critical section ($\mathtt{s}_i = \mathtt{u}$). The shared variable $\mathtt{x}$ gives each process its turn to enter the critical section: if $\mathtt{x} = \mathtt{0}$, process P_1 is in its critical section, and if $\mathtt{x} = \mathtt{1}$, process P_2 is in its critical section. □

Now we introduce the semantics of concurrent programs by using Kripke structures. We model a state u of a k-process concurrent program C by a $(k+1)$-tuple $\langle l_1, \ldots, l_k, d\rangle$, where: (i) the first k components are the values of the local state variables $\mathtt{s}_1, \ldots, \mathtt{s}_k$, and (ii) d is the value of the shared variable $\mathtt{x}$.

Definition 2 (Kripke Structure Associated with a *k*-Process Concurrent Program). Let C be a k-process concurrent program of the form

$$C: \quad \mathtt{s}_1 := l_1; \ldots; \mathtt{s}_k := l_k;\ \mathtt{x} := d_0;\ \mathtt{do}\ P_1 \,[]\, \ldots \,[]\, P_k\ \mathtt{od}$$

where the l_i's belong to L and d_0 belongs to D. The *Kripke structure* $\mathcal{K}$ *associated with* C is the 4-tuple $\langle S, S_0, R, \lambda\rangle$, where:
(i) the set S of states is $L^k \times D$,
(ii) the set S_0 of initial states is the singleton $\{\langle l_1, \ldots, l_k, d_0\rangle\}$,

(iii) the set $R \subseteq S \times S$ of transitions
$\{\langle u, v\rangle \mid i, j \in \{1, \ldots, k\} \ \wedge \ \mathtt{s}_i = l \ \wedge \ \mathtt{x} = d \rightarrow \mathtt{s}_i := l'; \ \mathtt{x} := d' \text{ in } P_i \ \wedge$
$u(\mathtt{s}_i) = l \ \wedge \ u(\mathtt{x}) = d \ \wedge \ v(\mathtt{s}_i) = l' \ \wedge \ v(\mathtt{x}) = d' \ \wedge \ u \neq v \ \wedge$
$\forall j \neq i, \ u(\mathtt{s}_j) = v(\mathtt{s}_j)\}$,
where for all states $t \in S$, for all variables $\mathtt{x} \in \mathit{Var}$, $t(\mathtt{x})$ denotes the value of the variable $\mathtt{x}$ in t, and

(iv) for all states t of the form $\langle l_1, \ldots, l_k, d\rangle$, the value $\lambda(t)$ is defined to be $\{\mathtt{s}_1 = l_1, \ \ldots, \mathtt{s}_k = l_k, \mathtt{x} = d\}$.
The set *Elem* of the elementary propositions is the set $\bigcup_{t \in S} \lambda(t)$. □

We make the following assumptions about k-process concurrent programs.
(i) Since, by definition, the transition relation R of any Kripke structure is total, we have that every concurrent program C we consider, is *nonterminating*, in the sense that, in every state there exists a process P_i of C and a guarded command $g \rightarrow c$ of P_i such that: (i.1) g evaluates to *true*, and (i.2) c cannot be abbreviated to *skip*. This assumption restricts the class of concurrent programs we consider.
(ii) Every k-process concurrent program consists of *deterministic* processes, that is, for $i = 1, \ldots, k$, in every state, at most one guard of the guarded commands of process P_i evaluates to *true* (a similar assumption is made in [17]).

Note that the usual assumption that every guarded command is executed *atomically* (in the sense that only one process at a time among the processes of a concurrent program is selected and executed) is taken into account in an implicit way when constructing the transition relation R of the Kripke structure.

Example 2. Given the 2-process symmetric concurrent program C of Example 1, the associated Kripke structure $\langle S, \{s_0\}, R, \lambda\rangle$ is depicted in Figure 1. We depict it as a graph whose nodes are the states in S and whose edges represent the transitions in R. The set S of states includes the four state depicted in Figure 1 and also the states $\langle \mathtt{t}, \mathtt{u}, 0\rangle$, $\langle \mathtt{u}, \mathtt{t}, 1\rangle$, $\langle \mathtt{u}, \mathtt{u}, 0\rangle$, and $\langle \mathtt{u}, \mathtt{u}, 1\rangle$, which have not been depicted because they are not reachable from the initial state $\langle \mathtt{t}, \mathtt{t}, 0\rangle$. Each transition from state u to state v is associated with the guarded command $g \rightarrow c$ whose guard g evaluates to *true* in u. For the labelling function λ, we have that $\lambda(\langle \mathtt{t}, \mathtt{t}, 0\rangle)$ is $\{\mathtt{s}_1 = \mathtt{t}, \mathtt{s}_2 = \mathtt{t}, \mathtt{x} = 0\}$ and, similarly, for the other states. □

Having defined the Kripke structure associated with a given program, now we can define the notion of a program satisfying a given behavioural property.

Definition 3 (Satisfaction relation for a Concurrent Program). Let C be a k-process concurrent program, $\mathcal{K}$ be the Kripke structure associated with C, s_0 be the initial state of $\mathcal{K}$, and φ be a CTL formula. We say that C *satisfies* φ, denoted $C \models \varphi$, iff $\mathcal{K}, s_0 \models \varphi$. □

Example 3. Let us consider the 2-process concurrent program C defined in Example 1. We associate with the local states $\mathtt{t}$ (short for 'think') and $\mathtt{u}$ (short for 'use') two regions of code, called the noncritical section and the critical section, respectively. We require that the region of code associated with state $\mathtt{u}$ should be executed in a mutually exclusive way. This is formalized by the CTL formula

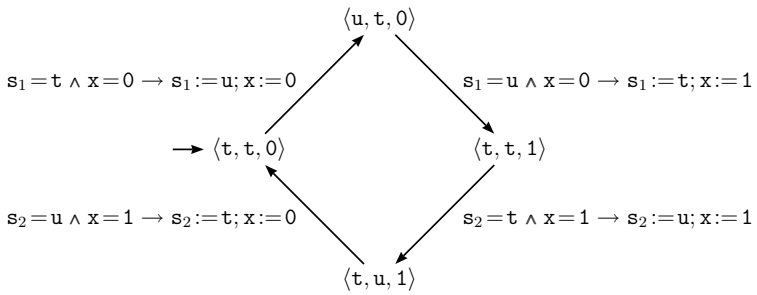


Fig. 1. The transition relation R of the Kripke structure $\mathcal{K} = \langle S, \{s_0\}, R, \lambda\rangle$ associated with the concurrent program C of Example 1. The initial state s_0 is $\langle \mathtt{t}, \mathtt{t}, \mathtt{0}\rangle$. The arcs are labelled by the guarded commands which are responsible for the transition.

$\varphi =_{def} \mathsf{AG}[\neg(\mathtt{s}_1 = \mathtt{u} \ \wedge \ \mathtt{s}_2 = \mathtt{u})]$, and we have that $C \models \varphi$ holds because for the Kripke structure $\mathcal{K}$ of Example 2 (see Figure 1), we have that $\mathcal{K}, s_0 \models \varphi$ (indeed, there is no path starting from the initial state $s_0 = \langle \mathtt{t}, \mathtt{t}, \mathtt{0}\rangle$ which leads the system to either the state $\langle \mathtt{u}, \mathtt{u}, \mathtt{0}\rangle$ or the state $\langle \mathtt{u}, \mathtt{u}, \mathtt{1}\rangle$). □

Often, in our setting a k-concurrent program consists of *symmetric processes*, the symmetry being determined by the fact that, for any two processes P_i and P_j, for $i \neq j$, we have that P_j can be obtained from P_i by permuting the values of the shared variable $\mathtt{x}$ in the guarded commands. Indeed, as shown in Example 1, the guarded commands in P_2 can be obtained from those in P_1 by interchanging $\mathtt{0}$ and $\mathtt{1}$. In practice, the property of symmetry is very common in many concurrent programs, and our task is precisely the one of automatically synthesizing symmetric processes. This observation motivates a notion of *symmetry* which we now introduce by using cyclic groups. A similar approach has been followed for the automated verification of concurrent systems in [8].

Definition 4 (*k*-Generating Function). Given an integer $k > 1$, and a finite domain D, we say that $f \in Perm(D)$ is *a k-generating function* iff either $f = id$ or f is a generator of a cyclic group $G_f = \{id, f, f^2, \ldots, f^{k-1}\}$ of order k. □

Let us introduce the following notation. Given a guarded command gc of the form:

$$\mathtt{s}_i = l \wedge \mathtt{x} = d \ \rightarrow \ \mathtt{s}_i := l'; \ \mathtt{x} := d';$$

and a k-generating function f, we denote by $f(gc)$ the guarded command:

$$\mathtt{s}_{(i \bmod k)+1} = l \wedge \mathtt{x} = f(d) \ \rightarrow \ \mathtt{s}_{(i \bmod k)+1} := l'; \ \mathtt{x} := f(d');$$

Definition 5 (*k*-Process Symmetric Concurrent Program). Given a k-generating function f, a *k-process symmetric concurrent program* C is a command of the form:

$$C: \quad \mathtt{s}_1 := l_0; \ldots; \mathtt{s}_k := l_0; \ \mathtt{x} := d_0; \ \mathtt{do} \ P_1 \ [\!] \ \ldots \ [\!] \ P_k \ \mathtt{od}$$

where, for all processes P_i, for all guarded commands gc, gc is in P_i iff $f(gc)$ is in $P_{(i \bmod k)+1}$. □

Example 4. Let us consider the 2-process concurrent program C of Example 1. The group $Perm(D)$ of permutations over $D = \{0, 1\}$ is made out of the following two permutations only: $f_1 = \{\langle 0, 0\rangle, \langle 1, 1\rangle\}$ and $f_2 = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$. The 2-generating function f_2 shows that the concurrent program C is symmetric.

P_1 : *true* → if
  $s_1{=}t \wedge x{=}0 \rightarrow s_1{:=}u;\ x{:=}0;$
 [] $s_1{=}t \wedge x{=}1 \rightarrow$ *skip*;
 [] $s_1{=}u \wedge x{=}0 \rightarrow s_1{:=}t;\ x{:=}1;$
fi

P_2 : *true* → if
  $s_2{=}t \wedge x{=}f_2(0) \rightarrow s_2{:=}u;\ x{:=}f_2(0);$
 [] $s_2{=}t \wedge x{=}f_2(1) \rightarrow$ *skip*;
 [] $s_2{=}u \wedge x{=}f_2(0) \rightarrow s_2{:=}t;\ x{:=}f_2(1);$
fi □

By definition, one can generate a k-process symmetric concurrent program C from one of the processes in C by applying the generating function f. Moreover, it is often the case that all processes of a given program C also share additional structural properties, besides those determined by f. For instance, in the case of Example 4, we have that both process P_1 and P_2 may move from the local state t to the local state u, or from t to t, or from u to t. These additional structural properties define a *local transition relation* $T \subseteq L \times L$ which together with the k-generating function f, defines a so called *symmetric program structure* $\Sigma = \langle f, T\rangle$. A pair $\langle l, l'\rangle$ in T will also be denoted by $l \mapsto l'$.

Our synthesis problem can be defined as follows.

Definition 6 (Synthesis Problem of a *k*-Process Symmetric Concurrent Program). The *synthesis problem of a k-process symmetric concurrent program* C starting from: (i) a CTL formula φ, and (ii) a symmetric program structure $\Sigma = \langle f, T\rangle$, where f is a k-generating function and T is a local transition relation, consists in finding C such that $C \models \varphi$ holds. □

Note that there exists a CTL formula that characterizes the set of initial states. In particular, the initial state $\langle l_1, \ldots, l_k, d_0\rangle$ can be characterized by the CTL formula $s_1 = l_1 \wedge \ldots \wedge s_k = l_k \wedge x = d_0$, where we assume that each conjunct belongs to *Elem*. However, for reasons of simplicity, we assume that the initial state s_0 is given to our synthesis procedure as an additional input (see clause 1 of the logic program Π_φ of Definition 7).

4 Synthesising Concurrent Programs

In this section we present our synthesis procedure based on ASP. We encode the desired behavioural property φ of our k-process concurrent program to be synthesized as a logic programs Π_φ, and the desired structural property Σ as a logic programs Π_Σ. Programs Π_φ and Π_Σ are defined in the following Definition 7 and 8, respectively.

Definition 7 (Logic program encoding a behavioural property). Let φ be a CTL formula expressing a behavioural property. The logic program Π_φ encoding φ is as follows:

1. $\leftarrow$ not $sat(s_0, \varphi)$

2. $sat(U, F) \leftarrow elem(F, U)$
3. $sat(U, not(F)) \leftarrow \texttt{not}\ sat(U, F)$
4. $sat(U, and(F_1, F_2)) \leftarrow sat(U, F_1) \wedge sat(U, F_2)$
5. $sat(U, ex(F)) \leftarrow tr(U, V) \wedge sat(V, F)$
6. $sat(U, eu(F_1, F_2)) \leftarrow sat(U, F_2)$
7. $sat(U, eu(F_1, F_2)) \leftarrow sat(U, F_1) \wedge tr(U, V) \wedge sat(V, eu(F_1, F_2))$
8. $sat(U, eg(F)) \leftarrow satpath(U, V, F) \wedge satpath(V, V, F)$
9. $satpath(U, V, F) \leftarrow sat(U, F) \wedge tr(U, V) \wedge sat(V, F)$
10. $satpath(U, Z, F) \leftarrow sat(U, F) \wedge tr(U, V) \wedge satpath(V, Z, F)$

11.1 $tr(s(S_1, \ldots, S_k, X), s(S'_1, \ldots, S'_k, X')) \leftarrow reachable(s(S_1, \ldots, S_k, X)) \wedge gc(1, S_1, X, S'_1, X') \wedge \langle S_1, X\rangle \neq \langle S'_1, X'\rangle$

$\ldots$

11.k $tr(s(S_1, \ldots, S_k, X), s(S'_1, \ldots, S'_k, X')) \leftarrow reachable(s(S_1, \ldots, S_k, X)) \wedge gc(k, S_k, X, S'_k, X') \wedge \langle S_k, X\rangle \neq \langle S'_k, X'\rangle$

12. $\leftarrow \texttt{not}\ out(S) \wedge reachable(S)$
13. $out(S) \leftarrow tr(S, Z)$
14. $reachable(s_0) \leftarrow$
15. $reachable(S) \leftarrow tr(Z, S)$

where the predicates are defined as follows: (i) $sat(U, F)$ holds iff the formula F holds in state U, (ii) $elem(b, u)$ holds iff $b \in \lambda(u)$, that is, the elementary proposition b holds in state u, (iii) $satpath(U, V, F)$ holds iff there exists a path from state U to state V such that every state in that path satisfies the formula F, (iv) $tr(s(S_1, \ldots, S_k, X), s(S'_1, \ldots, S'_k, X'))$ holds iff the pair of states $\langle\langle S_1, \ldots, S_k, X\rangle, \langle S'_1, \ldots, S'_k, X'\rangle\rangle$ belongs to the transition relation R of the Kripke structure associated with the program C to be synthesized, and (v) the predicates *out* and *reachable* force the relation R to be total (in particular, $out(S)$ holds iff from state S there is an outgoing edge, and $reachable(S)$ holds iff there is a path from the initial state s_0 to state S.) □

Rule 1 is required for ensuring that φ holds in the initial state s_0 representing the initialization $\mathtt{s}_1 := l_0; \ldots; \mathtt{s}_k := l_0;\ \mathtt{x} := d_0$ of the k-process symmetric concurrent program to be synthesized. Rule 11.i defines the interleaved execution of the guarded commands, that is, for all states U and V, $tr(U, V)$ holds iff U is a reachable state, and there exists a guarded command gc of process P_i whose guard evaluates to *true* in U and whose execution leads from state U to state V.

Definition 8 (Logic program encoding a structural property). Let L be the set of local states and D be the domain of the shared variable. Let $\Sigma = \langle f, T\rangle$ be a symmetric program structure of a k-process symmetric concurrent program. The logic program Π_Σ is defined as follows:

1.1 $\bigvee_{\langle S', X'\rangle \in Next(\langle S_1, X\rangle)} gc(1, S_1, X, S', X') \leftarrow reachable(S_1, S_2, \ldots, S_k, X)$

1.2 $\leftarrow gc(1, S, X, S', X') \wedge gc(1, S, X, S'', X'') \wedge \langle S', X'\rangle \neq \langle S'', X''\rangle$

2.1 $gc(2, S, f(X), S', f(X')) \leftarrow gc(1, S, X, S', X')$

2.2 $\leftarrow gc(2, S, X, S', X') \wedge \texttt{not}\ ps(2, S, X)$

$$
\begin{array}{ll}
2.3 & ps(2, S_2, X) \leftarrow \mathit{reachable}(S_1, S_2, \ldots, S_k, X) \\
 & \quad \ldots \\
k.1 & gc(k, S, f(X), S', f(X')) \leftarrow gc(k{-}1, S, X, S', X') \\
k.2 & \leftarrow gc(k, S, X, S', X') \wedge \texttt{not}\ ps(k, S, X) \\
k.3 & ps(k, S_k, X) \leftarrow \mathit{reachable}(S_1, S_2, \ldots, S_k, X)
\end{array}
$$

where: (i) $gc(i, S, X, S', X')$ holds iff $\mathtt{s_i} = l \wedge \mathtt{x} = d \rightarrow \mathtt{s_i} := l'; \mathtt{x} := d'$ is in P_i, (ii) f is a k-generating function, (iii) $ps(i, S, X)$ holds iff there exists a reachable state of the form $\langle S_1, \ldots, S_{i-1}, S, S_{i+1}, \ldots, S_k, X\rangle$, and (iv) for all $l \in L$, $d \in D$, $\mathit{Next}(l, d) = \{\langle l', d'\rangle \mid l \mapsto l' \in T \wedge d' \in D\}$. □

Rules 1.1 and 1.2 generate a set of guarded commands for process P_1. The disjunction in the head of Rule 1.1 is over all possible guarded commands that P_1 may execute. The set of those guarded commands is defined using the sets $\mathit{Next}(l, d)$, one for each $l \in L$ and $d \in D$. The integrity constraint 1.2 enforces the generation of a set of guarded commands in which any two guards of the guarded commands in P_1 are mutually exclusive (recall that we consider only deterministic processes).

For $j = 2, \ldots, k$, Rules $j.1$, $j.2$ and $j.3$ realize Definition 5. We use Rule $j.1$ to derive a guarded command in P_j from a guarded command of the process P_{j-1}. Rule $j.2$ ensures that for every guarded command $g \rightarrow c$ derived by $j.1$, there exists a reachable state U such that in U the guard g evaluates to *true*.

Now we present a theorem establishing the correctness of our synthesis procedure. It relates the k-process symmetric concurrent programs satisfying φ with the answer sets of the logic program $\Pi_\varphi \cup \Pi_\Sigma$. Obviously, the correctness of the synthesis procedure implies also the correctness of the programs Π_φ and Π_Σ encoding the behavioural properties and the structural properties, as specified in Definition 7 and 8, respectively.

Theorem 1 (Correctness of Synthesis). *Let* $\Pi = \Pi_\varphi \cup \Pi_\Sigma$ *be the logic program obtained, as specified by Definitions* 7 *and* 8*, from:* (i) *a* CTL *formula* φ *and* (ii) *a symmetric program structure* $\Sigma = \langle f, T\rangle$*. Then,*

$$\big(\mathtt{s_1} := l_0; \ldots; \mathtt{s_k} := l_0;\ \mathtt{x} := d_0;\ \mathtt{do}\ P_1\ [\!]\ \ldots\ [\!]\ P_k\ \mathtt{od}\big)\ \models\ \varphi$$

iff there exists an answer set M *in* $\mathit{ans}(\Pi)$ *such that*

$\forall i \in \{1, \ldots, k\}$, $\forall l, l' \in L$, $\forall d, d' \in D$,

$\big(\mathtt{s}_i = l \wedge \mathtt{x} = d \rightarrow \mathtt{s}_i := l'; \mathtt{x} := d'\big)$ *is in* P_i *iff* $M \models \mathtt{gc}(i, l, d, l', d')$.

5 Experimental Results

In this section we present some experimental results obtained by applying our synthesis procedure to mutual exclusion protocols. All experiments have been performed on an Intel Core 2 Duo E7300 2.66GHz under the Linux operating system.

The first synthesis we did is the one of a simple program, called 2-*mutex*-1, for two processes enjoying the mutual exclusion property only, and then we progressively increased the number of properties that the synthesized program should

satisfy (see Table 1). In that table the program k-*mutex*-p denotes a synthesized program for k processes satisfying p behavioural properties. For instance, program 2-*mutex*-4 is the synthesized program that works for 2 processes and enjoys the four behavioural properties: (i) *ME* (mutual exclusion), (ii) *SF* (starvation freedom), (iii) *BO* (bounded overtaking), and (iv) *MR* (maximal reactivity), defined by CTL formulas as follows.

(i) *Mutual Exclusion*, that is, it is not the case that process P_i is in its critical section ($\mathtt{s}_i = \mathtt{u}$), and process P_j is in its critical section ($\mathtt{s}_j = \mathtt{u}$) at the same time: for all i, j in $\{1, \ldots, k\}$, with $i \neq j$,

$$\mathsf{AG}\,\neg(\mathtt{s}_i = \mathtt{u} \wedge \mathtt{s}_j = \mathtt{u}) \qquad (ME)$$

(ii) *Starvation Freedom*, that is, if a process is waiting to enter the critical section ($\mathtt{s}_i = \mathtt{w}$), then after a finite amount of time, process P_i will execute its critical section ($\mathtt{s}_i = \mathtt{u}$): for all i in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,(\mathtt{s}_i = \mathtt{w} \rightarrow \mathsf{AF}\,\mathtt{s}_i = \mathtt{u}) \qquad (SF)$$

(iii) *Bounded Overtaking*, that is, while process P_i is in its waiting section, any other process P_j exits from its critical section at most once: for all i, j in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,((\mathtt{s}_i = \mathtt{w} \wedge \mathtt{s}_j = \mathtt{u}) \rightarrow \mathsf{AF}\,(\mathtt{s}_j = \mathtt{t} \wedge \mathsf{A}[\neg(\mathtt{s}_j = \mathtt{u})\,\mathsf{U}\,\mathtt{s}_i = \mathtt{u}])) \quad (BO)$$

(iv) *Maximal Reactivity*, that is, if process P_i is waiting to execute the critical section and all other processes are executing their noncritical sections, then in the next state P_i will enter its critical section: for all i in $\{1, \ldots, k\}$,

$$\mathsf{AG}\,((\mathtt{s}_i = \mathtt{w} \wedge \textstyle\bigwedge_{j \in \{1,\ldots,k\} \setminus \{i\}} \mathtt{s}_j = \mathtt{t}) \rightarrow \mathsf{EX}\,\mathtt{s}_i = \mathtt{u}) \qquad (MR)$$

In our synthesis experiments we have made the following choices for s_0, L, D, f, and T.

The initial state s_0 is $\langle \mathtt{t}, \mathtt{t}, \mathtt{0} \rangle$ and $\langle \mathtt{t}, \mathtt{t}, \mathtt{t}, \mathtt{0} \rangle$ for the 2- and 3-process symmetric concurrent programs, respectively.

The set L of the local states for the variables $\mathtt{s}_i$'s is $\{\mathtt{t}, \mathtt{w}, \mathtt{u}\}$, where $\mathtt{t}$ represents the *noncritical section*, $\mathtt{w}$ represents the *waiting section*, and $\mathtt{u}$ represents the *critical section*.

The domain D of the shared variable $\mathtt{x}$ is a finite set of natural numbers whose cardinality $|D|$ depends on: (i) the number k of the processes to be synthesized, and (ii) the properties that the concurrent program should satisfy. The value of $|D|$ is not known a priori, and we guess it at the beginning of our synthesis task. If the synthesis fails, we increase the value of $|D|$, hoping for a successful synthesis with a larger value of $|D|$.

The k-generating function f is chosen among the following ones: (i) id is the identity function, (ii) $f_1 = \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$, (iii) $f_2 = \{\langle 0, 1\rangle, \langle 1, 0\rangle, \langle 2, 2\rangle\}$, and (iv) $f_3 = \{\langle 0, 1\rangle, \langle 1, 2\rangle, \langle 2, 0\rangle\}$.

The local transition relation T is $\{\mathtt{t} \mapsto \mathtt{w},\ \mathtt{w} \mapsto \mathtt{w},\ \mathtt{w} \mapsto \mathtt{u},\ \mathtt{u} \mapsto \mathtt{t}\}$. The pair $\mathtt{t} \mapsto \mathtt{w}$ denotes that, once the noncritical section has been executed, a process enters the waiting section. The pairs $\mathtt{w} \mapsto \mathtt{w}$ and $\mathtt{w} \mapsto \mathtt{u}$ denote that a process may repeat (possibly an unbounded number of times) the execution of its waiting section and then may enter its critical section. The pair $\mathtt{u} \mapsto \mathtt{t}$ denotes that, once the critical section has been executed, a process enters its noncritical section.

Program	Satisfied Properties	$\|D\|$	f	$\|ans(\Pi)\|$	Time
2-*mutex*-1	*ME*	2	*id*	6	0.07
2-*mutex*-1	*ME*	2	f_1	7	0.70
2-*mutex*-2	*ME*, *SF*	2	f_1	3	0.71
2-*mutex*-3	*ME*, *SF*, *BO*	2	f_1	3	1.44
2-*mutex*-4	*ME*, *SF*, *BO*, *MR*	3	f_2	2	11.7
3-*mutex*-1	*ME*	2	*id*	5	0.95
3-*mutex*-1	*ME*	2	f_1	10	0.87
3-*mutex*-2	*ME*, *SF*	3	f_3	8	152
3-*mutex*-3	*ME*, *SF*, *BO*	3	f_3	8	1700

Table 1. Column named Program gives the names of the synthesized programs. k-*mutex*-p denotes the mutual exclusion program for k processes and p behavioural properties that are indicated in the column named Satisfied Properties. *ME*, *SF*, *BO* and *MR* stand for 'mutual exclusion', 'starvation freedom', 'bounded overtaking', and 'maximal reactivity', respectively. Column named $|D|$ gives the cardinality of the domain of the shared variable `x`. Column named f gives the k-generating functions (they are defined in the text). Column named $|ans(\Pi)|$ gives the cardinality of $ans(\Pi)$, that is, the number of answer sets of program $\Pi = \Pi_\varphi \cup \Pi_\Sigma$. In column named Time we indicate the times (in seconds) taken for the synthesis using the smodels [20].

In Figures 2 and 3 we present the syntax and the semantics of the synthesized program, called 2-*mutex*-4, for the 2-process mutual exclusion problem described in Example 3. (Program 2-*mutex*-4 is essentially the same as the Peterson algorithm [18], but it uses a single shared variable.)

6 Related Work and Concluding Remarks

Two well known, early works on synthesis of concurrent programs were those by Emerson and Clark [6] and Manna and Wolper [16].

In [6] Emerson and Clark introduce the notion of a synchronization skeleton as an abstraction of the actual processes in concurrent programs. They synthesize programs for a shared-memory model of execution by extracting the synchronization skeletons from the models of CTL specifications using a tableau-based decision procedure for the satisfiability of CTL formulas. This extraction procedure is not completely mechanized.

Similarly to [6] in [16] Manna and Wolper present a method for synthesizing synchronization instructions for processes in a message-passing model of execution from a Propositional Temporal Logic (PTL) using a tableau-based decision procedure for the satisfiability of PTL formulas. The instructions synthesized by their method are written as Communicating Sequential Processes [13].

In [19] Piterman, Pnueli, and Sa'ar consider the problem of the design of digital circuits from Linear Temporal Logic (LTL) specifications and give an

```
     P1 :                                    P2 :
     true → if                               true → if
(1)      s1 = t ∧ x = 0 → s1 := w; x := 2;       s2 = t ∧ x = 0 → s2 := w; x := 2;
(2)    [] s1 = t ∧ x = 1 → s1 := w; x := 2;    [] s2 = t ∧ x = 1 → s2 := w; x := 2;
(3)    [] s1 = t ∧ x = 2 → s1 := w; x := 1;    [] s2 = t ∧ x = 2 → s2 := w; x := 0;
(4)    [] s1 = w ∧ x = 0 → s1 := u; x := 0;    [] s2 = w ∧ x = 0 → skip;
(5)    [] s1 = w ∧ x = 1 → skip;               [] s2 = w ∧ x = 1 → s2 := u; x := 1;
(6)    [] s1 = w ∧ x = 2 → s1 := u; x := 2;    [] s2 = w ∧ x = 2 → s2 := u; x := 2;
(7)    [] s1 = u ∧ x = 2 → s1 := t; x := 1;    [] s2 = u ∧ x = 2 → s2 := t; x := 0;
(8)    [] s1 = u ∧ x = 0 → s1 := t; x := 2;    [] s2 = u ∧ x = 1 → s2 := t; x := 2;
     fi                                      fi
```

Fig. 2. The two synthesized processes P_1 and P_2 of the program 2-*mutex*-4: $\mathtt{s}_1 := \mathtt{t}$; $\mathtt{s}_2 := \mathtt{t}$; $\mathtt{x} := 0$; do P_1 [] P_2 od. It enjoys the following properties: mutual exclusion, starvation freedom, bounded overtaking, and maximal reactivity.

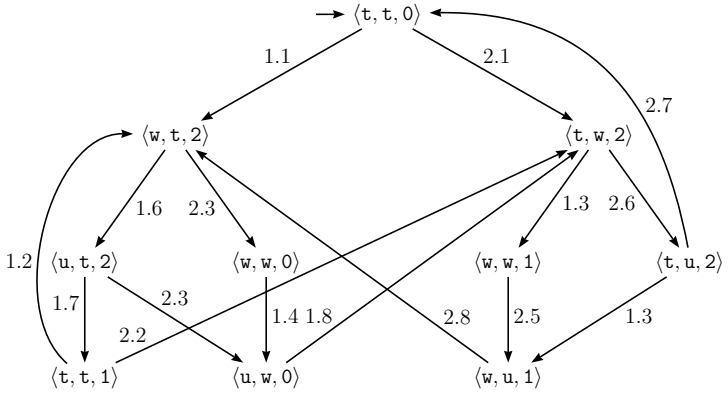


Fig. 3. The transition relation of the Kripke structure associated with the 2-process concurrent program 2-*mutex*-4. The initial state is $\langle \mathtt{t}, \mathtt{t}, 0 \rangle$. For $i = 1, 2$, an arc labelled $i.n$ indicates that the guarded command n of process P_i is responsible for that transition.

$O(N^3)$ algorithm to construct an automaton satisfying a formula of a particular class of LTL specifications.

We closely follow the approaches of [6] and [16]. In particular we synthesize concurrent processes that communicate with each other by means of shared variables starting from CTL specifications. The programs we synthesize are written as guarded commands [7].

In order to reduce the search space of our synthesis problem, we have used a notion of symmetric concurrent programs which is similar to the one which was introduced in [1,8] to overcome the state explosion problem. Our notion of symmetry is formalized using group theory, similarly to what has been done in [8] for model checking.

Similarly to Attie and Emerson [2], we also propose a method for the synthesis task and we separate the behavioural properties from the structural properties.

However, in our approach the structural properties, such as symmetry, are represented in the symmetric program structures, rather than an automata based formalism.

We have implemented our synthesis method in Answer Set Programming (ASP). One advantage of our method over [1,6,16] is its generality: besides temporal properties, we can specify structural properties, such as the above mentioned symmetry, and our ASP program will automatically synthesize concurrent programs satisfying the desired properties without the need for ad hoc algorithms.

To the best of our knowledge, there is only one paper by Heymans, Nieuwenborgh and Vermeir [12] who use Answer Set Programming for the synthesis of concurrent programs. They have extended the ASP paradigm by adding preferences among models and they have developed an answer set system, called OLPS. Using OLPS they perform the synthesis of concurrent programs following the approach proposed in [6]. The synthesis method is not completely automatic and, in particular, the shared variables are manually introduced during the extraction of the synchronization skeleton. We do not require any extension of the ASP paradigm, we use the by now standard ASP systems, such as DLV [9] and smodels [20], and every steps of our synthesis procedure is fully automatic.

As future work we plan to explore various techniques for reducing the search space of the synthesis procedure and, thus, we hope to synthesize protocols for a larger number of processes and more complex properties to be guaranteed. Among these techniques we envisage to apply those used in compositional model checking [4].

References

1. P. C. Attie and E. A. Emerson. Synthesis of Concurrent Programs with Many Similar Processes *ACM Trans. on Program. Lang. and Syst.*, 51–115, 1998.
2. P. C. Attie and E. A. Emerson. Synthesis of Concurrent Programs for an Atomic Read/Write Model of Computation. *ACM Trans. Program. Lang. Syst.*, 187–242, 2001.
3. F. Calimeri, S. Cozza, G. Ianni and N. Leone. Enhancing ASP by Functions: Decidable Classes and Implementation Techniques. *Proceedings of the* 24-*th AAAI Conference on Artificial Intelligence 2010*, 1666–1670, 2010.
4. E. M. Clarke Jr., D. E. Long, and K. L. McMillan. Compositional model checking. *Logic in Computer Science, LICS '89, Proceedings*, IEEE Computer Society, 353–362, 1989.
5. E. M. Clarke Jr., O. Grumber and D. A. Peled. *Model Checking*. The MIT Press, 1999.
6. E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic. *Workshop on Logic of Programs*, London, UK, Springer-Verlag, 52–71, 1982.
7. E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
8. E. A. Emerson and A. P. Sistla. Symmetry and Model Checking. *Formal Methods in System Design*: 9, 1–2, 105–131, 1996.

9. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri and F. Scarcello. The DLV system for knowledge representation and reasoning *ACM Trans. Comput. Logic*: 7, 499–562, 2006.
`http://www.dlvsystem.com/dlvsystem/index.php/DLV`
10. M. Gelfond and V. Lifschitz. The Stable Model Semantics For Logic Programming. *Proc. of the Fifth Intern. Conf. and Symp. on Logic Programming*, Seattle, MIT Press, 1070–1080, 1988.
11. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*: 9, 365–385, 1991.
12. S. Heymans, D. Van Nieuwenborgh and D. Vermeir. Synthesis from Temporal Specifications using Preferred Answer Set Programming. *Lecture Notes in Computer Science* no. 3701, Springer, 280–294, 2005.
13. C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.
14. V. Lifschitz. Answer Set Programming and Plan Generation. *Artificial Intelligence* no. 138, 39-54, 2002.
15. V. Lifschitz. What Is Answer Set Programming? *Proceedings of the AAAI Conference on Artificial Intelligence*, MIT Press, 1594–1597, 2008.
16. Z. Manna and P. Wolper: Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.*, 68–93, 1984.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive Systems: Specification.* Springer-Verlag, 1991.
18. G. L. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3):115–116, 1981.
19. N. Piterman, A. Pnueli and Y. Sa'ar. Synthesis of Reactive(1) Designs. *Lecture Notes in Computer Science* no. 3855, Springer, 364–380, 2006.
20. T. Syrjänen and I. Niemelä. The Smodels System. *Lecture Notes in Computer Science* no. 2173, Springer, 434–438, 2001.
`http://www.tcs.hut.fi/Software/smodels/`

PRODPROC - Product and Production Process Modeling and Configuration *

Dario Campagna and Andrea Formisano

Dipartimento di Matematica e Informatica, Università di Perugia, Italy
(dario.campagna|formis)@dmi.unipg.it

Abstract. Software product configurators are an emerging technology that supports companies in deploying mass customization strategies. Such strategies need to cover the management of the whole customizable product cycle. Adding process modeling and configuration features to a product configurator may improve its ability to assist mass customization development. In this paper, we describe a modeling framework that allows one to model both a product and its production process. We first introduce our framework focusing on its process modeling capabilities. Then, we outline a possible implementation based on Constraint Logic Programming of such product/process configuration system. A comparison with some of the existing systems for product configuration and process modeling concludes the paper.

1 Introduction

In the past years many companies started to operate according to *mass customization* strategies. Such strategies aim at selling products that satisfy customer's needs, preserving as much as possible the advantages of *mass production* in terms of efficiency and productivity. The products offered by such companies, usually called *configurable products*, have a predefined basic structure that can be customized by combining a series of available components and options (modules, accessories, etc.) or by specifying suitable parameters (lengths, tensions, etc.). Actually, a configurable product does not correspond to a specific physical object, but identify sets of (physical) objects that a company can realize. A *configured product* is a single variant of a configurable product, obtained by specifying each of its customizable attributes, which corresponds to a fully-specified physical object. The *configuration process* consists of a series of activities and operations ranging from the acquisition of information about the variant of the product requested by the customer, to the generation of data for its realization.

The mass customization operating mode involves a series of difficulties that companies struggle to resolve by using traditional software tools, designed for repetitive productions. As more companies started offering configurable products, different systems designed for supporting them in deploying mass customization strategies appeared. These systems are called *software product configurators* and allow one to effectively and efficiently deal with the configuration process [21]. They offer functionality for the

* Research partially founded by GNCS-2011 and MIUR-PRIN-2008 projects, and grants 2009.010.0336 and 2010.011.0403.

representation of configurable products through *product models*, and for organizing and managing the acquisition of information about the product variants to be realized.

Mass customization strategies need to cover the management of the whole customization product cycle, from customer order to final manufacturing. Current software product configurators focus only on the support to product configuration, and do not cover aspects related to the production process planning. Extending the use of configuration techniques from products to processes, may avoid or reduce planning impossibilities due to constraints introduced in the product configuration phase, as well as configuration impossibilities due to production planning requirements. Existing languages/tools for process modeling, such as BPMN [28] and YAWL [24], do not offer suitable features for specifying production processes and process configuration. Moreover, they lack the capability of modeling, in a single uniform setting, product models and their corresponding process models. The framework we propose, called PRODPROC, intends to overcome these limitations and act as a core for a full-fledged configuration system, covering the whole customization product cycle.

2 A Framework for Product/Production Modeling

In this section we present the PRODPROC framework by exploiting a working example that will be used throughout the paper (cf., Sections 2.1 and 2.2). We also provide a brief description of PRODPROC semantics in term of model instances (Sect. 2.3). See [6] for a description of PRODPROC graphical modeling language.

A PRODPROC *model* consists of a description of a product, a description of a process, and a set of constraints coupling the two. In order to introduce the PRODPROC features let us consider a rectangular base prefabricated component multi-story building, together with its construction process. More specifically, a building is composed by the followings parts: story, roof, heating service, ventilation service, sanitary service, electrical/lighting service, suspended ceiling, floor, partition wall system. For the purposes of this paper, we consider two types of building:

Warehouse: it is a single story building, it has no mandatory service except for the electrical/lighting service, it has no partition wall system and no suspended ceiling, it may have a basement.

Office building: it may have a basement and up to three stories, all services except ventilation are mandatory, suspended ceiling and floor are mandatory for each story, each story may have a partition wall system.

The building construction process can be split in four main phases: preparation and development of the building site; building shell and building envelope works; building services equipment; finishing works. (For a detailed description of such phases see [23].)

2.1 Product Description

A product is modeled as a multi-graph, called *product model graph*, and a set of constraints. The nodes of the graph represent the components of the product. The edges represent the *has-part*/*is-part-of* relations between product components. We require the presence of a node without entering edges in the product model graph. We call this

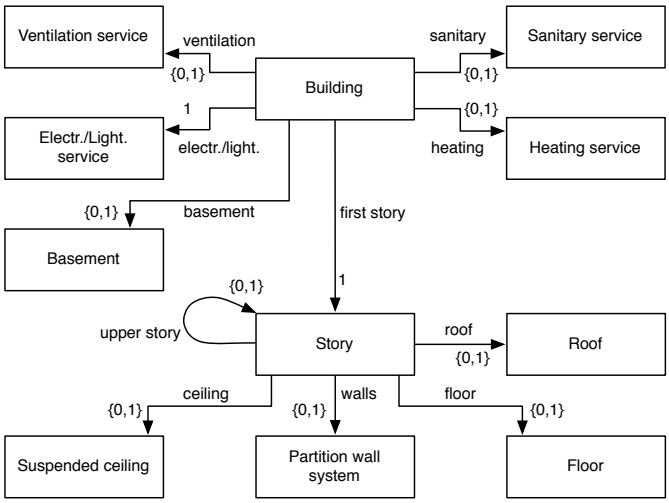


Fig. 1. Building product model graph.

node *root node*. Such a product description will represent a configurable product whose configuration can lead to the definition of different (producible) variants that can be represented as trees. Nodes of these trees correspond to physical components, whose characteristics are all determined. The tree structure describes how the single components taken together define a configured product. Fig. 1 shows the product model graph for our example. Edges are labeled with names describing the *has-part* relations and numbers indicating the admitted values for the cardinalities.

Each node/component of a product model graph is characterized by a name, a set of variables representing configurable features of the component, and a set of constraints that may involve variables of the node as well as variables of its ancestors in the graph. Each variable is endowed with a finite domain (typically, a finite set of integers or strings), i.e., the set of its possible values. In the description of a configured product, physical components will be represented as instances of nodes in the product model graph. An instance of a node $NodeName$ consists of the name $NodeName$, a unique id, and a set of variables equals to the one of $NodeName$. Each variable will have a value assigned. The instance of the *root node* will be the root of the configured product tree. For example, the node *Building* in Fig. 1, which is the root node of the product model graph, is defined as the triple $\langle Building, \mathcal{V}_{Building}, \mathcal{C}_{Building} \rangle$, where the involved variables and the set of constraint are as follows:

$$\mathcal{V}_{Building} = \{\langle BuildingType, \{\text{Warehouse}, \text{Office building}\}\rangle, \\ \langle StoryNum, [1,3]\rangle, \langle Width, [7,90]\rangle, \langle Length, [7,90]\rangle\},$$

$$\mathcal{C}_{Building} = \{BuildingType = \text{Warehouse} \Rightarrow StoryNum = 1\}.$$

Hence, a building is described by four features/variables, each one with a set of possible values. Note that the single constraint associated with the node imposes that if the building is a warehouse, then it must have exactly one story. The node representing a

story of the building is defined as $\langle Story, \mathcal{V}_{Story}, \mathcal{C}_{Story}\rangle$, where:

$$\begin{aligned}\mathcal{V}_{Story} &= \{\langle FloorNum, [1,3]\rangle, \langle Height, [3,15]\rangle\},\\ \mathcal{C}_{Story} = \{&FloorNum = \langle FloorNum, Story, [upper\ story]\rangle + 1,\\ &FloorNum \leq \langle StoryNum, Building, [first\ story, \star]\rangle,\\ &\langle BuildingType, Building, [first\ story, \star]\rangle = \text{Office building} \Rightarrow\\ &\quad \Rightarrow Height \geq 4 \wedge Height \leq 5\}.\end{aligned}$$

In this case we have two variables associated with the node *Story*, whose values are controlled by three constraints. Note that these constraints involve features/variables associated with ancestors of the node *Story*. To refer to specific variables in the ancestors of a node, we introduce the notion of *meta-variable*, i.e., a triple of the form $\langle VarName, AncestorName, MetaPath\rangle$. This triple denotes a variable $VarName$ in an ancestor node $AncestorName$ (e.g., $BuildingType$ in the node $Building$). The third component of a meta-variable, $MetaPath$, is a list of edge labels (see below) and describes a path connecting the two nodes in the graph (wildcards '_' and '$\star$' can be used to represent arbitrary labels and a sequence of arbitrary labels, respectively). $MetaPath$s are used to define constraints that will have effect only on particular instances of a node. For example, the first constraint in $\mathcal{C}_{Story}$ will have to hold only for those instances of node $Story$ which are connected to another instance of node $Story$ through an edge labeled $upper\ story$. Intuitively, a node constraint for the node N will have to hold for each instance of N, such that it has ancestors connected with it through paths matching with the $MetaPath$s occurring in the constraint.

An edge is defined by: a name, two node names indicating the parent and the child nodes in the *has-part* relation, the cardinality of such relation (expressed as either an integer number or a variable), and a set of constraints. Such constraints may involve the cardinality variable (if any) as well as the variables of the parent node and of any of its ancestors (referred to by using meta-variables). An instance of an edge labeled $label$ connecting a node N with a node M, will be an edge labeled $label$, connecting an instance of N and an instance of M. Let us consider the edges *first story* and *upper story* of our sample model. The former is the edge that relates the building and its first story. It is defined as $\langle first\ story, Building, Story, 1, \emptyset\rangle$. Note that the cardinality is imposed to be 1 and there is no constraint. The edge *upper story* represents the *has-part* relation over two adjacent stories of the building. It is defined as $\langle upper\ story, Story, Story, Card, \mathcal{CC}\rangle$, where the variable $Card$ is defined as $\langle Card, [0,1]\rangle$, while the set of constraints is defined as follows:

$$\begin{aligned}\mathcal{CC} = \{&FloorNum = \langle StoryNum, Building, [first\ story, \star]\rangle \Rightarrow Card = 0,\\ &FloorNum < \langle StoryNum, Building, [first\ story, \star]\rangle \Rightarrow Card = 1\}.\end{aligned}$$

The two constraints in $\mathcal{CC}$ control the number of instances of the node $Story$. An instance of the node $Story$ will have as child another instance of node $Story$, if and only if its floor number is not equal to the number of stories of the building. Intuitively, a cardinality constraint for and edge e will have to hold for each instance of the parent node P in e, such that P has ancestors connected with it through paths matching with $MetaPath$s occurring in the constraint.

As mentioned, a product description consists of a product model together with a set of global constraints. Such constraints, called *model constraints*, involve variables

of nodes not necessary related by *has-part* relations (*node model constraints*) as well as cardinalities of different edges exiting from a node (*cardinality model constraints*). Also, global constraints like `alldifferent` [27] and aggregation constraints can be used to define node model constraints. Intuitively, a node model constraint will have to hold for all the tuples of node instances reached by paths matching with $MetaPath$s occurring in the constraint. The following is an example of cardinality model constraint:

$$\langle upper\ story, Story, Story, Card\rangle \neq \langle roof, Story, Roof, Card\rangle.$$

This constraint states that, given an instance of the node $Story$ the cardinality of the edge $upper\ story$ and $roof$ exiting from it must be different, i.e., an instance of the node $Story$ can not have both an upper story and a roof.

2.2 Process Description

PRODPROC allows one to model a process in terms of activities and temporal relations between them. Moreover, PRODPROC makes it possible to model process resource production and consumption, and to intermix the product and the process modeling phases.

In general, a process consists of: a set of activities; a set of variables (as before, endowed with a finite domain of strings or of integers) representing process characteristics and involved resources; a set of temporal constraints between activities; a set of resource constraints; a set of constraints on activity durations.

There are three kinds of activity: *atomic activities*, *composite activities*, and *multiple instance activities*. An *atomic* activity A is an event that happens in a time interval. It has associated a name and the following parameters:

- two integer decision variables, t^{start} and t^{end}, denoting the start time and end time of the activity. They define the time interval $[t^{start}, t^{end}]$, subject to the implicit requirement that $t^{end} \geq t^{start} \geq 0$.
- a decision variable $d = t^{end} - t^{start}$ denoting the duration of the activity.
- a flag $exec \in \{0, 1\}$.

When $d = 0$ we say that A is an *instantaneous activity*. If $exec = 1$ holds, A *is executed*, otherwise (namely, if $exec = 0$) A is *not executed*. A composite activity is an event described in terms of a process. Hence, it has associated four variables analogously to an atomic activity, as explained earlier. Moreover, it is associated with a model of the process it represents. A *multiple instance* (atomic or composite) activity is an event that may occur multiple times. Together with the four variables (and possibly the sub-process model), a multiple instance activity has associated a decision variables (named $inst$) representing the number of times the activity can be executed.

Temporal constraints between activities are inductively defined starting from *atomic temporal constraints*. Let A and B be to activities. We consider as atomic temporal constraints all the thirteen mutually exclusive binary relations which capture all the possible ways in which two intervals might overlap or not (as introduced by Allen in [3]), and some further constraints inspired by the constraint templates of the language ConDec [19]. The following are some examples of atomic temporal constraints (for lack of space we avoid listing all the possibilities):

1. $A\ before\ B$ to express that A is executed before B.

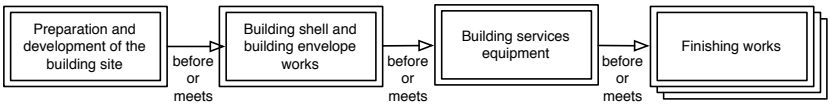


Fig. 2. Temporal constraint network for the building construction process.

2. $A\ meets\ B$ to express that the execution of A ends at time point in which the execution of B starts.
3. $A\ must{-}be{-}executed$ to express that A must be executed.
4. $A\ is{-}absent$ to express that A can never be executed.
5. $A\ not{-}co{-}existent{-}with\ B$ to express that either A or B can be executed (i.e., it is not possible to execute both A and B).
6. $A\ succeeded{-}by\ B$ to express that when A is executed than B has to be executed after A.

The constraints 1 and 2 are two of the binary relations of [3]. The constraints 3–6 have been inspired by the templates used in the language ConDec [19]. A *temporal constraint* is inductively defined as follows.

- An atomic temporal constraint is a constraint.
- If φ and ϑ are temporal constraint then $\varphi\ and\ \vartheta$ and $\varphi\ or\ \vartheta$ are temporal constraints.
- If φ is a temporal constraint and c is a constraint on process variables, then $c \rightarrow \varphi$ is an *if-conditional* temporal constraint, stating that φ has to hold whenever c holds. Also, $c \leftrightarrow \varphi$ is an *iff-conditional* temporal constraint, stating that φ has to hold if and only if c holds.

Plainly, the truth of the atomic temporal constraints is related with the execution of the activities they involve. For instance, whenever for two activities A and B it holds that $exec_A = 1 \wedge exec_B = 1$, then the atomic formulas of the forms 1 and 2 must hold. A *temporal constraint network* $\mathcal{CN}$ is a pair $\langle \mathcal{A}, \mathcal{C} \rangle$, where $\mathcal{A}$ is a set of activities and $\mathcal{C}$ is a set of temporal constraints on activities in $\mathcal{A}$. Fig. 2 shows the temporal constraint network for the building construction process. Fig. 3 shows the temporal constraint network for the sub-process represented by the composite activity called "Building services equipment". In the figures, atomic activities are depicted as rectangles, composite activities as nested rectangles, multiple instance activities as overlapped rectangles. Binary temporal constraints are represented as edges whose labels describe the temporal relations. If an activity is involved in a $must\ be\ executed$ or in a $is\ absent$ constraint, it is depicted as a dashed line rectangle or a dotted line rectangle, respectively. A conditional temporal constraints is depicted together with its activation condition.

PRODPROC allows one to specify constraints on resource amounts [15] and activity durations. A resource constraint is a quadruple $\langle A, R, q, TE \rangle$, where A is an activity; R is an variable endowed with a finite integer domain; q is an integer or a variable endowed with a finite integer domain, defining the quantity of resource R consumed (if $q < 0$) or produced (if $q > 0$) by executing A; TE is a time extent that defines the time interval where the availability of resource R is affected by the execution of activity A. The possibilities for TE are: $FromStartToEnd$, $AfterStart$, $AfterEnd$, $BeforeStart$, $BeforeEnd$, $Always$, with the obvious meaning. The following is an

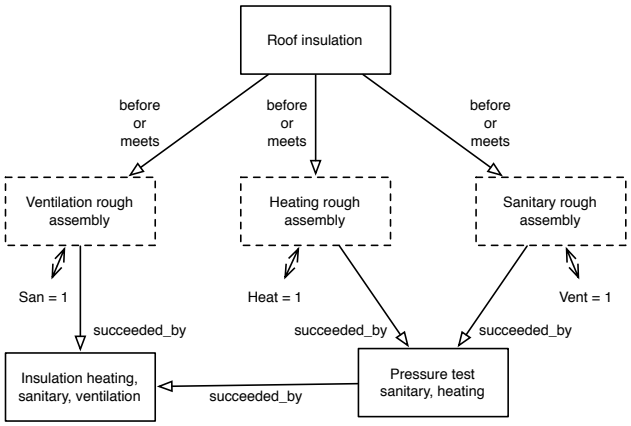


Fig. 3. Temporal constraint network for the composite activity "Building services equipment".

example of resource constraints for the third phase of the building construction process.

$$\langle \text{Roof insulation}, GeneralWorkers, \langle q_{GW}, [-10, -4] \rangle, FromStartToEnd \rangle.$$

This constraint specifies that the number of $GeneralWorkers$ available is reduced of an amount between 4 and 10 during the execution of the activity Roof insulation. All the workers will return available as soon as the activity ends. Note that resource constraints may (implicitly) imply constraints on the number of instances of multiple instance activities. Another form of resource constraints establishes initial level constraints, i.e., expressions defining the quantity of a resource available at the time origin of a process. The basic form is $initialLevel(R, iv)$, where R is a resource and $iv \in \mathbb{N}$.

An activity duration constraint has the form $\langle A, Constraint \rangle$, where A is the name of an activity, and $Constraint$ may involve the duration of A, process variables, and quantity variables for resource related to A. This is an example of activity duration constraint for the third phase of the building construction process (where $BuildingArea$ is a process variable, and q_T, q_C are quantity variables):

$$\Big\langle \text{Roof insulation}, d = \frac{BuildingArea}{2 \cdot |q_{GW}| + 2 \cdot |q_T| + 3 \cdot |q_C|} \Big\rangle.$$

PRODPROC also allows one to couple elements for modeling a process and elements for modeling a product through constraints involving process variables and product variables. The following are examples in our sample model:

$$\langle Building, sanitary, Card \rangle = San \ ,$$
$$\langle StoryNum, Building, [] \rangle = inst_{Finishing\ works}.$$

For instance, the last one states that the number of stories of a building has to be equal to the value of $inst_{Finishing\ works}$ (i.e., number of times the event $Finishing\ works$ is executed). In general, constraints involving both product and process variables may help to detect/avoid planning impossibilities due to product configuration, and configuration impossibilities due to product configuration, during the configuration of a product.

2.3 PRODPROC Instances

A PRODPROC model represents the collection of single (producible) variants of a configurable product and the processes to produce them. A PRODPROC *instance* represent one of such variant and its production process. To precisely define this notion we need to introduce first the notion of *candidate instance*. A PRODPROC candidate instance consists of the following components:

- A set $\mathcal{N}$ of *node instances*, i.e., tuples of the form $n = \langle N, i, \mathcal{V}_N \rangle$ where N is a node in the product model graph, $i \in \mathbb{N}$ is an index (different for each instance of a node), $\mathcal{V}_N$ is the set of variables of node N.
- a set $\mathscr{A}_{\mathsf{Nodes}}$ of *assignments* for all the node instance variables, i.e., expressions of the form $V = value$ where V is a variable of node instance n and $value$ belongs to the set of values for V.
- A tree, called *instance tree*, that specifies the pairs of node instances in the relation *has-part*. Such a tree is defined as $IT = \langle \mathcal{N}, \mathcal{E} \rangle$, where $\mathcal{E}$ is a set of tuples $f = \langle label, n, m \rangle$ such that there exists an edge $e = \langle label, N, M, Card, \mathcal{CC} \rangle$ in the product model graph, n is an instance of N and m is an instance of M.
- A set $\mathscr{A}_{\mathsf{Cards}}$ of *assignments* for all the instance cardinality variables, i.e., expressions of the form $IC_n^e = k$ where n is an instance of a node N, e is a quintuple $\langle label, N, M, Card, \mathcal{CC} \rangle$, $IC_n^e \equiv Card$, and k is the number of the edges $\langle label, n, m \rangle$, such that m is an instance of M, in the instance tree.
- A set $\mathcal{A}$ of *activity instances*, i.e., pairs $a = \langle A, i \rangle$ where A is the name of an activity such that $exec_A = 1$ and $i \in \mathbb{N}$ is a unique id for instances of A.
- A set $\mathscr{E}$ of flags $exec_A$, one for each activity A such that $exec_A \neq 1$.
- A set $\mathscr{A}_{\mathsf{Proc}}$ of *assignments* for all model variables and activity parameters (i.e., time instant variables, duration variables, execution flags, quantity resource variables, instance number variables), that is, expressions of the form $P = value$ where P is a model variable or an activity parameter, and $value \in \mathbb{Z}$ or $value$ belongs to the set of values for P.

A PRODPROC instance is a candidate instance such that the assignments in $\mathscr{A}_{\mathsf{Nodes}} \cup \mathscr{A}_{\mathsf{Cards}} \cup \mathscr{A}_{Proc}$ satisfy all the constraints in the PRODPROC model (node constraints, edges constraints, temporal constraints, resource constraints, etc.), appropriately instantiated with variables of node instances and activity instances in the candidate instance.

The (constraint) instantiation mechanism produces a set of constraints on candidate instance variables from each constraint in the PRODPROC model. A candidate instance must satisfy all these constraints to qualify as an instance. We give here an intuitive explanation of how the instantiation mechanism works on different constraint types. Let us begin with node and cardinality constraints. Let c be a constraint belonging to the node N, or a constraint for an edge e between nodes N and M. Let us suppose that $N_1, \ldots, N_k$ are ancestors of N whose variables are involved in c, and let $p_1, \ldots, p_k$ be $MetaPath$s such that, for $i = 1, \ldots, k$, p_i is a $MetaPath$ from N_i to N. We define L_{node} as the set of k-tuple of node instances $\langle n, n_1, \ldots, n_k \rangle$ where: n is an instance of N; for $i = 1, \ldots, k$ n_i is an instance of N_i, connected with n through a path q_i in the instance tree such that $\texttt{match}(q_i, p_i) = true$ holds. `match` is defined as follows.[1]

[1] Given two lists l_1 and l_2, $l_1 \circ l_2$ denotes their concatenation. We denote with $[x|l]$ the list obtained by prepending the element x to the list l.

$$\texttt{match}(q,p) = \begin{cases} true & \text{if } q = p \\ \texttt{match}(ps, mps) & \text{if } q = [label|ps] \wedge (p = [label|mps] \vee p = [\_|mps]) \\ true & \text{if } p = [\star, label|ps] \wedge \\ & \wedge \exists s.(q = s \circ [label|ps] \wedge \texttt{match}(ps, mps)) \\ false & \text{otherwise} \end{cases}$$

For each k-tuple $t \in L_{node}$, we obtain a constraint on instance variables appropriately substituting variables in c with variables of node instances in t. If c is a constraint for e, given a k-tuple $\langle n, n_1, \ldots, n_k \rangle$ on which to instantiate it, the cardinality occurring in it is substituted with the cardinality variable IC^e_n.

Node model constraints are instantiated in a slightly different way. Let c be a node model constraint. Let us suppose that $N_1, \ldots, N_k$ are the nodes whose variables are involved in c, let $p_1, \ldots, p_k$ be $MetaPath$s such that, for $i = 1, \ldots, k$, p_i is a $MetaPath$ that ends in N_i. We define L_{nmc} as the set of ordered k-tuples of node instances $\langle n_1, \ldots, n_k \rangle$, where for $i = 1, \ldots, k$ n_i is an instance of N_i connected by a path q_i with one of its ancestors in the instance tree, such that $\texttt{match}(q_i, p_i) = true$ holds. For each k-tuple $t \in L_{nmc}$, we obtain a constraint on instance variables appropriately substituting variables in c with variables of node instances in t. If c is an aggregation or an `alldifferent` constraint, then we define an equivalent constraint on the list consisting of all the node instances of $N_1, \ldots, N_k$ reached by a path matching with the corresponding $MetaPath$.

The instantiation of cardinality model constraint is very simple. Let c be a cardinality model constraint for the cardinalities of the edges with labels $e_1, \ldots, e_k$ exiting from a node N. Let $n_1, \ldots, n_h$ be instances of N. For all $i \in \{1, \ldots, h\}$, we instantiate c appropriately substituting the cardinality variables occurring in it, with the instance cardinality variables $IC^{e_1}_{n_1}, \ldots, IC^{e_k}_{n_k}$.

Let us now consider process constraints. Let A be an activity, let $a_1, \ldots, a_k$ be instances of A. Let r be the resource constraint $\langle A, R, q, TE \rangle$, we instantiate it on each instance of A, i.e., we obtain a constraint $\langle a_i, R, q_i, TE \rangle$ for each $i = 1, \ldots, k$, where $q_i = q$ is a fresh variable. Let c be an activity duration constraint for A, for each $i = 1, \ldots, k$ we obtain a constraint substituting in c d_A with d_{a_i}, and each quantity variable q with the corresponding variable q_i. Finally, let B an activity, let $b_1, \ldots, b_h$ be instances of B. If c is a temporal constraint involving A and B, we obtain a constraint on activity instances for each ordered couple $\langle i, j \rangle$, with $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, h\}$, substituting in c each occurrence of A with a_i, and of B with b_j. This mechanism can be easily extended to temporal constraints involving more than two activities.

3 Product and Process Configuration

On top of the framework we described in Sect. 2 it is possible to implement a configuration system based on Constraint Logic Programming (CLP) [13]. In this section, we first explain how such a system can support a user through the configuration of a product and its production process. Then, we show how we can generate a CLP program from a model and a (partial) candidate instance.

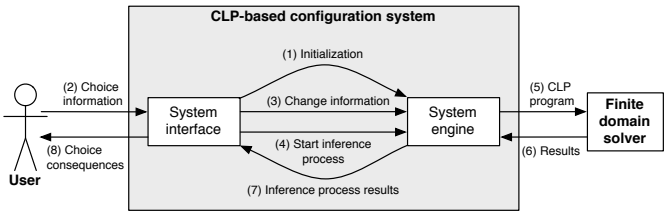


Fig. 4. Configuration process supported by a CLP-based system.

A possible general structure of a configuration process supported by a CLP-based system is pictorially described in Fig. 4. First, the user initializes the system (1) selecting the model of the product/process to be configured. After such an initialization phase the user starts to make her/his choices by using the system interface (2). The interface communicates to the system engine (i.e., the piece of software that maintains a representation of the product/process under configuration, and checks the validity and consistency of user's choices) each data variation specified by the user (3). The system engine updates the current partial configuration accordingly. Whenever an update of the partial configuration takes place, the user, through the system interface, can activate the engine inference process (4). The engine instantiates PRODPROC constraints on the current (partial) candidate instance, and encodes the product/process configuration problem in a CLP program (encoding a Constraint Satisfaction Problem, abbreviated to CSP). Then, it uses a finite domain solver to propagate the logical effects of user's choices (5). Once the inference process ends (6), the engine returns to the interface the results of its computation (7). In its turns, the system interface communicates to the user the consequences of her/his choices on the (partial) configuration (8).

In the following, we briefly explain how it is possible to obtain a CLP program from a PRODPROC model and a (partial) candidate instance (a candidate instance is partial when there are variables with no value assigned to) corresponding to it. We do this considering only the process side of a model, the operations necessary to obtain CLP variables and constraints for the product side are similar.

Given a PRODPROC model and a corresponding (partial) candidate instance defined by a user, we can easily obtain a CSP $\langle \mathcal{VAR}, \mathcal{DOM}, \mathcal{CONSTR} \rangle$, where $\mathcal{VAR}$ is a set of variables, $\mathcal{DOM}$ is a set of finite domain for variables in $\mathcal{VAR}$, and $\mathcal{CONSTR}$ is a set of constraints on variables in $\mathcal{VAR}$. $\mathcal{VAR}$ will contain a variable for each node instance variable, cardinality variable, activity parameter, process characteristic, resource, and quantity resource variable. $\mathcal{DOM}$ will contain a domain, obtained form the PRODPROC model, for each variable in $\mathcal{V}$. $\mathcal{CONSTR}$ will contain all the constraints that the (partial) candidate instance should satisfy. As we explained in Sect. 2.3, such constraints are determined by an instantiation mechanism. We give here a formalization of such mechanism for the process side of a model. We define a function μ that, given the set of activity instances $\mathcal{A}$, the set $\mathcal{RDC} = \mathcal{R} \cup \mathcal{D} \cup \mathcal{C}$, where $\mathcal{R}$ is the set of resource constraints, $\mathcal{D}$ is the set of activity duration constraints, $\mathcal{C}$ is the set of temporal constraints, generates a set of constraints instantiated on activity instances. To define μ we preliminary need to introduce some basic notions. If c is a temporal constraint $\texttt{acts}(c)$

is the list of activities involved in c. In the following we will denote with a an instance of an activity, and with $\texttt{pInsts}(a)$ the set of instances of the process associated to a composite activity instance a. We say that $a \leftrightarrow_{Act} A$ if and only if a is an instance of A. The function μ is defined as follows:

$$\mu(\mathcal{A}, \mathcal{RCP}, \mathcal{I}) = \bigcup_{a \in \mathcal{A}} \alpha(a) \cup \bigcup_{c \in \mathcal{RCP}} \gamma(c, \mathcal{A}).$$

The function α generates the set of default constraints on duration, start time, and finishing time for an activity instance a:

$$\alpha(a) = \begin{cases} t^{\mathsf{Comp}}(a) \text{ if } a \text{ is a composite activity instance} \\ t(a) \qquad \text{otherwise} \end{cases},$$

$$t^{\mathsf{Comp}}(a) = \{t_a^{start} = \min_{b \in \texttt{pInsts}(a)} t_b^{start}, t_a^{end} = \max_{b \in \texttt{pInsts}(a)} t_b^{end}, \\ t_a^{end} \geq t_a^{start}, d_a = t_a^{end} - t_a^{start}, exec_A = 1\},$$

$$t(a) = \{t_a^{start} \geq 0, t_a^{end} \geq t_a^{start}, d_a = t_a^{end} - t_a^{start}, exec_A = 1\}.$$

The function γ instantiate a constraint c on activity instances in $\mathcal{A}$.

$$\gamma(c, \mathcal{A}) = \begin{cases} \{\langle a, R, q_a, TE\rangle | a \in \mathcal{A} \wedge & \text{if } c \in \mathcal{R} \wedge \\ \quad \wedge\, a \leftrightarrow_{Act} A \wedge q_a = q_A\} & \quad \wedge\, c \equiv \langle A, R, q_A, TE\rangle \\ c & \text{if } c \in \mathcal{R} \wedge \\ & \quad \wedge\, c \equiv initialLevel(R, iv) \\ \{c[d_A/d_a, q_A/q_a] \mid a \in \mathcal{A} \wedge a \leftrightarrow_{Act} A\} & \text{if } c \in \mathcal{D} \\ \{c[A_1/a_1, \ldots, A_k/a_k] \mid & \text{if } c \in \mathcal{C} \\ \quad [A_1, \ldots, A_k] = \texttt{acts}(c) \wedge & \\ \quad \wedge\, [a_1, \ldots, a_k] \in L_{\mathsf{act}}(c, [A_1, \ldots, A_k], \mathcal{A})\} & \end{cases}$$

The function $L_{\mathsf{act}}(c, [A_1, \ldots, A_k], \mathcal{A})$ generates all the k-tuple of activity instances that are instances of activities involved in a constraint c:

$$L_{\mathsf{act}}(c, [A_1, \ldots, A_k], \mathcal{A}) = \{[a_1, \ldots, a_k] \mid \textstyle\bigwedge_{j=1}^{k} (a_j \in \mathcal{A} \wedge a_j \leftrightarrow_{Act} A_j)\}$$

From instantiated resource constraints and CSP variables for resources it is possible to generate a cumulative constraint [1,4]. To obtain CSP constraints from all other constraints it is sufficient to substitute the instance variables with the corresponding CSP variables. Temporal constraints are defined on activities, but it is possible to compile them into propositional formulas on activity durations, starting times, and finishing times. Table 1 shows the translation for some of the atomic temporal constraints.

Let φ and ϑ be temporal constraints, let $\varphi^{\mathcal{P}}$ and $\vartheta^{\mathcal{P}}$ the corresponding propositional formulas. Then $\varphi \; and \; \vartheta$, $\varphi \; or \; \vartheta$, $c \rightarrow \varphi$ and $c \leftrightarrow \varphi$ correspond to $\varphi^{\mathcal{P}} \wedge \vartheta^{\mathcal{P}}$, $\varphi^{\mathcal{P}} \vee \vartheta^{\mathcal{P}}$, $c \Rightarrow \varphi^{\mathcal{P}}$ and $c \Leftrightarrow \varphi^{\mathcal{P}}$, respectively.

Given the constraint satisfaction problem $\mathcal{CSP}$ it is straightforward to obtain a CLP program encoding it, once a specific CLP system has been chosen, e.g., SICStus Prolog, SWI Prolog, or ECLiPse.

4 A Comparison with Existing Product/Process Modeling Tools

In this section, we briefly compare the PRODPROC framework with some of the most important product configuration systems and process modeling tools to put in evidence its strength and limitations.

Atomic temporal constraint	Propositional formula
$A\,before\,B$	$t_A^{start} < t_B^{start} \wedge t_A^{end} < t_B^{start}$
$A\,meets\,B$	$t_A^{start} < t_B^{start} \wedge t_A^{end} = t_B^{start}$
$A\,must{-}be{-}executed$	$exec_A = 1$
$A\,is-absent$	$exec_A = 0$
$A\,not{-}co{-}existent{-}with\,B$	$exec_A + exec_B \leq 1$
$A\,succeeded{-}by\,B$	$exec_A = 1 \Rightarrow exec_B = 1 \wedge t_B^{start} \geq t_A^{end}$

Table 1. Atomic temporal constraints and corresponding propositional formulas.

Product configuration systems based on Answer Set Programming (ASP) [11], e.g., Kumbang Configurator [16], provide a number of features that are specifically tailored to the modeling of software product families. On the one hand, this makes these systems appealing for a relevant range of application domains. On the other hand, it results in a lack of generality, which is probably the major drawback of this class of systems. In particular, they do not support global constraints, and they encounter some problems in the management of arithmetic constraints related to the so called grounding stage [16].

Systems based on binary decision diagrams (BDDs) for product configuration, e.g., Configit Product Modeler [8], trade the complexity of the construction of the BDD, that basically provides an encoding of all possible configurations [12], for the simplicity and efficiency of the configuration process. Despite their various appealing features, BDD-based systems suffer from some significant limitations. First, even though some work has been done on the introduction of modules [25,26], they basically support flat models only. Moreover, they find it difficult to cope with global constraints. Some attempts at combining BDD with CSP to tackle `alldifferent` constraints have been recently done [17]; however, they are confined to the case of flat models. We are not aware of any BDD system that deals with global constraints in a general and satisfactory way.

Unlike ASP-based and BDD-based product configuration systems, CSP-based systems allow the user to define non-flat models and to deal with global constraints. Unfortunately, the modeling expressiveness of CSP-based systems has a cost, i.e., backtrack-free configuration algorithms for CSP-based systems are often inefficient, while non backtrack-free ones need to explicitly deal with dead ends. Some well-known CSP-based configuration systems, such as ILOG Configurator [14] and Lava [10], seem to be no longer supported. A recent CSP-based configuration system is Morphos Configuration Engine (MCE) [7]. From the point of view of process modeling, PRODPROC can be viewed as an extension of the MCE modeling language. In particular, it extends MCE modeling language with the following features: (1) *cardinality variables*, i.e., *has-part/is-part-of* relations can have non-fixed cardinalities; (2) *product model graph*, i.e., nodes and relations can define a graph, not only a tree; (3) *cardinality constraints* and *cardinality model constraints*, i.e., constraints can involve cardinalities of relations; (4) $MetaPath$s, i.e., a mechanism to refer node instance variables in constraints.

In [22] the authors present an ontology representing a synthesis of resource-based, connection-based, function-based and structure-based product configuration approches. The PRODPROC framework covers only a subset of these concepts. However, it is not limited to product modeling and it defines a rich (numeric) constraint language, while

it remains unclear to what extent the language used in [22] supports the formulation of configuration-domain specific constraints.

PRODPROC can be viewed as the source code representation of a configuration system with respect to the MDA abstraction levels presented in [9]. PRODPROC product modeling elements can be mapped to UML/OCL in order to obtain platform specific (PSM) and platform independent (PIM) models. The mapping to OCL of $MetaPath$s containing '$\star$' wildcards and of model constraints requires some attention. For example, the latter do not have an explicit context as OCL constraint must have.

In the past years, different formalisms have been proposed for process modeling. Among them we have: the Business Process Modeling Notation (BPMN) [28], Yet Another Workflow Language (YAWL) [24], DECLARE [19]. Languages like BPMN and YAWL model a process as a detailed specification of step-by-step procedures that should be followed during the execution. They adopt an *imperative* approach in process modeling, i.e., all possibilities have to be entered into their models by specifying their control-flows. BPMN has been developed under the coordination of the Object Management Group. PRODPROC has in common with BPMN the notion of atomic activity, sub-process, and multiple instance activity. The effect of BPMN joins and splits on the process flow can be obtained by using temporal constraints. In PRODPROC there are no notions such as BPMN events, exception flows, and message flows. However, events can be modeled as instantaneous activities and data flowing between activities can be modeled with model variables. YAWL is a process modeling language whose intent is to directly supported all control flow patterns. PRODPROC has in common with YAWL the notion of task, multiple instance task, and composite task. YAWL join and split constructs are not present in PRODPROC, but using temporal constraints it is possible to obtain the same expressivity. As opposed to traditional imperative approaches to process modeling, DECLARE uses a constraint-based declarative approach. DECLARE models rely on constraints to implicitly determine the possible ordering of activities (any order that does not violate constraints is allowed). With respect to DECLARE, PRODPROC has in common the notion of activity and the use of temporal constraints to define the control flow of a process. The set of atomic temporal constraints is not as big as the set of template constraints available in DECLARE, however it is possible to easily combine the available ones so as to define all complex constraints of practical interest. Moreover, in PRODPROC it is possible to define multiple instance and composite activities, features that are not available in DECLARE.

From the point of view of process modeling, PRODPROC combines modeling features of languages like BPMN and YAWL, with a declarative approach for control flow definition. Moreover, it presents features that, to the best of our knowledge, are not presents in other existing process modeling languages. These are: resource variables and resource constraints, activity duration constraints, and product related constraints. Thanks to these features, PRODPROC is suitable for modeling production processes and, in particular, to model mixed scheduling and planning problems related to production processes. Furthermore, a PRODPROC model does not only represent a process ready to be executed as a YAWL (or DECLARE) model does, it also allows one to describe a configurable process. Existing works on process configuration, e.g., [20], define process models with variation points, and aim at deriving different process model variants from

a given model. Instead, we are interested in obtaining process instances, i.e., solutions to the scheduling/planning problem described by a PRODPROC model.

The PRODPROC framework allows one to model products, their production processes, and to couple products with processes using constraints. The only works on the coupling of product and process modeling and configuration we are aware of are the ones by Aldanondo et al. [2]. They propose to consider simultaneously product configuration and process planning problems as two constraint satisfaction problems; in order to propagate decision consequences between the two problems, they suggest to link the two constraint based models using coupling constraints. The development of PRODPROC has been inspired by the papers of Aldanondo et al., in fact we have separated models for products and processes and, constraints for coupling them too. However, our modeling language is far more complex and expressive than the one presented in [2].

5 Conclusions

In this paper we focused on the problem of product and process modeling and configuration. In particular, we pointed out the lack of a tool covering both physical and production aspects of configurable products. To overcome this absence, we proposed a framework called PRODPROC, that allows one to model a configurable products and its production process. Moreover, we showed how it is possible to build a CLP-based configuration systems on top of this framework, and compared it to existing product configuration systems and process modeling tools.

We have already implemented a first prototype of a CLP-based configuration system that uses PRODPROC. It covers only product modeling and configuration, but we are working to add to it process modeling and configuration capabilities. PROPROC and SysML [18] have various commonalities in terms of modeling features, despite the fact that their purposes are different. We plan to further investigate the relations that exists between the two modeling languages. We also plan to experiment our configuration system on different real-world application domains, and to compare it with commercial products, e.g., [5].

References

1. A. Aggoun and N. Beldiceanu. Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17 (7):57–73, 1993.
2. M. Aldanondo and E. Vareilles. Configuration for mass customization: how to extend product configuration towards requirements and process configuration. *J. of Intelligent Manufacturing*, 19 (5):521–535, 2008.
3. J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26:832–843, 1983.
4. N. Beldiceanu and M. Carlsson. A New Multi-resource cumulatives Constraint with Negative Heights. In P. Van Hentenryck, editor, *CP 2002*, volume 2470 of *LNCS*, pages 63–79. Springer Berlin / Heidelberg, 2006.
5. U. Blumöhr, M. Münch, and M. Ukalovic. *Variant Configuration with SAP*. SAP Press, 2009.

6. D. Campagna. A Graphical Framework for Supporting Mass Customization. In *Proc. of the IJCAI'11 Workshop on Configuration*, pages 1–8, 2011.
7. D. Campagna, C. D. Rosa, A. Dovier, A. Montanari, and C. Piazza. Morphos Configuration Engine: the Core of a Commercial Configuration System in CLP(FD). *Fundam. Inform.*, 105 (1-2):105–133, 2010.
8. Configit A/S. Configit Product Modeler. `http://www.configit.com`.
9. A. Felfernig. Standardized Configuration Knowledge Representations as Technological Foundation for Mass Customization. *IEEE Trans. on Engineering Management*, 54 (1):41–56, 2007.
10. G. Fleischanderl, G. Friedrich, A. Haselböck, H. Schreiner, and M. Stumptner. Configuring Large Systems Using Generative Constraint Satisfaction. *IEEE Intelligent Systems*, 13 (4):59–68, 1998.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.
12. T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Moller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *Proc. of the International Conference on Economic, Technical and Organizational Aspects of Product Configuration Systems*, pages 131–138. 2004.
13. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *J. Log. Program.*, 19/20:503–581, 1994.
14. U. Junker. The Logic of ILOG (J)Configurator: Combining Constraint Programming with a Description Logic. In *Proc. of the IJCAI'03 Workshop on Configuration*, pages 13–20. 2003.
15. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: existing approaches and new results. *Artif. Intell.*, 143:151–188, February 2003.
16. V. Myllärniemi, T. Asikainen, T. Männistö, and T. Soininen. Kumbang configurator - a configurator tool for software product families. In *Proc. of the IJCAI'05 Workshop on Configuration*, pages 51–56. 2005.
17. A. H. Nørgaard, M. R. Boysen, R. M. Jensen, and P. Tiedemann. Combining Binary Decision Diagrams and Backtracking Search for Scalable Backtrack-Free Interactive Product Configuration. In *Proc. of the IJCAI'09 Workshop on Configuration*, 2009.
18. OMG. OMG Systems Modeling Language. `http://www.omgsysml.org`.
19. M. Pesic, H. Schonenberg, and W. van der Aalst. DECLARE: Full support for loosely-structured processes. In *EDOC'07*, pages 287–287, 2007.
20. M. L. Rosa. *Managing Variability in Process-Aware Information Systems*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2009.
21. D. Sabin and R. Weigel. Product configuration frameworks-a survey. *IEEE Intelligent Systems*, 13:42–49, 1998.
22. T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen. Towards a general ontology of configuration. *Artif. Intell. Eng. Des. Anal. Manuf.*, 12:357–372, September 1998.
23. H. Sommer. *Project Management for Building Construction*. Springer, 2010.
24. A. H. M. ter Hofstede, W. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010.
25. E. R. van der Meer and H. R. Andersen. BDD-based Recursive and Conditional Modular Interactive Product Configuration. In *Proc. of Workshop on CSP Techniques with Immediate Application (CP'04)*, pages 112–126, 2004.
26. E. R. van der Meer, A. Wasowski, and H. R. Andersen. Efficient interactive configuration of unbounded modular systems. In *Proc. of the 2006 ACM symposium on Applied computing*, SAC '06, pages 409–414. ACM, 2006.
27. W. J. van Hoeve. The alldifferent Constraint: A Survey, 2001.
28. S. A. White and D. Miers. *BPMN modeling and reference guide: understanding and using BPMN*. Lighthouse Point, 2008.

A Model Instantiation and CSP creation

In this section, exploiting the building model introduced in Sect. 2, we show an example of PRODPROC partial candidate instance, and describe the CSP we obtain from it. The purpose of the example is twofold: first, to show how multiple instances of a node affect the constraint instantiation and the CSP corresponding to a model instance; second, to better describe the encoding of the process description into a CSP, in particular the generation of a cumulative constraint from resources and instantiated resource constraints.

Fig. 5 shows the instance tree of the partial candidate instance we consider. It consists of one instance of the root node (i.e., the node *Building*) of the product model graph depicted in Fig. 1, one instance of the node *Electr./Light. service*, two instances of the node *Story*, and one instance of the node *Roof*.

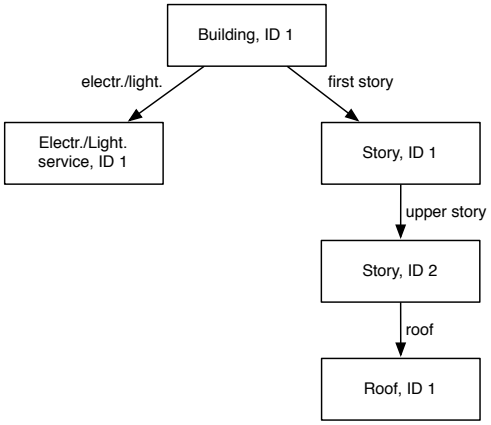


Fig. 5. Instance tree of a building partial candidate instance.

The activity instances and the instantiated temporal constraints of the construction process for the building instance showed in Fig. 5 are depicted in Fig. 6.

In the following we will denote as $\langle Var, Node\text{-}i \rangle$ the variable Var of the instance with id i of the node $Node$. Since we are considering a partial candidate instance, some of the node instance variables and process variables may have a value assigned to. For example, we may have $\langle StoryNum, Building\text{-}1 \rangle = 2$ and $San = 0$.

As mentioned in Sect. 2.3, a candidate instance is an instance if it satisfies all the constraints defined in the model, appropriately instantiated on instance variables. The instantiation of the node constraints for the nodes *Building* and *Story* listed in Sect. 2.1 leads to the following constraints on the variables of node instances in Fig.5.

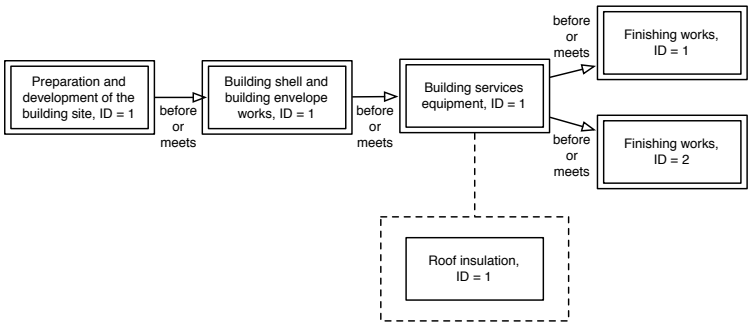


Fig. 6. Activities and temporal constraints for the building partial candidate instance.

$\langle BuildingType, Building\text{-}1\rangle = \text{Warehouse} \Rightarrow \langle StoryNum, Building\text{-}1\rangle = 1,$
$\langle FloorNum, Story\text{-}1\rangle \leq \langle StoryNum, Building\text{-}1\rangle,$
$\langle BuildingType, Building\text{-}1\rangle = \text{Office building} \Rightarrow$
$\Rightarrow \langle Height, Story\text{-}1\rangle \geq 4 \wedge \langle Height, Story\text{-}1\rangle \leq 5,$
$\langle FloorNum, Story\text{-}2\rangle = \langle FloorNum, Story\text{-}1\rangle + 1,$
$\langle FloorNum, Story\text{-}2\rangle \leq \langle StoryNum, Building\text{-}1\rangle,$
$\langle BuildingType, Building\text{-}1\rangle = \text{Office building} \Rightarrow$
$\Rightarrow \langle Height, Story\text{-}2\rangle \geq 4 \wedge \langle Height, Story\text{-}2\rangle \leq 5.$

Instantiating the cardinality constraints for the edge *upper story*, introduced in Sect 2.1, we obtain:

$\langle FloorNum, Story\text{-}1\rangle = \langle StoryNum, Building\text{-}1\rangle \Rightarrow IC^{upper\ story}_{Story\text{-}1} = 0,$
$\langle FloorNum, Story\text{-}1\rangle < \langle StoryNum, Building\text{-}1\rangle \Rightarrow IC^{upper\ story}_{Story\text{-}1} = 1,$
$\langle FloorNum, Story\text{-}2\rangle = \langle StoryNum, Building\text{-}1\rangle \Rightarrow IC^{upper\ story}_{Story\text{-}2} = 0,$
$\langle FloorNum, Story\text{-}2\rangle < \langle StoryNum, Building\text{-}1\rangle \Rightarrow IC^{upper\ story}_{Story\text{-}2} = 1.$

Finally, the instantiation of the cardinality model constraint showed in Sect. 2.1 leads to the constraint:

$$IC^{upper\ story}_{Story\text{-}2} \neq IC^{roof}_{Story\text{-}2}.$$

For each activity instance we have constraints on duration, starting and finishing time. For example, for the composite activity instance "Finishing works" with id 1 we have:

$t^{start}_{\text{Finishing works-1}} = \min_{b \in \texttt{pInsts}(\text{Finishing works-1})} t^{start}_b,$
$t^{end}_{\text{Finishing works-1}} = \max_{b \in \texttt{pInsts}(\text{Finishing works-1})} t^{end}_b,$
$t^{end}_{\text{Finishing works-1}} \geq t^{start}_{\text{Finishing works-1}},$
$d_{\text{Finishing works-1}} = t^{end}_{\text{Finishing works-1}} - t^{start}_{\text{Finishing works-1}}.$

While for the activity instance "Roof insulation" with id 1 we have:

$t^{start}_{\text{Roof insulation-1}} \geq 0, t^{end}_{\text{Roof insulation-1}} \geq 0,$
$t^{end}_{\text{Roof insulation-1}} \geq t^{start}_{\text{Roof insulation-1}},$
$d_{\text{Roof insulation-1}} = t^{end}_{\text{Roof insulation-1}} - t^{start}_{\text{Roof insulation-1}}.$

Instantiating the resource and duration constraints for the activity *Roof insulation* introduced in Sect. 2.2 we obtain:

$\langle$Roof insulation-1, $GeneralWorkers$, $\langle q_{GW}, [-10, -4]\rangle$, $FromStartToEnd\rangle$,

$\big\langle$Roof insulation-1, $d = \dfrac{BuildingArea}{2 \cdot |q_{GW}| + 2 \cdot |q_T| + 3 \cdot |q_C|}\big\rangle$.

The instantiation of the constraint involving both product and process variables showed in Sect. 2.2 leads to the following constraints:

$$IC^{sanitary}_{Building\text{-}1} = San\ ,$$
$$\langle StoryNum, Building\text{-}1\rangle = inst_{Finishing\ works}.$$

From the PRODPROC partial candidate instance we just described and its instantiated constraints, we can construct a CSP with the following characteristics (we use the SWI-Prolog notation for variables, domains and constraints).

- A finite domain (FD) variable for each node instance variable, e.g., for the variable $\langle StoryNum, Building\text{-}1\rangle$ the FD variable `StoryNum_Building_1`;
- A FD variable for each instance cardinality variable, e.g, for $IC^{roof}_{Story\text{-}2}$ the FD variable `IC_roof_Story_2`;
- FD variables for starting time, ending time, duration of each activity instance, e.g., `T_start_Roof_insulation_1`, `T_end_Roof_insulation_1`, and `D_Roof_insulation_1` for the activity instance "Roof insulation" with id 1;
- FD variables for execution flags of activities with no instance;
- FD variables for process and resource variables, e.g., `BuildingArea` for the process variable $BuildingArea$, `GeneralWorkers` for the resource variable $GeneralWorkers$;
- A domain constraint for each FD variable, e.g., `IC_roof_Story_2 in 0..1`;
- A constraint on an FD variable for each assignments, obtained by substituting each instance variable with the corresponding FD variable;
- A constraint on FD variables for each instantiated constraint, obtained by substituting each instance variable with the corresponding FD variable;
- For each composite activity instance, a `minimum` and a `maximum` constraint on start and end times, e.g., for the instance with id 1 of the activity "Finishing works" the constraint `minimum(T_start_Finishing_works_1,Ts)` and the constraint `maximum(T_end_Finishing_works_1,Te)`, where `Ts`, `Te` are respectively the list of start and end times of the activity in the process related to the instance with id 1 of "Finishing works";
- A constraint on FD variables for each instantiated temporal constraint, obtained by substituting start times, end times, and execution flags with the corresponding FD variables in the propositional formula equivalent to the temporal constraint;
- A constraint on FD variables for each instantiated duration constraint, obtained by substituting duration, process and resource variables with the corresponding FD variables;
- A constraint of the form `cumulatives(Tasks,Machines)` where `Tasks` is a list of `task` predicates, one for each instantiated resource constraint, and `Machines` is a list of `machine` predicates, one for each resource. For example, for the resource constraint showed in Sect. 2.2 and the resource $GeneralWorkers$ we define the predicates

```
task(T_start_Roof_insulation_1,D_Roof_insulation_1,
     T_end_Roof_insulation_1,Q_GW,GeneralWorkers,
     FromStartToEnd)

machine(GeneralWorkers,0..10,10)
```

B CLP-based Configuration System

We are using SWI-Prolog to develop a CLP-based configuration system that exploits the close relation that exists between configuration problems and CSPs.[2] In particular, we are using the SWI-Prolog `pce` library to implement the system graphical user interface, and the `clpfd` library for constraint propagation and labeling purposes. The current version of the system is limited to product modeling. Fig. 7 shows the graphical user interface that allows a user to define a product description using PRODPROC. The interface presents (on the left, from top to bottom) controls for graphical element selection, creation of nodes, creation of edges, and creation of sets of model constraints. Moreover, there is a menu named "Check" with controls for checking model syntactic correctness, and for automatically generate product instances to check model validity.

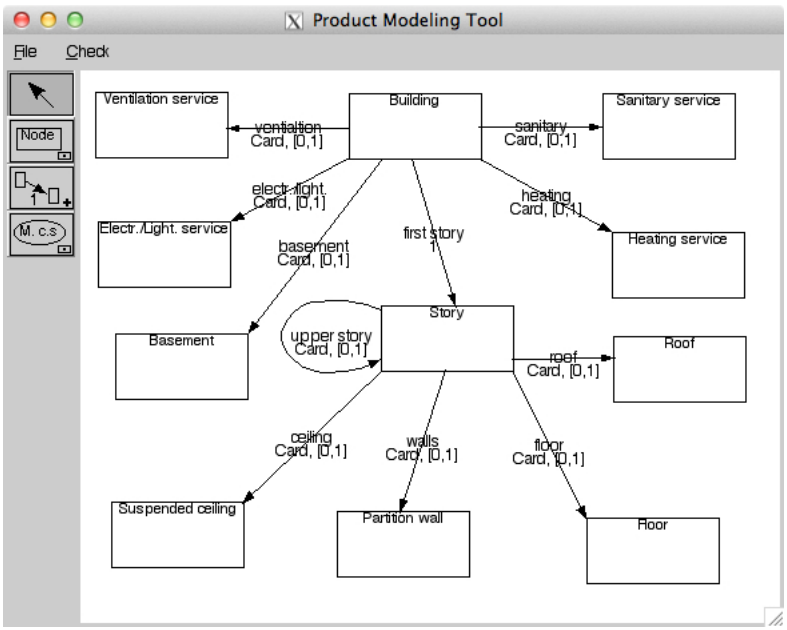


Fig. 7. Graphical user interface for product description creation and checking.

[2] We chose CLP instead of Constraint Programming for the advantages the former gives in terms of rapid software prototyping.

PrettyCLP: a Light Java Implementation for Teaching CLP

Alessio Stalla[1], Davide Zanucco[2], Agostino Dovier[2], and Viviana Mascardi[1]

[1] DISI - Univ. of Genova,
alessiostalla@gmail.com,mascardi@disi.unige.it
[2] DIMI - Univ. of Udine,
zanucco.davide@spes.uniud.it,agostino.dovier@uniud.it

Abstract. Recursion is nowadays taught to students since their first programming days in order to embed it deeply in their brains. However, students' first impact on Prolog programs execution sometimes weakens their faith in recursive programming thus invalidating our initial efforts. The selection and computation rules implemented by all Prolog systems, although clearly explained in textbooks, are hard to be interiorized by students also due to the poor system debugging primitives. Problems increase in Constraint Logic Programming when unification is replaced by constraint simplification in a suitable constraint domain. In this paper, we extend PrettyProlog, a light-weight Prolog interpreter written in Java capable of system primitives for SLD tree visualization, to deal with Constraint Logic Programming over Finite Domains. The user, in particular, can select the propagation strategies (e.g. arc consistency vs bound consistency) and can view the (usually hidden) details of the constraint propagation stage.

1 Introduction

PrettyProlog was developed two years ago by a team of the University of Genova, for providing concrete answers to demands raised by Prolog novices [14]. Teaching experience demonstrated that one of the hardest concepts for Prolog students is to understand the construction and the visit strategy of the SLD tree. PrettyProlog was developed from scratch, without reusing any existing Prolog implementation, and designed to be simple, modular, and easily expandable. Research on visualization of the execution of Prolog programs has a long history (just to make some examples, [13, 7, 15], many papers collected in [6], and [9]). Nevertheless, nowadays few Prolog implementations offer a Stack Viewer and an SLD tree visualizer as graphical means for debugging. The open-source implementations that provide these facilities are even fewer. Among them, SWI-Prolog[1] offers a debugging window showing current bindings, a diagrammatic trace of the call history, and a highlighted source code listing. No SLD tree visualization is given. On the other hand, many Java implementations of a Prolog

[1] http://www.swi-prolog.org/

interpreter exist, starting from W-Prolog[2]. Although at a prototypical stage, PrettyProlog presents three features that, to the best of our knowledge, cannot be found together in any other Prolog implementation:

1. it provides Stack and SLD Tree visualizers;
2. it is open source;
3. it is written in Java, and fully compliant with Java ME CDC application framework.

The desirable architectural features of PrettyProlog have been exploited in the research activity described in this paper where PrettyProlog has been extended for dealing with Constraint Logic Programming on Finite Domains (briefly, CLP(FD)). As a matter of fact, experience in teaching CLP(FD) evidenced further problems for students, first of all the replacement of unification with constraint solving. The term $1 + 3$ does not unify with the term $3 + 1$. However, they are both considered as 4 by CLP(FD). Moreover, the constraint propagation stage is parametric on some choices. For instance, bounds consistency and arc consistency return different "results" to the constraint $2X = Y$ where the domain D_X and D_Y of the variables X and Y are both the intervals 0..3 ($D_X = \{0, 1\}, D_Y = \{0, 1, 2\}$ in the former case, $D_X = \{0, 1\}, D_Y = \{0, 2\}$ in the latter case). Although different propagation techniques are studied in theory, Prolog interpreters supporting CLP(FD) usually implement only one of them, and students using different systems can be confused. Another source of confusion is introduced by some implementations of the SLD resolution with constraints that manage the ordering of literals in goals in a different way depending on whether they are constraint literals or user-defined literals. Moreover, during constraint's solution search (if explicitly required by a `labeling`) an auxiliary tree named prop-labeling tree is created and visited. This tree is sometimes wrongly confused with the SLD tree.

The proposed extension of PrettyProlog, called PrettyCLP, has been developed to help the new CLP programmers in a deeper understanding of what happens during the execution of a CLP(FD) program. The basic procedures for constraint propagation have been implemented in Java, either in the case of (hyper) arc consistency or in the case of (hyper) bounds consistency. A `labeling` built-in has also been developed for the solution's search using a prop-labeling tree.

This paper is organized in the following way: Section 2 recalls the functionalities of PrettyProlog and its implementation; Section 3 provides some background on CLP; Section 4 describes the original contribution of this paper, namely the design and implementation of PrettyCLP. In particular, it discusses PrettyCLP syntax, the supported mechanism for constraint propagation and labeling, and the output renderer. Section 5 analyzes the related work and concludes by outlining some future extensions.

[2] `http://waitaki.otago.ac.nz/~michael/wp/`

2 PrettyProlog

2.1 Functionalities

PrettyProlog implements a Prolog engine able to deal with basic data types (integer and real numbers, lists, strings), and offering metaprogramming facilities that, combined with the "cut" predicate, make the definition of negation as failure possible. Despite to some simplifications that were made during its design and implementation, sophisticated programs may be implemented with PrettyProlog thanks to these features.

The main functionality of PrettyProlog, however, is that it allows the user to visualize how the Stack and the SLD Tree evolve during a computation made by the interpreter to solve a given goal.

The SLD tree viewer panel shows the steps the PrettyProlog engine has performed as a tree. Each branch represents the selection of a clause from the theory, which can be selected in the "theory panel", whereas leaves are either solutions or dead ends, i.e. goals that could not be solved. The substitution that was valid at a given point is shown aside the corresponding node in the tree. Also, the SLD tree shows which frames are removed from the stack as the effect of a cut, by printing them with a different font and icon.

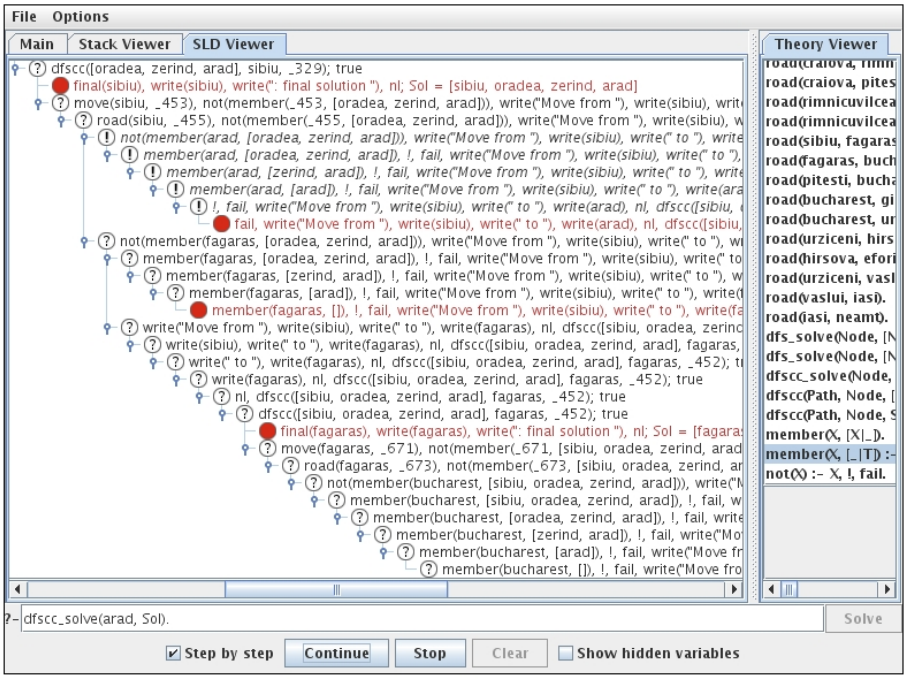


Fig. 1. SLD Viewer.

Figure 1 shows the SLD tree of a Prolog program that implements a classical instance of a search problem: that of moving from a city in Romania (Arad, in our case) to Bucharest [11, Chapter 3]. We implemented a depth first search with control of cycles, as well as the auxiliary `not` and `member` predicates.

Because of space constraints, we do not show here the code of the implemented "DFS with control of cycles" program. It can be found in [14], as well as in most Prolog textbooks.

Besides showing what happens both to the stack and to the SLD tree (while it is built), PrettyProlog correctly visualizes the effect of a "cut" on the SLD tree. In the upper part of Figure 1 there are goals written in italic (from *`not(member(arad, [oradea, zerind, arad])), ...`* to *`!, fail, write(....)`*). These nodes are cut after the execution of the `!` in the first clause defining `not`, called with `member(arad, [oradea, zerind, arad]))` as argument. PrettyProlog SLD viewer keeps the cut goals for didactic purposes, but shows them in a different font to emphasize that they no longer belong to the tree. The system predicates supported by PrettyProlog, although limited, include simple predicates for input-output, such as `write` and `nl`.

The stack viewer shows each frame pushed onto the stack. When the user clicks on a frame, its content is displayed: the goal that still had to be solved at the time the frame was pushed on the stack; the substitution that is the partial solution to such goal at this point; the clause that has been used to obtain the goal; the index from where, on backtracking, the engine will search for the next clause.

When the PrettyProlog engine solves a goal step-by-step, the clause used in each resolution step is highlighted in the theory panel.

2.2 Implementation

PrettyProlog is made of several modules, each one corresponding roughly to a Java package. Modules are pretty much organized in a layered fashion, with the lower-level ones providing services to the upper-level ones. Currently implemented modules include: the Data Types Module, the Parser Module, the Engine Module, the GUI Module.

- The `Data Types` Module includes data types that are commonly used throughout many other PrettyProlog modules. From this point of view, the Data Types Module is the lowest-level one.
- The `Parser` Module contains the `Parser` class and some parser exception classes. This module lies just above the Data Types Module; its task is to read characters from a stream and produce instances of PrettyProlog data types, or throw an exception if something goes wrong.
- The `GUI Module` includes the classes that make up the PrettyProlog GUI, including the viewers for the Stack, Theory, and SLD Tree.
- The `Engine Module` is the main PrettyProlog module. It contains the `Engine` class as well as many helper classes such as `Theory`, `Goal` and `Clause`. This module contains also two sub-modules: `EventListeners` , which provides

classes and interfaces used to attach listeners to the `Engine`, the `Stack`, and the `Theory`; and `Syspreds`, which defines the built-in system predicates and gives the programmer the possibility to easily add new ones.

The classes that make up the Engine Module are the following:

`Unifier.` This class provides a single public method, `unify(Term, Term)`, that returns a substitution that unifies the two terms passed as arguments, or null if they are not unifiable. This class exists as a separate class for reasons of modularity and extensibility.

`Clause.` A clause is an object made of a `Callable` (the head) and a body, again a nameless callable.

`Theory.` This class implements a list of clauses, with the usual operations for adding to, removing from, or navigating through the list.

`Frame.` A `Frame` is a single piece of data that is contained in a stack.

`Stack.` In addition to the usual stack operations, this class can register `StackListeners` which are notified of every change in the stack's state. The lack of an explicit representation of the stack in many prolog implementations, and the requirement to have such a data structure for inspecting the behavior of the Prolog engine were the main motivations for building a new interpreter from scratch.

`Goal.` A goal is a list of callables.

`Engine.` The main class of the Engine module.

3 Constraint Logic Programming

We briefly recall here some basic notions of Constraint Logic Programming (CLP). The reader is referred to [10] for a recent survey. We mix syntax and semantics to shorten the presentation.

Let us consider a first-order language $\langle \Pi, \mathcal{F}, \mathcal{V} \rangle$, where $\Pi, \mathcal{F}, \mathcal{V}$ are the sets of predicate symbols, functional symbols, and variables, respectively. The set Π is partitioned in the two sets Π_C and Π_P ($\Pi = \Pi_C \cup \Pi_P$ and $\Pi_C \cap \Pi_P = \emptyset$). Π_C (Π_P) is the set of constraints (resp., program defined") predicate symbols. Π_C is assumed to contain the equality symbol "=". Similarly, $\mathcal{F}$ is partitioned into $\mathcal{F}_C$ and $\mathcal{F}_P$. In this paper we focus on CLP on finite domains (CLP(FD)), therefore, we assume that $\mathcal{F}_C$ contains the binary arithmetic function symbols $+, -, *, /,$ `mod` etc. as well as a constant symbol for any integer number, and Π_C contains $\leq, <$, etc. `false` is assumed to be a special predicate in Π_P which has no rules defining it. `domain` is assumed to be a predicate in Π_C assigning a domain to a list of variable or refining it, if the variables already have one.

An atom built on $\langle \Pi_P, \mathcal{F}_P, \mathcal{V} \rangle$ (resp. $\langle \Pi_C, \mathcal{F}_C, \mathcal{V} \rangle$) is said to be a program (resp. constraint) atom. Any constraint atom and any subterm based on $\langle \mathcal{F}_C, \mathcal{V} \rangle$ is interpreted in a *constraint domain*, namely, fulfilling the intended semantics of its symbols (in this case, the arithmetical properties on integer numbers). The same happens to constraint atoms. To this aim, each variable X used in constraint atoms is associated with a domain D_X. We will use the functions min and max that return the smallest (resp., largest) value of a domain.

A *primitive constraint* is a constraint atom or its negation and a *constraint* is a conjunction of primitive constraints. We denote the empty conjunction by `true`. This syntactic notion of constraint has a semantic counterpart: a constraint C on n variables $X_1, \ldots, X_n$ with domains $D_1, \ldots, D_n$, respectively, is a relation on $D_1 \times \cdots \times D_n$. A *solution* for C is a mapping $[X_1/d_1, \ldots, X_n/d_n]$ such that $\langle d_1, \ldots, d_n \rangle \in C$. If there are no solutions, then C is inconsistent.

A *goal CLP* is of the form $\leftarrow \bar{B}$, where $\bar{B}$ is a conjunction of program atoms and primitive constraints. A *CLP rule* is of the form $A \leftarrow \bar{B}$ where A is a program atom and $\leftarrow \bar{B}$ is a CLP goal. A *CLP program* is a set of CLP rules.

The operational semantics of CLP is parametric on the function `solve` that given a constraint C should detect whether C is satisfiable (consistent) in the constraint domain chosen (in this paper finite domains). During its computation, `solve`(C) might rewrite C to an equivalent simplified constraint. In practice, for complexity reasons, `solve` is an incomplete procedure, in the sense that instead of verifying consistency of the (entire) constraints, acts locally in each primitive constraint, removing some values in domains that cannot belong to any solution until a local property is satisfied. Typical local properties are (hyper)arc consistency and (hyper)bounds consistency (see below for details). This operation is called *constraint propagation* and is required to be as fast as possible. During this computation, inconsistency of C can be detected. In this case `solve`(C) returns `false`. But, even if `solve`$(C) \neq$ `false` we cannot be sure of the consistency of the constraint. As we will see below, the user might require the search for a solution using the labeling predicate.

As an example, let us consider the constraint $X \neq Y, X \neq W, X \neq Z, Y \neq W, Y \neq Z, W \neq Z$, where $D_X = D_Y = D_Z = D_W = \{0, 1, 2\}$. Although it is inconsistent, default options in Prolog implementations are such that it is left unaltered by `solve` and, therefore, inconsistency is not detected. This constraint is the encoding of the 3-coloring problem of a graph (in this case, of four nodes, $\{X, Y, W, Z\}$, disequations are added for each edge). Checking consistency of this class of constraints is therefore NP-complete and a fast propagation algorithm can not check it (unless P=NP).

Operational semantics of CLP is based on the notion of state. Some variants are possible. The one presented here is the one we believe is the closest to standard SLD resolution.

A *state* is a pair $\langle G \,|\, C \rangle$ where G is a CLP goal and C is a constraint (also known as the *constraint store*). A state $\langle G \,|\, C \rangle$ is said to be:

- *successful* if $G =$ `true` and `solve`$(C) \neq$ `false`.
- *failing* if either `solve`$(C) =$ `false` or there are no clauses in P with the same predicate of the head of the selected atom in G.
- *unsolved* if $G \neq$ `true` and it is not failing.

Let $\langle G_1 \,|\, C_1 \rangle$ be an unsolved state, where $G_1 = \leftarrow L_1, \ldots, L_m$, and P a program. A *CLP-derivation step* $\langle G_1 \,|\, C_1 \rangle \Rightarrow \langle G_2 \,|\, C_2 \rangle$ is defined as follows:

- Let L_i be the selected literal in G_1 (for simplicity, let us assume it is L_1).

– Then $\langle G_2 \,|\, C_2 \rangle$ is obtained from S and P in one of the following ways:
  - L_1 is a primitive constraint, $C_2 = L_1 \wedge C_1$. If $\texttt{solve}(C_2) = \texttt{false}$, then $G_2 = \leftarrow \texttt{false}$, otherwise $G_2 = \leftarrow L_2, \ldots, L_n$.
  - If $L_1 = p(t_1, \ldots, t_n)$ is a program atom, and $p(s_1, \ldots, s_n) \leftarrow \bar{B}$ is a renaming of a clause of P then $G_2 = \leftarrow t_1 = s_1, \ldots, t_n = s_n, \bar{B}, L_2, \ldots, L_n$ and $C_2 = C_1$.

A *derivation* for a state S_0 in P is a maximal sequence of derivations such that $S_0 \Rightarrow S_1 \Rightarrow \cdots$. A derivation for a goal G is a derivation for the state $\langle G \,|\, \texttt{true} \rangle$.

A finite *derivation* $S_0 \Rightarrow \cdots \Rightarrow S_n$ is said *successful* (resp. *failing*) if S_n is a successful (resp., failing) state. In the case of a successful derivation the computed answer is the projection of the constraint store of S_n on the variables in S_0. Of course, a simplification, based on `solve`, is usually employed to make the output readable.

Although the computed answer is returned in implicit form, explicit enumeration of the solutions can be forced by using the built-in predicate `labeling`. In this stage inconsistency of a constraint is discovered. The main parameter is a list of variables to be instantiated (labeled). Other optional parameters are related to the search heuristics and are different in different Prolog implementations. We use here the choice of not allowing extra parameters.

Basically, starting from a successful state $\langle \texttt{true} \,|\, C \rangle$ (or equivalently, by a constraint C) the labeling builds a search tree that alternates two stages: a constraint propagation stage followed by a non deterministic assignment of a (selected) variable. During the propagation stage the constraint is simplified and, possibly, its inconsistency is detected. In this case, the search backtracks to the last non deterministic choice. If all variables are assigned (labeled) a solution is found. If all possible backtracks are applied and no solution is found, the constraint is inconsistent. The derivation step is extended with:

- If $L_1 = \texttt{labeling}([V_1, \ldots, V_n])$ then $G_2 = \leftarrow V_1 = v_1, \ldots, V_n = v_n, L_2, \ldots, L_n$ and $C_2 = C_1$ if $[V_1/v_1, \ldots, V_n/v_n]$ is an assignment that do not lead C_1 to inconsistency. If there are not such assignments, then $G_2 = \leftarrow \texttt{false}$.

Every variable X used in constraint is associated with a domain D_X. This is done initially by the built-in predicate `domain`; domains and later reduced by effect of the computation. As common in CLP, we denote the interval $\{a, a+1, a+2, \ldots, b\}$ by $a..b$.

Let us consider a primitive constraint c on the variables $X_1, \ldots, X_n$. c is *arc consistent* (hyper arc consistent if $n > 2$) if for all $i \in \{1, \ldots, n\}$ and for all $d_i \in D_i$ exist $d_1 \in D_1, \ldots d_{i-1} \in D_{i-1}, d_{i+1} \in D_{i+1}, \ldots, d_n \in D_n$ such that $[X_1/d_1, \ldots, X_n/d_n]$ is a solution of c.

As explained in [5] there are several definitions of bounds consistency in literature. We refer to the one implemented by SICStus Prolog[3], by B Prolog[4], and by SWI Prolog, just to cite a few, and and called *interval consistency* in [2].

[3] `http://www.sics.se/isl/sicstuswww/site/`
[4] `http://www.probp.com/`

Let us consider a primitive constraint c on the variables $X_1, \ldots, X_n$. c is *bounds consistent* (hyper bounds consistent if $n > 2$) if for all $i \in \{1, \ldots, n\}$ and for all $d_i \in \{\min D_i, \max D_i\}$ (the two interval bounds), exist

$$d_1 \in \min D_1 .. \max D_1, \ldots, d_{i-1} \in \min D_{i-1} .. \max D_{i-1},$$
$$d_{i+1} \in \min D_{i+1} .. \max D_{i+1}, \ldots, d_n \in \min D_n .. \max D_n$$

such that $[X_1/d_1, \ldots, X_n/d_n]$ is a solution of c.

Going back to the example in the Introduction, the constraint $2X = Y$ where the domains $D_X = \{0, 1\}, D_Y = \{0, 1, 2\}$ is bounds consistent but not arc consistent.

4 PrettyCLP

This section introduces PrettyCLP: Section 4.1 reports the concrete syntax of the CLP part of PrettyCLP, Section 4.2 describes the procedures implemented for constraint propagation and labeling, and Section 4.3 shows how the output primitives have been modified and reports some system screenshots.

4.1 Concrete CLP syntax for PrettyCLP

Concretely, the set Π_C contains the constraint predicate symbols `#=` and `#=<`; symbols `#\=`, `#>`, `#>=`, `#<` are accepted as a syntactic sugar for building negated constraint literals. Standard arithmetic functional symbols are allowed as well.

The built-in instruction `domain` for assigning a finite domain to variables can be used in two ways (lists are usual Prolog lists):

- `domain(VARS, min, max)`, where `VARS` is a list of variables, and the domain is the interval `min..max`.
- `domain(VARS, DOMAIN)`: where `VARS` is a list of variables, and `DOMAIN` is a list of integer numbers.

The labeling built-in has a unique argument: `labeling(VARS)`, where `VARS` is a list of variables assigned to a finite domain.

4.2 Constraint Propagation and Labeling

Every finite domain variable is assigned to a domain, namely a set of points that can monotonically decrease during the computation, or increase again due to backtracking. Domains are stored in a vector of integer values (we recall that Java vectors are in fact a dynamic data structures that can be increased and decreased as needed). This is realized by modifying the class `Variable` within the module `Data Types`.

More in detail, a Java constructor is changed in order to characterize a variable by a `symbol`. `symbol` is a PrettyProlog class, and a symbol can be the string naming a variable or a constant or functional symbol of $\mathcal{F}_C$.

Moreover, the class `Domain` is defined in the module `Engine`. This class stores an array (again, a Java dynamic array) with an entry for each of the variables occurring in the derivation and implements the methods for domain manipulation. Among them, we would like to point out the methods:

- `getDomain(Var X)` that returns the (vector storing the) domain of a variable `X`.
- `domainVar(Var X, int min, int max)` that initializes the domain of the variable `X` with the interval `min..max`.
- `updateDomain(Var X, Vector D)` that replaces the current domain of the variable `X` with the domain array `D`. This method simplifies propagation and backtracking operations.

The parser of Pretty Prolog has been slightly modified to be able to deal with constraint terms and atoms.

The class `Engine` is the core of the computation. Since the SLD resolution is now coupled with constraint solving, we need to act on the class `Engine`. Unification is called for standard terms, constraint solving for constraint terms. The constructor that creates a new instance has been modified for dealing with the array domains. Then the method `ContinueSolving` is called. Its role is to either call the Constraint Solver procedure, called `SolveConstraint`, or the unification procedures already developed in PrettyProlog.

The Constraint Solver returns a Boolean value: `false` can be obtained as effect of constraint propagation; if constraint propagation ends without failing it returns `true`. Constraint Propagation operates on both arc and bounds consistency. The procedures called by the Constraint Solver are:

- `Solve` that receives as input a constraint and the domains of the variables in it and elaborates it on the basis of its main predicate symbol. When selected, unary constraints are used to reduce the domain of the corresponding variable and removed. Constraints with two or more variables, instead, are dealt with by:
  - `ArcSolveConstraint`, implementing arc consistency, and
  - `BoundSolveConstraint`, that implements bounds consistency.

  In these two procedures, one variable per time is selected, then every value in its domain is considered and a support for it is looked for in the domains of the remaining variables (but with a different rule for arc vs bounds). In the case of bounds, of course, a faster algorithm based on the bounds of the domains is employed.
- `SolveConstraint` repeatedly applies the `Solve` procedures on all the constraints until a fixpoint is reached.
- `ExprEval` computes the various expressions involved in constraints when values are assigned to variables and is used as auxiliary procedure by `ArcSolve`/ `BoundSolveConstraint`.

The handling of `labeling` is made by a homonymous method. Variables in the argument list are selected from left to right (heuristics `leftmost` of other CLP(FD) systems) and smallest domain values are tried first (heuristics `up`).[5]

[5] It it easy to implement other heuristics here. This will be done as future work.

This method also deals with backtracking, handling,choice points, and storing and retrieving intermediate constraint stores.

Remark 1. As a final observation for this section, we would like to underline a typical problem in CLP implementations coming from the weak typing of constraint functional symbols and variables. In theory, terms and variables are sorted, in practice this is not true. If the two terms $1 + 3$ and $3 + 1$ are found within a unification they are assumed to be different even if the arguments are integer numbers and the binary symbol $+ \in \mathcal{F}_C$. On the contrary the constraint $1 + 3$ `#=` $3 + 1$ is true. This is also the behavior of our interpreter, but this may lead a student, the target of PrettyCLP, to confusion.

Similarly, let us assume to find the constraint atom X `#<` 3 and the variable X is not yet assigned to a domain. Some Prolog implementations will answer X `in -inf..2`, thus implicitly assuming a starting domain `-inf..+inf` for each finite domain variable. We have chosen a more rigid option: if the variable has been not yet associated with a domain, it cannot be used in a constraint atom. An error (a sort of type error) is returned.

4.3 Output rendering

After the parametric propagation procedures and the procedures for the labeling have been implemented, we modified the graphical applet of PrettyProlog for showing the new information. In particular:

- buttons for visualizing and erasing domains have been added to the applet window;
- the field Constraint can be inspected from the `Stack Viewer`;
- two windows for inspecting Arc Consistency and Bounds Consistency based propagation have been made available to PrettyCLP users.

Changes are done in the method `TheoryViewer`.

In Figures 2 and 3 we report the rendering of the two alternative executions to a goal `p(X,Y)` where the predicate `p` is defined by the constraint:

```
domain([X],0,2), domain([Y],0,5), Y #= 2 * X.
```

In the current implementation, we decided to leave an unique SLD tree, while different computed answers are returned in the Arc Consistency and Bounds Consistency windows. Since Arc Consistency is more effective in reducing the domains, it can be the case that an inconsistent branch of the SLD tree is found some steps before than using bounds consistency. However, this case is extremely rare. We have preferred to leave the tree obtained using Bounds Consistency (the same computed by Standard Prolog system) only. However, the user can view what would have happened with the other propagation technique looking at the Arc Consistency window. In particular, all domains computed during the computation are included in those computed with Bounds Consistency. Sometimes they are strictly included and it may happen that one domain becomes empty

before arriving at the end of the tree. This choice can be changed as future work, namely, we could leave the user to select in advance (with a button) the propagation choice and report the selected computation only.

In Figures 4–5 we report the execution of PrettyCLP on a Knapsack problem. The explicit labeling is required for variable `W` only. The computed solution is shown in the main picture (Fig 4). In the labeling case the differences between Arc and Bounds become evident. An excerpt of the executed constraint propagation is shown in Fig 5.

5 Related Work and Conclusions

Although visualization and tracing of constraint programs do not constitute a new research area, implemented systems that provide dynamic visualization and control functionalities for CLP(FD) are still few.

Among the oldest ones we may mention the Oz-Explorer system by C. Schulte [12] which uses the search tree of a constraint problem as its central metaphor, and exploration and visualization of the search tree are user-driven and interactive. Within the community of constraint programming, ILOG debugger[6] is a configurable tool with nice graphics capabilities. But, according to our opinion, it is not suitable for beginners (and it is not a CLP visual debugger). Similarly, CPVisu tracer[7] is a module of the Java constraint solver on finite domains CHOCO, producing XML files that, once interpreted using CPViz[8], allow to see information on the tree search, the states of constraints and variables at different points of computations, and a configuration file. It is a professional tool (together with CHOCO, which is developed by a large group of people including Francois Laburthe, Narendra Jussien, Xavier Lorca, and other contributors, such as Nicolas Beldiceanu), but its scope is for debugging large programs rather than for learning CP (and, in any case, it does not deal with CLP).

In [3], M. Carro and M. V. Hermenegildo address the design and implementation of visual paradigms for observing the execution of constraint logic programs, aiming at debugging, tuning and optimization, and teaching. They describe two tools, VIFID and TRIFID, exemplifying the devised depictions. In the companion paper [4] they describe the APT tool for running constraint logic programs while depicting a (modified“) search tree, keeping information about the state of the variables at every moment in the execution. This information can be used to replay the execution at will, both forwards and backwards in time. The search-tree view is used as a framework onto which constraint-level visualizations can be attached.

The integration of explanations in the trace structure and some ideas on how to implement the trace structure in a high-end system like SICStus are addressed by [1].

[6] `http://www.cs.cornell.edu/w8/iisi/ilog/cp11/pdf/debugsolver.pdf`
[7] `http://www.emn.fr/z-info/choco-solver/cpvisu-tracer/index.html`
[8] `http://cpviz.sourceforge.net/`

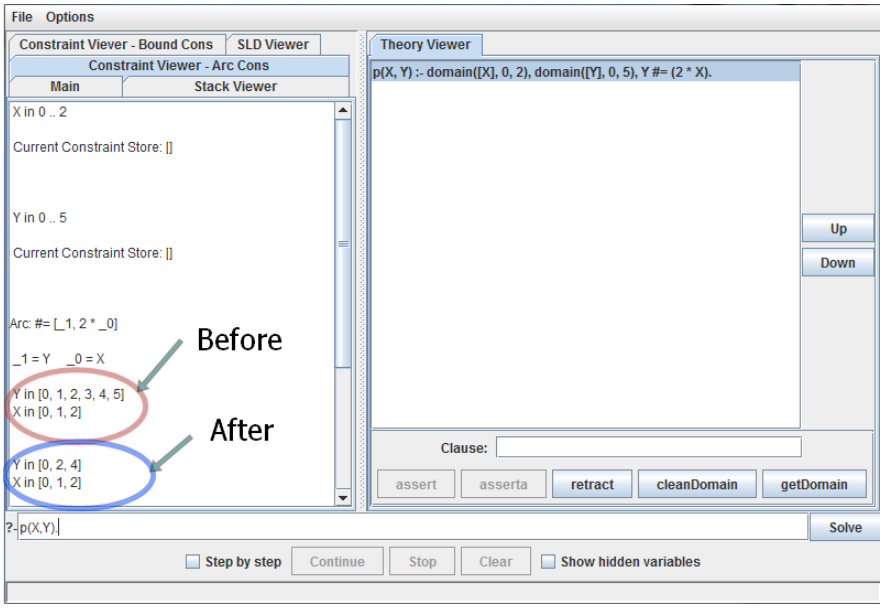


Fig. 2. PrettyCLP in action: Arc Consistency Propagation

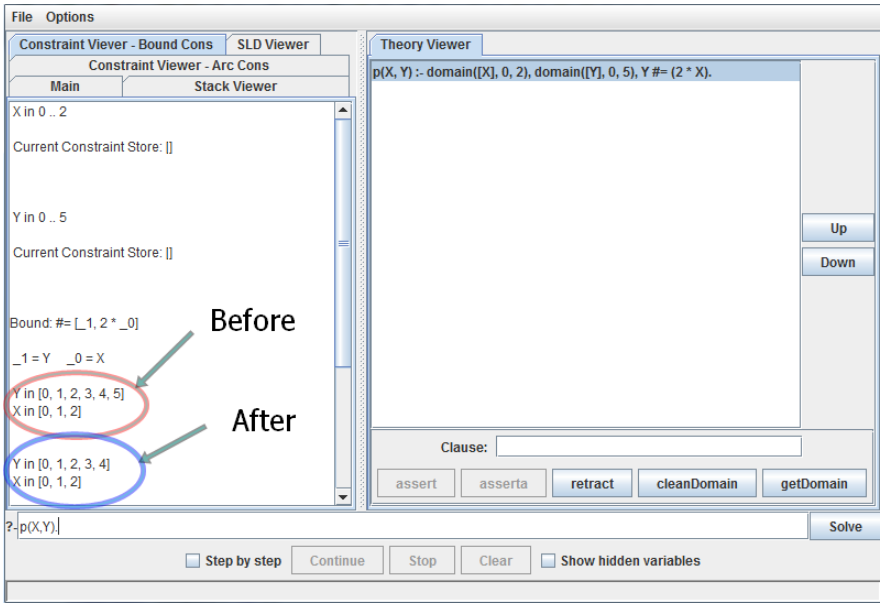


Fig. 3. PrettyCLP in action: Bounds Consistency Propagation

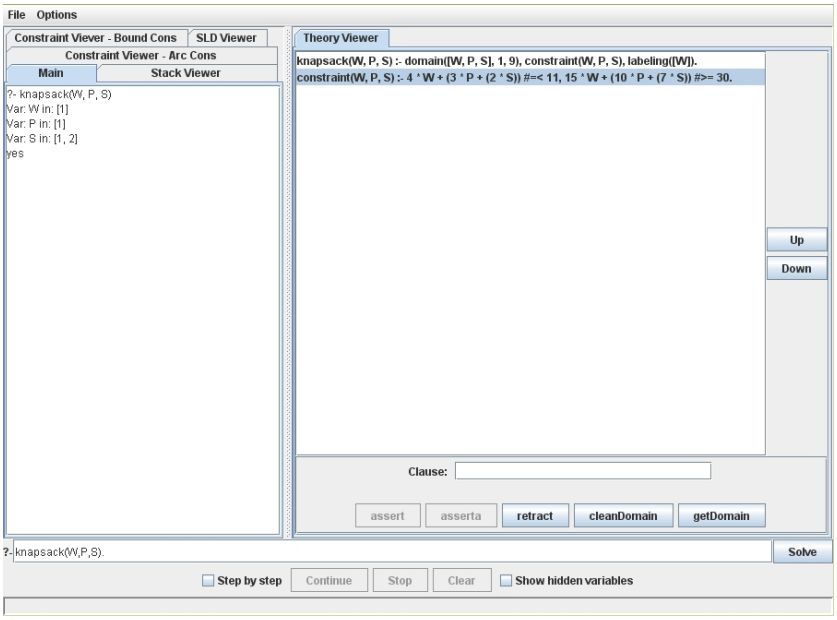


Fig. 4. PrettyCLP in action: Knapsack main output

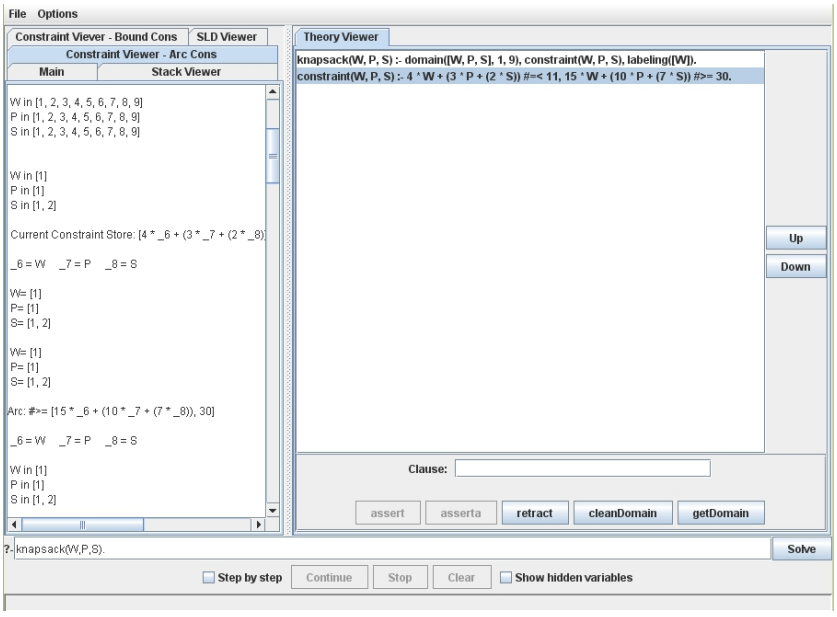


Fig. 5. PrettyCLP in action: Knapsack, propagation details

More recently, F. Fages, S. Soliman, and R. Coolen developed CLPGUI [8], a generic graphical user interface for visualizing and controlling the execution of constraint logic programs. CLPGUI is based on a client-server architecture for connecting a CLP process to a Java-based GUI process, and integrates a non-intrusive tracing and control method based on annotations in the CLP program. Arbitrary constraints and goals can be posted incrementally from the GUI in an interactive manner, and arbitrary states can be recomputed. Several generic 2D and 3D viewers of the variables and of the search tree are supported.

Although definitely simpler than some of the above systems as far as the visualization of variables is concerned, PrettyCLP shows the original feature of allowing the user to select the propagation strategies (e.g. arc consistency vs bound consistency), which – to the best of our knowledge – is not supported by any other tool.

As part of our close future activities we are planning to incorporate some global constraints, such as the `alldifferent` one, into PrettyCLP. Propagation procedures of global constraints allow to sensibly prune the search tree. This has effects on the size and on the form of the SLD tree too. Therefore, we will add buttons to enable/disable global consistency vs simple consistency so as to select one or the other search tree.

Because of our teaching mission, from which the development of both PrettyProlog and PrettyCLP stemmed, we are currently facing the problem of guiding the student in his/her CLP learning activity. To this aim, we are designing a set of benchmarks for supporting self-evaluation and for helping students in identifying those aspects of CLP design and programming that they still need to better understand. This benchmark will heavily ground upon PrettyCLP as the tool that the students will be suggested to use in order to appreciate CLP not only on the stage, but also behind the scene.

All the material relevant to PrettyProlog and PrettyCLP can be found at `http://code.google.com/p/prettyprolog/`.

Acknowledgments

This work is partially supported by INdAM-GNCS 2010, INdAM-GNCS 2011, and PRIN 20089M932N. We would like to thank Maurizio Martelli for the precious discussions during the design and implementation of the PrettyProlog visualizer.

References

1. Ågren, M., Szeredi, T., Beldiceanu, N., and Carlsson, M. Tracing and explaining execution of clp(fd) programs. In *WLPE* (2002), pp. 1–16.
2. Carlsson, M., Ottosson, G., and Carlson, B. An open-ended finite domain constraint solver. In *Proc. Programming Languages: Implementations, Logics, and Programs* (1997), vol. 1292 of *LNCS*, Springer, pp. 191–206.

3. Carro, M., and Hermenegildo, M. V. Tools for constraint visualisation: The vifid/trifid tool. In *Analysis and Visualization Tools for Constraint Programming* (2000), P. Deransart, M. V. Hermenegildo, and J. Maluszynski, Eds., vol. 1870 of *Lecture Notes in Computer Science*, Springer, pp. 253–272.
4. Carro, M., and Hermenegildo, M. V. Tools for search-tree visualisation: The apt tool. In *Analysis and Visualization Tools for Constraint Programming* (2000), P. Deransart, M. V. Hermenegildo, and J. Maluszynski, Eds., vol. 1870 of *Lecture Notes in Computer Science*, Springer, pp. 237–252.
5. Choi, C. W., Harvey, W., Lee, J. H. M., and Stuckey, P. J. Finite domain bounds consistency revisited. In *Australian Conference on Artificial Intelligence* (2006), vol. 4304 of *LNCS*, Springer, pp. 49–58.
6. Ducassé, M., Emde, A.-M., Kusalik, T., and Levy, J., Eds. *Logic Programming Environments, ICLP'90 Preconference Workshop*. 1990. ECRC Technical Report IR-LP-31-25.
7. Eisenstadt, M., and Brayshaw, M. A fine-grained account of Prolog execution for teaching and debugging. *Instructional Science 19*, 4/5 (1990), 407–436.
8. Fages, F., Soliman, S., and Coolen, R. Clpgui: A generic graphical user interface for constraint logic programming. *Constraints 9*, 4 (2004), 241–262.
9. Gavanelli, M. SLDNF-Draw: a visualisation tool of prolog operational semantics. In *CILC'07, Messina* (June 2007).
10. Gavanelli, M., and Rossi, F. Constraint logic programming. In *A 25-Year Perspective on Logic Programming* (2010), vol. 6125 of *LNCS*, pp. 64–86.
11. Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach, Second Edition*. Prentice Hall, 2003.
12. Schulte, C. Using the oz explorer for the development of constraint programs. In *LPE* (1997), pp. 55–56.
13. Shinomi, H. Graphical representation and execution animation for Prolog programs. In *MIV* (1989), IEEE Computer Society, pp. 181–186.
14. Stalla, A., Mascardi, V., and Martelli, M. PrettyProlog: A Java Interpreter and Visualizer of Prolog Programs. In *CILC'09, Ferrara* (June 2009). System available at `http://code.google.com/p/prettyprolog/`.
15. Tamir, D. E., Ananthakrishnan, R., and Kandel, A. A visual debugger for pure Prolog. *Inf. Sci. Appl. 3*, 2 (1995), 127–147.

A framework for structured knowledge extraction and representation from natural language via deep sentence analysis

Stefania Costantini[1], **Niva Florio**[1], and **Alessio Paolucci**[1]

Dip. di Informatica, Università di L'Aquila, Coppito 67100, L'Aquila, Italy
stefania.costantini@univaq.it
niva.florio@univaq.it
alessio.paolucci@univaq.it

Abstract. We present a framework that we are currently developing, that allows one to extract knowledge from natural language sentences using a deep analysis technique based on linguistic dependencies. The extracted knowledge is represented in OOLOT, an intermediate format that we have introduced, inspired by the Language of Thought (LOT) and based on Answer Set Programming (ASP). OOLOT uses an ontology-oriented lexicon and syntax. Therefore, it is possible to export the extracted knowledge into OWL and native ASP.

1 INTRODUCTION

Many intelligent systems have to deal with knowledge expressed in natural language, either extracted from books, web pages and documents in general, or expressed by human users. Knowledge acquisition from these sources is a challenging matter, and many attempts are presently under way towards automatically translating natural language sentences into an appropriate knowledge representation formalism [1]. Although this task is a classic Artificial Intelligence challenge (mainly related to Natural Language Processing and Knowledge Representation [2]), with the Semantic Web growth new interesting scenarios are opening. The Semantic Web aims at complementing the current text-based web with machine interpretable semantics; however, the manual population of ontologies is very tedious and time-consuming, and practically unrealistic at the web scale [3, 4]. Given the enormous amount of textual data that is available on the web, to overcome the knowledge acquisition bottleneck, the ontology population task must rely on the use of natural language processing techniques to extract relevant information from the Web and transforming it into a machine-processable representation.

In this paper we present a framework that we are currently developing. It allows one to extract knowledge from natural language sentences using a deep analysis technique based on linguistic dependencies and phrase syntactic structure. We also introduce OOLOT (Ontology-Oriented Language of Thought). It is

an intermediate language based on ASP, specifically designed for the representation of the distinctive features of the knowledge extracted from natural language. Since OOLOT is based on an ontology-oriented lexicon, our framework can be easily integrated in the context of the Semantic Web.

It is important to emphasize that the choice of ASP is a key point and it is of fundamental relevance. In fact according to [5], this formalism is the most appropriate one to deal with normative statements, default statements, exceptions and many other characteristic aspects of knowledge encoded through Natural Language.

Though this is an ongoing work, we believe to be able to argue in favour of the usefulness and the potential of the proposed approach.

In particular, in Section 2 we introduce the framework architecture. In Section 3 we analyze the state of the art for parsing and extraction of dependencies taking into account our translation needs, taking into particular account three kinds of parsers. Section 4 describes the context disambiguation and lexical item resolution methods that we have devised. Section 5 introduces the intermediate format OOLOT, while Section 6.3 describes the translation methodology with the help of an example. Finally, Section 7 shows an exporting from OOLOT into OWL example, and in Section 8 we conclude with a brief resume of achieved goals and future works.

2 FRAMEWORK ARCHITECTURE

The proposed framework aims at allowing automatic knowledge extraction from plain text, like a web page, producing a structured representation in OWL or ASP as output. Thus, the framework can be seen as a standalone system, or can be part of wider workflow, e.g. a component of complex semantic web applications.

Starting from plain text written in natural language, as first step we process the sentence through a statistical parser (see Section 3). If we use a parser with embedded dependency extractor, we can perform a single step and have as output both the parse tree (constituents), and in the meantime the dependency graph. Otherwise, if we use two different components, the workflow is that of Fig.1. In this case, we use a simple algorithm for context disambiguation (see Section 4). Then, each token is resolved w.r.t. popular ontologies including DBPedia and OpenCYC and the context is used in case of multiple choices.

At this point we have enough information to translate the knowledge extracted from a natural language sentence into our intermediate OOLOT format. OOLOT stands for “Ontological Oriented Language Of Though”, a language mainly inspired by [6], that we have introduced as an intermediate representation language for the extracted knowledge in a way that is totally independent from the original lexical items, and therefore, from the original language. OOLOT is itself a language, but its lexicon is ontology-based; it uses Answer Set Programming as basic host environment that allows us to compose a native, high expressive knowledge representation and reasoning environment. For the

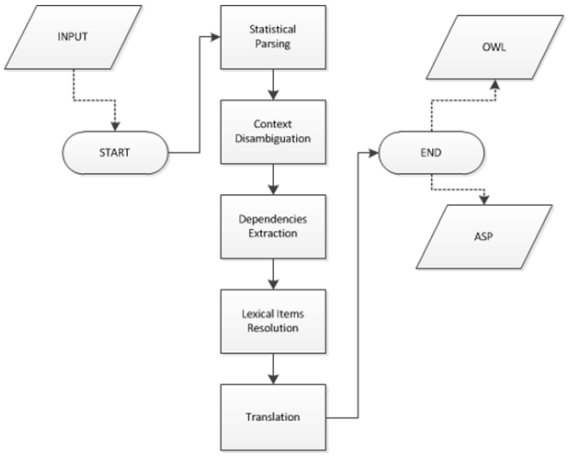


Fig. 1. The framework architecture

translation process described in Section 6, we employ a λ-calculus engine that drives the translation into OOLOT, using information about the deep structure of the sentence extracted in the previous steps.

From the ontology-based Language of Thought it is possible to directly translate the encoded knowledge into OWL. In addition, it is also possible to export the knowledge base in pure ASP.

3 PARSING

3.1 Background

In informatics and linguistics, parsing is the process that can determine the morpho-syntactic structure of a sentence; parsing associates a sentence expressed in a natural language with a structure (e.g. a parse tree structure), that analyses the sentence by a certain point of view; thus there are morphological parsing, syntactic parsing, semantic parsing, etc.

With regard to the syntactic parsing, analysis consists of a definition of the phrases building up the sentence in their hierarchical order, likewise the constituent analysis proposed by Chomsky [7]. In the 1950s Noam Chomsky said natural language sentences can be generated by a formal grammar [8, 7]; this is the so-called generative approach, motivated by the fact that people are able to generate sentences that are syntactically correct and totally new (i.e., that have never been heard before). Syntactic parser decomposes a text into a sequence of tokens (for example, words), and attributes them their grammatical functions and thematic or logical roles, with respect to a given formal grammar. The task of syntactic parser is to say if the sentence can be generated by the grammar and, if so, it gives the appropriate sentence syntactic representation (called parse tree), showing also the relations between the various elements of the sentence [8,

7].

Most of today's syntactic parsers are mainly statistical [9–12]. They are based on a corpus of training data that have been previously annotated (parsed) by hand. This approach allows the system to gather information on the frequency with which the various constructs are needed in specific contexts. A statistical parser can use a search procedure on the space of all candidates, and it would provide the probability of each candidate and makes it possible to derive the most probable parse of a sentence.

In the '90, Collins proposes a conditional and generative model which describes a straightforward decomposition of a lexicalized parse tree [11, 12] based on a Probabilistic Context Free Grammar (PCFG). Charniak and Johnson's parser [10, 13] is based on a parsing algorithm for PCFG, but it is a lexicalized N-Best PCFG parser: it is a generative and discriminative reranking parser which uses the MaxEnt reranker to select the best among the possible parses. The Berkeley parser uses an automatically induced PCFG, parsing sentences with a hierarchical state-splitting. Only statistics is not enough to determine when to split each symbol in sub-symbols [14]; thus [14, 15] present an automatic approach for obtaining annotation trees through a split-merge method. At a first stage, this parser considers a simple PCFG derived from a treebank, but then it iteratively refines this grammar, in order to sub-categorize basic symbols (like NP and VP) into sub-symbols. So non-terminal basic symbols are split and merged in order to maximize the training treebank and to add a larger number of annotations to the previous grammar. This parser can learn automatically the type of linguistic distinction showed in the manually annotated treebank and then it can create annotation trees thanks to a more complex and complete grammar.

Statistical parsing is useful to solve problems like ambiguity and efficiency, but with this kind of parsing we lose part of the semantic information; this aspect is recovered thanks to dependency representation [16].

Dependency grammars (DGs) were proposed by the French linguist Tesnière [17] and have recently received renewed attention (cfr. [18] and the references therein). In Dependency Grammars, words in a sentence are connected by means of binary, asymmetrical governor-dependent relationships. In fact, Tesnière assumes that each syntactic connection corresponds to a semantic relation. In a sentence, the verb is seen as the highest level word, governing a set of complements, which govern their own complements themselves. Opposed to the notion of the sentence division into a subject and predicate, the grammatical subject in Tesnière's work is also considered subordinate to the verb. The valence of a verb (its property of requiring certain elements in a sentence) determines the structure of the sentence it occurs in. Tesnière distinguishes between actants, which are required by the valence of the verb, and circonstants which are optional.

3.2 Parser analysis

It is difficult to evaluate parsers; we can compare them in many ways, such as the speed with which they examine a sentence or their accuracy in the analysis

(e.g. [23]). The task based evaluation seems to be the best one [16, ?]: we must choose whether to use a parser rather than another simply basing on our needs. At this stage of our ongoing research, we use the Stanford parser because it is more suited to our requirements, both for the analysis of the constituents and for that of the dependencies.

Stanford parser performs a dependency and constituent analysis [19, 20]. This parser provides us with different types of parsing. In fact, it can be used as an unlexicalized PCFG parser [19] to analyse sentences, or it can be used as a lexicalized probabilistic parser [20]. Thanks to an A* algorithm, the second version combines the PCFG analysis with the lexical dependency analysis. At the moment the Stanford parser provides us a typed dependency and a phrase structure tree. The Stanford typed dependencies (cfr. [16]) describe the grammatical relations in a sentence. The relations are binary and are arranged hierarchically; as Tesnière suggested, Stanford dependency relations have a head and its dependent but, unlike Tesnière, the head of a dependency can be any content words, not only verbs. Thanks to rules [21] applied on phrase structure trees (also created by the Stanford parser), typed dependencies are generated.
In particular, for constituent analysis, we choose to analyse the sentence *"Many girls eat apples."*. Seeing Fig.2, we can notice that the parser attributes to each token its syntactic roles, and it provides us also the grammatical function of each word. In order to better understand the hierarchy of the syntactic structure is useful to represent it as a tree (Fig.3).

```
(ROOT
  (S
    (NP (JJ Many) (NNS girls))
    (VP (VBP eat)
      (NP (NNS apples)))
    (. .)))
```

Fig. 2. Phrase structure produced by the Stanford parser for the sentence "Many girls eat apples".

With regard to dependency analysis, the Stanford parser gives us two versions of this analysis: the typed dependency structure (Fig.4) and the collapsed typed dependency structure (Fig.5). In the first, each node of the sentence is a node connected with a binary relation to another node; in the second, prepositions are turned into relations (unfortunately, in this example, you may not notice the difference). Thus, Fig.4 and Fig.5 show us that *girls* and *many* are connected with an *amod* relation, that means an adjective phrase modifies the meaning of the noun phrase; *eat* is connected to *girls* with a *nsubj* relation, where the noun phrase is the syntactical subject of the verb; *eat* and *apple* are connected with a *dobj* relation because the direct object of the verb phrase is the object of the verb [16].

As reference for the following steps, we use the Stanford syntactic phrase structure (Fig.2) and the Stanford collapsed typed dependencies structure (Fig.5).

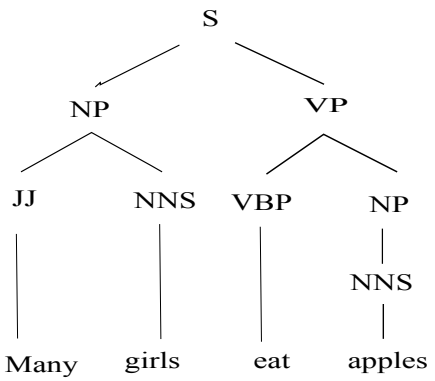


Fig. 3. Parse tree of the sentence "Many girls eat apples." for the analysis done by the Stanford parser.

amod(girls-2, Many-1)
nsubj(eat-3, girls-2)
dobj(eat-3, apples-4)

Fig. 4. Typed dependency structure produced by the Stanford parser for the sentence "Many girls eat apples.".

amod(girls-2, Many-1)
nsubj(eat-3, girls-2)
dobj(eat-3, apples-4)

Fig. 5. Collapsed typed dependency structure produced by the Stanford parser for the sentence "Many girls eat apples.".

4 CONTEXT DISAMBIGUATION AND LEXICAL ITEM RESOLUTION

The context disambiguation task is a very important step in our work flow, as we need to assign each lexical unit to the correct meaning, and this is particularly hard due to the polysemy. For this task, we use a simple algorithm: we have a finite set of contexts (political, technology, sport, ...), and as first step we built a corpus of web pages for each context, and then we used each set as a training set to build a simple lexical model. Basically we build a matrix where for each row we have a lexical item, and for each column we have a context. The relation (lexical item, context) is the normalized frequency of each lexical item into the given context. The model is then used to assign the correct context to a given sentence. We use a $n \times m$ matrix, where n is the number of lexical tokens (or items), and m is the number of contexts. In other words, we give a score for each lexical token in relation to each context. To obtain the final score we perform a simple sum of the local values to obtain the global score, and thus assign the final context to the sentence. Our method for context disambiguation

can certainly be improved, in particular [25], seems to be a good development direction.

The context is crucial to choose the correct reference when a lexical item has multiple meanings, and thus, in an ontology can be resolved in multiple references. The context becomes the key factor for the resolution of each lexical item to the relative reference.

We perform a lookup in the ontology for each token, or a set of them (using a sliding window of length k). For example, using DBPedia, for each token (or a set of tokens of length k), we perform a SPARQL query, assuming the existence of a reference to the lexical item: if this is true, we've found the reference, otherwise we go forward. If we found multiple references, we use the context to choose the most appropriate one.

Given the sentence *Many girls eat apples* and it's syntactic phrase structure (Fig.2) and dependencies structure (Fig.5), as first step we tokenize the sentence, obtaining:

Many, *girls*, *eat*, *apples*.

Before the lookup, we use the part of speech tagging from the parse tree to group the consecutive tokens that belong to the same class. In this case, such peculiar aspect of natural language is not present and thus the result is simply the following:

(*Many*), (*girls*), (*eat*), (*apples*).

Excluding the lexicon for which we have a direct form, for each other lexicon the reference ontology is resolved through a full text lookup; thus we obtain the lexical item resolution in Table 1.

Table 1. Lexical item resolution example

Lexicon	*Ontology*	*URI*
girls	DBPedia	http://dbpedia.org/resource/Girl
eat	DBPedia	http://dbpedia.org/class/Eating
apples	DBPedia	http://dbpedia.org/resource/Apple

5 FROM THE LANGUAGE OF THOUGHT TO OOLOT

The Language of Thought (LOT) is an intermediate format mainly inspired by [6]. It has been introduced to represent the extracted knowledge in a way that

is totally independent from original lexical items and, therefore, from original language.

Our LOT is itself a language, but its lexicon is ontology oriented, so we adopted the acronym OOLOT (Ontology-Oriented Language Of Thought). This is a very important aspect: OOLOT is used to represent the knowledge extracted from natural language sentences, so basically the bricks of OOLOT (lexicons) are ontological identifier related to concepts (in the ontology), and they are not a translation at lexical level.

In [26] a translation methodology from natural language sentences into ASP that takes into accounts all words of the sentence is presented. In this method, the final representation is itself dependent from the original lexical structure, and this is sub-optimal if we want to export our knowledge base into a more general formalism like, e.g., OWL.

In the next section we present the translation process.

6 TRANSLATING INTO OOLOT

6.1 Background

[26] describes a technique for extracting knowledge from natural language and automatically translate it into ASP. To achieve this result we built an extension of λ-calculus and we have introduced meta expressions to fully automate the translation process, originally inspired by [5].

This is a key point in the process of representing knowledge extracted from natural language, and thus for using it into other contexts, e.g. the Semantic Web. The selection of a suitable formalism plays an important role: in fact, though under many aspects first-order logic would represent a natural choice, it is actually not appropriate for expressing various kinds of knowledge, i.e. for dealing with default statements, normative statements with exceptions, etc. Recent work has investigated the usability of non-monotonic logics, like ASP [5] with encouraging results in terms of dealing with the kind of knowledge represented through natural language.

OOLOT allows us to have native reasoning capabilities (using ASP) to support the syntactic and semantic analysis tasks. Embedded reasoning is of fundamental importance for the correct analysis of complex sentences, as shown in [27]. The advantages of adopting an ASP-based host language is not limited to the previous aspects: in fact, the integration of ASP and the Semantic Web is not limited to the Natural Language Processing side. Answer Set Programming fits very well with Semantic Web as demonstrated by the recent research efforts of integrating rule-based inference methods with current knowledge representation formalisms in the Semantic Web [28, 29].

Ontology languages such as OWL and RDF Schema are widely accepted and successfully used for semantically enriching knowledge on the Web. However, these languages have a restricted expressivity if we have to infer new knowledge from existing one. Semantic Web needs a powerful rule language complementing

its ontology formalisms in order to facilitate sophisticated reasoning tasks. To overcome this gap, different approaches have been presented on how to combine Description Logics with rules, like in [29].

6.2 Lambda Calculus

λ-calculus is a formal system designed to investigate function definition, function application and recursion. Any computable function can be expressed and evaluated via this formalism [30]. In [26] we extended the λ-calculus introducing the λ-ASP-$Expression_T$ that allows a native support for ASP, and, at the same time, permits to formally instantiated to λ-ASP-Expression [5, 26]. For the purpose of our running example, the set of λ-ASP-$Expression_T$ is available in Table 2.

In the following subsection, we illustrate the translation process based on λ-calculus. It is important to note that the choice of the lambda calculus was made because it fully matches the specifications of the formal tool we need to drive the execution of the steps in the right way.

6.3 Lambda-based translation

According to the workflow in Fig.1, the translation from plain text to the OOLOT intermediate format makes use of the information extracted in several steps. The information on the deep structure of the sentence is now used to drive the translation using the λ-calculus according to the λ- expression definitions in Table2.

Table 2. The λ-ASP-expression template

Lexicon	SemClass	$\lambda - ASP - expression\ T$
-	noun	$\lambda x.\langle noun\rangle(x)$
-	verb	$\lambda y.\langle verb\rangle(y)$
-	transVerb	$\lambda y.\lambda w.\langle verb\rangle(y, w)$
many	det	$\lambda u\lambda v.($ $v@X \leftarrow u@X,$ $not\ \neg v@X,$ $possible(v@X, u@X),$ $usual(v@X, u@X)$ $)$

For each lexicon, we use the phrase structure in Fig.2 to determine the semantic class to which it belongs. In this way, we are able to fetch the correct *lambda*-ASP-expression template from the Table 2.

For the running example, as result we have the λ-ASP-expressions of Table 3.

Table 3. The λ-ASP-expressions

Lexicon	$\lambda - ASP - expression$
apples	$\lambda x.dbpedia : Apple(x)$
eat	$\lambda y \lambda w.dbpedia : Eating(y, w)$
girls	$\lambda z.dbpedia : Girl(x)$
many	$\lambda u \lambda v.($ $v@X \leftarrow u@X,$ $not\ \neg v@X,$ $possible(v@X, u@X),$ $usual(v@X, u@X)$ $)$

Now, differently from [26], we use the dependencies, that is, we use the deep structure information, to drive the translation.

According to the dependency in Fig.5, the first relation that we use is $amod(girls-2,\ many-1)$. Thus, for the λ-calculus definition, we apply the λ-ASP-expression for $girls$ to the λ-ASP-expression for $many$:

$\lambda u \lambda v.($
$v@X \leftarrow u@X,$
$not\ \neg v@X,$
$possible(v@X, u@X),$
$usual(v@X, u@X)$
$)@@(\lambda x.(dbpedia : Girl(x))$

obtaining:

$\lambda v.($
$v@X \leftarrow dbpedia : Girl(X),$
$not\ \neg v@X,$
$possible(v@X, dbpedia : Girl(X)),$
$usual(v@X, dbpedia : Girl(X))$
$)$

The second relation, $nsubj(eat-3,\ girls-2)$, drives the application of the λ-expression for *eat* to the expression for *girls* that we obtained in the previous step:

$$\begin{aligned}&\lambda v.(\\&\quad v@X \leftarrow dbpedia:Girl(X),\\&\quad not\ \neg v@X,\\&\quad possible(v@X, dbpedia:Girl(X)),\\&\quad usual(v@X, dbpedia:Girl(X))\\&)@@(\lambda z\lambda w.(dbpedia:Eating(z,w))\end{aligned}$$

that reduces to:

$$\begin{aligned}&dbpedia:Eating(X,W) \leftarrow dbpedia:Girl(X),\\&\quad not\ \neg dbpedia:Eating(X,W),\\&\quad possible(dbpedia:Eating(X,W), dbpedia:Girl(X)),\\&\quad usual(dbpedia:Eating(X,W), dbpedia:Girl(X))\end{aligned}$$

Then, we apply *apple* to the expression we have seen, obtaining the final result:

$$\begin{aligned}&dbpedia:Eating(X, dbpedia:Apple) \leftarrow dbpedia:Girl(X),\\&\quad not\ \neg dbpedia:Eating(X, dbpedia:Apple),\\&\quad possible(dbpedia:Eating(X, dbpedia:Apple), dbpedia:Girl(X)),\\&\quad usual(dbpedia:Eating(X, dbpedia:Apple), dbpedia:Girl(X))\end{aligned}$$

7 EXPORTING INTO OWL

Our framework has been designed to export the knowledge base from OOLOT into a target formalism. For now, we are working on a pure ASP and OWL exporter.

Exporting into OWL is a very important feature, because it allows endless possibilities due to its native Semantic Web integration. In this way, the framework as a whole becomes a power tool that starting from plain text produces the RDF/OWL representation of the sentences; through ASP it takes care of special reasoning and representation features of natural language.

To complete the example, the resulting RDF representation is:

```
< http://dbpedia.org/resource/Girl,
  http://dbpedia.org/ontology/Eating,
  http://dbpedia.org/resource/Apple >
```

Fig. 6. RDF

The export into OWL has, at the present stage, some drawbacks, including loosing of some aspects of natural language that instead are perfectly managed in OOLOT. The export procedure is ongoing work, so there is room for improvement. Due to the nature of the problem, that is very complex, these aspects will be the subject of a future work.

8 CONCLUSION

In this paper, we have proposed a comprehensive framework for extracting knowledge from natural language and representing the extracted knowledge in suitable formalisms so as to be able to reason about it and to enrich existing knowledge bases. The proposed framework is being developed and an implementation is under way and will be fully available in short time. The proposed approach incorporates the best aspects and results from previous related works and, although in the early stages, it exhibits a good potential and its prospects for future development are in our opinion really interesting.

Future improvements concern many aspects of the framework. First of all, we have to choose the best parser or establish how to combine the best aspects of all them. On the OOLOT side, there is the need to better formalize the language itself, and better investigate the reasoning capabilities that it allows, and how to take the best advantage from them.

The ontology-oriented integration is at a very early stage, and there is room for substantial improvements, including a better usage of the current reference ontologies, and the evaluation study about using an upper level ontology, in order to have a more homogeneous translation.

References

1. Bos, J., Markert, K.: Recognising textual entailment with logical inference. In: HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2005) 628–635
2. Pereira, F., Shieber, S.: Prolog and natural-language analysis. Microtome Publishing (2002)
3. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. The Semantic Web (2007) 722–735
4. Kasneci, G., Ramanath, M., Suchanek, F., Weikum, G.: The YAGO-NAGA approach to knowledge discovery. SIGMOD Record **37**(4) (2008) 41–47
5. Baral, C., Dzifcak, J., Son, T.C.: Using answer set programming and lambda calculus to characterize natural language sentences with normatives and exceptions. In: Proceedings of the 23rd national conference on Artificial intelligence - Volume 2, AAAI Press (2008) 818–823
6. Kowalski, R.: Computational Logic and Human Thinking: How to be Artificially Intelligent - In Press
7. Chomsky, N.: Syntactic Structures. The MIT Press (1957)
8. Chomsky, N.: Three models for the description of language. IEEE Transactions on Information Theory **2**(3) (1956) 113–124

9. Charniak, E.: Tree-bank grammars. In: Proceedings of the National Conference on Artificial Intelligence. (1996) 1031–1036
10. Charniak, E., Johnson, M.: Coarse-to-fine n-best parsing and maxent discriminative reranking. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, Association for Computational Linguistics (2005) 173–180
11. Collins, M.: A new statistical parser based on bigram lexical dependencies. In: Proceedings of the 34th annual meeting on Association for Computational Linguistics, Association for Computational Linguistics (1996) 184–191
12. Collins, M.: Three generative, lexicalised models for statistical parsing. In: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics, Association for Computational Linguistics (1997) 16–23
13. McClosky, D., Charniak, E., Johnson, M.: Effective self-training for parsing. In: Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Association for Computational Linguistics (2006) 152–159
14. Petrov, S., Barrett, L., Thibaux, R., Klein, D.: Learning accurate, compact, and interpretable tree annotation. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, Association for Computational Linguistics (2006) 433–440
15. Petrov, S., Klein, D.: Improved inference for unlexicalized parsing. In: Proceedings of NAACL HLT 2007. (2007) 404–411
16. de Marneffe, M., Manning, C.: The stanford typed dependencies representation. In: Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, Association for Computational Linguistics (2008) 1–8
17. Tesnière, L.: Elèments de syntaxe structurale. Klincksieck, Paris (1959) ISBN 2252018615.
18. Neuhaus, P., Bröker, N.: The complexity of recognition of linguistically adequate dependency grammars. In: Proc. of ACL-97/EACL-97. (1997)
19. Klein, D., Manning, C.: Accurate unlexicalized parsing. In: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1, Association for Computational Linguistics (2003) 423–430
20. Klein, D., Manning, C.: Fast exact inference with a factored model for natural language parsing. Advances in neural information processing systems (2003) 3–10
21. De Marneffe, M., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC. Volume 6., Citeseer (2006) 449–454
22. Sleator, D., Temperley, D.: Parsing english with a link grammar. Arxiv preprint cmp-lg/9508004 (1995)
23. Cer, D., de Marneffe, M., Jurafsky, D., Manning, C.: Parsing to stanford dependencies: Trade-offs between speed and accuracy. LREC 2010 (2010)
24. : Index to link grammar documentation `http://www.link.cs.cmu.edu/link/dict/`.
25. Banerjee, S., Pedersen, T.: An adapted lesk algorithm for word sense disambiguation using wordnet. Computational Linguistics and Intelligent Text Processing (2002) 117–171
26. Costantini, S., Paolucci, A.: Towards translating natural language sentences into asp. In: Proc. of the Intl. Worksh. on Answer Set Programming and Other Computing Paradigms (ASPOCP), Edimburgh. (2010)

27. Costantini, S., Paolucci, A.: Semantically augmented DCG analysis for next-generation search engine. CILC (July 2008) (2008)
28. Eiter, T.: Answer set programming for the semantic web. Logic Programming (2010) 23–26
29. Schindlauer, R.: Answer-set programming for the Semantic Web (2006)
30. Church, A.: A set of postulates for the foundation of logic. The Annals of Mathematics **33**(2) (1932) 346–366

Logic-based reasoning support for SBVR

Dmitry Solomakhin[1)], Enrico Franconi[2)], and Alessandro Mosca[2)]

Free University of Bozen-Bolzano, Italy
Piazza Domenicani 3, 39100 Bolzano, Italy
[1)]Dmitry.Solomakhin@stud-inf.unibz.it, [2)][surname]@inf.unibz.it

Abstract. Automated support to enterprize modeling has increasingly become a subject of interest for organizations seeking solutions for storage, distribution and analysis of knowledge about business processes. This interest has recently resulted in approving the standard for specifying Semantics of Business Vocabulary and Business Rules (SBVR). Despite the existence of formally grounded notations, up to now SBVR still lacks a sound and consistent logical formalization which would allow developing automated solutions able to check the consistency of a set of business rules. This work reports on the attempt to provide logical foundations for SBVR by the means of defining a specific first-order deontic-alethic logic (FODAL). The connections of FODAL with the modal logic $\boldsymbol{QK}$ and the description logic $\mathcal{ALCQI}$ have been investigated and, on top of the obtained theoretical results, a special tool providing automated support for consistency checks of a set of $\mathcal{ALCQI}$-expressible deontic and alethic business rules has been implemented.

1 Introduction

Automated support to enterprize modeling has increasingly become a subject of interest for organizations seeking solutions for storage, distribution and analysis of knowledge about business processes. One of the most common approaches for describing business and the information used by that business is the rule-based approach [4], which was adopted by the Object Management Group (OMG) for a standard for specifying business objects and rules. The *Semantics of Business Vocabulary and Business Rules* (SBVR) [19] standard provides means for describing the structure of the meaning of rules, so called "semantic formulation", expressed in one of the intuitive notations, including the natural language that business people use [2] and Object-Role Modeling (ORM2) diagrams [8]. ORM2 has recently become widely used as conceptual modeling approach combining both formal, textual specification language and graphical modeling language [9]. It consists in identifying and articulating the rules that define the structure (alethic) and control the operation (deontic) of an enterprize [18]. The main expectation from automated solutions built upon this approach is the ability to automatically determine consistency of business rules in a business model, so that they can be further exploited to build information systems and relational databases that are coherent with the intended domain business logic.

Several attempts have been made so far in order to provide a logical formalization for structural and operational rules in SBVR and its notations. The most significant related work includes several formalizations of the purely structural fragment of ORM2, including translation to first-order predicate logic (FOL) [10] and some description logics (DL), e.g. [14] and [12]. However, none of the existing approaches enables consistency checks for a combined set of possibly interacting alethic and deontic business rules.

In this paper we define a multimodal *first-order deontic-alethic logic (FODAL)* with sound and complete axiomatization that captures the desired semantics of and interaction between business rules. We then report on the logical properties of such formalization and its connections with the modal logic $\boldsymbol{QK}$ and the description logic $\mathcal{ALCQI}$. Finally we present the tool which provides automated support for consistency checks of a set of $\mathcal{ALCQI}$-expressible deontic and alethic ORM2 constraints. Additionally, it implements the translation of aforementioned class of ORM2 constraints into an OWL2 ontology.

The rest of the paper is organized as follows. In the second section an overview of the SBVR standard and its ORM2 notation is given. Third section describes the proposed logical formalization in terms of first-order deontic-alethic logic (FODAL) along with its syntax, semantics and complete and sound axiomatization. Next two paragraphs are devoted to modeling SBVR rules with FODAL and checking their consistency with the help of this logic, while in the sixth section a connection with standard modal logic is introduced. Finally, the last paragraph describes the tool developed to provide automated support for consistency checks together with translation to OWL2.

2 SBVR Overview

A core idea of business rules formally supported by SBVR is the following [19]: "Rules build on facts, and facts build on concepts as expressed by terms. Terms express business concepts; facts make assertions about these concepts; rules constrain and support these facts". The notions of terms and facts of this "business rules mantra" correspond to SBVR *noun concepts* and *verb concepts* (or *fact types*) respectively.

Noun and verb concepts. According to the SBVR 1.0 specification [19] a *noun concept* is defined as a "concept that is the meaning of a noun or noun phrase". It has several subtypes: *object type*, *individual concept* and *fact type role*. An object type is defined as "noun concept that classifies things on the basis of their common properties", while individual concept is "a concept that corresponds to only one object [thing]". A role is a "noun concept that corresponds to things based on their playing a part, assuming a function or being used in some situation".

A verb concept (or fact type) represents the notion of relations and is defined as "a concept that is the meaning of a verb phrase". A fact type can have one (*characteristic*), two (*binary*) or more fact type roles.

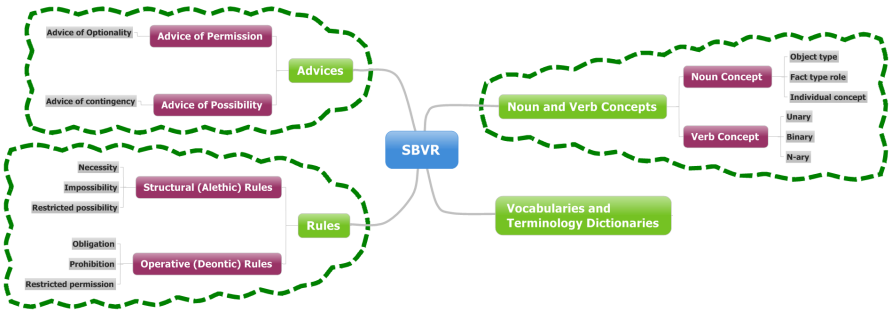


Fig. 1. SBVR overview

Business rules. The main types of rules defined in SBVR standard are *structural business rules* and *operative business rules* (See Figure 1). Structural (*definitional*) rules specify what the organization takes things to be, how do the members of the community agree on the understanding of the domain [5]. They define the characteristics of noun concepts and put constraints on verb concepts and can not be broken. Operative (*behavioral*) business rules are intended to describe the business processes in organization and can be either ignored or violated by people.

Conceptual model. An SBVR *conceptual model* $CM = \langle S, F \rangle$ is a structure intended to describe a business domain, where S is a *conceptual schema*, declaring fact types and rules relevant to the business domain, and F is a *population of facts* that conform to this schema. Business rules defined in the conceptual schema S can be considered as high-level facts (i.e., facts about propositions) and play a role of ***constraints***, which are used to impose restrictions concerning fact populations.

The SBVR standard provides means for formally expressing business facts and business rules in terms of fact types of pre-defined schema and certain logical operators, quantifiers, etc. These formal statements of rules may be transformed into logical formulations, which can in turn be used for exchange with other rules-based software tools. Such logical rule formulations are equivalent to formulae in 2-valued first-order predicate calculus with identity [19]. In addition to standard universal ($\forall$) and existential ($\exists$) quantifiers, for the sake of convenience, SBVR standard allows logical formulation to use some pre-defined [8] numeric quantifiers, such as *at-most-one* ($\exists^{0..1}$), *exactly-n* ($\exists^{n}, n \geq 1$) and others.

In order to express the structural or operational nature of a business rule, the corresponding rule formulation uses any of the basic alethic or deontic modalities. Structural rule formulations use alethic operators: $\Box$ = *it is necessary that* and $\Diamond$ = *it is possible that*; while operative rule formulations use deontic modal operators $\boldsymbol{O}$ = *it is obligatory that*, $\boldsymbol{P}$ = *it is permitted that*, as well as $\boldsymbol{F}$ = *it is forbidden that.*

Notations for business vocabulary and rules. There are several common means of expressing facts and business rules in SBVR, namely through statements, diagrams or any combination of those, each serving best for different purposes ([16], [19, Annex C, Annex L]). While graphical notations are helpful for demonstrating how concepts are related, they are usually impractical when defining vocabularies or expressing rules. We use r to denote a business rule in SBVR regardless the particular format in which it is written. For the sake of readability we will denote any necessity claim as $r_{\Box}$, possibility claim as $r_{\Diamond}$, obligation claim as r_{O} and permission claim as r_{P}.

One of the most promising notations for SBVR is *Object-Role Modeling (ORM2)*, which is a conceptual modeling approach combining both formal, textual specification language and formal graphical modeling language [9]. ORM2 specification language applies to mixfix predicates of any arity and contains predefined patterns covering a wide range of constraints typical for business domains. An example of a structural rule expressed as necessity statement in ORM2 specification language is the following:

$$r = \texttt{Each visitor } \textit{\texttt{has}} \texttt{ at most one passport.}$$

An example illustrating ORM2 graphical notation is introduced on Figure 2.

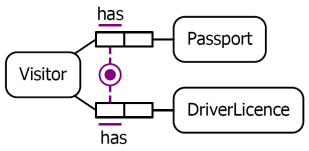


Fig. 2. Example of ORM2 diagram

The advantage of ORM2 over other notations is that it is a formal language *per se*, featuring rich expressive power, intelligibility, and semantic stability [11]. There exist several translations from non-modal ORM2 expressions to standard logics, including translation to first-order logic ([10]) and some description logics ([14], [7]). We will hereafter denote by $\phi_{\hat{r}}$ a first-order representation of a non-modal ORM2 expression[1] $\hat{r}$ from a rule r. Similarly, we will denote by $\phi_{\hat{r}}^{DL}$ a description logic representation of a non-modal ORM2 expression $\hat{r}$.

Aforementioned existing translations to standard logics may be seen as attempts to provide a logical formalization for structural and operational rules. However, since they consider only the purely structural fragment of ORM2, they are not capable of providing consistency checks for a combined set of possibly interacting alethic and deontic business rules.

[1] Since the nature of business rules implies the absence of uncertainty, it means that the resulting first-order formulae will not contain free variables, i.e. will be closed formulae. Then an SBVR rule may be represented by an expression resulted from application of modalities and boolean connectives to a set of closed FOL formulae $\phi_{\hat{r}_i}$.

3 First-order deontic-alethic logic (FODAL)

In this section we describe our attempt to provide logical foundations for SBVR by the means of defining a specific multi-modal logic. The basic formalisms we use to model business rule formulations are standard deontic logic (***SDL***) and normal modal logic ***S4***, which are both propositional modal logics. We then construct a *first-order deontic-alethic logic* (FODAL) – a multimodal logic, as a first-order extension of a combination of ***SDL*** and ***S4*** to be able to express business constraints defined in SBVR. In order to construct the first-order extension for the combined logic we follow the procedure described in [6].

3.1 Syntax

The alphabet of FODAL contains the following symbols:

- a set of *propositional connectives*: $\neg, \wedge$.
- a universal quantifier: $\forall$ (*for all*).
- an infinite set $\mathcal{P} = \{P_1^1, P_2^1, ..., P_1^2, P_2^2, ..., P_1^n, P_2^n, ...\}$ of n-place *relation symbols* (also referred to as *predicate symbols*).
- an infinite set $\mathcal{V} = \{v_1, v_2, ...\}$ of *variable* symbols.
- modal operators: alethic – $\Box$ (*necessity*) and deontic – $\boldsymbol{O}$ (*obligation*).

FODAL formulae. The *formulae* of FODAL are defined inductively in the following way:

- Every atomic formula is a formula.
- If X is a formula, so is $\neg X$.
- If X and Y are formulae, then $X \wedge Y$ is a formula.
- If X is a formula, then so are $\Box X$ and $\boldsymbol{O}X$.
- If X is a formula and v is a variable, then $\forall v X$ is a formula.

The existential quantifier ($\exists$) as well as other propositional connectives ($\vee, \rightarrow, \leftrightarrow$) are defined as usual, while additional modal operators ($\Diamond, \boldsymbol{P}, \boldsymbol{F}$) are defined in the following way:

$$\Diamond\phi \equiv \neg\Box\neg\phi \qquad \boldsymbol{P}\phi \equiv \neg\boldsymbol{O}\neg\phi \qquad \boldsymbol{F}\phi \equiv \boldsymbol{O}\neg\phi \tag{1}$$

A FODAL formula with no free variable occurrences is called a *closed formula* or a *sentence*. A *modal sentence* is a sentence whose main logical operator is a modal operator. An *atomic modal sentence* is a modal sentence which contains one and the only modal operator.

3.2 Semantics

Since SBVR itself interprets constraints in the context of possible worlds which correspond to states of the fact model (i.e. different fact populations), the choice of varying domain Kripke semantics is intuitively justified. Also, since SBVR rule formulations may includes two types of modalities: deontic and alethic, - we utilize the notion of two-layer Kripke frames with accessibility relations $R_{\boldsymbol{O}}$ and $R_{\Box}$ respectively.

Augmented frame. A varying domain augmented bimodal frame is a relational structure $\mathfrak{F}_{var} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D} \rangle$, where $\langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box} \rangle$ is a two-layer Kripke frame, $\mathcal{W}$ is a non-empty set of worlds, $R_{(\cdot)}$ are binary relations on $\mathcal{W}$ and $\mathfrak{D}$ is a domain function mapping worlds of $\mathcal{W}$ to non-empty sets. A *domain of a possible world* w is then denoted as $\mathfrak{D}(w)$ and a *frame domain* is defined as $\mathfrak{D}(\mathfrak{F}) = \bigcup\{\mathfrak{D}(w_i) | w_i \in \mathcal{W}\}$.

In order to correctly capture the behavior and interaction of the alethic and deontic modal operators it is necessary to constrain the corresponding accessibility relations: the alethic accessibility is usually taken to be a reflexive and transitive relation (***S4***) [3], while the behavior of a deontic modality is classically considered to be captured by a serial relation (***KD***) [15]. We refer to the corresponding class of bimodal frames as $\boldsymbol{S4} \otimes \boldsymbol{KD}$*-frames*.

Moreover, since one of the objectives of the formalization under development is to define the consistency of the set of business rules, it should also take into account the existing interaction between alethic and deontic modalities. The desired interaction can be verbalized as *"Everything which is necessary is also obligatory"* and then expressed as a following FODAL formula:

$$\Box X \rightarrow \boldsymbol{O} X \tag{2}$$

It can be proved that the formula 2 defines a special subclass of $\boldsymbol{S4} \otimes \boldsymbol{KD}$-frames.

Theorem 1. *The modal formula* $\Box X \rightarrow \boldsymbol{O} X$ *defines the subclass of augmented bimodal* $\boldsymbol{S4} \otimes \boldsymbol{KD}$*-frames* $\mathfrak{F} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D} \rangle$ *such that* $R_{\boldsymbol{O}} \subseteq R_{\Box}$*, where* $R_{\Box}$ *is a preorder and* $R_{\boldsymbol{O}}$ *is serial. We then call such frame a* **FODAL** *frame.*
Proof: For the complete proof please refer to [17].

Interpretation and model. An *interpretation* $\mathfrak{I}$ in a varying domain augmented frame $\mathfrak{F}_{var} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D} \rangle$ is a function which assigns to each m-place relation symbol P and to each possible world $w \in \mathcal{W}$ some m-place relation on the domain $\mathfrak{D}(w)$ of that world. $\mathfrak{I}$ can be also interpreted as a function that assigns to each possible world $w \in \mathcal{W}$ some first-order interpretation $\mathfrak{I}(w)$. A ***FODAL*** *varying domain first-order model* is a structure $\mathfrak{M} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D}, \mathfrak{I} \rangle$, where $\langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D} \rangle$ is a ***FODAL*** frame and $\mathfrak{I}$ is an interpretation in it.

Truth in a model. The satisfiability relation between ***FODAL*** models and formulae is then defined in the usual way, using the notion of valuation which maps variables to elements of the domain.
Let $\mathfrak{M} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_{\Box}, \mathfrak{D}, \mathfrak{I} \rangle$ be a ***FODAL*** model, X, Y and Φ be FODAL formulae. Then for each possible world $w \in \mathcal{W}$ and each valuation σ on $\mathfrak{D}(\mathfrak{M})$ the following holds:

- if P is a m-place relation symbol, then $\mathfrak{M}, w \models_{\sigma} P(x_1, ..., x_m)$ if and only if $(\sigma(x_1), ..., \sigma(x_m)) \in \mathfrak{I}(P, w)$ or, equivalently, $\mathfrak{I}(w) \models_{\sigma}^{FOL} P(x_1, ..., x_m)$,
- $\mathfrak{M}, w \models_{\sigma} \neg X$ if and only if $\mathfrak{M}, w \not\models_{\sigma} X$,
- $\mathfrak{M}, w \models_{\sigma} X \wedge Y$ if and only if $\mathfrak{M}, w \models_{\sigma} X$ and $\mathfrak{M}, w \models_{\sigma} Y$,
- $\mathfrak{M}, w \models_{\sigma} \forall x \Phi$ if and only if for every x-variant σ' of σ at w, $\mathfrak{M}, v \models_{\sigma} \Phi$,
- $\mathfrak{M}, w \models_{\sigma} \exists x \Phi$ if and only if for some x-variant σ' of σ at w, $\mathfrak{M}, v \models_{\sigma} \Phi$,

- $\mathfrak{M}, w \vDash_\sigma \Box X$ if and only if for every $v \in \mathcal{W}$ such that $wR_\Box v$, $\mathfrak{M}, v \vDash_\sigma X$,
- $\mathfrak{M}, w \vDash_\sigma \Diamond X$ if and only if for some $v \in \mathcal{W}$ such that $wR_\Box v$, $\mathfrak{M}, v \vDash_\sigma X$,
- $\mathfrak{M}, w \vDash_\sigma \boldsymbol{O}X$ if and only if for every $v \in \mathcal{W}$ such that $wR_{\boldsymbol{O}} v$, $\mathfrak{M}, v \vDash_\sigma X$,
- $\mathfrak{M}, w \vDash_\sigma \boldsymbol{P}X$ if and only if for some $v \in \mathcal{W}$ such that $wR_{\boldsymbol{O}} v$, $\mathfrak{M}, v \vDash_\sigma X$.

3.3 Axiomatization

A *FODAL* axiom system for first-order alethic-deontic logic is defined following the approach presented in [6] and is obtained by combining the axiom systems for the propositional modal logics $\boldsymbol{S4}$ and $\boldsymbol{KD}$ and extending the resulting combination with additional axiom schemas and the axiom 9 reflecting desired interaction between alethic and deontic modalities.

Axioms. All the formulae of the following forms are taken as axioms.

(Tautologies $\boldsymbol{S4}$)	Any FOL substitution-instance of a theorem of $\boldsymbol{S4}$	(3)
(Tautologies $\boldsymbol{KD}$)	Any FOL substitution-instance of a theorem of $\boldsymbol{KD}$	(4)
(Vacuous $\forall$)	$\forall x\phi \equiv \phi$, provided x is not free in ϕ	(5)
($\forall$ Distributivity)	$\forall x(\phi \to \psi) \to (\forall x\phi \to \forall x\psi)$	(6)
($\forall$ Permutation)	$\forall x \forall y\phi \to \forall y \forall x\phi$	(7)
($\forall$ Elimination)	$\forall y(\forall x\phi(x) \to \phi(y))$	(8)
(Necessary $\boldsymbol{O}$)	$\Box\phi \to \boldsymbol{O}\phi$	(9)

Rules of inference.

$$\textit{(Modus Ponens)}\ \frac{\phi \quad \phi \to \psi}{\psi} \qquad \textit{(Alethic Necessitation)}\ \frac{\phi}{\Box\phi} \tag{10}$$

$$\textit{(Deontic Necessitation)}\ \frac{\phi}{\boldsymbol{O}\phi} \qquad \textit{($\forall$ Generalization)}\ \frac{\phi}{\forall x\phi} \tag{11}$$

Theorem 2. *The FODAL axiom system is complete and sound with respect to the class of $\boldsymbol{FODAL}$ frames.*
Proof: For the complete proof please refer to [17].

4 Modeling SBVR vocabulary and rules with FODAL

Given an SBVR conceptual schema $\mathcal{S}$ we define the following translation $\tau(\cdot)$ from elements of $\mathcal{S}$ to notions of first-order deontic-alethic logic:

- For each noun concept A from $\mathcal{S}$, $\tau(A)$ is an unary predicate in FODAL.
- For each verb concept R from $\mathcal{S}$, $\tau(R)$ is an n-ary predicate in FODAL ($n \geq 2$).

Recall that an SBVR business rule may be represented by an expression resulted from application of modalities and boolean connectives to a set of closed first-order formulae $\phi_{\hat{r_i}}$. Then for each SBVR rule r from $\mathcal{S}$, its FODAL formalization $\tau(r)$ is defined inductively as follows:

- $\tau(\hat{r}) = \phi_{\hat{r}}$, where $\hat{r}$ is an non-modal SBVR expression and $\phi_{\hat{r}}$ is its first-order translation,
- $\tau(\neg r) = \neg\tau(r)$,
- $\tau(r_1 \circ r_2) = \tau(r_1) \circ \tau(r_2), \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, where r_1 and r_2 are rule formulations,
- $\tau(\Box\hat{r}) = \Box\tau(\hat{r})$ and $\tau(\boldsymbol{O}\hat{r}) = \boldsymbol{O}\tau(\hat{r})$.

Example 1. Assume the following set of business rules, expressed in SBVR Structured English:

(r_1) Each car rental *is insured by* exactly one credit card.

(r_2) Each luxury car rental *is* a car rental.

(r_3) It is obligatory that each luxury car rental *is insured by* at least two credit cards.

Then the corresponding FODAL formulas are the following:

$$\tau(r_1) = \forall x \exists^1 y(CarRental(x) \wedge Insured(x, y)),$$
$$\tau(r_2) = \forall x(LuxuryCarRental(x) \rightarrow CarRental(x)),$$
$$\tau(r_3) = \boldsymbol{O}(\forall x \exists^{\geq 2} y(LuxuryCarRental(x) \wedge Insured(x, y))).$$

While our *FODAL* formalization of SBVR rules provides logical mechanism supporting rule formulations with multiple occurrences of modalities, SBVR standard mostly focuses on normalized business constraints [19, p.108] that may be expressed by rule statements of the form of atomic modal sentences or by statements reducible to such a form via mechanisms provided by *FODAL* axiomatization. As a matter of fact, restricting the domain of interest only to such atomic modal rule formulations allows to obtain some useful results concerning satisfiability reduction and connection to standard logics, as shown in [17].

Hereafter we will only consider SBVR rules expressible in one of the following forms of atomic modal sentences:

$$\Box\phi \qquad \Diamond\phi \qquad \boldsymbol{O}\phi \qquad \boldsymbol{P}\phi \tag{12}$$

where ϕ is any closed wff of first-order logic.
In the case of having negation in front of the modal operator, we assume application of the standard modal negation equivalences in order to obtain the basic form of the initial rule.

FODAL regulation. A *FODAL regulation* Σ is a set of FODAL atomic modal sentences formalizing structural and operational rules of an SBVR conceptual schema S. We introduce the following designations:

$$\tau(r_{\Box}) = \Box\eta, \qquad \tau(r_{\Diamond}) = \Diamond\pi,$$
$$\tau(r_{\boldsymbol{O}}) = \boldsymbol{O}\theta, \qquad \tau(r_{\boldsymbol{P}}) = \boldsymbol{P}\rho,$$
$$\Sigma = \{\Box\eta_1, ..., \Box\eta_k, \Diamond\pi_1, ..., \Diamond\pi_l, \ \boldsymbol{O}\theta_1, ..., \boldsymbol{O}\theta_m, \boldsymbol{P}\rho_1, ..., \boldsymbol{P}\rho_n\} \tag{13}$$
$$\Sigma_{\wedge} = \bigwedge_{i=1}^{k} \Box\eta_i \wedge \bigwedge_{i=1}^{l} \Diamond\pi_i \ \wedge \bigwedge_{i=1}^{m} \boldsymbol{O}\theta_i \wedge \bigwedge_{i=1}^{n} \boldsymbol{P}\rho_i \tag{14}$$

where every $\eta_i, \pi_i, \theta_i, \rho_i$ is a closed first-order logic formula.

5 Consistency of a set of business rules

The final objective of the proposed formalization is to provide an automation solution with reasoning support for SBVR business modeling and business processes monitoring. It is well known that when reasoning about some particular universe of discourse, consistency is essential.

Assume a FODAL regulation Σ representing a set of structural and operative business rules. The *task of consistency check for* Σ is defined as procedure which analyzes the given set Σ and decides whether the rules do not contradict each other, i.e. there is no formula ψ such that $\Sigma \vdash \psi \wedge \neg\psi \equiv \bot$.
A FODAL regulation Σ is called *internally inconsistent* when the specified constraints contradict each other when the system is populated. We then define a *minimal inconsistent set* $\Sigma_\bot \subseteq \Sigma$ such that $\Sigma_\bot \vdash \bot$ and $\forall \Delta \subset \Sigma_\bot$, $\Delta \nvdash \bot$.

We distinguish several types of inconsistency depending on types of modalities of rules involved. The set Σ is called *alethic inconsistent* if it is inconsistent and the minimal inconsistent set $\Sigma_\bot$ contains formulae of only alethic nature, i.e. $\Sigma_\bot \subseteq \Sigma_\Box$. The set Σ is called *deontic inconsistent* if it is inconsistent and the minimal inconsistent set $\Sigma_\bot$ contains formulae of only deontic nature, i.e. $\Sigma_\bot \subseteq \Sigma_{\boldsymbol{O}}$. Otherwise, if $\Sigma_\bot \subseteq \Sigma_\Box \cup \Sigma_{\boldsymbol{O}}$, the set Σ is called *cross inconsistent*.

According to the completeness of the FODAL logic we have that $\Sigma \nvdash \psi$ if and only if there exists a **FODAL** model $\mathfrak{M}$ and a possible world w in it, such that $\mathfrak{M}, w \vDash \Sigma \wedge \neg\psi$. Therefore, it is sufficient to state the satisfiability of the conjunction of all formulae of the set:

$$\Sigma_\wedge = \bigwedge_{i=1}^{k} \Box\eta_i \wedge \bigwedge_{i=1}^{l} \Diamond\pi_i \wedge \bigwedge_{i=1}^{m} \boldsymbol{O}\theta_i \wedge \bigwedge_{i=1}^{n} \boldsymbol{P}\rho_i$$

Bearing in mind the fact that the regulation Σ may only contain FODAL atomic modal sentences and taking into account the properties of accessibility relations of the **FODAL** frame $\mathfrak{F}$, we can obtain the following result:

Theorem 3. *A FODAL regulation* $\Sigma_\wedge = \bigwedge_{i=1}^{k} \Box\eta_i \wedge \bigwedge_{i=1}^{l} \Diamond\pi_i \wedge \bigwedge_{i=1}^{m} \boldsymbol{O}\theta_i \wedge \bigwedge_{i=1}^{n} \boldsymbol{P}\rho_i$ *is FODAL-satisfiable if and only if each of the following formulae* $\mathcal{N}, \mathcal{O}, \mathcal{Q}_j, \mathcal{P}_j$ *is independently first-order satisfiable:*

$$\mathcal{N} = \bigwedge_{i=1}^{k} \eta_i \tag{15a}$$

$$\mathcal{O} = \bigwedge_{i=1}^{m} \theta_i \wedge \bigwedge_{i=1}^{k} \eta_i \tag{15b}$$

$$\mathcal{Q}_j = \pi_j \wedge \bigwedge_{i=1}^{k} \eta_i, \quad \forall j = \overrightarrow{1 \ldots l} \tag{15c}$$

$$\mathcal{P}_j = \rho_j \wedge \bigwedge_{i=1}^{m} \theta_i \wedge \bigwedge_{i=1}^{k} \eta_i, \quad \forall j = \overrightarrow{1 \ldots n} \tag{15d}$$

Proof: For the complete proof please refer to [17].

Observe that satisfiability of $\mathcal{N}$ follows naturally from satisfiability of any $\mathcal{Q}_j$. The same holds for $\mathcal{O}$ and $\mathcal{P}_j$ respectively. However, the satisfiability checks for 15a and 15b should be examined explicitly, since Σ may only contain necessity and obligation rules. Moreover, such definition allows to detect the actual source of unsatisfiability of the FODAL regulation Σ.

Modularity of the approach. It should be noted that the developed approach of satisfiability reduction possesses a property of *modularity*, i.e. it does not depend on the formalism behind the rule bodies η_i, θ_i, π_i and ρ_i, as long as formalism-specific satisfiability relation is provided.

6 Reduction from FODAL to monomodal logic $\boldsymbol{QK}$

As a matter of fact, the FODAL logic inherits the property of undecidability from both its component logics: standard predicate modal logics $\boldsymbol{QS4}$ and $\boldsymbol{QKD}$ are undecidable [13]. However, decidability results have been obtained for several well-studied fragments of quantified modal logics [20]. This section defines a *truth-preserving translation* of atomic modal sentences of the FODAL logic into standard predicate modal logic $\boldsymbol{QK}$, which allows to use those results.

Monomodal simulating pointed frame. Given a **FODAL** frame $\mathfrak{F} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_\Box, \mathfrak{D} \rangle$ and a possible world $w_0 \in \mathcal{W}$, a *monomodal simulating pointed frame* $\mathfrak{F}^s_{w_0}$ is defined as a tuple $\langle \mathcal{W}^s, R^s, \mathfrak{D}^s, w_0 \rangle$, such that:

- $\mathcal{W}^s$ includes w_0 and all its deontic and alethic successors: $\mathcal{W}^s = \{w_0\} \cup \{v \mid (w_0, v) \in R_{\boldsymbol{O}}\} \cup \{v \mid (w_0, v) \in R_\Box\} = |\text{since } R_{\boldsymbol{O}} \subseteq R_\Box \text{ and } R_\Box \text{ is reflexive}| = \{v \mid (w_0, v) \in R_\Box\}$.
- $R^s = \{(w_0, v) \mid (w_0, v) \in R_\Box\}$, and $\Box^s, \Diamond^s$ are modal operators associated with R^s.
- $\mathfrak{D}^s$ is a domain function on $\mathcal{W}^s$ such that $\mathfrak{D}^s(v) = \mathfrak{D}(v) \cup \{\pi^{\mathfrak{D}^s}\}, \forall v \in \mathcal{W}^s$, where $\pi^{\mathfrak{D}^s} \notin \mathfrak{D}$ is a new service domain symbol.

Since the definition of R^s does not preserve specific properties of $R_{\boldsymbol{O}}$ and $R_\Box$, the resulting frame $\mathfrak{F}^s_{w_0}$ belongs neither to serial nor to transitive nor to reflexive class of frames and therefore can be classified as a **K**-frame.

Monomodal translation. Given a FODAL regulation Σ expressed as a conjunction of FODAL atomic modal sentences 14, a *monomodal translation of regulation* $MTR(\Sigma_\wedge)$ is defined inductively as follows:

$$MTR(\phi) = \phi, \text{where } \phi \text{ is an objective FODAL formula,}$$

$$MTR(\phi_1 \wedge \phi_2) = MTR(\phi_1) \wedge MTR(\phi_2), \text{where } \phi_1 \text{ and } \phi_2 \text{ are FODAL atomic modal sentences,}$$

$$MTR(\Box\psi) = \Box^s MTR(\psi), \qquad MTR(\boldsymbol{O}\psi) = \Box^s(\neg \Pi \rightarrow MTR(\psi)),$$

$$MTR(\Diamond\psi) = \Diamond^s(MTR(\psi) \wedge \Pi), \qquad MTR(\boldsymbol{P}\psi) = \Diamond^s(MTR(\psi) \wedge \neg\Pi),$$

where ψ is a objective FODAL formula and Π is a 0-place predicate symbol, i.e. propositional letter, encapsulating the nature of the original modality of the rules of possibility and permission.

Simulated pointed model. Given a **FODAL** model $\mathfrak{M} = \langle \mathfrak{F}, \mathfrak{I} \rangle$ and a possible world $w_0 \in \mathcal{W}$, a *simulated pointed model* $\mathfrak{M}^s_{w_0}$ is defined as a tuple $\langle \mathfrak{F}^{\mathfrak{s}}_{\mathfrak{w}_\circ}, \mathfrak{I}^{\mathfrak{s}} \rangle$ such that:

- $\mathfrak{F}^s_{w_0} = \langle \mathcal{W}^s, R^s, \mathfrak{D}^s, w_0 \rangle$ is a monomodal simulating pointed frame for $\mathfrak{F} = \langle \mathcal{W}, R_{\boldsymbol{O}}, R_\Box, \mathfrak{D} \rangle$ and a possible world $w_0 \in \mathcal{W}$,
- $\mathfrak{I}^s$ is a first-order interpretation on the frame $\mathfrak{F}^s_{w_0}$ such that:
  - For each $v \in \mathcal{W}^s$ and for every n-place predicate P, $\mathfrak{I}^s(P, v) = \mathfrak{I}(P, v)$,
  - For each $v \in \mathcal{W}^s$ such that $(w_0, v) \in R_{\boldsymbol{O}}$, $\mathfrak{I}^s(\Pi, v) = \varnothing$,
  - For each $v \in \mathcal{W}^s$ such that $(w_0, v) \notin R_{\boldsymbol{O}}$, $\mathfrak{I}^s(\Pi, v) = \{\pi^{\mathfrak{D}^s}\}$.

We now state formally that the translation given above is truth-preserving with respect to varying domain semantics.

Theorem 4. *For any FODAL regulation Σ, any **FODAL** model $\mathfrak{M}$ and any possible world w_0 of a model, we have that*

$$\mathfrak{M}, w_0 \vDash \Sigma \quad \textit{if and only if} \quad \mathfrak{M}^s_{w_0}, w_0 \vDash MTR(\Sigma), \tag{16}$$

where $\mathfrak{M}^s_{w_0}$ is a simulated pointed model *for $\mathfrak{M}$ and w_0.*

Proof: For the complete proof please refer to [17].

Therefore, the truth-preserving translation MTR defined for FODAL regulations enables the transfer of decidability results from well-studied fragments of predicate modal logics ([20], [1]) to FODAL. In particular, the following fragments of FODAL logic are decidable:

- the set of atomic modal sentences with at most two variables,
- the set of *monadic* atomic modal sentences, all predicate symbols in which are at most unary,
- the set of atomic modal sentences, modal operators in which are applied to subformulas from the *guarded fragment* of first-order logic.

7 Implementation of automated reasoning support tool

7.1 General description of the tool

The ORM2 automated reasoning support tool is implemented in Java and includes a parser for ORM2 Formal Syntax [7], a set of Java classes representing the ORM2 knowledge database, a translator into an OWL2 ontology and a modal reasoning engine using HermiT or FaCT++ as an underlying reasoner. The workflow diagram of the tool is depicted on Figure 3. Currently, the tool provides the following functionality:

- Checking the consistency of a given ORM2 schema which may include both alethic (necessities and possibilities) and deontic (obligations and permissions) constraints. One of the advantages of the underlying approach is the straight-forward possibility to determine whether the inconsistency is caused purely by alethic or deontic constraints or by their combination. Additionally to the result of the consistency check, the tool prints out the list of concepts which are involved in conflicting constraints.

– Translating a given ORM2 schema into OWL2 ontology which can then be saved in various formats for further use. However, this translation does not support modalities in their diversity and, therefore, takes into account only structural constraints (i.e. alethic rules).

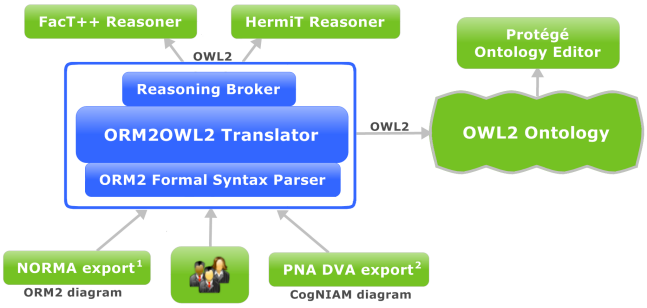


Fig. 3. Workflow Diagram

[1] *Neumont ORM Architect for Visual Studio*
[2] *PNA Group Discovery and Validation Assistant*

7.2 Logical foundations of implementation

The algorithm of the developed automated reasoning support tool relies on two fundamental results.

Firstly, it implements the procedure defined in [7] to translate a set of constraints from ORM2 Formal Syntax to $\mathcal{ALCQI}$ description logic, which is in fact a fragment of OWL2. Since the implemented ORM2 reasoning procedure involves less expressive $\mathcal{ALCQI}$ logic as underlying formalism [17], it does not support reasoning about the following information about the ORM2 conceptual schema:

– frequency constraints on multiple roles,
– generalized subset constraints on relations,
  • *still supported:* simple case of stand-alone roles,
  • *still supported:* special case of contiguous full-set of roles;
– ring constraints *(NB: drawback of mapping n-ary relations via reification).*

The same information is lost by translating a given ORM2 schema into OWL2 ontology.

Secondly, in order to check the consistency of a set of business rules expressed in $\mathcal{ALCQI}$-definable fragment of ORM2, we utilize the modularity of the approach defined in Section 5 and adapt the result of satisfiability reduction for the case of general description logic DL. The satisfiability relation for ORM2 is then provided by the semantic-preserved translation from ORM2 Formal Syntax to $\mathcal{ALCQI}$ [7].

Theorem 5. *A FODAL regulation* $\Sigma = \{\Box\eta_1, ..., \Box\eta_k, \Diamond\pi_1, ..., \Diamond\pi_l, \boldsymbol{O}\theta_1, ..., \boldsymbol{O}\theta_m, \boldsymbol{P}\rho_1, ..., \boldsymbol{P}\rho_n\}$*, expressed in DL-definable fragment of* ORM2*, is internally consistent if and only if each of the following description logic formulae* $\mathcal{N}^{DL}$*,* $\mathcal{O}^{DL}$*,* $\mathcal{Q}_j^{DL}$*,* $\mathcal{P}_j^{DL}$ *is independently satisfiable in DL:*

$$\mathcal{N}^{DL} = \bigsqcap_{i=1}^{k} \eta_i \qquad \mathcal{O}^{DL} = \bigsqcap_{i=1}^{m} \theta_i \sqcap \bigsqcap_{i=1}^{k} \eta_i \tag{17}$$

$$\mathcal{Q}_j^{DL} = \pi_j \sqcap \bigsqcap_{i=1}^{k} \eta_i, \qquad \mathcal{P}_j^{DL} = \rho_j \sqcap \bigsqcap_{i=1}^{m} \theta_i \sqcap \bigsqcap_{i=1}^{k} \eta_i, \quad \forall j = \overrightarrow{1 \ldots n}$$

Thus, we can reduce the consistency of a given set of constraints to $\mathcal{ALCQI}$ satisfiability, which in turn can be interpreted as unsatisfiable concepts' check in resulting OWL2 ontology. Indeed, whenever a formula in $\mathcal{ALCQI}$ is unsatisfiable, it means that the concept definition expressed by this formula contains a contradiction which prevents the concept from having a model, i.e. the concept is forced to *not* have any instances, hence is unsatisfiable.

7.3 Usage of the tool

In the following section we will demonstrate the functionality of the implemented tool by means of a real-life example of its usage. The graphical user interface of a tool is introduced on Figure 5 and contains controls which allow to select an input file, underlying reasoner, output file and output format for resulting ontology (if needed). The consistency check for an input ORM2 schema is performed after loading the input file and the result of the check is communicated by visual flag as well as by a detailed log in the corresponding window.

Checking the consistency of a given ORM2 schema. In order to illustrate the functionality of the consistency check we will consider the ORM2 schema obtained by merging two conceptual models (e.g. A and B) and depicted on the Figure 4. This schema contains the following set of conflicting business rules:

(R_1^A) Each car rental *is insured by* exactly one credit card.

(R_1^B) Each luxury car rental *is* a car rental.

(R_2^B) It is obligatory that each luxury car rental *is insured by* at least two credit cards.

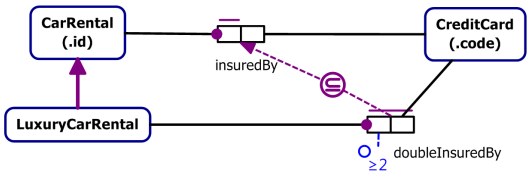


Fig. 4. Inconsistent ORM2 Schema

The given set of constraints can be fully expressed in $\mathcal{ALCQI}$ description logic, therefore we can use the developed reasoning support tool for consistency check. The schema on Figure 4 is internally inconsistent with respect to obligation (R_2^B) since the latter clearly contradicts the structural constraint (R_1^A)

which, together with *is-a* constraint on luxury car rental, simply does not support more than one credit card. Therefore, for any luxury car rental the obligatory cardinality constraint cannot be satisfied. The same conclusion is indeed inferred by the implemented tool on Figure 5.

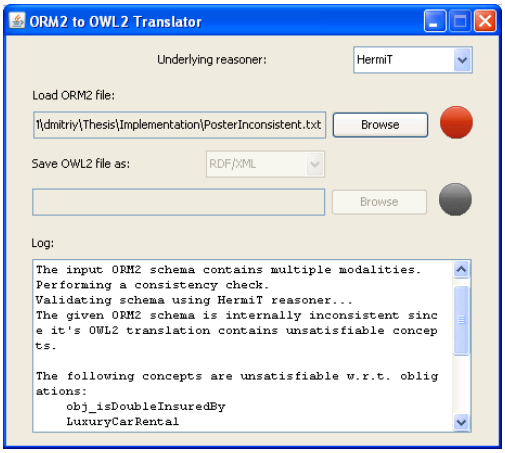


Fig. 5. The Graphical User Interface

8 Conclusion

In this paper we introduced a logical formalization of the Semantics of Business Vocabulary and Rules standard (SBVR) by defining a first-order deontic-alethic logic (FODAL) with its syntax, semantics and complete and sound axiomatization, that captures the semantics of and interaction between business rules.

We also showed that satisfiability in FODAL logic may be reduced to a standard first-order satisfiability for a class of formulas restricted to atomic modal sentences. Moreover, in order to establish a relationship with a standard logical formalism, we defined a truth-preserving translation from a fragment of bimodal FODAL into quantified monomodal logic $\boldsymbol{QK}$, that can be used to facilitate the transfer of decidability results from well-studied fragments of predicate modal logics to FODAL.

Finally we presented the ORM2 reasoning tool which provides an automated support for consistency checks of the conceptual model along with its translation to OWL2 ontology. The main functionality of the tool is a consistency check of a set of $\mathcal{ALCQI}$-expressible deontic and alethic business rules. Another important task supported by the tool is translation of the aforementioned fragment of an ORM2 schema into an OWL2 ontology, which, however, does not support any modalities except necessity due to lack of notions representing deontic constraints in OWL2.

The future research in the field of logical formalization of SBVR aims at studying the problem of entailment with respect to possible interaction of alethic and deontic modalities. Another future course of work includes defining an approach to translate an ORM2 schema with its alethic and deontic rules to SWRL or some other extension of OWL2.

References

1. Hajnal Andréka, Johan Van Benthem, and István Németi. Modal languages and bounded fragments of predicate logic, 1996.
2. Donald E. Baisley, John Hall, and Donald Chapin. Semantic formulations in SBVR. In *Rule Languages for Interoperability*. W3C, 2005.
3. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Number 53 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, UK, 2001.
4. Paolo Ceravolo, Cristiano Fugazza, and Marcello Leida. Modeling semantics of business rules. In *Proceedings of the Inaugural IEEE International Conference On Digital Ecosystems and Technologies (IEEE-DEST)*, February 2007.
5. Donald Chapin. SBVR: What is now possible and why? *Business Rules Journal*, 9(3), 2008.
6. Melvin Fitting and Richard L. Mendelsohn. *First-order modal logic*. Kluwer Academic Publishers, Norwell, MA, USA, 1999.
7. Enrico Franconi, Alessandro Mosca, and Dmitry Solomakhin. ORM2: Syntax and semantics. Internal report, KRDB Research Centre for Knowledge and Data, 2011.
8. Terry Halpin. *A Logical Analysis of Information Systems: Static Aspects of the Data-oriented Perspective*. PhD thesis, Department of Computer Science, University of Queensland, 1989.
9. Terry Halpin. Object-Role Modeling (ORM/NIAM). In *Handbook on Architectures of Information Systems*, pages 81–102. Springer-Verlag, 1998.
10. Terry Halpin and Tony Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2008.
11. Terry A. Halpin and Jan Pieter Wijbenga. FORML 2. In Ilia Bider, Terry A. Halpin, John Krogstie, Selmin Nurcan, Erik Proper, Rainer Schmidt, and Roland Ukor, editors, *BMMDS/EMMSAD*, volume 50 of *Lecture Notes in Business Information Processing*, pages 247–260. Springer, 2010.
12. Rami Hodrob and Mustafa Jarrar. Mapping ORM into OWL 2. In *Proceedings of the 1st International Conference on Intelligent Semantic Web-Services and Applications*, ISWSA '10, pages 9:1–9:7, New York, NY, USA, 2010. ACM.
13. G. E. Hughes and M. J. Cresswell. *A New Introduction To Modal Logic*. Routledge, 1996.
14. C. Maria Keet. Mapping the Object-Role Modeling language ORM2 into Description Logic language DLRifd. *CoRR*, abs/cs/0702089, 2007.
15. Paul McNamara. Deontic logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Fall 2010 edition, 2010.
16. Ronald G. Ross. *Principles of the Business Rule Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
17. Dmitry Solomakhin. *Logical Formalization of Semantic Business Vocabulary and Rules*. MSc thesis, Faculty of Informatics, Vienna University of Technology, `http://media.obvsg.at/AC07810206-2001`, 2011.
18. The Business Rules Group. Defining business rules. What are they really? Technical report, The Business Rules Group, 2001.
19. The Object Management Group. Semantics of Business Vocabulary and Business Rules (SBVR). Formal specification, v1.0, 2008.
20. Frank Wolter and Michael Zakharyaschev. Decidable fragments of first-order modal logics. *J. Symb. Log.*, 66(3):1415–1438, 2001.

Winning CaRet Games with Modular Strategies*

Ilaria De Crescenzo and Salvatore La Torre

Dipartimento di Informatica
Università degli Studi di Salerno

Abstract. Recursive state machines are a well-accepted formalism for modelling the control flow of systems with potentially recursive procedure calls. In the open systems setting, i.e., systems where an execution depends on the interaction of the system with the environment, the natural counterpart is two-player recursive game graphs which essentially are recursive state machines where vertices are split into two sets each controlled by one of the players.
We focus on solving games played on such graphs with respect to winning conditions expressed by a formula of the temporal logic CaRet and such that the protagonist can only use modular strategies (modular CaRet games). In a modular strategy, the protagonist may use as memory only the portion of the play which is local to the current activation of the current module. Therefore, every time a module is entered, the memory used by the protagonist gets reset.
The main motivation for considering modular strategies is related to the synthesis of system controllers. In fact, a modular strategy would correspond to a modular controller for the considered system. Modular strategies have been already studied with winning conditions expressed as reachability, safety, Büchi automata or LTL formulas. In this paper we extend these results to non-regular winning conditions by considering specifications expressed in CaRet. In particular, we show that deciding whether the protagonist has a winning modular strategy in a CaRet game is 2ExpTime-complete, that matches the complexity of deciding modular LTL games.

1 Introduction

The interest for games naturally arises in many contexts. In the formal analysis of systems, games are closely related to the *controller synthesis* problem and to the verification of *open systems*, and are useful tools for solving decision problems such as, for example, the model-checking of the μ-calculus formulas (see [6, 9]).

In controller synthesis, given a description of the system where some of the choices depend upon the input and some represent uncontrollable internal nondeterminism, the goal is to design a *controller* that supplies inputs to the system

* This work was partially funded by the MIUR grants FARB 2009-2010 Università degli Studi di Salerno (Italy).

so that the product of the controller and the system satisfies the correctness specification, that clearly corresponds to computing winning strategies in two-player games. See [10] for a survey.

In the open systems setting, for instance, the *Alternating Temporal Logic* allows specification of requirements such as "module A can ensure delivery of the message no matter how module B behaves" [2]; *module checking* deals with the problem of checking whether a module behaves correctly no matter in which environment it is placed [8].

In this paper, we focus on pushdown systems. Pushdown systems accurately model the control flow in programs of sequential imperative programming languages with recursive procedure calls. A large number of hardware and software systems can be captured by this model, such as programs of object oriented languages, systems with distributed architectures and communication protocols. The study of games on such systems has traditionally focused on determining the existence of a winning strategy, that is a mapping that specifies for each play ending into a controlled state the next move such that each resulting play satisfies the winning conditions [5, 12].

Here, we consider *modular strategies* [4], that are strategies where the next move is determined only looking at the local memory of the current activation of the current module. It is known that modular reachability games are NP-complete [4]. Also, modular strategies have been considered for winning conditions expressed as an ω-regular language, using Büchi, Co-Büchi automata or linear temporal logic formulas, and in particular, modular LTL games are known to be 2ExpTime-complete [3].

We extend the results on modular strategies to a more general class of winning conditions. We allow winning conditions expressed as formulas of the temporal logic CaRet [1]. The logic CaRet combines the temporal modalities of LTL with different kinds of successor (*global, abstract* and *caller*) and can express both regular requirements and a variety of non-regular properties such as partial and total correctness of program blocks or inspection of the stack. By using an automaton theoretic approach, we show that solving the modular CaRet games is decidible within double exponential time. Since modular LTL games are already 2ExpTime-hard and CaRet sintactically includes LTL, we get that also modular CaRet games are 2ExpTime-complete.

2 Preliminary

Recursive game graph. A recursive game graph (RGG) is composed of *game modules* that are essentially two-player graphs (i.e., graphs whose vertices are partitioned into two sets depending on the player who controls the outgoing moves) with *entry* and *exit* nodes and two different kinds of vertices: the nodes and the boxes. A *node* is a standard graph vertex and a *box* corresponds to invocations of other game modules in a potentially recursive manner (in particular, entering into a box corresponds to a module *call* and exiting from a box corresponds to a *return* from a module). Each RGG has a distinct game module

which is called the *start* module. A *state* of an RGG is composed by a call stack and a node. The notion of *run* can be defined analogously to the computation of a standard procedural program (the modules corresponding to the procedures). A *play* of an RGG is a run starting from an entry node of the start module.

For a formal definition of an RGG we refer the reader to [4].

Strategies. Given a player P, a *strategy* is a function that associates a move to every run that ends in a node controlled by P. A *modular strategy* consists of a set of local strategies, one for each game module, that are used together as a global strategy for a player . A local strategy for a module can only refer to the local memory of the module, i.e. the sequence of vertices that correspond to internal nodes, call or returns of the module. This sequence corresponds to the portion of the play concerning to the current invocation of the module.

For detailed comments and formal definitions on recursive game graphs, strategies and modular strategies we refer the reader to [4].

Syntax	$\varphi := p \mid \varphi \vee \varphi \mid \neg\ \varphi \mid \ \bigcirc^g \varphi \mid \ \bigcirc^a \varphi \mid \ \bigcirc^- \varphi \mid \varphi\ \mathcal{U}^g \varphi \mid \varphi\ \mathcal{U}^a \varphi \mid \varphi\ \mathcal{U}^- \varphi$ (where $p \in AP \cup$ {call, ret, int})
Semantics	for a word $\alpha = \alpha_1, \alpha_2, \alpha_3, ..., \alpha_n, ... \in \hat{\Sigma}^\omega$ and n $\in \mathbb{N}$: – $(\alpha, n) \models p$ iff $\alpha_n = (X, d)$ and $p \in X$ or $p = d$ – $(\alpha, n) \models \varphi_1 \vee \varphi_2$ iff $(\alpha, n) \models \varphi_1$ or $(\alpha, n) \models \varphi_2$ – $(\alpha, n) \models \neg\varphi$ iff $(\alpha, n) \not\models \varphi$ – $(\alpha, n) \models \bigcirc^g \varphi$ iff $(\alpha, succ^g_\alpha(n)) \models \varphi$, i.e., iff $(\alpha, n+1) \models \varphi$ – $(\alpha, n) \models \bigcirc^a \varphi$ iff $succ^a_\alpha(n)) \neq \perp$ and $(\alpha, succ^a_\alpha(n)) \models \varphi$ – $(\alpha, n) \models \bigcirc^- \varphi$ iff $succ^-_\alpha(n)) \neq \perp$ and $(\alpha, succ^-_\alpha(n)) \models \varphi$ – $(\alpha, n) \models \varphi_1\ \mathcal{U}^b \varphi_2$ (for any $b \in \{g, a, -\}$) iff there is a sequence of position $i_0, i_1, ..., i_k$, where $i_0 = n, (\alpha, i_k) \models \varphi_2$ and for every $0 \leq j \leq k-1, i_j + 1 = succ^b_\alpha(i_j))$ and $(\alpha, i_j) \models \varphi_1$

Fig. 1. Syntax and semantics of CaRet.

CaRet. Let $\Sigma = 2^{AP}$ where AP is a finite set of atomic propositions. We consider the augmented alphabet of Σ that is $\hat{\Sigma} = \Sigma \times \{call, ret, int\}$.

The syntax and the semantics of CaRet are reported in Fig. 1. We refer the reader to [1] for a detailed definition of CaRet.

In this logic, three different notions of successor are used:

- the global-successor ($succ^g$) which is the usual successor function. It points to next node, whatever module it belongs;
- the abstract-successor ($succ^a$) which, for internal moves, corresponds to the global successor and for calls corresponds to the matching returns;
- the caller successor ($succ^-$) which is a "past" modality that points to the innermost unmatched call.

Typical properties that can be expressed by the logic CaRet are pre and post conditions. An example is the formula $\Box[(call \wedge p \wedge p_A) \rightarrow \bigcirc^a q]$. If we assume that all calls to procedure A are characterized by the proposition p_A, the formula expresses that if the pre-condition p holds when the procedure A is invoked, then the procedure terminates and the post-condition q is satisfied

upon the return. This is the requirement of total correctness. Observe the use of the abstract next operator to refer to the return associated with the call.

Modular CaRet *games.* A modular CaRet game is a pair $\langle G, \varphi \rangle$ where G is a RGG whose vertices (nodes, calls and returns) are labeled with a set of propositions and φ is a CaRet formula. Given a modular CaRet game $\langle G, \varphi \rangle$ we want solve the problem of deciding whether there exists a modular strategy such that the resulting plays are guaranteed to satisfy φ.

Detailed comments and formal definitions for modular CaRet games can be found in [13].

3 Solving modular CaRet games

In this section, we briefly sketch our solution to CaRet games. For the omitted details, we refer the interested reader to [13].

Our solution consists of three main steps.

Let $\langle G, \varphi \rangle$ be a modular CaRet game.

The first step consists of constructing from $\langle G, \varphi \rangle$ an equivalent game $\langle G', color \rangle$ with parity winning conditions such that $|G'| = O(2^{|\varphi|})$. We build on the top of the construction given in [1] for model checking CaRet formulas. The main differences are that we apply the construction to the negation of the formula φ instead of to the formula φ directly, and introduce fresh nodes to ensure the correct semantics of the interaction of the two players. Negating the formula is needed to use correctly the construction from [1]. We recall that this construction, such as all the constructions which are tableau based, are nondeterministic and therefore cannot be directly combined with a game graph in a cross product. Starting from the negated formula, we can apply the same tableau construction but now interpreting the nondeterminism as universality, and thus dualizing also the winning conditions we obtain a game graph which is equivalent to the starting one with respect to the considered decision problem. The resulting winning condition is the conjunction of a Büchi condition with a generalized co-Büchi one, that can be simplified to a single pair Rabin condition and thus to an equivalent parity condition.

Also, notice that both G and φ can define context-free languages, and problems such as inclusion and emptiness of intersection are undecidable for context-free languages [7]. Therefore, translating φ into an automaton and then intersecting it with the recursive game graph does not seem feasible.

The second step consists of constructing from the parity game $\langle G', color \rangle$ a two-way alternating parity tree automaton A_{win} similarly to what is done in [3]. The resulting automaton accepts a strategy tree iff it corresponds to a winning modular strategy.

For the last step of our solution, we observe that by using [11] we can convert A_{win} to a one-way nondeterministic tree automaton and take its intersection with the A_{strat} that is a tree automaton accepting strategy trees. Thus, we get a one-way nondeterministic automaton A' which accepts a tree iff it corresponds

to a winning strategy tree. Checking the emptiness of this automaton takes polynomial time in the number of states and exponential in the number of colors in the parity condition [10].

Since the constructed recursive game graph G' is doubly exponential in the formula φ and linear in the recursive game graph G, *color* is constant and LTL games are already 2ExpTime-hard, we get:

Theorem 1. *Deciding modular* CaRet *games is* 2ExpTime*-complete.*

References

1. R. Alur, K. Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Proc. of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'04*, LNCS 2988, pages 467–481. Springer, 2004.
2. R. Alur, T. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM*, 49(5):1–42, 2002.
3. R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for infinite games on recursive graphs. In *Proc. of the 15th International Conference on Computer Aided Verification, CAV'03*, LNCS 2725, pages 67-79. Springer, 2003.
4. R. Alur, S. La Torre, and P. Madhusudan. Modular strategies for recursive game graphs. In *Proc. 9th Intern. Conf. on Tools and Algorithms for the Construction and the Analysis of Systems, TACAS'03*, LNCS 2619, pages 363–378. Springer, 2003.
5. T. Cachat. Symbolic strategy synthesis for games on pushdown graphs. In *Automata, Languages and Programming, 29th Int'l Coll., ICALP, Malaga, Spain, July 8-13, 2002, Proceedings*, LNCS 2380, pages 704–715. Springer.
6. E. A. Emerson. Model checking and the mu-calculus. In N. Immerman and P. Kolaitis, editors, *Proceedings of the DIMACS Symposium on Descriptive Complexity and Finite Models*, pages 185–214. American Mathematical Society Press, 1997.
7. J. E. Hopcroft and J. D. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, 1979.
8. O. Kupferman, M. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164(2):322–344, 2001.
9. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
10. W. Thomas. Infinite games and verification. In *Proceedings of the International Conference on Computer Aided Verification CAV'02*, LNCS 2404, pages 58–64. Springer, 2002.
11. M. Vardi. Reasoning about the past with two-way automata. In *Proc. 14th Intern. Coll. on Automata, Languages, and Programming, ICALP'98*, LNCS 1443 , pages 628–641. Springer, 1998.
12. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, January 2001.
13. Giochi su Grafi Pushdown rispetto a Strategie Modulari. *http://www.dia.unisa.it/professori/latorre/ilaDec/tesi2011.pdf*, 2011

A Note on the Approximation of Mean-Payoff Games

Raffaella Gentilini[1]

[1]University of Perugia, Italy

Abstract. We consider the problem of designing approximation schemes for the values of mean-payoff games. It was recently shown that (1) mean-payoff with rational weights scaled on $[-1, 1]$ admit additive fully-polynomial approximation schemes, and (2) mean-payoff games with positive weights admit relative fully-polynomial approximation schemes. We show that the problem of designing additive/relative approximation schemes for general mean-payoff games (i.e. with no constraint on their edge-weights) is P-time equivalent to determining their exact solution.

1 Introduction

Two-player mean-payoff games are played on weighted graphs[1] with two types of vertices: in player-0 vertices, player 0 chooses the successor vertex from the set of outgoing edges; in player-1 vertices, player 1 chooses the successor vertex from the set of outgoing edges. The game results in an infinite path through the graph. The long-run average of the edge-weights along this path, called the *value* of the play, is won by player 0 and lost by player 1.

The *decision problem* for mean-payoff games asks, given a vertex v and a threshold $\nu \in \mathbb{Q}$, if player 0 has a strategy to win a value at least ν when the game starts in v. The *value problem* consists in computing the maximal (rational) value that player 0 can achieve from each vertex v of the game. The associated *(optimal) strategy synthesis problem* is to construct a strategy for player 0 that secures the maximal value.

Mean-payoff games have been first studied by Ehrenfeucht and Mycielski in [1], where it is shown that memoryless (or positional) strategies suffice to achieve the optimal value. This result entails that the decision problem for these games lies in NP $\cap$ coNP [2, 18], and it was later shown to belong to[2] UP $\cap$ coUP [10]. Despite many efforts [19, 18, 13, 5, 6, 20, 9, 12], no polynomial-time algorithm for the mean-payoff game problems is known so far.

Beside such a theoretically engaging complexity status, mean-payoff games have plenty of applications, especially in the synthesis, analysis and verification of reactive (non-terminating) systems. Many natural models of such systems include quantitative information, and the corresponding question requires the solution of quantitative games, like mean-payoff games. Concrete examples of applications include various kinds of scheduling, finite-window online string matching, or more generally, analysis of online problems and algorithms, as well as selection with limited storage [18]. Mean-payoff games can even be used for solving the max-plus algebra $Ax = Bx$ problem, which in

[1] in which every edge has a positive/negative (rational) weight

[2] The complexity class UP is the class of problems recognizable by unambiguous polynomial time nondeterministic Turing machines [14]. Obviously P$\subseteq$ UP $\cap$ coUP $\subseteq$ NP $\cap$ coNP.

	Problems		
Algorithms	***Decision Problem***	***Value Problem***	**Note**
[12]	$\mathcal{O}(\|E\| \cdot \|V\| \cdot W)$	$\mathcal{O}(\|E\| \cdot \|V\|^2 \cdot W \cdot (log\|V\| + logW))$	*Deterministic*
[18]	$\Theta(\|E\| \cdot \|V\|^2 \cdot W)$	$\Theta(\|E\| \cdot \|V\|^3 \cdot W)$	*Deterministic*
[20]	$\mathcal{O}(\|E\| \cdot \|V\| \cdot 2^{\|V\|})$	$\mathcal{O}(\|E\| \cdot \|V\| \cdot 2^{\|V\|} \cdot logW)$	*Deterministic*
[9]	$min(\mathcal{O}(\|E\| \cdot \|V\|^2 \cdot W), 2^{\mathcal{O}(\sqrt{\|V\| \cdot log\|V\|})} \cdot logW)$	$min(\mathcal{O}(\|E\| \cdot \|V\|^3 \cdot W \cdot (logV + logW)), 2^{\mathcal{O}(\sqrt{\|V\| \cdot log\|V\|})} \cdot logW)$	*Randomized*

Table 1. Complexity of the main algorithms to solve mean-payoff games.

turn has further applications [6]. Beside their applicability to the modeling of quantitative problems, mean-payoff games have tight connections with important problems in game theory and logic. For instance, parity games [8] and the model-checking problem for the modal mu-calculus [11] are poly-time reducible to mean-payoff games [7], and it is a long-standing open question to know whether these problems are in P.

Table 1 summarize the complexity of the main algorithms for solving mean-payoff games in the literature. In particular, all *deterministic* algorithms for mean-payoff games are either pseudopolynomial (i.e., polynomial in the number of vertices $|V|$, the number of edges $|E|$, and the maximal absolute weight W, rather than in the binary representation of W) or exponential [19, 18, 13, 12, 20, 17]. The works in [9, 3] define a *randomized* algorithm which is both subexponential and pseudopolynomial. Recently, the authors of [15, 4] show that the pseudopolynomial procedures in [18, 13, 12] can be used to design (fully) polynomial value approximation schemes for certain classes of mean-payoff games: namely, mean-payoff games with positive (integer) weights or rational weights with absolute value less or equal to 1. In this paper, we consider the problem of extending such positive approximation results for general mean-payoff games, i.e. mean-payoff games with weights arbitrary shifted/scaled on the line of rational numbers.

2 Preliminaries and Definitions

Game graphs A *game graph* is a tuple $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ where $G^\Gamma = (V, E, w)$ is a weighted graph and $\langle V_0, V_1 \rangle$ is a partition of V into the set V_0 of player-0 vertices and the set V_1 of player-1 vertices. An *infinite game* on Γ is played for infinitely many rounds by two players moving a pebble along the edges of the weighted graph G^Γ. In the first round, the pebble is on some vertex $v \in V$. In each round, if the pebble is on a vertex $v \in V_i$ $(i = 0, 1)$, then player i chooses an edge $(v, v') \in E$ and the next round starts with the pebble on v'. A *play* in the game graph Γ is an infinite sequence $p = v_0 v_1 \dots v_n \dots$ such that $(v_i, v_{i+1}) \in E$ for all $i \geq 0$. A *strategy* for player i $(i = 0, 1)$ is a function $\sigma : V^* \cdot V_i \to V$, such that for all finite paths $v_0 v_1 \dots v_n$ with $v_n \in V_i$, we have $(v_n, \sigma(v_0 v_1 \dots v_n)) \in E$. A *strategy-profile* is a pair of strategies $\langle \sigma_0, \sigma_1 \rangle$, where σ_0 (resp. σ_1) is a strategy for player 0 (resp. player 1). We denote by Σ_i $(i = 0, 1)$ the set of strategies for player i. A strategy σ for player i is *memoryless*

if $\sigma(p) = \sigma(p')$ for all sequences $p = v_0 v_1 \dots v_n$ and $p' = v'_0 v'_1 \dots v'_m$ such that $v_n = v'_m$. We denote by Σ_i^M the set of memoryless strategies of player i. A play $v_0 v_1 \dots v_n \dots$ is *consistent* with a strategy σ for player i if $v_{j+1} = \sigma(v_0 v_1 \dots v_j)$ for all positions $j \geq 0$ such that $v_j \in V_i$. Given an initial vertex $v \in V$, the *outcome* of the *strategy profile* $\langle \sigma_0, \sigma_1 \rangle$ in v is the (unique) play $\mathsf{outcome}^\Gamma(v, \sigma_0, \sigma_1)$ that starts in v and is consistent with both σ_0 and σ_1. Given a memoryless strategy π_i for player i in the game Γ, we denote by $G^\Gamma(\pi_i) = (V, E_{\pi_i}, w)$ the weighted graph obtained by removing from G^Γ all edges (v, v') such that $v \in V_i$ and $v' \neq \pi_i(v)$.

Mean-Payoff Games A *mean-payoff game* (MPG) [1] is an infinite game played on a game graph Γ where player 0 wins a payoff value defined as the long-run average weights of the play, while player 1 loses that value. Formally, the payoff value of a play $v_0 v_1 \dots v_n \dots$ in Γ is

$$\mathsf{MP}(v_0 v_1 \dots v_n \dots) = \liminf_{n \to \infty} \frac{1}{n} \cdot \sum_{i=0}^{n-1} w(v_i, v_{i+1}).$$

The value *secured* by a strategy $\sigma_0 \in \Sigma_0$ in a vertex v is

$$\mathsf{val}^{\sigma_0}(v) = \inf_{\sigma_1 \in \Sigma_1} \mathsf{MP}(\mathsf{outcome}^\Gamma(v, \sigma_0, \sigma_1))$$

and the *(optimal) value* of a vertex v in a mean-payoff game Γ is

$$\mathsf{val}^\Gamma(v) = \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \mathsf{MP}(\mathsf{outcome}^\Gamma(v, \sigma_0, \sigma_1)).$$

We say that σ_0 is *optimal* if $\mathsf{val}^{\sigma_0}(v) = \mathsf{val}^\Gamma(v)$ for all $v \in V$. Secured value and optimality are defined analogously for strategies of player 1. Ehrenfeucht and Mycielski [1] show that mean-payoff games are *memoryless determined*, i.e., memoryless strategies are sufficient for optimality and the optimal (maximum) value that player 0 can secure is equal to the optimal (minimum) value that player 1 can achieve.

Theorem 1 ([1]). *For all* MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ *and for all vertices* $v \in V$*, we have*

$$\mathsf{val}^\Gamma(v) = \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \mathsf{MP}(\mathsf{outcome}^\Gamma(v, \sigma_0, \sigma_1)) = \inf_{\sigma_1 \in \Sigma_1} \sup_{\sigma_0 \in \Sigma_0} \mathsf{MP}(\mathsf{outcome}^\Gamma(v, \sigma_0, \sigma_1)),$$

and there exist two memoryless strategies $\pi_0 \in \Sigma_0^M$ *and* $\pi_1 \in \Sigma_1^M$ *such that*

$$\mathsf{val}^\Gamma(v) = \mathsf{val}^{\pi_0}(v) = \mathsf{val}^{\pi_1}(v).$$

Moreover, *uniform* optimal strategies exist for both players, i.e. there exists a strategy profile $\langle \sigma_0, \sigma_1 \rangle$ that can be used to secure the optimal value independently of the initial vertex [1]. Such a strategy profile is said the *optimal strategy profile*.

The following lemma characterizes the shape of MPG values in a MPG $\Gamma = (V, E, w, \langle V_0, V_1 \rangle)$ with integer weights in $\{-W, \dots, W\}$. Note that solving MPG with rational weights is P-time reducible to solving MPG with integer weights [20, 18].

Lemma 1 ([1, 20]). *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be a* MPG *with integer weights and let $W = \max_{(v,v')\in E}|w(v, v')|$. For each vertex $v \in V$, the optimal value $\mathsf{val}^\Gamma(v)$ is a rational number $\frac{n}{d}$ such that $1 \leq d \leq |V|$ and $|n| \leq d \cdot W$.*

We consider the following three classical problems [18, 9] for a MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$:

1. *Decision Problem*. Given a threshold $\nu \in \mathbb{Q}$ and a vertex $v \in V$, decide if $\mathsf{val}^\Gamma(v) \geq \nu$.
2. *Value Problem*. Compute for each vertex $v \in V$ the value $\mathsf{val}^\Gamma(v)$.
3. *(Optimal) Strategy Problem* . Given an MPG Γ, compute an (optimal) strategy profile for Γ.

Approximate Solutions for MPG

Dealing with approximate MPG solutions, we can take into consideration either absolute or relative error measures, and define the notions of *additive* and *relative* MPG *approximate value*.

Definition 1 (MPG additive ε-value). *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be a* MPG*, let $v \in V$ and consider $\varepsilon \geq 0$. The value $\widetilde{val} \in \mathbb{Q}$ is said an* additive ε-value *on v if and only if:*

$$|\widetilde{val} - val^\Gamma(v)| \leq \varepsilon$$

Definition 2 (MPG relative ε-value). *Let $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ be a* MPG*, let $v \in V$ and consider $\varepsilon \geq 0$. The value $\widetilde{val} \in \mathbb{Q}$ is said a* relative ε-value *on v if and only if:*

$$\frac{|\widetilde{val} - val^\Gamma(v)|}{|val^\Gamma(v)|} \leq \varepsilon$$

Note that additive MPG ε-values are shift-invariant. More precisely, if $\widetilde{val}$ is an additive approximate ε-value on the vertex v in $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, then $\widetilde{val} + k$ is an additive approximate ε-value in the MPG $\Gamma' = (V, E, w + k, \langle V_0, V_1\rangle)$, where all the weights are shifted by k. On the contrary, additive MPG ε-values are not scale-invariant. In fact, if $\widetilde{val}$ is a relative ε-value for v in the MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, then $k \cdot \widetilde{val}$ is a relative $\varepsilon \cdot k$-value for v in the MPG $\Gamma' = (V, E, w \cdot k, \langle V_0, V_1\rangle)$, where all the weights are multiplied by k. In other words, the additive error on Γ is amplified by a factor k in the scaled version of the game, Γ'. Conversely, relative MPG ε-values are scale invariant but not shift invariant.

The notions of (fully) polynomial approximation schemes w.r.t relative and additive errors are formally defined below.

Definition 3 (MPG Fully Polynomial Time Approximation Scheme (FPTAS)). *An additive (resp. relative)* fully polynomial approximation scheme *for the* MPG *$\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ is an algorithm $\mathcal{A}$ such that for all $\varepsilon > 0$, $\mathcal{A}$ computes an additive (resp. relative) ε-value in time polynomial w.r.t. the size[3] of Γ and $\frac{1}{\varepsilon}$.*

Definition 4 (MPG Polynomial Time Approximation Scheme (PTAS)). *An additive (resp. relative)* polynomial approximation scheme *for the* MPG *$\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ is an algorithm $\mathcal{A}$ such that for all $\varepsilon > 0$, $\mathcal{A}$ computes an additive (resp. relative) ε-value in time polynomial w.r.t. the size of Γ.*

[3] Given $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$, *size*$(\Gamma) = |E| + |V| + \log(W)$, where W is the maximum (absolute value) of a weight in Γ.

3 Mean-Payoff Games and Additive Approximation Schemes

Recently, [15] provides an additive fully polynomial scheme for the MPG value problem on graphs with rational weights in the interval $[-1, +1]$. A natural question is whether we could efficiently approximate the value in MPG with no restrictions on the weights. The next theorem shows that a generalization of the positive approximation result in [15] on MPG with arbitrary (rational) weights would indeed provide a polynomial time *exact* solution to the MPG value problem.

Theorem 2. *The MPG value problem does not admit an additive FPTAS, unless it is in* P.

Proof. We start to consider the MPG problem on graphs with integer weights. Assume that the MPG value problem on graphs with integer weights admits an additive FPTAS. Given a MPG $\Gamma = (V, E, w, \langle V_0, V_1\rangle)$ and a vertex $v \in V$, let $|V| = n$ and $\varepsilon = \frac{1}{2n(n-1)}$. Then, our FPTAS computes an additive ε-value $\widetilde{val}$ on v in time polynomial w.r.t. n. By Lemma 1, $val^{\Gamma}(v)$ is a rational number with denominator d such that $1 \leq d \leq n$. Two rationals with denominator d for which $1 \leq d \leq n$ have distance at least $\frac{1}{n(n-1)}$. Hence, there is a unique rational with denominator d, $1 \leq d \leq n$, within the interval $I = \{q \in \mathbb{Q} \,|\, \widetilde{val} - \varepsilon \leq q \leq \widetilde{val} + \varepsilon\}$, where $\varepsilon = \frac{1}{2n(n-1)}$. Such unique rational is $val^{\Gamma}(v)$ and can be easily found in time logarithmic w.r.t. n [16]. Thus, we have an algorithm $\mathcal{A}$ to solve the value problem on Γ in time polynomial w.r.t. n. The MPG problem on graphs with rational weights can be reduced in polynomial time (w.r.t. the size of Γ) to the MPG on graphs with integer weights by simply resizing the weights in the original graph [20, 12]. □

In view of the proof of the above theorem, we could still hope to obtain some positive approximation results for general (i.e. arbitrarly scaled) MPG by considering weaker notion of approximations with respect to FPTAS. Unfortunately, the next lemma shows that the following is sufficient to show that the MPG value problem is in P: determining in time polynomial w.r.t. the size of a given MPG Γ a k-approximate value of v, where $v \in V$ and k is an arbitrary constant.

Theorem 3. *For any constant k: If the problem of computing an additive k-approximate* MPG *value can be solved in polynomial time (w.r.t. the size of the input* MPG*), then the MPG value problem belongs to* P.

Proof. We start to consider MPG with integer weights. Let v be a vertex in the MPG $\Gamma = (V, E, w : E \mapsto [-W, W], \langle V_0, V_1\rangle)$ and denote $|V| = n$. If $2k + 1 > (n-2)!$, then the problem of determining $val^{\Gamma}(v)$ can be solved in time $\mathcal{O}(k^k) = \mathcal{O}(1)$ by simply enumerating all the strategies available to the players.

Otherwise, assume $2k+1 \leq (n-2)!$. Consider the game $\Gamma' = (V, E, w', \langle V_0, V_1\rangle)$, where $\forall e \in E : w'(e) = w(e) \cdot n!$. By hypothesis, there is an algorithm $\mathcal{A}$ that computes a k-approximate value $\widetilde{val}$ for v on Γ' in time T polynomial w.r.t. the size of Γ'. Since $\log(W \cdot n!) = \mathcal{O}(n \cdot \log(n) + \log(W))$, T is also polynomial w.r.t. the size of Γ. By construction, $val^{\Gamma'}(v)$ is an integer. There are at most $2k + 1$ integers in the interval $[\widetilde{val} - k, \widetilde{val} + k]$, thus we have at most $2k + 1$ candidates $\{\frac{\lfloor \widetilde{val}-k \rfloor}{n!}, \ldots, \frac{\lceil \widetilde{val}+k \rceil}{n!}\}$ for $val^{\Gamma}(v)$. Moreover, those candidates lie in an interval of

length $L \leq \frac{2k+1}{n!} \leq \frac{(n-2)!}{n!} = \frac{1}{n\cdot(n-1)}$. The minimum distance between two possible candidates for $val^{\Gamma}(v)$ is $\frac{1}{n\cdot(n-1)}$.

The exact value $val^{\Gamma}(v)$ is thus the unique rational number with denominator of size at most n that lies in the interval $[\frac{\lfloor \widetilde{val}-k \rfloor}{n!}, \frac{\lfloor \widetilde{val}+k \rfloor}{n!}]$ and can be easily found in time logarithmic w.r.t. n [16].

The MPG problem on graphs with rational weights can be reduced in polynomial time (w.r.t. the size of Γ) to the MPG on graphs with integer weights by simply resizing the weights in the original graph [20, 12]. □

A direct consequence of Theorem 3 is that the MPG value problem does not admit a PTAS, unless it is in P. More precisely, Theorem 2 and Theorem 3 entail a result of P-time equivalence between the exact MPG value problem and the three classes of approximations listed in Corollary 1.

Corollary 1. *The following problems are* P*-time equivalent:*

1. *Solving the* MPG *value problem.*
2. *Determining an additive* FPTAS *for the* MPG *value problem.*
3. *Determining an additive* PTAS *for the* MPG *value problem.*
4. *Computing an additive k-approximate* MPG *value in polynomial time, for any constant k.*

4 Mean-Payoff Games and Relative Approximation Schemes

In the recent work in [4], the authors consider the design approximation schemes for the MPG value problem based on the relative–rather than absolute–error. In particular, they provide a relative FPTAS for the MPG value problem on graphs with *nonnegative* weights. Note that negative weights are necessary to encode parity games and the μ-calculus model checking into MPG games [10]. The following theorem considers the problem of designing (fully) polynomial approximation schemes for the MPG value problem on graphs with arbitrary (positive and negative) rational weights. It shows that solving such a problem would indeed provide an exact solution to the MPG value problem, computable in time polynomial w.r.t. the size of the MPG.

Theorem 4. *The MPG value problem does not admit a relative PTAS, unless it is in* P.

Proof. Let $\Gamma = (V, E, p, \langle V_0, V_1 \rangle)$ be a MPG, let $v \in V$. Assume that MPG admit a relative PTAS and consider $\varepsilon = \frac{1}{2}$. Our assumption entails that we have an algorithm $\mathcal{A}$ that computes a relative $\frac{1}{2}$-value $\widetilde{val}$ for v in time polynomial w.r.t. the size of Γ. We show that $\widetilde{val} \geq 0$ if and only if $val^{\Gamma}(v) \geq 0$. In other words, we show that the MPG decision problem is PTIME reducible to the computation of a relative $\frac{1}{2}$-value. By definition of relative ε-value, for $\varepsilon = \frac{1}{2}$, we have:

$$\frac{|\widetilde{val} - val^{\Gamma}(v)|}{|val^{\Gamma}(v)|} \leq \frac{1}{2} \tag{1}$$

We have four cases to consider:

1. In the first case, assume that $\widetilde{val} - val^{\Gamma}(v) \geq 0$ and $\widetilde{val} \geq 0$. By contradiction, suppose $val^{\Gamma}(v) < 0$. Then, Disequation implies:

$$\begin{array}{l} \widetilde{val} - val^{\Gamma}(v) \leq \frac{1}{2} \cdot |val^{\Gamma}(v)| \qquad \Rightarrow \\ \widetilde{val} \leq val^{\Gamma}(v) + \frac{1}{2} \cdot |val^{\Gamma}(v)| < 0 \end{array}$$

   that contradicts our hypothesis.
2. In the second case, assume that $\widetilde{val} - val^{\Gamma}(v) \geq 0$ and $\widetilde{val} < 0$. Then, $0 > \widetilde{val} \geq val^{\Gamma}(v)$. that contradicts our hypothesis.
3. In the third case, assume that $\widetilde{val} - val^{\Gamma}(v) < 0$ and $\widetilde{val} < 0$. By contradiction, suppose $val^{\Gamma}(v) > 0$. Then, Disequation implies:

$$\begin{array}{l} val^{\Gamma}(v) - \widetilde{val} \leq \frac{1}{2} \cdot |val^{\Gamma}(v)| \qquad \Rightarrow \\ \widetilde{val} \geq val^{\Gamma}(v) - \frac{1}{2} \cdot |val^{\Gamma}(v)| \geq 0 \end{array}$$

   that contradicts our hypothesis.
4. The last case to consider is: $\widetilde{val} - val^{\Gamma}(v) < 0$ and $\widetilde{val} \geq 0$. Then, $val^{\Gamma}(v) > \widetilde{val} \geq 0$.

Provided a P-time algorithm for deciding whether $val^{\Gamma}(v) \geq 0$, a dichotomic search can be used to determine $val^{\Gamma}(v)$ in time polynomial w.r.t. the size of Γ [12, 20]. □

As a direct consequence of Theorem 4 we obtain the following result of P-time equivalence involving the computation of MPG exact and approximate solutions.

Corollary 2. *The following problems are* P*-time equivalent:*

1. *Solving the* MPG *value problem.*
2. *Determining a relative* FPTAS *for the* MPG *value problem.*
3. *Determining a relative* PTAS *for the* MPG *value problem.*

References

1. A. Ehrenfeucht and J. Mycielski. International journal of game theory. *Positional Strategies for Mean-Payoff Games*, 8:109–113, 1979.
2. A. V. Karzanov and V. N. Lebedev. Cyclical games with proibitions. *Mathematical Programing*, 60:277–293, 1993.
3. D. Andersson and S. Vorobyov. Fast algorithms for monotonic discounted linear programs with two variables per inequality. Technical Report Preprint NI06019-LAA, Isaac Newton Institute for Mathematical Sciences, Cambridge, UK, 2006.
4. Y. Boros, K. Elbassioni, M. Fouz, V. Gurvich, K. Makino, and B. Manthey. Stochastic mean-payoff games: Smoothed analysis and approximation schemes. In *Proc. of ICALP: Colloquium on Automata, Languages and Programming*, 2011.
5. A. Condon. On algorithms for simple stochastic games. In *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–73. American Mathematical Society, 1993.
6. V. Dhingra and S. Gaubert. How to solve large scale deterministic games with mean payoff by policy iteration. In *Proc. Performance evaluation methodolgies and tools,* article no. 12. ACM, 2006.

7. E. A. Emerson, C. Jutla, and A. P. Sistal. On model checking for fragments of the μ-calculus. In *Proc. of CAV: Computer Aided Verification*, LNCS 697, pages 385–396. Springer, 1993.
8. Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. of STOC: Symposium on Theory of Computing*, pages 60–65. ACM, 1982.
9. H. Bjorklund and S. Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155:210–229, 2007.
10. M. Jurdzinski. Deciding the winner in parity games is in UP $\cap$ coUP. *Inf. Process. Lett.*, 68(3):119–124, 1998.
11. D. Kozen. Results on the propositional mu-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.
12. L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J-F. Raskin. Faster algorithms for mean payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.
13. N. Pisaruk. Mathematics of operations research. *Mean Cost Cyclical Games*, 4(24):817–828, 1999.
14. C. M. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.
15. A. Roth, M. Balcan, A. Kalai, and Y. Mansour. On the equilibria of alternating move games. In *Proc. of SODA: Symposium on Discrete Algorithms*, pages 805–816. ACM, 2010.
16. S. Kwek and K. Mehlhorn. Optimal search for rationals. *Information Processing Letters*, 86:23–26, 2003.
17. S. Schewe. From parity and payoff games to linear programming. In *Proceedings of MFCS: Mathematical Foundations of Computer Science*, LNCS 5734, pages 675–686. Springer, 2009.
18. U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158:343–359, 1996.
19. V. A. Gurvich, A. V. Karzanov, and L. G. Kachiyan. Ussr computational mathematics and mathematical physics. *Cyclic Games and an Algorithm to Find Minmax Cycle Means in Directed Graphs*, 5(28):85–91, 1988.
20. Y. Lifshits and D. Pavlov. Potential theory for mean payoff games. *Journal of Mathematical Sciences*, 145(3):4967–4974, 2007.

Coalitional Games with Priced-Resource Agents*

Dario Della Monica, Margherita Napoli, and Mimmo Parente

Dept. of Computer Science, Università di Salerno, Fisciano (Italy),
{ddellamonica|napoli|parente}@unisa.it

Abstract. *Alternating-time Temporal Logic* (ATL) and *Coalition Logic* (CL) are well-established logical formalisms particularly suitable to model games between dynamic coalitions of agents (like e.g. the system and the environment). Recently, the ATL formalism has been extended in order to take into account boundedness of the resources needed for a task to be performed. The resulting logic is known as *Resource-bounded* ATL (RB-ATL) and has been presented in quite a variety of scenarios. Model checking RB-ATL in very general setting is usually undecidable. Nevertheless, model checking procedures for semantically or syntactically restricted versions of RB-ATL have been proposed. In this paper, we analyze the problem of coalitions of agents that need to perform complex tasks, by using resources with a variable price. We highlight a certain number of problems and considerations, based on different interpretations of shortage of resources, leading to different scenarios.

1 Introduction

Automated verification of multi-agent systems is a significant topic in the recent literature in artificial intelligence [1]. The need of modeling this kind of systems has inspired logical formalisms, the most famous being the *Alternating-time Temporal Logics* [4] and the *Coalition Logic* (CL) [8,9], oriented towards the description of collective behaviors.

The idea of such logics is that agents can join together in team (or coalitions) and share resources to accomplish a task (or reach a goal). In particular, Alternating-time Temporal Logics have been introduced in [4], where the full alternating-time temporal language, denoted by ATL*, has been presented, along with two significant fragments, namely, ATL and ATL$^+$.

In [7], Goranko has studied the relationship between the (expressive power of the) two formalisms. In particular, he has shown that CL can be embedded into ATL. Anyway, none of the two logics takes into account the boundedness of the resources. Approaches towards verification of multi-agent systems under resource constraints can be found in [2,3,5]. In [2], Alechina et al. introduce the logic RBCL, whose language extends the one of CL with explicit representation of resource bounds. In [3], the same authors propose an analogous extension for ATL, called RB-ATL, and give a PTIME model checking procedure mostly based

* This is a preliminary work. Ideas and concepts presented in this paper have been investigated more deeply in [6].

on the one for ATL. In [5], Bulling and Farwer introduce the logics RAL and RAL*. The former represents a generalization of Alechina et al.'s RB-ATL, the latter is ATL* extended with resource bounds. The authors study several syntactic and semantic variants of RAL and RAL* with respect to the (un)decidability of the model checking problem. In particular, while previous approaches only conceive actions consuming resources, they introduce the notion of actions producing resources. It turned out that such a new notion makes the model checking problem undecidable.

The paper is structured as follows. In the next section, we make some considerations about scenarios proposed in the literature, we illustrate our proposals, and show that the model checking problem is still decidable. Then, in the last section, we propose new scenarios for which the model checking problem is under study.

2 Our scenario

This section contains an epistemic discussion about the formalization of a multi-agent system in which agents can cooperate to perform a task and are subject to a limited availability of resources, that is an intrinsic feature of any real-world system. Our discussion hinges on existing approaches in the literature (see e.g. [2,3,5]) and represents an attempt to do a further step towards the formalization of such complex systems.

Formulas of the formalisms proposed in [2,3,5] allow one to assign an endowment of resources to the agents by means of the so-called *team operators*, (borrowed from ATL) and to state that a team of agents can perform a task. Due to the nesting of the team operators in a formula (which reflects the fact that coalitions may change in a game), during the execution of the task, the agents can be provided with a new endowment of resources to perform subtasks. This is somehow unrealistic, as it does not take into account issues related to procurement of resources. In particular, a very significant present-day issue is that resources are available on the market (or in nature) in limited amount, and the cost for achieving them depends on such an availability.

First improvement. Thus, our first proposal is to introduce the notion of *price* of resources. Unlike the existing approaches, agents are equipped with an amount of money instead of an endowment of resources. They can use money for getting resources. Notice that money cannot be considered as a resource like the others for at least two reasons. First, according to our aim of assigning, to each resource, a price that is variable depending on several factors (e.g., the current availability of the resource on the market), it is necessary to introduce the new component money with the special ability of "measuring" the value of all the resources, thus making it possible for the agent to acquire them when needed. Second, since the money as the special ability of "measuring" the value of the resources makes sense to consider problems of optimization (e.g., minimization of the amount of money needed to acquire the resources to perform a task). Formulas of our logic state that a team of agents is able to perform a given

task provided with a given amount of money. We also introduce a notion of *global availability* of resources on the market, the intended meaning being that, whenever an agent acquires resources from the market, the global availability is decreased, whenever it produces resources, the global availability is increased. The price of resources can be any function of the several components into play. In our approach, prices of resources depend on their global availability, the acting agent, and the physical location.

Second improvement. Another aspect that has not been fully analyzed in the literature is the problem of actions producing resources. On the one hand, in [2,3], actions can only consume resources; on the other hand, in [5], the authors state that whenever actions can produce resources the model checking problem is undecidable. In this paper, we show how to constrain the way in which actions can produce resources, still preserving the decidability of the model checking problem. The idea is that it is possible, at a given time, for an action to produce a resource in a quantity that is not greater than the amount that has already been consumed so far. This implies that, even if actions can produce resources, the global availability of the market will never be greater than the initial global availability, that is crucial for the model checking algorithm. Such a notion makes sense as, in practical terms, it allows one to model significant real-world scenarios, such as, acquiring memory by a program, leasing a car during a travel, and, in general, any scenario in which an agent is releasing resources previously acquired.

2.1 Team and task

So far, we have talked about teams (or coalitions) of agents performing a task. But we have not clarified yet the two notions of team and task. First of all, a task is a goal that has to be reached and, for what concerns us, is represented by a logical formula that has to be satisfied. A team of agents is a subset of agents that act collectively in order to perform a task. To this end, they select a strategy that univocally determines their behavior in each possible configuration of the system. Nevertheless, the behavior of the remaining agents, that we collectively denote as the *opponent*, is undetermined. Aim of the team is to guarantee that the task is performed independently of the opponent's behavior, that is, the task must be guaranteed for each possible opponent's strategy.

The formalism that naturally fits our intention is the logic ATL, that allows one to fix a strategy for the agents of a team and to force an 'LTL-like' property, representing the task, to be true over all the possible executions (or outcomes) of the system. Obviously, its syntax and semantics will be extended in order to deal with resource constraints.

2.2 The special resource 'time'

One can be interested in answering questions of the kind "is it possible for the team A of agents to complete the task in x time-unit?". It is clear that the

resource 'time' neither can be bought nor rented. It is in a certain sense out of the control of the agents, as it is only possible to specify that a task should be executed within some given time constraints, while it is not possible to administer it. Thus, resource 'time' will be treated in a special way with respect to other resources.

2.3 Game structure

A *game structure* G is based on a graph whose vertices, called *locations*, are labeled by atomic propositions. In each location, each agent can choose among a non-empty set of actions to be performed. Any possible combination of actions gives rise to *transitions*, that are the edges of the graph. In general, actions consume and produce resources. Each resource has a price that is variable and depends on, inter alia, the current availability of that resource on the market. Thus, a transition can be executed if the resources needed to perform the actions are available and each agent has enough money to acquire them.

Let $\mathbb{Z}$ denote the set of integers, $\mathbb{N}$ denote the set of non-negative integers, and let $\mathcal{M} = (\mathbb{N} \cup \{\infty\})^r$. A game structure G is a tuple $\langle Q, \mathcal{AP}, V, Ag, \Sigma, \Delta, R, t, c, \rho \rangle$, where:

- Q is the finite set of *locations*, $\mathcal{AP}$ is the finite set of *propositional letters*, and $V : Q \to 2^{\mathcal{AP}}$ is the *valuation function*;
- $Ag = \{a_1, a_2, \ldots, a_n\}$ is the finite set of *agents*, and Σ is the finite set of *actions*, denoted by $\alpha_1, \alpha_2, \ldots$,
- $\Delta : Q \times Ag \to 2^{\Sigma}$ is the *action function* that defines the possible actions of an agent in a given location. By an abuse of notation, we use Δ also to denote the function from Q to 2^{Σ^n} defined as $\Delta(q) = \Delta(q, a_1) \times \ldots \times \Delta(q, a_n)$;
- $R = \{R_1, \ldots, R_r\}$ is the finite set of *resource types*. It contains the particular resource *time*, denoted by R_1;
- $t : Q \times \Sigma^n \to Q$ is the *transition function* over the set of locations. It is a partial function defined for any pair $(q, \langle \alpha_1, \ldots, \alpha_n \rangle)$ such that $\langle \alpha_1, \ldots, \alpha_n \rangle \in \Delta(q)$;
- $c : \Sigma \times Ag \to \mathbb{Z}^r$ is the *cost* function, assigning a cost to each action performed by an agent. A negative cost represents a resource consumption, while a positive cost represents a resource production;
- $\rho : \mathcal{M} \times Ag \times Q \to \mathbb{N}^r$ is the *price function*, denoting the price of each resource, depending on the current resource availability, the acting agent, and the current location. Without loss of generality, we can assume the price of the resource 'time' to be always zero, as it is a resource that cannot be acquired and thus its price is meaningless.

2.4 A logical formalization: PRB-ATL

We now define the logic *Priced* RB-ATL (PRB-ATL). The formulae are given by the following grammar.

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle A^{\$} \rangle\rangle \bigcirc \varphi \mid \langle\langle A^{\$} \rangle\rangle \Box \varphi \mid \langle\langle A^{\$} \rangle\rangle \varphi \mathcal{U} \varphi \mid \sim \vec{b}$$

where $p \in \mathcal{AP}$, $A \subseteq Ag$, $\sim \in \{<, \leq, =, \geq, >\}$, and $\vec{b} \in \mathcal{M}$. Moreover, $\$ \in (\mathbb{N} \cup \{\infty\})^n$ is a vector representing the availability of money of the agents. For each agent $a \in Ag$, by $\$_a$ we denote the availability of money for the agent a. Intuitively, formulae of the kind $\sim \vec{b}$ tests the current availability of resources on the market. Formulae of the kind $\langle\langle A^{\$}\rangle\rangle\psi$, with $\psi \in \{\bigcirc\varphi, \Box\varphi, \varphi\mathcal{U}\varphi\}$ state that the team A has a strategy such that, for every action performed by the opponent (i.e, $Ag \setminus A$), ψ is satisfied, and such that the total expenses of each agent $a \in A$ is less than or equal to $\$_a$. Without loss of generality, we can assume $\$_a = \infty$ for each $a \notin A$.

In order to give the formal semantics we must first define the following notions. From now on, let G be a generic game structure. We extend the sum operation to sum between vectors component-wise. Additionally, we use the usual component-wise comparison relations between vectors.

Definition 1 (configuration and computation). *A* configuration *of G is a pair $c = \langle q, \vec{m}\rangle \in Q \times \mathcal{M}$. A* finite computation *(resp.,* infinite computation*) over G is a finite (resp., infinite) sequence of configurations (of G) $\pi = c_1 c_2 \ldots$, such that, for each i, if $c_i = \langle q_i, \vec{m_i}\rangle$ and $c_{i+1} = \langle q_{i+1}, \vec{m_{i+1}}\rangle$, then there exists a transition $t(q_i, \vec{\alpha}) = q_{i+1}$, with $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_n\rangle$, such that $\vec{m_{i+1}} = \vec{m_i} + \sum_{j=1}^{n} c(\alpha_j, a_j)$.*

Given a team A, a *move* of A, denoted $\vec{\alpha_A}$, is a vector of actions α_a, for all $a \in A$, representing the action performed by the agents of A. To represent the possible moves of the team A at any location q, we extend the function Δ with the function $\hat{\Delta}_A : Q \to 2^{\Sigma^{|A|}}$ representing the Cartesian product of $\Delta(q, a)$, for all $a \in A$.

Definition 2 (strategy). *A* strategy *F_A for the team of agents A is a function which associates, to each finite computation $\pi = c_1 c_2 \ldots c_s$, with $c_s = \langle q, \vec{m}\rangle$, a move $\vec{\alpha_A}$, such that $\vec{\alpha_A} \in \hat{\Delta}_A(q)$. A strategy is said to be* memoryless *if $F_A(\pi) = F_A(\pi')$ for each pair of computations π, π' such that $\pi = c_1 c_2 \ldots c_s$, $\pi' = c'_1 c'_2 \ldots c'_{s'}$, $c_s = \langle q, \vec{m}\rangle$, $c'_s = \langle q, \vec{m'}\rangle$.*

In other words, a strategy F_A determines the behavior of the agents in the team A. Anyway, for each move $\vec{\alpha_A}$ (of the team A) and location $q \in Q$, depending on the move of the opponent, there are several possibilities for the next location. The set including all such possibilities is called the set of *outcomes of the move $\vec{\alpha_A}$ (of the team A) at location q*, denoted by $out(q, \vec{\alpha_A})$. As a consequence, given an initial location q_1, a strategy F_A corresponds to a tree of computations, called *outcomes of the strategy F_A from the location q_1* and denoted by $out(q_1, F_A)$.

Finally, in order to prevent actions producing resources to cause a reimbursement of money to the agent, we define $cons : \Sigma \times Ag \to \mathbb{Z}^r$ in such a way that $cons(\alpha, a)$ returns the vector obtained from the vector $c(\alpha, a)$ by replacing the positive components with zeros and the negative components with the corresponding absolute value.

Let $\pi = c_1 c_2 \ldots \in out(q_0, F_A)$, where $c_i = \langle q_i, \vec{m}_i \rangle$ for all i, be a computation and let $\vec{\alpha}^i = \langle \alpha_a^i \rangle_{a \in Ag}$ be the move performed by the agents at the configuration c_i for all i.

Definition 3 (consistent computations). *The computation π is $(\$, \vec{m}_1)$-consistent for a strategy F_A if, for each $i \geq 0$, $\vec{0} \leq \vec{m}_i \leq \vec{m}_1$, and $a \in A$*

$$\sum_{j=1}^{i} \rho(\vec{m}_j, a, q_j) \cdot cons(\alpha_a^j, a) \leq \$_a.$$

The semantics of the logic can be defined as usual and we omit it here.

2.5 Model checking

The model checking problem consists in verifying whether a formula φ is satisfied in a location q of a game structure G, with an initial resource availability $\vec{m} \in \mathcal{M}$.

The algorithm for model checking our logic is mostly based on the one proposed in [4] and used in [3] for model checking, respectively, ATL and its resource-bounded extension RB-ATL. Roughly speaking, it works by computing, for each sub-formula ψ of the formula φ to be model checked, the set of states in which ψ holds. The main difficulties when dealing with bounds on resources are the following. First, the set of sub-formulae must be replaced by an extended set of formulae (see [3]), that includes, for each sub-formula of the form $\langle\langle A^{\$} \rangle\rangle \psi$, all the formulae $\langle\langle A^{\$'} \rangle\rangle \psi$ for each $\$' < \$$. Second, the state does not correspond anymore to the vertices of the game structure, but to configurations, that is, pairs $\langle q, \vec{m} \rangle \in Q \times \mathcal{M}$. Third, during the analysis of the computations over the game structure, the algorithm must take into account the resource availability on the market in order to guarantee that in each instant of the computation all the resources are still available, as well as to be able to compute the current prices of resources, that depend also on their availability. Finally, it must be ensured that, even if actions can produce resources, availability of each resource may not be higher than the initial availability.

Thus, we now state the main result, without showing the proof in this preliminary version. Full details of both the formalization and the algorithm will appear in a forthcoming paper.

Let M be the greater component appearing in the initial resource availability vector $\vec{m}$.

Theorem 1. *The model checking problem for* PRB-ATL *is decidable in time* $O(M^r \times |\varphi|^{r+1} \times |G|)$.

3 Discussion

A further line of research in which we intend to investigate is when, given a formula in our logic, the coalitions are unknown, that is they are not specified

and we may ask whether, for each nested sub-formula, there exists a team and a money endowment such that the formula is satisfied. More precisely, given a formula Ψ where $\langle\langle X_i^{\$_i}\rangle\rangle$ are the team operators occurring in it, we want to compute the coalitions X_i such that Ψ is satisfied with minimum expense in terms of both money and resources. Let us notice that if the minimality condition is not requested, then the problem can be trivially solved.

Another feature we are investigating is when each agent has a price. In this scenario, in which agents are themselves resources to be acquired to perform the task, it makes sense to consider the problem of deciding which team is able to perform the task at the minimum cost.

References

1. Thomas Ågotnes, Wiebe van der Hoek, and Michael Wooldridge. On the logic of coalitional games. In *AAMAS*, pages 153–160, 2006.
2. Natasha Alechina, Brian Logan, Nguyen Hoang Nga, and Abdur Rakib. A logic for coalitions with bounded resources. In *Proc. of the 21st International Joint Conference on Artificial Intelligence*, IJCAI '09, pages 659–664, 2009.
3. Natasha Alechina, Brian Logan, Nguyen Hoang Nga, and Abdur Rakib. Resource-bounded alternating-time temporal logic. In *Proc. of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*, AAMAS '10, pages 481–488, 2010.
4. Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of ACM*, 49:672–713, September 2002.
5. Nils Bulling and Berndt Farwer. On the (un-)decidability of model checking resource-bounded agents. In *Proc. of the 19th European Conference on Artificial Intelligence*, ECAI '10, pages 567–572, 2010.
6. D. Della Monica, M. Napoli, and M. Parente. On a Logic for Coalitional Games with Priced-Resource Agents. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 278:215–228, 2011. Proc. of the 7th Workshop on Methods for Modalities (M4M 2011) and the 4th Workshop on Logical Aspects of Multi-Agent Systems (LAMAS 2011).
7. Valentin Goranko. Coalition games and alternating temporal logics. In *Proc. of the 8th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '01, pages 259–272, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
8. Marc Pauly. A logical framework for coalitional effectivity in dynamic procedures. *Bulletin of Economic Research*, 53(4):305–324, 2001.
9. Marc Pauly. A modal logic for coalitional power in games. *Journal of Logic and Computation*, 12(1):149–166, 2002.

Cyclic Pregroups and Natural Language: a Computational Algebraic Analysis

Claudia Casadio and Mehrnoosh Sadrzadeh

[1]Faculty of Psychology, Chieti University, IT
casadio@unich.it
[2]Computing Laboratory, Oxford University, UK
mehrs@comlab.ox.ac.uk

Abstract. The calculus of pregroups is introduced by Lambek [1999] as an algebraic computational system for the grammatical analysis of natural languages. Pregroups are non commutative structures, but the syntax of natural languages shows a diffuse presence of cyclic patterns exhibited in different kinds of word order changes. The need of cyclic operations or transformations was envisaged both by Z. Harris and N. Chomsky, in the framework of generative transformational grammar. In this paper we propose an extension of the calculus of pregroups by introducing appropriate cyclic rules that will allow the grammar to formally analyze and compute word order and movement phenomena in different languages such as Persian, French, Italian, Dutch and Hungarian. This cross-linguistic analysis, although necessarily limited and not at all exhaustive, will allow the reader to grasp the essentials of a pregroup grammar, with particular reference to its straightforward way of computing linguistic information.

1 Introduction

In this paper we apply logical cyclic rules to the analysis of word order changes in natural languages. The need of some kind of cyclic operations or transformations was envisaged both by Harris [1966, 1968] and Chomsky [1981, 1986] for the treatment of the linguistic contexts referred to with the term *movement*. In the paper we present a formal approach to natural language based on two cyclic rules that hold in the systems of Noncommutative and Cyclic Multiplicative Linear Logic (NMLL,CyMLL), developed by Abrusci [1991, 2002] from Yetter [1990]. A critical move of this paper is to embed such cyclic rules into the calculus of Pregroups recently introduced by Lambek [1999, 2001, 2008]. The calculus has been succesfully applied to a variety of natural languages from English and German, to French and Italian, and others [see Casadio and Lambek 2008].

We show that the formal grammar obtained by so extending the pregroup calculus allows one to compute string of words belonging to various kinds of natural languages, deriving grammatical sentences involving different types of word order changes or *movements*, with particular reference to the way in which unstressed clitic pronouns attach to their verbal heads. Cross-linguistic evidence

is provided comparing languages belonging to the Indo-European family, like Persian, on the one side, French and Italian, on the other, as representatives of the Romance group. Moreover the analysis is extended to include Dutch, as a representative of the West Germanic group, and Hungarian, as a representative of the Uralic family, non related to the Indo-European languages. Such cross-linguistic prespective extends the results of preceding work [Casadio and Sadrzadeh 2011, Sadrzadeh 2010], and the analysis proposed for Dutch is new.

We conclude with a short discussion of the logical and methodological connections of the present analysis to cyclic linear logic [Yetter 1990, Abrusci 1991, 2002].

2 Cyclic rules for the Calculus of Pregroups

2.1 Pregroup grammar

Pregroups are introduced by Lambek in [1999] as an alternative to the Syntactic Calculus, a well known model of categorial grammar largely applied in the fields of theoretical and computational linguistics; see e.g. Moortgat [1997], Morrill [2010]. The calculus of pregroups is a particular kind of substructural logic that is compact and non-commutative [Buszkowsi 2001, 2007]. Pregroups in fact are non conservative extensions of Noncommutative Multiplicative Linear Logic (NMLL) in which *left* and *right* iterated negations, equivalently *left* and *right* iterated adjoints, do not cancel [Abrusci 2001, Casadio 2001, Casadio and Lambek 2002, Lambek 2001, 2008].

A pregroup $\{G, \cdot, 1, {}^{\ell}, {}^{r}, \rightarrow\}$ is a partially ordered monoid in which each element a has a *left adjoint* a^{ℓ}, and a *right adjoint* a^{r} such that

$$a^{\ell} a \rightarrow 1 \rightarrow a\, a^{\ell}$$
$$a\, a^{r} \rightarrow 1 \rightarrow a^{r} a$$

where the dot "$\cdot$", that is usually omitted, stands for multiplication with unit 1, and the arrow denotes the partial order[1]. In linguistic applications syntactic types (or categories) are assigned to the words in the dictionary of a language, the symbol 1 is assigned to the empty string of types, and the operation of multiplication is interpreted as linguistic concatenation. Adjoints are unique and the following results are proved (see Lambek [2008] for details)

$$1^{\ell} = 1 = 1^{r},$$
$$(a \cdot b)^{\ell} = b^{\ell} \cdot a^{\ell} \quad , \quad (a \cdot b)^{r} = b^{r} \cdot a^{r},$$
$$\frac{a \rightarrow b}{b^{\ell} \rightarrow a^{\ell}} \quad , \quad \frac{a \rightarrow b}{b^{r} \rightarrow a^{r}} \quad , \quad \frac{b^{\ell} \rightarrow a^{\ell}}{a^{\ell\ell} \rightarrow b^{\ell\ell}} \quad , \quad \frac{b^{r} \rightarrow a^{r}}{a^{rr} \rightarrow b^{rr}} \quad .$$

[1] A partial order '$\leq$' (here denoted by the arrow '$\rightarrow$') is a binary relation which is reflexive: $x \leq x$, transitive: $x \leq y$ and $y \leq z$ implies $x \leq z$, and anti-symmetric: $x \leq y$ and $y \leq x$ implies $x = y$. We may read $x \leq y$ as saying that everything of type x is also of type y. The arrow is introduced to show the inference between types, like in type logical grammmars.

Linguistic applications make particular use of the equation $a^{r\ell} = a = a^{\ell r}$, allowing the cancellation of double opposite adjoints, and of the rules

$$a^{\ell\ell}\ a^{\ell} \to 1 \to a^{\ell}\ a^{\ell\ell} \quad , \quad a^{r}\ a^{rr} \to 1 \to a^{rr}\ a^{r}$$

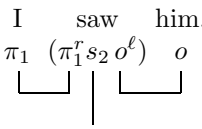
contracting and expanding *left* and *right* adjoints respectively; just *contractions* are needed to check and determine if a given string of words is a sentence:

$$a^{\ell}\, a \to 1 \ \text{ and } \ a\, a^{r} \to 1\ .$$

A pregroup is *freely generated* by a partially ordered set of *basic* types. From each basic type a we form *simple* types by taking single or repeated adjoints: $\ldots\ a^{\ell\ell}$, a^{ℓ}, a, a^{r}, $a^{rr}\ldots$. *Compound* types or just *types* are strings of simple types.

Like in categorial grammars we have two essential steps: (i) assign one or more (basic or compound) types to each word in the dictionary; (ii) check the grammaticality and sentencehood of a string of words by a calculation on the corresponding types, where the only rules involved are *contractions* and *ordering postulates* such as $\alpha \to \beta$ (α, β basic types).

Taking as basic types: n (noun), π (nominative argument), o (accusative argument), w (dative argument), λ (locative argument), i (infinitive verb), s (sentence), we obtain *simple types* such as n^{ℓ}, n^{r}, π^{ℓ}, π^{r}, o^{ℓ}, o^{r}, $\ldots$, and *compound types* such as $(\pi^{r}\, s\, o^{\ell})$, the type of a transitive verb with subject in the nominative case and object in the accusative case. For example, the types of the constituents of the sentence "I saw him." are as follows, where the subscript 1 in π_1 means first person singular, and the subscript 2 in s_2 indicates the past tense[2]



We say that a sentence is grammatical iff the computation (or calculation) of the types assigned to its words *reduces* to the type s, a procedure depicted by the under-link diagrams[3].

2.2 Cyclic rules in theoretical linguistics

In the Sixties Zellig Harris developed a cyclic cancellation automaton [1966, 1968] as the simplest device to recognize sentence structure by computing strings of words through cancellations of a given symbol with its left (or right) inverse.

[2] We analyze a sentence of the form SUBJ VP by assigning types $(\pi_k\ s_j)$, for $j = 1, \ldots, 7$ denoting the seven basic *tenses*, and k denoting the six verbal persons (singular $k = 1, 2, 3$, plural $k = 4, 5, 6$).

[3] These diagrams are reminescent of the planar poof nets of non-commutative linear logic, connecting the formulas, decorated by a left or right adjoint with their positive counterparts, by means of under-links that satisfy the requirements of parallelism and planarity (Abrusci 2002, Lambek 1999, 2008, Buszkowski 2007).

The formalism proposed by Harris is sufficient for many languages, requiring just string concatenation for sentence derivation, but the same limitations of context free grammars are met [Francez and Kaminski 2007, Buszkowski and Moroz 2008]. Different kinds of cyclic transformations were explored by Chomsky [e.g. 1981] to compute constituents movement in long distance dependencies. As argued by Lambek [2008], the analysis of modern European languages requires that word symbols (logical types) take double superscripts, like in Harris [1968], or the double adjoints defined in pregroup grammar, wherever Chomsky's approach postulates a trace. The calculus of pregroups meets in this sense the requirements of Chomsky's transformational grammar expressing *traces* by means of *double adjoints*.

2.3 Introducing cyclic rules into pregroups

We extend the pregroup calculus with two cyclic rules that will allow us to analyse a variety of movement phenomena in natural languages. It is important to point out that the addition of cyclic rules is not equivalent to the reintroduction of the structural rule of *Commutativity* into the pregroup calculus (a logic without structural rules like the Syntactic Calculus).

These rules are derivable into NMLL (or also CyMLL) cf. Abrusci [2002]

$$\frac{\vdash \Gamma, \Delta}{\vdash \Delta^{+2}, \Gamma}(rr) \qquad\qquad \frac{\vdash \Gamma, \Delta}{\vdash \Delta, \Gamma^{-2}}(\ell\ell)$$

In the notation of pregroups (*positive* formulae as $right$ adjoints and *negative* formulae as $left$ adjoints), the formulation of the two cyclic rules becomes

$$(1)\quad qp \leq p^{rr}q \qquad\qquad (2)\quad qp \leq pq^{\ell\ell}$$

The monoid multiplication of the pregroup is *n*on-commutative, but if we add to the pregroup calculus the cyclic rules defined above as metarules, then we obtain a limited form of commutativity, for $p, q \in P$.

Metarules are postulates introduced into the dictionary of the grammar to simplify lexical assignments and make syntactic calculations quicker: the types assigned to the words of a given language are assumed to be stored permanently in the speaker's 'mental' dictionary; to prevent overloading this mental dictionary, the grammar includes metarules asserting that, if the dictionary assigns a certain type to a word, then this word may also have certain other types. The effect of the two cyclic metarules is that the *cyclic* type of each verb form is derivable from its original type.

3 Word Order and Cyclicity in Natural Languages

In the following section we present a cross-linguistic analysis comparing languages belonging to the Indo-European family, like Persian, on the one side, French and Italian, on the other side, as representatives of the Romance group.

The analysis is also extended to include Dutch, as a representative of the West Germanic group, and Hungarian, as a representative of the Uralic family, which is not related to the Indo-European family.

3.1 Cross-linguistic motivations

In Persian the subject and object of a sentence occur in pre-verbal position (Persian is a SOV language), but they may attach themselves as clitic pronouns to the end of the verb and form a one-word sentence. By doing so, the word order changes from SOV to VSO. A similar phenomenon happens in Romance languages like Italian and French, but the movement goes in the opposite direction: verbal complements occurring in post-verbal position, can take a clitic form and move to a pre-verbal position.

These movements have been accounted for in the pregroup grammar for French [Bargelli and Lambek 2001] and Italian [Casadio and Lambek 2001] by assigning clitic words types with double adjoints. In this paper we present a different approach offering a unified account of clitic movement by adding two *c*yclic rules (or metarules) to the lexicon of the pregroup grammar. The import of these rules is that the *c*litic type of the verb is derivable from its *o*riginal type.

Clitic Rule (1): If $p^r q$ is the *original* type of the verb, then so is $q\overline{p}^{\ell}$.

Clitic Rule (2): If qp^{ℓ} is the *original* type of the verb, then so is $\overline{p}^r q$.

The over-lined types $\overline{p}^{\ell}$, $\overline{p}^r$ are introducend as a notational convenience to distinguish the clitic pronouns from the non-clitic stressed pronouns or arguments. For any clitic pronoun p, we postulate the partial order $\overline{p} \leq p$ to express the fact that a clitic pronoun is also a kind of pronoun. We assume that for all $p, q \in P$, we have $\overline{pq} = \overline{p}\ \overline{q}$.

3.2 Clitic movement in Persian

In Persian the subject and object of a sentence occur in pre-verbal position (Persian is a SOV language), but they may attach themselves as clitic pronouns to the end of the verb and form a one-word sentence (word order changes from SOV to VSO). The clitic clusters (pre-verbal vs. post-verbal) for the sentence *I saw him*, "man u-ra didam" in Persian, exhibit the following general pattern:

The over-lined types $\overline{\pi}, \overline{o}$, stand for the clitic versions of the subject and object pronouns.

Including **clitic rule** (1) in the lexicon of the pregroup grammar of Persian, we obtain the clitic form of the verb from its original type. The original Persian verb has the type: $o^r\pi^r s = (\pi o)^r s$, which is of the form $p^r q$; after applying the clitic rule we obtain: $s(\overline{\pi o})^\ell = s(\overline{\pi}\,\overline{o})^\ell = s\overline{o}^\ell\overline{\pi}^\ell$, i.e. the type of the verb with postverbal clitics. The clitic rule can be seen as a re-write rule and the derivation can be depicted as a one-liner as follows

$$o^r\pi^r s = (\pi o)^r s \quad \rightsquigarrow \quad s(\overline{\pi o})^\ell = s\overline{o}^\ell\overline{\pi}^\ell$$

To form these one-word sentences, one does not necessarily have pronouns for subject and object in the original sentence. They can as well be formed from sentences with nominal subjects and objects, for example the sentence *I saw Nadia*, in Persian "man Nadia-ra didam", becomes "did-am-ash" and is typed exactly as above.

Hassan	Nadia	saw		saw	he	her
Hassan	Nadia-ra	did.		di	d	ash.
π	o	$(o^r\pi^r s) \to$ s		$(s\ \overline{o}^\ell\ \overline{\pi}^\ell)$	$\overline{\pi}$	$\overline{o} \to$ s

One can form a yes-no question from any of the sentences above, by adding the question form "aya" to the beginning of the sentence. Since in Persian the word order of the question form is the same as that of the original sentence, the clitic movement remains the same and obeys the same rule [Sadrzadeh 2008]

Did	Hassan	Nadia	see?		Did	see	he	her?
aya	Hassan	Nadia-ra	did?		aya	di	d	ash?
qs^ℓ	π	o	$(o^r\pi^r s) \to$ q		qs^ℓ	$(s\overline{o}^\ell\overline{\pi}^\ell)$	$\overline{\pi}$	$\overline{o} \to$ q

3.3 Clitic movement in French

In French, the clitic clusters move in the opposite direction with respect to Persian. We need therefore the **clitic rule** (2). Using this rule we can derive the type of the clitic form of the verb from its original type. Consider a simple example, the sentence "Jean voit Marie." (*Jean sees Marie*) and its clitic form "Jean la voit". We type these as follows

Jean	voit	Marie.		Jean	la	voit.	
π	$(\pi^r s\, o^\ell)$	o	$\to$ s	π	$\overline{o}$	$(\overline{o}^r\pi^r s)$	$\to$ s

To derive the clitic type of the verb from its original type, we start with the original type of "voit" : $(\pi^r s\, o^\ell)$ take $q = (\pi^r s)$ and $p^\ell = o^\ell$, apply **clitic rule** (2) and obtain the type: $(\overline{o}^r\pi^r s)$. The following is an example with the locative object λ and its clitic pronoun $\overline{\lambda}$.

Jean	va	à Paris.		Jean	y	va.	
π	$(\pi^r s\lambda^\ell)$	λ	$\to$ s	π	$\overline{\lambda}$	$(\overline{\lambda}^r\pi^r s)$	$\to$ s

Again the **clitic rule** (2) easily derives $(\overline{\lambda}^r\pi^r s)$ from $(\pi^r s\lambda^\ell)$. Now consider the more complicated example "Jean donne une pomme à Marie" (*Jean gives an apple to Marie*); we type it as follows

$$\begin{array}{cccccc} \text{Jean} & \text{donne} & \text{une pomme} & \text{à} & \text{Marie.} \\ \pi & (\pi^r s\, w^\ell o^\ell) & o & & w \end{array}$$

While learning French at school, it's difficulty to remember the order of the clitic pronouns in these sentences; **clitic rule** (2) offers a hint: according to it a verb of the type $(\pi^r s\, w^\ell o^\ell)$ can also be of type $\overline{w}^r \overline{o}^r \pi^r s$, taking $q = (\pi^r s)$ and $p = (ow)^\ell$. This type will result in the following grammatical sentence

$$\begin{array}{cccccc} \text{Jean} & \text{la} & \text{lui} & \text{donne.} \\ \pi & \overline{o} & \overline{w} & \overline{w}^r \overline{o}^r \pi^r s \end{array}$$

But it will *not* make the following incorrect order grammatical

$$\begin{array}{cccccc} \text{Jean} & \text{lui} & \text{la} & \text{donne.} \\ \pi & \overline{w} & \overline{o} & \overline{w}^r \overline{o}^r \pi^r s\ . \end{array}$$

3.4 Clitic movement in Italian

Sentences with one occurrence of a pre-verbal clitic can be obtained exactly like in French, as shown in the following examples corresponding to the French sentences given above: "Gianni vede Maria" and its clitic form "Gianni la vede"

$$\begin{array}{cccccc} \text{Gianni} & \text{vede} & \text{Maria.} & & \text{Gianni} & \text{la} & \text{vede.} \\ \pi & (\pi^r s\, o^\ell) & o & \to \text{s} & \pi & \overline{o} & (\overline{o}^r \pi^r s) \to \text{s} \end{array}$$

To derive the clitic type of the verb we start with the original type $(\pi^r s o^\ell)$, take $q = \pi^r s$ and $p^\ell = o^\ell$, apply **clitic rule** (2) and obtain the type $(\overline{o}^r \pi^r s)$. The same process is obtained with a locative argument λ and the corresponding clitic pronoun $\overline{\lambda}$, where the clitic rule derives $(\overline{\lambda}^r \pi^r s)$ from $(\pi^r s \lambda^\ell)$.

$$\begin{array}{cccccc} \text{Gianni} & \text{va} & \text{a Roma.} & & \text{Gianni} & \text{ci} & \text{va.} \\ \pi & (\pi^r s\, \lambda^\ell) & \lambda & \to \text{s} & \pi & \overline{\lambda} & (\overline{\lambda}^r \pi^r s) \to \text{s} \end{array}$$

When we consider the more complicated cases of a verb with two arguments like in "Gianni da un libro a Maria" (*Gianni gives a book to Maria*), or "Gianni mette un libro sul tavolo" (*Gianni puts a book on the table*), we find that clitics pronouns occur in the opposite order with respect to French: e.g. the verb "dare" (*to give*) has the clitic form "Gianni glie lo da" (*Gianni to-her it gives*).

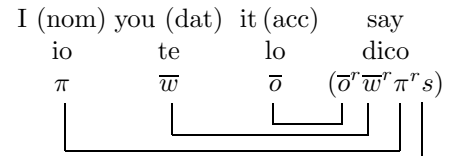
In Casadio and Lambek [2001] this problem was handled by introducing a second type for verbs with two complements $(\pi^r s\, o^\ell w^\ell)$ and $(\pi^r s\, o^\ell \lambda^\ell)$; assuming these verb types and applying **clitic rule** (2) we obtain the correct clitic verb forms to handle the cases of pre-verbal cliticization:

$$(\pi^r s\, o^\ell w^\ell) = (\pi^r s (wo)^\ell) \rightsquigarrow ((\overline{wo})^r\ \pi^r\ s) = (\overline{o}^r\ \overline{w}^r\ \pi^r\ s)$$

the same with λ in place of o.

$$\begin{array}{cccccc} \text{Gianni} & \text{glie} & \text{lo} & \text{da.} & \text{Gianni} & \text{ce} & \text{lo} & \text{mette.} \\ \pi & \overline{w} & \overline{o} & (\overline{o}^r \overline{w}^r \pi^r s) \to \text{s} & \pi & \overline{\lambda} & \overline{o} & (\overline{o}^r \overline{\lambda}^r \pi^r s) \to \text{s} \end{array}$$

The following diagram shows the general pattern of preverbal cliticization in Italian with a verb taking two arguments:



4 Insights into Hungarian and Dutch word order

In the previous section we have dealt with a special kind of movement: the clitic movement, limited to certain words moving from before to after the verb (or the other way around) and becoming clitics. In this section we show that similar cyclic rules can be used to reason about movement of words in general. This movement is more free: firstly all words, or relevant words strings, can move; secondly the movement is not restricted to the context surrounding the verb.

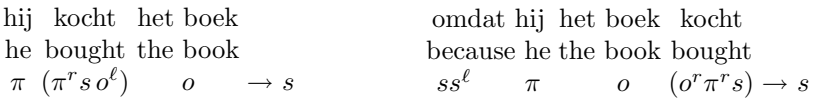
4.1 Word order in Dutch subordinate clauses

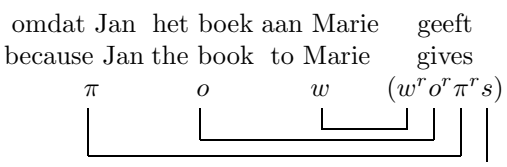
In Dutch (like in German), the position of the finite verb in main clauses differs from that in subordinate clauses. The unmarked order of the former is SVO, while the latter exhibit an SOV pattern. Also concerning word order Dutch is similar to German in that the finite verb always occurs in second position in declarative main clauses (V2), while the verb appears in final position in subordinate clauses: a sentence like "hij kocht het boek" (*he bought the book*) in subordinate clauses becomes "... hij het boek kocht" (*he the book bought*); with more arguments, "Jan geeft het boek aan Marie" (*Jan gives the book to Marie*) becomes "... Jan het boek aan Marie geeft" (*Jan the book to Marie gives*).

In order to reason about these kinds of movement, we generalize our **clitic rule** (2), corresponding to the *r*ight cyclic axiom, to all words by removing the bar from the types and the word 'original' from the definition, obtaining the following rule allowing verb argoments to move up the string from right to left

Move Rule (1): If qp^ℓ is the type of the verb, so is $p^r q$.

The rule allows us to correctly type the examples mentioned above





Consider now an example with a modal verb "Ann wil Marie kussen" (*Ann wants to kiss Marie*) and the corresponding embedded clause "dat Ann Marie wil Kussen" (*that Ann wants to kiss Marie*)

Ann	wil	Marie	kussen		dat	Ann	Marie	(wil kussen)	
Ann	wants	Marie	kiss		that	Ann	Marie	(wants kiss)	
π	$(\pi^r s\ i^\ell)$	o	$(o^r i) \to s$		ss^ℓ	π	o	$(o^r \pi^r s)$	$\to s$

By contraction we obtain the type of the string "wil kussen": $(\pi^r s\ i^\ell)\ (i\, o^\ell) \to (\pi^r s\, o^\ell)$; then by applying **move rule** (1) we obtain the type $(o^r \pi^r s)$ expecting the object to occur before the verb string. The clause-final verb clusters in Dutch and German have been estensively studied in different linguistic theories, see Steedman [1985], Haegeman and van Riemsdijk [1986], Moortgat [1997], Lambek [2000]: a common observation is that while German prefers nested dependencies, between verbs and their arguments, Dutch prefers crossed dependencies. Consider the following sentences where "geld": NP_2 and "Marie": NP_3 are arguments of "geven": V_2, "Piet": NP_1 is an argument of the perception verb "zag": V_1. In the second example, an embedded clause, the dependencies between the two verbs and their arguments are crossed.

Jan	zag	Piet	geld	Marie	geven	
Jan	saw	Piet	money	Marie	give	
π	$(\pi^r s\, i^\ell\, o_1^\ell)$	o_1	o_2	w	$(w^r o_2^r\, i)$	$\to$ s

...	Jan	Piet	Marie	geld	(zag	geven)	
...	Jan	Piet	Marie	money	(saw	give)	
	π	o_1	w	o_2	$(o_2^r\, w^r\, o_1^r \pi^r s)$	$\to$ s	

In the first example, "Jan zag Piet geld Marie geven" (*Jan saw Piet give money to Marie*), the type $(i\, w^\ell o_2^\ell)$ of "geven" is converted by **move rule** (1) into $(w^r o_2^r\, i)$ where o_1 = "Piet", o_2 = "geld", w = "Marie"; for $q = i$ and $p^\ell = (w^\ell o_2^\ell) = (o_2 w)^\ell$, we have $(o_2 w)^\ell \rightsquigarrow (o_2 w)^r = (w^r o_2^r)$. In the second example, first we apply **move rule** (1) to the type $(\pi^r s\, i^\ell\, o_1^\ell)$ of "zag" and obtain $(o_1^r \pi^r s\, i^\ell)$, for $p = o_1$; then we get the type of the verb string "zag geven" by contraction: $(o_1^r \pi^r s\, i^\ell)\ (i\, o_2^\ell w^\ell) \to (o_1^r \pi^r s\, o_2^\ell w^\ell)$; finally, applying again the cyclic rule, we obtain $(o_2^r\, w^r\, o_1^r \pi^r s)$, for $p^\ell = (w\, o_2)^\ell$. A similar analysis applies to the sentence "Jan Piet Marie zag laten zwemmen" (*Jan saw Piet make Marie swim*).

...	Jan	Piet	Marie	(zag laten zwemmen)	
...	Jan	Piet	Marie	(saw make swim)	
	π	o_1	o_2	$(o_2^r\, o_1^r\, \pi^r\, s)$	$\to$ s

4.2 Word order changes in Hungarian

Examples of still more radical word order changes are offered by languages such as Hungarian[4] , where the movement is caused by a change of focus in the sentence. Words move within the sentence to reflect or focus on a certain meaning. For instance the following Hungarian sentence, which has no focus in it, simply means "János took two books to Péternek yesterday".

[4] agglutinative

János tegnap elvitt két könyvet Péternek.
János yesterday took two books to Péternek.

This can become as follows

János	tegnap	**két könyvet**	vitt el	Péternek.
János	yesterday	**two books**	took	to Péternek.
π	λ	o	$(o^r\lambda^r\pi^r s w^\ell)$	w

which means "Only two books were taken by János to Péternek yesterday". This is an example of a *single move*: **két könyvet** has moved from after the verb to before it. More sophisticated movements are also possible, for instance in the following sentence

Péternek	vitt el	tegnap	János	két könyvet.
To Péternek	took	yesterday	János	two books.
w	$(w^r s o^\ell \pi^\ell \lambda^\ell)$	λ	π	o

which means "It was to Péternek and to no one else that the two books were taken". This is an example of a *multi move*: not only *Péternek* has moved to the beginning of the sentence, but also first *tegnap* and then *János* have moved from before the verb to after it, and in so doing have changed their order with regard to each other. For more details on single and multi moves and a formalization of a notion of *focus*, we refer the reader to Sadrzadeh [2010]; here instead we review some examples. In order to reason about these kinds of movement, we generalize our previous cyclic rules in the following way

Move Rule (2): If $p^r q$ is $\boxed{in}$ the type of the verb, so is qp^ℓ.

Move Rule (3): If qp^ℓ is $\boxed{in}$ the type of the verb, so is $p^r q$.

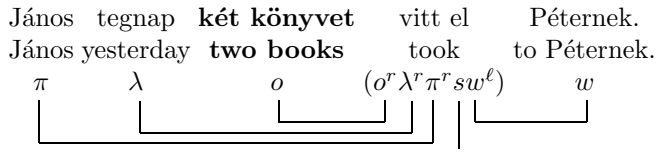
Taking π to stand for the type of the subject, o for the first object, w for the second object, and λ for the adverb, we assign the following types to the constituents of our example sentence, which had no focus in it yet

János	tegnap	elvitt	két	könyvet	Péternek.
π	λ	$(\lambda^r\pi^r s\, w^\ell o^\ell)$		o	w

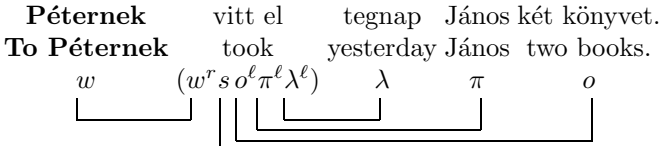
The focus can be on the subject or either of the objects. In each case, they will appear right before the verb after the movement. For the case of the subject, i.e. *János* the temporal adverb *yesterday* moves to after the verb, as follows

János	vitt el	tegnap	két	könyvet	Péternek.
János	took	yesterday	two	books	to Péternek.
π	$(\pi^r s w^\ell o^\ell \lambda^\ell)$	λ		o	w

We use our new cyclic rules to derive the new type of the verb as follows: apply **move rule** (2) to the type of the verb $(\lambda^r \pi^r s w^\ell o^\ell)$, by taking q to be $(\pi^r s\, w^\ell o^\ell)$ and p to be λ. If the focus is on the first object, i.e. *two books*, then it moves before the verb and the sentence above and its typing change as follows



To derive the new verb type, we apply **move rule** (3) to the original type of the verb $(\lambda^r \pi^r s w^\ell o^\ell)$, by taking q to be $(\lambda^r \pi^r s w^\ell)$, and p to be o. The focus can also be on the second object *Péternek* and the verb; for details see Sadrzadeh [2010]. As an example of multi move, consider our above sentence, typed as follows



Here *Péternek* has moved to the beginning of the sentence, but also first *tegnap* and then *János* have moved from before the verb to after it, and in so doing have changed their order with regard to each other. The calculation for deriving the new type of the verb reflects the above complications and needs repetitive applications of the rules. It is as follows

> Start from the original type of the verb $(\lambda^r \pi^r s w^\ell o^\ell)$ and, first *Péternek* moves to the front; to obtain this we apply rule (3) to the subtype $(\lambda^r \pi^r s w^\ell)$, take p to be w, and obtain $(w^r \lambda^r \pi^r s o^\ell)$. Then *tegnap* moves after the verb, for this we apply rule (2) to the subtype $(\lambda^r \pi^r s o^\ell)$, take p to be λ, and obtain $(w^r \pi^r s o^\ell \lambda^\ell)$. Finally *János* moves to after *tegnap*, for this we apply rule (2) to the subtype $(\pi^r s o^\ell)$, take p to be π, and obtain $(w^r s o^\ell \pi^\ell \lambda^\ell)$.

Free as they might seem, we need some restrictions to avoid certain over generations, mainly caused by the presence of the word *in*. Formulation of these exceed the purpose of this paper and can be found in Sadrzadeh [2010]. In a nutshell, they will prevent formation of types such as $(\pi^r \lambda^\ell s w^\ell o^\ell)$ and $(\lambda^r \pi^r s o^r w^\ell)$.

5 Clitic Rules and Cyclic Pregroups

Following Lambek [1999, 2001, 2008], we have formulated the clitic rules as *metarules*. At some risk of overgeneration, one is tempted to formulate these rules as axioms and add them to the pregroup calculus, or add their rule version to the sequent calculus of compact bilinear logic [Buszkowski 2001, 2002]. Note that the addition of our cyclic axioms (or cyclic rules) is not equivalent to the

reintroduction of the structural rule of *Commutativity* into the pregroup calculus (a logic without structural rules like the Syntactic Calculus)[5]. These axioms belong to the *cyclic* calculus studied by Abrusci [1991, 2002] and introduced in the following way

$$\frac{\vdash \Gamma, \Delta}{\vdash \Delta^{+2}, \Gamma}(rr) \qquad\qquad \frac{\vdash \Gamma, \Delta}{\vdash \Delta, \Gamma^{-2}}(ll)$$

Via the standard translation from the Syntactic Calculus to pregroups [Lambek 1999, Buszkowski 2001] (*positive* formulae as *right* adjoints and *negative* formulae as *left* adjoints), the axiomatic version of these rules becomes

$$(1)\ qp \leq pq^{ll} \qquad\qquad (2)\ qp \leq p^{rr}q$$

We can refer to (1) and (2) as *cyclic axioms*, in particular to the first one as the *left cyclic axiom* and to the second one as the *right cyclic axiom*. We can then re-formulate our clitic *metarules* as *clitic axioms*

$$\text{Persian} \quad p^r q \leq q\overline{p}^l \qquad\qquad \text{French-Italian} \quad qp^l \leq \overline{p}^r q$$

where the latter is derivable from the former, and prove the following results:

Proposition 1. *The* clitic *axioms are derivable from the* cyclic *axioms.*

Proof. The axiom for French and Italian is derivable form the *right cyclic axiom* as follows, take p to be p^l and observe that $(p^l)^{rr} = p^r$, then one obtains $qp^l \leq p^r q$. Since $\overline{p} \leq p$, and since adjoints are contravariant, we have $p^r \leq \overline{p}^r$, thus $p^r q \leq \overline{p}^r q$, and by transitivity of order we obtain $qp^l \leq \overline{p}^r q$. The axiom for Persian is derivable from the *left cyclic axiom* as follows: take q to be p^r and p to be q. Now since $(p^r)^{ll} = p^l$, we obtain $p^r q \leq qp^l$, and since $\overline{p} \leq p$, by contravariance, $p^l \leq \overline{p}^l$, thus $qp^l \leq q\overline{p}^l$, and by transitivity of order $p^r q \leq q\overline{p}^l$.

It is interesting that the rules for clitic movement correspond to logical rules of cyclicity. Accordingly, one may call French and Italian *right cyclic* languages and Persian a *left cyclic* language. The consequences of enriching a pregroup with these cyclic axioms (or rules) are however not so desirable.

Proposition 2. *A pregroup P with either of the cyclic axioms is a partially ordered group.*

Proof. Consider the left cyclic axiom; if one takes $q = 1$, we obtain $p^r \leq p^l$ for all $p \in P$, from which one obtains $p^{ll} \leq p$. Here take $p = w^r$ for some $w \in P$ and obtain $w^l \leq w^r$. Now since we have $p^r \leq p^l$ for all p, we obtain $w^r = w^l$. A similar argument can be made for the right cyclic axiom.

[5] An approach in this line is proposed by Francez and Kaminski [2007], where a free pregroup grammar is extended by a finite set of additional (commutative) inequations between types, leading to a class of mildly context-sensitive languages, allowing the analysis of crossed dependencies and extractions.

Although, as proven by Abrusci and Lambek, cyclic bilinear logic is a conservative extension of bilinear logic (or non-commutative linear logic), this is not the case for cyclic compact bilinear logic and compact bilinear logic (the logical calculus of pregroups) [Lambek 2008, Barr 2004]. The relations among these system are however of real interest to be studied both from the logical and, particularly, from the linguistic point of view.

We conclude observing that the present analysis is consistent with previous work on French [Bargelli and Lambek 2001] and Italian [Casadio and Lambek 2001], where *iterated* adjoints are used to type clitic pronouns. We can prove in fact that iterated adjoints show up in our work too, since as observed by Lambek, the $\overline{p}^r$ used in the metarule for French and Italian is nothing but $(\overline{p}^l)^{rr}$, and the $\overline{p}^l$ used for Persian is nothing but $(\overline{p}^r)^{ll}$.

6 Conclusions

We have applied the calculus of pregroups to a selected set of sentences involving word order changes in different languages: Persian, French, Italian, Dutch and Hungarian. The cross-linguistic results we have obtained provide evidence in favour of the theoretical and computational advantages offered by the pregroup calculus extended with appropriate cyclic rules. These rules in turn represent a stimulating challenge for the development of logical grammars. We have in fact shown that those calculations, or computations, that in pregroups are dealt with logical types involving double adjoints (corresponding to Chomskian traces), can be performed, in the different languages, by means of appropriate cyclic operations.

References

1. Abrusci, V. M.: Phase Semantics and Sequent Calculus for Pure Noncommutative Classical Linear Propositional Logic. J. Symbolic Logic 56(4), 1403–1451 (1991)
2. Abrusci, M.: Classical Conservative Extensions of Lambek Calculus. *Studia Logica* 71, 277–314 (2002).
3. Abrusci, V.M., Casadio C. eds: New Perspectives in Logic and Formal Linguistics. Proceedings of the 5th Roma Workshop. Rome, Bulzoni (2002)
4. Ajdukiewicz, K.: Die syntaktische Konnexitaat, Studia Philosophica, 1, 1-27 (1935). Eng. trans., Syntactic connexion. In Polish Logic, ed. S. McCall. Oxford, Clarendon Press (1967)
5. Bargelli, D., Lambek J.: An Algebraic Approach to French Sentence Structure. In *Logical Aspects of Computational Linguistics*, edited by P. de Groote, G. Morrill, and C. Retoré, 62–78. Berlin, Springer-Verlag (2001)
6. Barr, M.: *-Autonomous Categories Revisited, Journal of Pure and Applied Algebra, 111, 1–20 (1996)
7. Barr, M.: On Subgroups of The Lambek Pregroup. *Theory and Application of Categories* 12(8), 262–269 (2004)
8. Buszkowski, W.: Lambek Grammars Based on Pregroups. In *Logical Aspects of Computational Linguistics* edited by P. de Groote, G. Morrill, and C. Retoré, 95–109. Berlin, Springer-Verlag (2001)

9. Buszkowski, W.: Type Logics and Pregroups. *Studia Logica* 87(2–3), 145-169 (2007)
10. Buszkowski, W., Moroz, K.: Pregroup Grammars and Context-free Grammars. In Casadio and Lambek eds., 1–21 (2008)
11. Casadio, C.: Non-Commutative Linear Logic in Linguistics. *Grammars* 4(3), 167–185 (2001)
12. Casadio, C.: Applying Pregroups to Italian Statements and Questions. *Studia Logica* 87, 253–268 (2007)
13. Casadio, C., Lambek, J.: An Algebraic Analysis of Clitic Pronouns in Italian. In *Logical Aspects of Computational Linguistics* edited by P. de Groote, G. Morrill, and C. Retoré, 110–124. Berlin, Springer-Verlag (2001)
14. Casadio, C., Lambek, J.: A Tale of Four Grammars. *Studia Logica* 71(2), 315–329 (2002)
15. Casadio, C., Lambek, J. (eds.): *Recent Computational Algebraic Approaches to Morphology and Syntax*. Milan, Polimetrica (2008)
16. Casadio, C., Sadrzadeh, M.: Clitic Movement in Pregroup Grammar: a Cross-linguistic Approach. Proceeding 8th International Tbilisi Symposium on Language, Logic and Computation, Springer (2011)
17. Chomsky, N.: *Lectures on Government and Binding*. Dordrecht, Foris (1981)
18. Chomsky, N.: Barriers. Cambridge, The MIT Press (1986)
19. Francez, N., Kaminski, M: Commutation-Augmented Pregroup Grammars and Mildly Context-Sensitive Languages. Studia Logica 87(2/3), 295-321 (2007)
20. Grishin, V. N. : On a generalization of the Ajdukiewicz-Lambek system. In Studies in Nonclassical Logics and Formal Systems, Moscow, Nauka 315-343 (1983). Eng. trans. by D. Cubric, edited by author. In Abrusci and Casadio (eds.) 9-27 (2001)
21. Haegeman, L., van Riemsdijk, H.: Verb Projection Raising, Scope, and the Typology of Rules Affecting Verbs. Linguistic Inquiry, 17 (3), 417–466 (1986)
22. Harris, Z. S.: *Methods in Structural Linguistics*. Chicago (1951)
23. Harris, Z. S.: A Cycling Cancellation-Automaton for Sentence Well-Formedness. International Computation Centre Bulletin 5, 69–94 (1966)
24. Harris, Z. S.: *Mathematical Structures of Language*, Interscience Tracts in Pure and Applied Mathematics, John Wiley & Sons., New York (1968)
25. Kiślak - Malinowska, A.: Pregroups as a tool for typing relative pronouns in Polish, *Proceedings of Categorial Grammars. An Efficient Tool for Natural Language Processing*, Montpellier, 114-128 (2004)
26. Kiślak - Malinowska, A.: Polish Language in Terms of Pregroups. In: Casadio, C., Lambek, J. (eds.) Recent Computational Algebraic Approaches to Morphology and Syntax. Polimetrica, Milan, 145–172 (2008)
27. Klavans, J. L.: *Some Problems in a Theory of Clitics*. Bloomington, Indiana Linguistics Club (1982)
28. Kusalik, T.: Product Pregroups as an Alternative to Inflectors. In *Recent Computational Algebraic Approaches to Morphology and Syntax*, edited by C. Casadio and J. Lambek, 173–190. Milan, Polimetrica (2008)
29. Lambek, J.: The Mathematics of Sentence Structure. *American Mathematics Monthly* 65, 154–169 (1958)
30. Lambek, J. : Deductive Systems and Categories I. Syntactic Calculus and Residuated Categories. *Mathematical Systems Theory*, 2(4), 287–318 (1968)
31. Lambek, J.: From Categorial Grammar to Bilinear Logic. In Doŝen, K., P. Schroeder-Heister, eds. *Substructural Logics*. Oxford, Oxford University Press, 207–237 (1993)
32. Lambek, J.: Type Grammar Revisited. In *Logical Aspects of Computational Linguistics*, edited by A. Lecomte et al., 1–27. Springer LNAI 1582 (1999)

33. Lambek, J.: Type Grammar Meets German Word Order. *Theoretical Linguistics* 26, 19–30 (2000)
34. Lambek, J.: Type Grammars as Pregroups. *Grammars* 4(1), 21–39 (2001)
35. Lambek, J.: A computational Algebraic Approach to English Grammar. *Syntax* 7(2), 128–147 (2004)
36. Lambek, J.: From Word to Sentence: a Pregroup Analysis of the Object Pronoun Who(m). *Journal of Logic, Language and Information* 16, 303–323 (2007)
37. Lambek, J.: From Word to Sentence. A Computational Algebraic Approach to Grammar. Polimetrica, Monza (MI) (2008)
38. Monachesi, P.: *A Grammar of Italian Clitics*. ITK Dissertation Series, Tilburg (1995)
39. Morrill, G.: Categorial Grammar. Logical Syntax, Semantics, and Processing. Oxford University Press, Oxford (2010)
40. Moortgat, M.: Categorical Type Logics. In *Handbook of Logic and Language*, edited by J. van Benthem and A. ter Meulen, 93–177. Amsterdam: Elsevier (1997)
41. Moortgat, M.: Symmetric Categorial Grammar. J. Philos. Logic 38, 681–710 (2009)
42. Preller, A., Lambek, J.: Free Compact 2-categories. *Mathematical Structures for Computer Sciences* 17, 309–340 (2007)
43. Preller, A., Prince, V.: Pregroup Grammars with Linear Parsing of the French Verb Phrase. In *Recent Computational Algebraic Approaches to Morphology and Syntax* edited by C. Casadio and J. Lambek, 53-84. Milan, Polimetrica (2008)
44. Preller, A., Sadrzadeh, M.: Semantic Vector Space and Functional Models for Pregroup Grammars. Journal of Logic, Language and Information (2011)
45. Sadrzadeh, M.: Pregroup Analysis of Persian Sentences. In Casadio and Lambek eds., 121–143 (2008)
46. Sadrzadeh, M.: An Adventure into Hungarian Word Order with Cyclic Pregroups. In AMS-CRM proceedings of Makkai Fest. (2010)
47. Steedman, M.: Dependency and Cordination in the Grammar of Dutch and English. Language, 61 (3), 523–568 (1985)
48. Stabler, E. P.: Tupled Pregroup Grammars. In *Recent Computational Algebraic Approaches to Morphology and Syntax* edited by C. Casadio and J. Lambek, 23–52. Milan, Polimetrica (2008)
49. Yetter, D. N.: Quantales and (non-Commutative) Linear Logic. Journal of Symbolic Logic, 55 (1990)
50. Wanner, D.: *The Development of Romance Clitic Pronouns. From Latin to Old Romance*. Amsterdam, Mouton de Gruyter (1987)
51. Zwicky, A. M., Pullum, G. K.: Cliticization vs. Inflection: English *n't*. *Language* 59(3), 502-513 (1983)

TERENCE: An Adaptive Learning System for Reasoning about Stories with Poor Comprehenders and their Educators

Tania di Mascio[1], Rosella Gennari[2], and Pierpaolo Vittorini[3]

[1] DIEI – University of L'Aquila – Via G. Gronchi – 67100, L'Aquila, IT.
[2] CS Faculty, Free University of Bozen-Bolzano - P.za Domenicani, 3 – 39100 Bolzano, IT.
[3] MISP – University of L'Aquila – P.le S. Tommasi, 1 – 67100 Coppito, L'Aquila, IT.

Abstract. Text comprehension skills and strategies develop enormously from the age of 7-8 until the age of 11, when children advance as independent readers. Nowadays, more and more young children turn out to be poor (text) comprehenders: they demonstrate text comprehension difficulties, related to inference-making skills, despite proficiency in word decoding and other low-level cognitive skills. Though there are several pencil-and-paper reading interventions for improving inference-making skills on text, and specifically addressed to poor comprehenders, the design and development of adaptive learning systems for this purpose are lagging behind. The use of more intelligent adaptive learning systems to tailor such interventions to poor comprehenders has a tremendous potential. TERENCE embodies that potential. It is a Collaborative Project funded by the EC under the ICT Call 5 FP7-ICT-2009-5 (1 October 2010-30 September 2013). TERENCE will design and develop an intelligent adaptive learning system. In particular, the system's smart games will ask children to draw inferences about events of stories, in Italian and in English. Moreover, the system will allow teachers to choose and custom-tailor the types of stories and games according to the needs of their learners. The TERENCE consortium involves European experts in diverse and complementary fields (art and design, computer science, engineering, linguistics, evidence-based medicine, psychology), and sees the constant involvement of the end-users (poor comprehenders and their educators) from schools in the UK and in Italy. The paper overviews the TERENCE project.

1 Introduction

Developing the capabilities of children to comprehend written texts is key to their development as young adults. From the age of 7-8 until the age of 11, children develop as independent readers. Nowadays, more and more children in that age range turn out to be poor (text) comprehenders: they demonstrate difficulties in deep text comprehension, despite well developed low-level cognitive skills like vocabulary knowledge, e.g., see [2] for hearing poor comprehenders, and [5] for deaf poor comprehenders. In particular, several studies experimentally demonstrate that poor comprehenders fail to master the following reasoning skills in processing written stories:

(s1) coherent use of cohesive devices such as temporal connectives,

(s2) inference-making from different or distant parts of a text, integrating them coherently,
(s3) detection of inconsistencies in texts.

Nowadays, there is clear evidence that such reasoning skills (s1, s2, and s3) are very likely to be causally implicated in the development of deep text comprehension. In particular, experiments show that inference-making questions centred around (s1, s2, and s3), together with adequate visual aids, are pedagogically effective in fostering deep comprehension of stories, e.g., see [4].

However, finding stories and educational material that are appropriate for poor comprehenders is a challenge, and hence educators are left alone in their daily interaction with poor comprehenders. Most learning material for novice comprehenders is paper based, and is not easily customisable to the specific requirements of poor comprehenders, e.g., in the types, number or position of temporal connectives. Few systems promote general reading interventions, but they have high-school or university textbooks as learning material, instead of stories, and are developed for old children or adults, and not specifically for poor comprehenders.

TERENCE is a Collaborative Project funded by the EC under the ICT Call 5 FP7-ICT-2009-5 (1 October 2010-30 September 2013) and aims at designing and developing an intelligent *adaptive learning system* (ALS) [1]. The main end-users of the TERENCE ALS are: *learners*, namely, primary-school poor comprehenders, hearing and deaf, older than 7-8; *educators*, namely, primary-school teachers and support teachers, as well as parents of the TERENCE learners. The learning material (in English and in Italian) of the TERENCE ALS will be stories adapted to the specific requirements of poor comprehenders, and its reading interventions will be mainly interactive question-games centred around reasoning skills, like (s1), (s2), and (s3) above, that foster the development of deep text comprehension, both accompanied by adequate visual aids.

This paper outlines the TERENCE project and the work already conducted therein. It first specifies the chosen design methodology in Section 2. Then it outlines the conceptual model of the TERENCE ALS in Section 2. It continues with the design of the architecture in Section 4 and ends with a recap conclusive section.

2 The Design Methodology

The TERENCE system is developed following the user-centred design (UCD) methodology [3]. Generally speaking, the UCD places the end user, actively, at the centre of the design process, which is iteratively repeated until attaining the usability of the system. The iterative process revolves around the following main activities: a) analysis and specification of the context of use; b) analysis and specification of the user requirements; c) design prototypes; d) evaluate the prototypes against the specification of the requirements. In TERENCE we went through the first phase of the analysis of the context of use and the requirements, which resulted into the semi-formal specification of:

1. the characteristics of the users, like knowledge, skills, experience, education, training, physical attributes, habits and capabilities;

2. the tasks, like successful reading interventions by class teachers for improving reading comprehension;
3. the environments, divided into organisational, physical and socio-cultural characteristics that may influence the usage and acceptance of the system.

Our design of the conceptual model of the TERENCE ALS, which is outlined in Section 3 below, is based on such specifications and on the study of the state of the art of conceptual modelling for ALSs. In the UCD, the design evolves cyclically through refinements of the requirements, thus also the conceptual models of TERENCE will be updated accordingly. The following section outlines the current conceptual model.

3 The TERENCE Conceptual Model

As mentioned above, TERENCE is developed as an ALS. Its conceptual model includes:

1. the *domain model* for the learning material, that are stories and games,
2. the *user model* for the users, including the educator and learner sub-models,
3. the *adaptation model* for the adaptation learning process.

In the remainder, for space limitations, we only sketch the design of the domain model, and of the learner sub-model of the user model.

3.1 The Learner Sub-Model

The study of the state of the art for user modeling in ALSs and the specifications of the user characteristics, in particular, the poor comprehenders' reading skills, allowed the consortium to define the following sub-models of the user model: the learner sub-model; the educator sub-model; the expert sub-model.

The learner sub-model consists of three parts. The first structures general basic data about the learner such as name. The second is really specific to TERENCE: it structures information concerning the reading comprehension skills of poor comprehenders, analysed and specified in semi-formal format by the TERENCE consortium. By analysing such skills, the system will be able to adapt, to its learner, its learning material, structured as in the domain model. The third part of the learner sub-model structures data about the learner's interaction with the system.

3.2 The Domain Model

The main learning material of the TERENCE ALS consists in illustrated stories and games. During the analysis of the requirements and of the context of use of TERENCE, the tasks of the users were analyses and specified. Their specification allowed us to structure the learning material as in the domain model. This is divided into the story and game sub-models.

The story model represents data related to the stories that the learners interact with. For instance, stories are structured into books, each book has a genre and related avatars.

Furthermore, each story is annotated using natural language processing tools. The annotations, for instance, provide us with the events of which the story is made, specifying their actors, location and temporal relations.

Each story has associated games. Preliminary prototypes of the games are being designed by following assessed interventions for stimulating the reasoning skills on texts outlined in the introductory section and resulting from our user requirements' semi-formal analysis. For instance, according to this, poor comprehenders are in need of inference-making games that pose and solicit questions about relations between events in the stories, monitoring the learners' comprehension of the story flow, e.g., "Does the big eggshell crack before Mummy Duck watches it?". The game model represents the data for the TERENCE games, which are divided into smart games and relaxing games. Smart games are centered around inference-making questions that are built upon the annotations of the stories. As such, smart games are taxing for poor comprehenders and need to be paused by relaxing games (e.g., draw your favourite character), which keep the learners' attention and motivation alive. More details concerning their generation and resolution are in the following section.

4 The Architecture of the TERENCE System

The current logical architecture of our ALS is divided into three layers, namely, the data layer, the application layer, and the interaction layer. The data layer stores, for instance, the stories and games according to the domain model, and the information specific to each learner according to the learner model.

The application layer implements the adaptation engine, and the intelligent backbone responsible for the feedback on games (e.g., correct or incorrect answers to games). Finally, the interaction layer contains the users' interfaces.

The main modules of the adaptation engine of our ALS are a constraint-based automated reasoner and a natural language processing (NLP) module, consisting of a processor for English stories and one for Italian stories, that lay at the core of the adaptation engine and constitute its intelligent backbone, so to speak. The NLP modules serve primarily to annotate stories with specific XML tags. The tags are used for classifying stories as in the story sub-model, as well as for generating and giving feedback on the TERENCE smart games. According to the assessment of the students' performance on a class of games, the adaption engine will attempt to guide the students to the most adequate games and stories, following the adaptation model.

In the specific context of our ALS, tasks will be implemented through web services and their composition. For instance, let us refer to the generation of a temporal question-game centred around the question "Does the big eggshell crack *before* Mummy Duck watches it?". The composition will go as follows:

1. firstly, the system invokes an NLP web service operation, which takes the story as input and returns the annotations in the novel annotation language;
2. secondly, these annotations are taken as input, and an operation of the Automated Reasoner service deduces further temporal annotations as output, updating them;
3. finally, the updated annotations are taken as input and a further operation of the NLP web service generates as output the grammatically correct question-game.

A first clear advantage of such an approach is that it allows for the reuse of our architecture in other languages, by implementing the appropriate NLP services. Furthermore, since web services are accessible through HTTP calls, they can be invoked directly in their respective organisations, e.g. for keeping protected any eventual patent. Finally, the high-level operations might also be implemented with a programming-in-the-large paradigm, e.g. BPEL, thus allowing for an easy deploy of further operations, that become web services, and therefore re-usable to build up more complex tasks.

5 Conclusions

TERENCE is European project for the design and evaluation of an ALS specific for poor comprehenders and their educators. More in general, the project aims at offering innovative usability and evaluation methodologies, pedagogical models, AI technologies, and an ALS for reasoning about stories through smart games, in Italian and in English, all developed via a coordinated and cross-disciplinary effort of European experts in diverse and complementary fields (art and design, computer science, engineering, linguistics, psychology), and with the constant involvement of the end-users (poor comprehenders, deaf children and their educators) from schools in the UK and in Italy.

Acknowledgments. The authors' work was supported by TERENCE project. TERENCE is funded by the European Commission through the Seventh Framework Programme for RTD, Strategic Objective ICT-2009.4.2, ICT, Technology-enhanced learning. The contents of the paper reflects only the authors' view and the European Commission is not liable for it.

References

1. P. Brusilovsky. Adaptive Hypermedia. *User Modeling and User-Adapted Interaction*, 11:87–110, March 2001.
2. K. Cain and J. Oakhill, editors. *Children's Comprehension Problems in Oral and Written Language: A Cognitive Perspective*. Guildford Press, 2007.
3. J. Gulliksen, B. Göransson, I. Boivie, S. Blomkvist, J. Persson, and A. Cajanger. Key Principles for User-centred Systems Design. 22(6):397–409, 2003.
4. National Reading Panel. Teaching Children to Read: An evidence-based Assessment of the Scientific Research Literature on Reading and its Implications for Reading Instruction. Technical report, National Institute of Child Health and Human Development, 2000.
5. P. Spencer and M. Marschark. *Evidence-based Practice in educating Deaf and Hard-of-hearing Students*. Oxford University Press, New York, NY, USA, 2010.

Nested Weight Constraints in ASP*

Stefania Costantini[1] and Andrea Formisano[2]

[1] Università di L'Aquila, Via Vetoio, Loc. Coppito, I-67010 L'Aquila, Italy
stefania.costantini@univaq.it
[2] Università di Perugia, via Vanvitelli, 1, I-06123 Perugia, Italy
formis@dmi.unipg.it

Abstract. Weight constraints are a powerful programming construct that has proved very useful within the Answer Set Programming paradigm. In this paper, we argue that practical Answer Set Programming might take profit from introducing some forms of nested weight constraints. We define such empowered constraints (that we call "Nested Weight Constraints") and discuss their semantics and their complexity.

1 Introduction

Answer Set Programming (ASP for short) [8], has evolved over more than two decades as a paradigm that allows for very elegant solutions to many combinatorial problems: in fact, ASP has been successfully applied to many forms of knowledge representation and commonsense reasoning (cf. among others, [1, 7] and the references therein). The paradigm is based upon describing a problem by a logic program in such a way that its answer sets correspond to the solutions of the considered problem.

The ASP paradigm has become even more powerful by extending ASP programs by means of weight constraints [11, 13]. Intuitively, weight constraints allow one to associate weights to the literals occurring in specific subsets of a (candidate) model. Then, bounds can be imposed on the overall weight of each subset. A model is accepted if all these bounds are satisfied. Cardinality constraints are a special case where all weights are equal to one. Weight constraints have proved to be a very useful programming tool in many applications such as planning and configuration. For instance, in the product configuration domain, we need to express cardinality, cost, and resource constraints, which are very difficult to capture using logic programs without weights.

Weight constraints are nowadays adopted (in some form) by most of the ASP inference engines (usually called "ASP solvers").

All common algorithmic tasks related to programs with weight constraints, such as checking the consistency of a program (i.e., whether a program admits stable models), are intractable [5]. Though, as shown in [12], tractability can be achieved by imposing some restrictions on program structure.

We propose an improved form of constraints that admits nesting of weight constraints. Syntactically, nesting allows one to specify a set of weight constraints within a

* Research partially funded by GNCS-2011 and MIUR-PRIN-2008 projects, and grants 2009.010.0336 and 2010.011.0403. We are grateful to the anonymous referees for their helpful advice and suggestions.

"container" weight constraint. In turn, such "contained" constraints may include other constraints, and so on. Semantics is given by requiring that the satisfaction of the internal constraints has to be evaluated with respect to the *context* defined by the containing constraints. Hence, the new construct introduces a form of locality in program rules: two identical weight constraints might be differently evaluated depending on the context in which they occur. We will see that nesting can be introduced without affecting complexity.

We argue that practical ASP programming might take profit from the introduction of nested weight constraints. In particular, our proposal is aimed at improving *elaboration tolerance* where, [10]:

> "A formalism is elaboration tolerant to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances. Representations of information in natural language have good elaboration tolerance when used with human background knowledge. Human-level AI will require representations with much more elaboration tolerance than those used by present AI programs, because human-level AI needs to be able to take new phenomena into account. The simplest kind of elaboration is the addition of new formulas. We'll call these additive elaborations. Next comes changing the values of parameters. Adding new arguments to functions and predicates represents more of a change. However, elaborations not expressible as additions to the object language representation may be treatable as additions at a meta-level expression of the facts ..."

One can say that elaboration tolerance implies the ability to cope with minor changes to input problems without major revisions. The introduction of constructs involving forms of locality, as well as modularity, goes in this direction. In what follows, we will take a sample problem (which is however a representative of a wide class) and we will show that the formalization in ASP benefits from the use of nested weight constraints.

The paper is structured as follows. In Section 2 we recall the notions of weight (and cardinality) constraints. Section 3 introduces the enhancements we intend to propose, for the case of ground programs. In Section 4 we further extend the formalism by introducing conditional literals [11] and the use of variables to denote collections of literals. An example is exploited in Section 5 to illustrate nested weight constraints. The complexity issue is addressed in Section 6. Finally, in Section 7 we conclude.

2 Weight and Cardinality Constraints in ASP

Weight and cardinality constraints were introduced in [11, 13], where their semantics is also presented, as well as their implementation in the context of the ASP solver smodels. Deciding whether a program involving ground weight constraints has an answer set is still NP-complete, and computing an answer set is still FNP-complete. Though the computational complexity remains the same, the modeling power of the extended language is higher, as proved by the wide application of this construct.

In what follows we recall the syntax and semantics of (ground) programs with weight constraints by abstracting away from any particular concrete syntax. We assume known the usual notions of constant, predicate, term, atom, literal, etc. Let us consider

as fixed an underlying language and consequently let $\mathcal{B}$ denote the corresponding Herbrand base, namely the set of all ground atoms of the given language.

Atoms have the form $p(t_1, \ldots, t_k)$ where p is a predicate symbol and each t_i is a term. For a literal ℓ, let $\pi(\ell)$ denote the predicate symbol of ℓ (e.g., $\pi(p(t_1, \ldots, t_k) = p$). For a set of literals S, let $\pi(S) = \{\pi(\ell) | \ell \in S\}$.

A *weight literal* over is a pair (a, j) or $(\neg a, j)$ for $a \in \mathcal{B}$ and $j \in \mathbb{N}$, the weight of the literal and $\neg$ denotes *default negation*.[3] A *weight constraint* is a triple (S, l, u) where S is a set of weight literals and $l \leq u$ are non-negative integers, the lower and upper bound. We will often use the symbol ∞ to denote an arbitrarily large upper bound. (This will be useful in situations in which the upper bound is not specified.) Moreover, as a shorthand notation, we denote by a the weight constraint $(\{(a, 1)\}, 1, 1)$.

For a given constraint $c = (S, l, u)$, we indicate S with $Cl(c)$, l with $l(c)$ and u with $u(c)$. A weight constraint where for every weight literal (a, j) and $(\neg a, j)$ we have $j = 1$ is called a *cardinality constraint*.

A rule r is a pair (h, b) where h (the head) is a weight constraint and b (the body) is a set of weight constraints. We indicate h with $\mathrm{H}(r)$ and b with $\mathrm{B}(r)$.

A (ground) program with weight constraints (for short, PWC) is a set of rules.

Given a weight constraint c and a set of atoms I, we define the *weight* of c in I as $W(c, I) = \sum_{(a,j) \in Cl(c) \wedge a \in I} j + \sum_{(\neg a,j) \in Cl(c) \wedge a \notin I} j$.

A set of atoms I is a model of c (denoted by $I \models c$) iff $l(c) \leq W(c, I) \leq u(c)$. (Notice that the second inequality always holds if $u(c) = \infty$.)

For a set of weight constraints C, $I \models C$ iff $I \models c$ for all $c \in C$. Moreover, I is a model of a rule r (denoted by $I \models r$) iff $I \models \mathrm{H}(r)$ whenever it $I \models c$ holds for each $c \in \mathrm{B}(r)$. For a set of rules R, $I \models R$ iff $I \models r$ for all $r \in R$.

Stable models of a PWC are obtained by means of an extension to the GL-reduct [1] that, instead of removing rules where some negative literals in the body are not modeled in a given set of atoms (candidate stable model) I, it removes rules where the upper bound of some weight constraints in the body are not satisfied. The upper bounds of constraints are removed and the lower bounds are rearranged in order to eliminate negative literals. Each rule r is then replaced by a set of rules each of them having as head one of the positive literals in $\mathrm{H}(r)$ which belongs to I. In this manner, a positive PWC is obtained where the heads of rules are atoms. Finally, I is a stable model if it is the unique minimal model of this resulting program. Following [13], we have:

Definition 1 (PWC Semantics). *Let P be a PWC and let $I \subseteq \mathcal{B}$. The* reduct c^I *of a constraint c w.r.t. I is obtained from c by removing all negative literals, by setting the upper bound to be ∞, and by replacing the lower bound with the value $l' = max\big(0, l(c) - \sum_{(\neg a,j) \in Cl(c) \wedge a \notin I} j\big)$.*

The reduct P^I of the program P w.r.t. I is obtained by first removing each rule whose body contains a constraint c with $W(c, I) > u(c)$. Afterwards, each remaining rule r is replaced by the set of all rules of the form (h, b), for $(h, j) \in Cl(\mathrm{H}(r))$ such that $h \in I$ and $b = \{c^I : c \in \mathrm{B}(r)\}$.

[3] For the sake of simplicity, in this paper we will deal with non-negative integer weights only. Generalizations involving negative values, as well as real numbers, are possible [13].

The set I is a stable model of P iff it is a model of P^I and there exists no $J \subsetneq I$ such that J is a model of P^I.

3 Nested Weight Constraints

In this section we introduce an extension of ASP where weight constraints can be arbitrarily nested. As we will see, in this extension one can specify within an "external" weight constraint a collection of "internal" weight constraints. These represent conditions on the satisfiability of the outer constraint. Conversely, the external constraint affects the interpretation of internal weight literals and defines the *local context* where these weights literals have to be evaluated.

Definition 2. *A* nested weight constraint *(NWC) is a tuple (S, N, l, u) where*

- *S is a finite set of weight literals,*
- *$l \leq u$ are two non-negative integers (as before u can be ∞),*
- *N is a (possibly empty) finite collection of nested weight constraints*

The definitions of rule and program are given as one expects. We also extend to NWCs the notation introduced earlier and, moreover, for any given NWC $c = (S, N, l, u)$ we denote N with $N(c)$. The *depth* of a given NWC c, denoted by $depth(c)$, is defined as:

$$depth(c) \;=\; \begin{cases} 1 & \text{if } N(c) = \emptyset \\ 1 + \max_{c' \in N(c)} depth(c') & \text{otherwise} \end{cases}$$

The depth of a given program P is the maximum value among the depths of its NWCs.

For the purposes of this paper, it is not restrictive to assume the finiteness of the Herbrand universe of the underlying language. We will also consider only programs with finite depth.

The notion of satisfaction for NWCs requires some preliminary definitions. In particular, let $X, Y \subseteq \mathcal{B}$ be two disjoint sets of atoms and $c = (S, N, l, u)$ an NWC. Then, we define the *weight* of the constraint c (w.r.t. X, Y) as follows:

$$W(c, X, Y) = \sum_{(a,j) \in S \wedge a \in X} j + \sum_{(\neg a, j) \in S \wedge a \in Y} j \tag{1}$$

We say that a pair of sets of atoms X, Y satisfies the NWC $c = (S, N, l, u)$, and write $(X, Y) \models c$, if the following two conditions hold:

1. $l \leq W(c, X, Y) \leq u$;
2. for all $c' \in N$ it holds that $(U, V) \models c'$, where

$$\begin{aligned} U &= \{a \mid a \in X \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in X \wedge a \in S\} \\ V &= \{a \mid a \in Y \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in Y \wedge \neg a \in S\} \end{aligned} \tag{2}$$

Given a set of atoms I we say that I models an NWC c and write $I \models c$, iff $(I, \mathcal{B} \setminus I) \models c$. For a set Q of NWCs we write $I \models Q$ iff $I \models c$ for each $c \in Q$.

Notice that, in absence of nesting, namely, for an NWC $c = (S, N, l, u)$ with $N = \emptyset$, we obtain the notion introduced earlier for weight constraints. If $N \neq \emptyset$, the satisfaction of c also depends on the satisfaction of the NWCs in N. In turn, the

satisfaction of each $c' \in N$ has to be evaluated *within the context* determined by S. In particular, consider the above definition of satisfaction for an NWC c. The weight of a nested constraint $c' \in N$ is evaluated by considering only those atoms belonging to the subsets $U \subseteq X$ and $V \subseteq Y$ (recall that for the overall constraint c we have $X = I$ and $Y = \mathcal{B} \setminus X$ for a given set of atoms I). In this way, all weight literals in c' having an atom in $\mathcal{B} \setminus (U \cup V)$ are assumed to have null weight. More precisely, in evaluating the weight of c' we ignore the weights of all literals ℓ with $\pi(\ell) \in \pi(S)$ not occurring in S. The same procedure is recursively applied in evaluating the weights of the constraints $c'' \in N(c')$, and so on.

Note that, the above definition of satisfaction implicitly exploits, in (2), a partition of a Herbrand base $\mathcal{B}$. Each block of this partition corresponds to a single predicate symbol and consists of all the atoms having such leading symbol. Observe that the approach can be generalized since any partition of $\mathcal{B}$ can be used.

As before, a set of atoms I is a model of a rule r (denoted by $I \models r$) iff $I \models \mathrm{H}(r)$ whenever $I \models \mathrm{B}(r)$ holds. Given a program P, $I \models P$ iff $I \models r$ for all $r \in P$.

Now, we adapt the notion of reduct to deal with the nesting of constraints. Given an NWC $c = (S, N, l, u)$ and a pair of disjoint sets of atoms X, Y, the *reduct* of c w.r.t. X, Y is so defined (a denotes an atom):

$$c^{(X,Y)} = \big(\{(a,j) | (a,j) \in S\},\ \{d^{(U,V)} | d \in N\},\ \max(0, l - \textstyle\sum_{(\neg a, j) \in S \wedge a \in Y} j),\ \infty\big)$$

where U and V are obtained from X, Y and S as explained earlier (cf., (2) of page 4).

For a set Q of NWCs we denote by $Q^{(X,Y)}$ the set $\{c^{(X,Y)} \mid c \in Q\}$.

Given a program with NWCs, the reduct P^I of P w.r.t. a set of atoms I is so defined:

$$P^I = \Big\{ \big(a, (\mathrm{B}(r))^{(I, \mathcal{B} \setminus I)}\big) \mid r \in P,\ (a,j) \in Cl(\mathrm{H}(r)),\ a \in I, \\ W(c, I, \mathcal{B} \setminus I)) \leq u(c) \text{ for all } c \in \mathrm{B}(r) \Big\} \tag{3}$$

Notice that each rule r in P^I has an head of the form $(\{(a,1)\}, 1, 1)$, for some atom a. Moreover, no negative literal occurs in the body of r. Similarly to the case of ordinary weight constraints, we introduce an operator T_{P^I} defined as follows:

$$T_{P^I}(J) = \{a \mid \exists\, r \in P^I,\ a = \mathrm{H}(r),\ J \models \mathrm{B}(r)\} \tag{4}$$

Proposition 1. *Given a program P^I and two sets of atoms J_1 and J_2, if $J_1 \subseteq J_2$ then $T_{P^I}(J_1) \subseteq T_{P^I}(J_2)$.*

Proof. (Sketch). Let $a \in T_{P^I}(J_1)$. There exists a rule $r \in P^I$ of the form $(a, \mathrm{B}(r)$ such that $J_1 \models \mathrm{B}(r)$. Hence, for each NWC $c = (S, N, l, \infty) \in \mathrm{B}(r)$, we have that $l \leq W(c, J_1, \mathcal{B} \setminus J_1)$ and for each $c' \in N$, $(U_1, V_1) \models c'$, with $U_1 = \{a \mid a \in J_1 \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in J_1 \wedge a \in S\}$ and $V_1 = \{a \mid a \in (\mathcal{B} \setminus J_1) \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in (\mathcal{B} \setminus J_1) \wedge \neg a \in S\}$. If $J_1 \subseteq J_2$, then $l \leq W(c, J_2, \mathcal{B} \setminus J_2)$ plainly follows because there are no negative literals in S. Observe now that $U_2 = \{a \mid a \in J_2 \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in J_2 \wedge a \in S\} \supseteq U_1$ and $V_2 = \{a \mid a \in (\mathcal{B} \setminus J_2) \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in (\mathcal{B} \setminus J_2) \wedge \neg a \in S\} \subseteq V_1$. The fact that $(U_2, V_2) \models c'$ holds for each $c' \in N$ can be shown by induction on the maximum depth of nesting in N. In

particular, in absence of nesting (namely, if $N = \emptyset$) the result is immediate. The proof of inductive step relies on the fact that no negative literal occurs in N. This allows us to conclude that $J_2 \models \mathrm{B}(r)$, hence $a \in T_{P^I}(J_2)$. □

Given a program P and a set of atoms I, by the previous result, the operator T_{P^I} is monotone and has an unique least fix-point which is obtainable by iterated applications of T_{P^I} starting from the empty set. Let us denote such a fix-point by $T_{P^I}\uparrow$.

We have the following notion of stable model for programs with NWCs.

Definition 3. *Given a program with NWCs P, a set I of atoms is a stable model for P iff $I \models P$ and $I = T_{P^I}\uparrow$*

4 Conditional literals and the use of variables

Similarly to the approach of [11], in this section we adapt the treatment described in Section 3 to deal with *conditional literals*.

A conditional literal has the form $\ell{:}s$ where ℓ is a weight literal and s is a (possibly empty) set of atoms. The intended meaning is that the conjunction of the atoms in s constitutes a precondition for the satisfiability of ℓ. (Empty conditions, i.e., $s = \emptyset$, are trivially satisfied. Conditional literals of the form $\ell{:}\emptyset$ correspond to weight literals as introduced in Section 3. We will often write ℓ in place of $\ell{:}\emptyset$.)

All the notions introduced in Section 3 can be easily adapted to deal with conditional literals. In what follows we outline the main steps of such an adaptation. For the sake of readability, in doing this we will maintain the same notational conventions. The next definition is the counterpart of Def. 2:

Definition 4. *A* nested weight constraint *(NWC) is a tuple (S, N, l, u) where*

- *S is a finite set of conditional literals,*
- *$l \leq u$ are two non-negative integers (u can be ∞),*
- *N is a (possibly empty) finite collection of nested weight constraints*

Rules and programs are defined as one expects.

The notion of satisfaction for NWCs is slightly complicated w.r.t. the one in Section 3. This is so because the initial set of atoms (i.e., the candidate model, Z in the following) has to be considered in evaluating the preconditions of all conditional literals. Let $Z, X, Y \subseteq \mathcal{B}$ be sets of atoms such that $X \subseteq Z$ and $Y \subseteq (Z \setminus \mathcal{B})$. We define the *weight* of the NWC $c = (S, N, l, u)$, w.r.t. Z, X, Y, as follows:

$$W(c, Z, X, Y) = \sum_{(a,j):s \in S \wedge a \in X \wedge s \subseteq Z} j + \sum_{(\neg a,j):s \in S \wedge a \in Y \wedge s \subseteq Z} j$$

We say that a the sets of atoms Z, X, Y satisfy the NWC $c = (S, N, l, u)$, and write $(Z, X, Y) \models c$, if the following two conditions hold:

1. $l \leq W(c, Z, X, Y) \leq u$;

2. for all $c' \in N$ it holds that $(Z, U, V) \models c'$, where

$$\begin{array}{l} U = \{a \mid a \in X \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in X \wedge (a, j){:}s \in S \wedge s \subseteq Z\} \\ V = \{a \mid a \in Y \wedge \pi(a) \notin \pi(S)\} \cup \{a \mid a \in Y \wedge (\neg a, j){:}s \in S \wedge s \subseteq Z\} \end{array} \quad (5)$$

where, with abuse of notation, we denote by $\pi(S)$ the set $\{\pi(\ell)|\ell{:}s \in S\}$.

Given a set I of atoms, we say that I models an NWC c and write $I \models c$, iff $(I, I, \mathcal{B} \setminus I) \models c$. For a set Q of NWCs we write $I \models Q$ iff $I \models c$ for each $c \in Q$.

Analogously to what seen in Section 3, the satisfaction of each $c' \in N$ has to be evaluated within the context determined by S. The same recursive scheme outlined in Section 3 applies here in evaluating the satisfiability of NWCs.

As before, a set I of atoms is a model of a rule r (denoted by $I \models r$) iff $I \models \mathrm{H}(r)$ whenever $I \models \mathrm{B}(r)$ holds. Given a program P, $I \models P$ iff $I \models r$ for all $r \in P$.

In defining the notion of reduct we have to take into account all preconditions of conditional literals. Let Z, X, Y be sets of atoms. The *reduct* of an NWC $c = (S, N, l, u)$, w.r.t. Z, X, Y, is so defined:

$$c^{(Z,X,Y)} = \big(\{(a, j){:}s \mid (a, j){:}s \in S\},\ \{d^{(Z,U,V)} \mid d \in N\},\ \max(0, l - \textstyle\sum_{(\neg a,j):s \in S \wedge a \in Y \wedge s \subseteq Z} j),\ \infty\big)$$

where U and V are obtained from X, Y, Z, and S as explained earlier (cf., (5)).

In analogy with the cases of plain [11, Def. 2] and nested (Section 3) weight constraint, given a program P and a set of atoms I, the reduct P^I is so defined:

$$P^I = \Big\{\big(a, \mathrm{B}(I, r, s)\big) \mid r \in P,\ (a, j){:}s \in Cl(\mathrm{H}(r)),\ \{a\} \cup s \subseteq I,\ W(c, I, I, \mathcal{B} \setminus I)) \leq u(c) \text{ for all } c \in \mathrm{B}(r)\Big\}$$

where $\mathrm{B}(I, r, s)$ denotes the set

$$\mathrm{B}(I, r, s) = \Big\{c^{(I,I,\mathcal{B}\setminus I)} \mid c \in \mathrm{B}(r)\Big\} \cup \Big\{(\{(b, 1)\}, \emptyset, 1, \infty) \mid b \in s\Big\} \cup \quad (6)$$

$$\bigcup_{c \in \mathrm{B}(r)} \Big\{(\{(a, 1)\}, \emptyset, 1, \infty) \mid (\neg b, j){:}r \in Cl(c),\ a \in r \text{ s.t. } b \notin I,\ r \subseteq I\Big\} \quad (7)$$

The definition of the reduct P^I of a program is slightly more involute than the homologous definition given in Section 3. This is so because each negative literal $(\neg b, j){:}r$, with $b \notin I$, occurring in an NWC of the body of a rule r, will give its contribution to the weight of the NWC only if the precondition r holds in I. This requirement must be reflected in the program P^I by adding the set shown in (7). In this manner the body of the resulting rule will be falsified whenever any of such preconditions is false.

Now, we can define an operator T_{P^I} exactly as done in (4). Such an operator is monotone (an analogous to Proposition 1 can be stated) and has an unique least fixpoint. Def. 3 can be properly generalized to the case of NWCs with conditional literals:

Definition 5. *Given a program P with NWCs involving conditional literals, a set of atoms I is a stable model for P iff $I \models P$ and $I = T_{P^I}\uparrow$*

Variables can be exploited to denote collections of weight literals. This is done by admitting non-ground conditional weight literals $\ell{:}\varphi$ where ℓ has, in general,[4] the form $(p(X_1, \ldots, X_n), j)$ (or the form $(\neg p(X_1, \ldots, X_n), j)$) and each X_i is a variable (for $n \geq 0$). Similarly, φ is a set of not necessarily ground atoms. Let $var(\varphi) = \{X_1, \ldots, X_n, Y_1, \ldots, Y_m\}$ be the set of all the variables occurring in φ (for $n, m \geq 0$). The variables $X_1, \ldots, X_n$ are said to be *local* to the literal. The variables $Y_1, \ldots, Y_m$ are said to be *global*.

Non-ground NWCs, rules, and programs, are then defined as one expects.

Given a program, it is not restrictive to impose that each local variable occurs in a single conditional literal. We will make this assumption in what follows.

Considering a rule with non-ground NWC, all its global variables should be intended as being universally quantified. The instantiation of a rule is defined as the set of the ground rules each of them obtainable, first, by grounding all global variables (i.e., by uniformly substituting them by ground terms from the Herbrand universe of the underlying language) and then by replacing each non-ground conditional weight literal with the collection of all its ground instances that are obtainable by grounding the local variables. Notice that, in a literal such as (a, j) (or $(\neg a, j)$), we admit j to be a (global) variable. In this manner, each instantiation of this literal may have a different weight, determined through the grounding process. Analogously, the lower and upper bounds of an NWC can be expressed using variables, provided that the grounding process suitably instantiates them to non-negative integers. (In what follows we will adopt this option.)

The instantiation of a program P is defined as the set of all instantiations of rules in P. Stable models of programs involving variables are easily defined as follows:

Definition 6. *Given a (non-ground) program P, a set of ground atoms I is a stable model for P iff it is a stable model for the instantiation of P.*

Concrete syntax. In the following section, we describe a concrete encoding of a running example. In doing this we resort to the smodels-like notation, for programs, rules, and literals. In particular, we indicate by p an NWC of the form $(\{p\}, \emptyset, 1, 1)$. Also, we denote a weight constraint

$$(\{(a_1, w_{a_1}), \ldots, (a_1, w_{a_1}),\ (\neg b_1, w_{b_1}), \ldots,\ (\neg b_m, w_{b_m})\},\ l,\ u)$$

as $l[a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathit{not}\ b_1 = w_{b_1}, \ldots, \mathit{not}\ b_m = w_{b_m}]u$ and, similarly, an NWC $(\{(a_1, w_{a_1}), \ldots, (a_1, w_{a_1}),\ (\neg b_1, w_{b_1}), \ldots,\ (\neg b_m, w_{b_m})\},\ \{W_1, \ldots, W_k\},\ l,\ u)$ as $l[a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathit{not}\ b_1 = w_{b_1}, \ldots, \mathit{not}\ b_m = w_{b_m} \mid W_1, \ldots, W_k]u$. In both cases, we omit u whenever $u = \infty$.

For the special case of cardinality constraints, i.e., when $w_i = 1$ for all i, we adopt the shorthand notation $l\,\{a_1, \ldots, a_n, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_m\}\,u$. Conditional literals of the form $(a, j){:}\{b_1, \ldots, b_k\}$ will be denoted as $a : b_1, \ldots, b_k = j$. Moreover, in expressing weights and bounds of constraints we might use variables (intended to be suitably instantiated by the grounding phase). Finally, we denote a program rule as W_0 :- $W_1, \ldots, W_n$, where the W_is are (nested) weight constraints (for $n \geq 0$).

[4] Note that, in concrete encodings, constants are admissible in place of (some of) the X_is. For simplicity, and without loss of generality, we assume that each X_i is a variable.

5 A Case-study

To motivate the introduction of NWC into ASP, we resort to a case-study. Our running example is freely inspired by the Italian Computer Science undergraduate Program, that we shortly describe here in its basic features. Then, we provide an encoding using NWCs.

In order to get a bachelor degree in Computer Science, an Italian student is required to obtain 180 credits. Most of them must be obtained by attending courses and passing the corresponding exams. The remaining ones can be obtained by means of internships and a short thesis. There is a certain flexibility, so usually the number of credits that should be obtained from courses is allowed to vary within a range (say between 153 and 171, in the following encoding; actual ranges vary among different Universities and tracks). There are different possible choices for the courses to attend, so students are required to present what is called a "plan of studies", that must be approved by a Committee. Some courses must be taken at a certain year, for others there is some flexibility. For simplicity, we assume that the latter can be taken at any year and we neglect constraints related to the order in which certain courses should be taken.

Basically, the above (as described up to now) might be summarized by the following rule that characterizes possible plans of studies. (The atom `in_ps(c,j)` means that the course `c` is inserted into the plan of studies, at year `j`.)

```
Min [in_ps(X,Y):course(X,W),course_year(X,Y)} = W ] Max  :-
                          credits_bounds(Min,Max).
```

This knowledge base describes a possible problem instance:

```
year(1..3).   credits_bounds(153,171).
course_desc(programming, comp_science, 12).
course_desc(computer_architectures, comp_science, 6).
course_desc(databases, comp_science, 12).
course_desc(algorithms, comp_science, 12).
course_desc(theoretical_cs, comp_science, 6).
  ...
course_desc(calculus, mathematics, 6).
course_desc(optimization, mathematics, 6).

course(Course,Creds) :- course_desc(Course,Area,Creds).
```

where each fact `course_desc(c,a,n)` specifies that the course `c` belongs to the area `a` (see below) and corresponds to an amount of `n` credits. Moreover, we might assume the presence of facts of the form `course_year(c,y)` specifying, for each year `y`, the admissible courses `c` for that year.

Clearly, this simple encoding does not model all aspects of the problem at hand. For instance, an aspect which is not represented is that a plan of study cannot include the same course several times. This can be imposed by adding this NWC (actually a cardinality constraint):

```
0 {in_ps(X,Y):in_ps(X,Y1),neq(Y,Y1)} 0.
```

In our case-study, it is always the case that some mandatory courses must be situated at a certain course year. To model this requirement we add this NWC to the initial rule:

```
  0 {in_ps(X,Y):mandatory(X,Y1),neq(Y,Y1)} 0.
```

For courses that must be included in the solution, but can be situated at any year, we add this extra rule to the encoding:

```
  1{in_ps(C,Y):year(Y)}1 :- mandatory_course(C).
```

The specific instance might specify, for example, this piece of knowledge:

```
mandatory(programming, 1).  mandatory(computer_architectures, 1).
mandatory(algorithms, 2).   mandatory(theoretical_cs, 3).
mandatory_course(databases).
```

Finally, to avoid a student giving too many exams, there is a statement that enforces at least a minimum number of courses of the first two years to weigh 12 credits each. Also, courses are allowed to belong to certain scientific areas, namely Computer Science, Mathematics, Physics, and other different though related topics (within a list). However, there are directions stating that every subject should contribute to the plan of studies for a quota ranging between a minimum and a maximum number of credits.

The next NWCs specify both the number of the 12 credit courses in the first years, and range of credits that can be allowed to the different areas. Here, the constants `comp_science`, `mathematics`, ..., identify (through the facts `course_desc` listed earlier), the area of each course.

```
Min12 {in_ps(X,Y):course(X,W),leq(Y,2),eq(W,12)}
L1 [in_ps(X,Y):course_desc(X,comp_science,W),course_year(X,Y)=W] U1
 ...
Ln [in_ps(X,Y):course_desc(X,mathematics,W),course_year(X,Y)=W] Un
```

where the variables `Min12`, `L1`, `U1`, ..., `Ln`, and `Un`, (to be instantiated through atoms in the rule body) express the bounds on the minimum number of courses worth 12 credits in the first two years, and the minimum/maximum amounts of credits in the different areas.

Summing up, the encoding of our sample problem is as follows (to be joined with a specific instance specifying, together with the pieces of knowledge seen earlier, the predicate `area_bounds`):

```
Min [ in_ps(X,Y):course(X,W),course_year(X,Y)}=W  |
  0 {in_ps(X,Y):in_ps(X,Y1),neq(Y,Y1)} 0,
  0 {in_ps(X,Y):mandatory(X,Y1),neq(Y,Y1)} 0,
  Min12 {in_ps(X,Y):course(X,W),leq(Y,2),eq(W,12)},
  L1 [in_ps(X,Y):course_desc(X,comp_science,W),course_year(X,Y)=W] U1,
          ...
  Ln [in_ps(X,Y):course_desc(X,mathematics,W),course_year(X,Y)=W] Un
] Max  :-  credits_bounds(Min,Max), min_12(Min12),
           area_bounds(comp_science,L1,U1),
                   ...
           area_bounds(mathematics,Ln,Un).

1{in_ps(C,Y):year(Y)}1 :- mandatory_course(C).
```

We hope at this point to have convinced the reader that NWC can easily cope with aspects that can be relevant in a number of applications. Our case-study, in fact, is a simple example of a scheduling problem, where this kind of problems are an important realm of application of ASP. We believe therefore that many kinds of ASP applications might profit from programming constructs that allow for some degree of nesting. In other words, we deem it appropriate to introduce some kind of *contextual* constructs.

6 On the Complexity of Nested Weight Constraints

In [13] it is proved that introducing weight constraints does not affect the complexity of ASP. That is, for instance, the complexity of the problem of checking whether a program has a stable model does not depend on the presence of weight constraints. Here, we are more generally concerned with checking whether a program with NWCs admits stable models.

In what follows we address the complexity issue for NWC programs by focusing on the particular case of ground programs containing NWCs of bounded depth, as defined in Section 3.

Let k-NWC be the class of programs with depth not greater then k, for $k \geq 0$. For $k = 2$ we have the following proposition.

Proposition 2. *Deciding whether a ground 2-NWC program admits stable models is NP-complete.*

Proof. (Sketch) The problem of deciding whether a ground 2-NWC program admits a stable model is NP-hard. This follows from the NP-completeness of ASP with weight constraints in absence of nesting [13]. As regards inclusion in NP, this can be verified by showing that, given a set M of atoms, it can be checked in polynomial time whether M is a stable model of P. To do this we have to show that: (a) given a rule $r \in P$, checking if $M \models r$ takes polynomial time; (b) the reduct P_M of the program P has polynomial size w.r.t. the size of P; (c) $T_{P^M}\uparrow$ can be computed in polynomial time.

As regards (a), observe that checking whether $M \models c$ for an NWC c involves the evaluation of the weight of c (cf., (1)) and the computation of the sets U, V (cf., (2)). Both the computations can be completed in polynomial time (recall that $depth(c) \leq 2$). Concerning (b), for each rule r in P a linear number of rules is introduced in P_M (cf., (3)). Moreover, for each NWC c occurring in r the computation of the reduct of c takes polynomial time. Finally, (c) can be shown by observing that the computation of the set $I_2 = T_{P^M}(I_1)$, for a given I_1, can proceed by processing, one-by-one, the unsatisfied rules whose head is not in I_1, and checking the satisfaction of their bodies. By (b) we conclude that $T_{P^M}\uparrow$ can be computed in polynomial time. □

The previous result generalizes to the case of k-NWC programs, for any fixed k.

From this result, it follows that NWCs might be rephrased in plain ASP. As shown in [13, 6], for weight constraints this can be done at the expense of introducing a (polynomial, but not insignificant) number of new atoms and rules. Moreover, except for cardinality constraints, the translation is quite involved. Therefore, it turns out that weight constraints are a quite substantial programming construct, rather than simple syntactic

sugar. This is of course true also for NWCs. Notice that, how to represent NWCs in plain ASP is far from easy to understand. Outlining a translation into plain ASP and evaluating the necessary number of additional atoms and rules is a subject of future work.

7 Concluding Remarks

In this paper, we have introduced an extension to the weight constraint construct, widely used in ASP practical programming. We have illustrated by means of a significant example the potential usefulness of the extension. We have formally defined the extension involving arbitrary nesting of weight constraints and provided a semantics for the enhanced framework. In the case when the depth of nesting is bounded, we proved that the new construct does not affect the complexity of ASP.

Much remains to be done. First of all, the complexity issue for NWC programs has not been completely investigated in the general case (when no bound on the nesting depth is assumed). Moreover, the proposed construct has not been implemented yet and no translation in plain ASP has been designed (this, by Proposition 2, at least for the bounded-depth case, should be achievable). When an implementation will be available, practical use will help us explore the feasibility of further extensions and generalizations. Also, we intend to explore the enrichment of weight constraints by means of complex preferences. In particular, the present work can be easily integrated with the approach to preference handling devised in [2, 3] and extended to weight constraints in [4]. In the resulting setting, referring to the above example one might enrich the formulation with student's preferences, stating for instance with kind of courses are preferred and in which conditions.

We will have to explore both the usefulness in practical applications of nested-constraints, as well as their feasibility in cases where both negative weights and circular definitions are admitted.

From the formal point of view, we intend to extend the method of [6] so as to be able to extend the concept of *strong equivalence* to ASP programs with NWC. Strong equivalence [9] in fact, as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization. In the case of NWC programs, the form of locality implicitly present in NWCs might have interesting consequences. A further issue for future research regards the relation between NWC and (nested) aggregates.

References

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

[2] S. Costantini and A. Formisano. Conditional preferences in P-RASP. In *Proceedings of LANMR'08*, 2008.

[3] S. Costantini and A. Formisano. Modeling preferences and conditional preferences on resource consumption and production in ASP. *Journal of of Algorithms in Cognition, Informatics and Logic*, 64(1), 2009.

[4] S. Costantini and A. Formisano. Weight constraints with preferences in ASP. In *Proc. of LPNMR'11*, LNCS. Springer, 2011.
[5] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33 (3):374–425, 2001.
[6] P. Ferraris and V. Lifschitz. Weight constraints as nested expressions. *TPLP*, 5:45–74, 2005.
[7] M. Gelfond. Answer sets. In *Handbook of Knowledge Representation*. Elsevier, 2007.
[8] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proc. of ICLP/SLP'88*. The MIT Press, 1988.
[9] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM TOCL*, 2:526–541, 2001.
[10] J. McCarthy. Elaboration tolerance. In *Proc. of Commonsense'98*, 1998.
[11] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proc. of LPNMR'99*, number 1730 in LNCS, pages 317–331. Springer, 1999.
[12] R. Pichler, S. Rümmele, S. Szeider, and S. Woltran. Tractable answer-set programming with weight constraints: Bounded treewidth is not enough. In *Proc. of KR'10*. AAAI Press, 2010.
[13] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138 (1-2):181–234, 2002.

The temporal representation and reasoning of complex events

Francesco Mele[1], Antonio Sorgente[1],

[1] Istituto di Cibernetica, Consiglio Nazionale delle Ricerche,
Via Campi Flegrei, 34 Pozzuoli(Naples) Italy
{f.mele, a.sorgente}@cib.na.cnr.it

Abstract. This paper introduces a formalization of complex events. In particular, a formalism is presented to represent intentional and causal events in narrative contexts, and in their mechanisms of composition. Complex events have been defined through classes of a formal ontology that has been called the Ontology of Complex Events (OntoCE). This approach has allowed for the applications (reuse) of existing axiomatizations belonging to a large repertoire of temporal reasoning techniques and, the definition of new axiomatizations presented and discussed in this work. The focus in this work has been placed on three particular temporal aspects: the analysis of consistency, the discovery of new temporal relations in a knowledge base of events, and the causal reasoning in narrative contexts.

1 Introduction

The concept of event has been highly examined and much debated in philosophy [CAS, DAV] and Artificial Intelligence. In this area, some well-founded formalisms, like the Event Calculus [MIL], the Situation Calculus [LEV, LIN], and ALAN [BAR, Gon] have been proposed. Recently, a new point of attention, that regards the concept of the "complex event" [WIN], has been born (although this name is not explicitly mentioned by all the research projects that deal with these issues).

This concept emerges, particularly in the context of the Internet, where the broad set of information in unstructured form, hides a multitude of events that are connected by relationships extremely difficult to detect, but where one feels that these events are components of an implicit totality (suggested without being directly expressed).

The particular aim of this research was to build a model and a formalism to represent three main types of complex events (intentional events, causal events, and narrative events) and their mechanisms of composition.

The paper introduces a representation of complex events, in which an event is not only an aggregation of simple events (how it would be in the case of a narrative of events, consisting of a set of simple events and a set of relationships between those events). The modeling of narratives that have as components other complex events has been addressed. For example, casual events are considered such as: (the church of Santa Chiara was built (e1) for desire (e2) of Roberto D'Angiò) (e0), where e0 is composed by the events e1 and e2 and could also represent the component of a

narrative. A mechanism for determining the interval in which a complex event (intensional, narrative or causal) occurs has also been proposed. Such intervals have been calculated considering the intervals of happening of its component events (e0): in the example the event e0 has an occurrence interval that is calculated as union of occurrence intervals of events e1 and e2.

By inserting intentional events, in representation of a narrative, one can not only annotate or discover causal connections between events, but with appropriate axioms (eg *(e1 cause e2) implies (e1 precedes e2)*) one can convert the causal relations into temporal order relations, thus eliminating the existing deficiencies of connectivity among the events of a narrative.

A formalism has been constructed to represent the complex events in explicit form, with the main motivation that such a representation can be used as an ontological reference for various types of semantic annotations, in particular:

- to aggregate, as complex events, multimedia elements (photos, video or texts whose contents represent events (historical events, news, cultural events, etc.) in the same way as proposed in the Event-Centric in [GIU] and [MEL]; and,
- to annotate and aggregate complex events in natural language, starting from annotations represented by TimeML [PUS03, PUS08] formalism.

An annotation process of natural language texts or media, especially if this is done through a process of multiple annotation (by more than one operator), can easily generate some inconsistencies or lack of connections between events in the bases of annotations. For these reasons, it is necessary to identify inconsistencies and non-connected events in order to remove such anomalies among the annotated events (see Fig. 1).

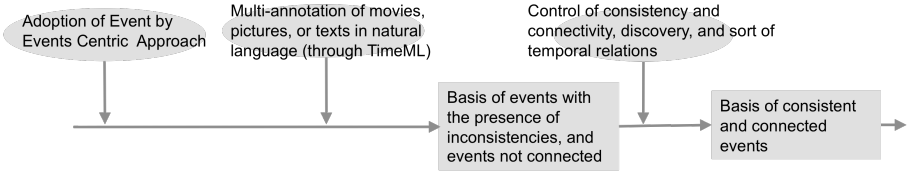


Fig. 1. Phases to control the consistency and connectivity of a narrative

The formalism (OntoCE) that has been defined in this paper is represented by a formal ontologies, where each entity has been defined as a class of an ontology. This methodological approach was chosen to facilitate the building of modules (algorithms) for the discovery of temporal relations, with the objective also, to reuse existing axiomatizations and facilitate the creation of new ones (some of these proposals are in this paper).

In this paper the logic programs have been used to represent the events (simple and complex) and their relations, to analyse the temporal consistency of a complex event, to discover new temporal relations between events, to apply causal reasoning to events, and to integrate the latter with axioms of temporal reasoning.

Related work

In recent research [WIN] there are several proposals for representing events. The basic motivation of this research stands from the claim that events can constitute an

excellent framework for aggregating knowledge. The large quantity of data and (fragmented and unstructured) knowledge on the Internet, makes this research very attractive. An emerging methodology for representing through events knowledge distributed on the Internet has been named Event-Centric [WIN]. In this methodology, an event is a structure of reference that is independent from the metadata of media that one intends to annotate. An example of the Event-Centric approach, which uses high-level ontology DOLCE, is the F model [SHE]. In the F model the methodological choices are motivated by a number of functional and non-functional requirements.

With respect to the functional requirements, the representation of an event must have the attribute for the participants. It is also necessary to be able to define relationships between parts and wholes of an event. It must be possible to define cause-effect relationships between two events (no matter of the degree of difficulty of the automatic process discovery), and, finally, it must be possible to represent correlation relationships between events (two events that have a common cause).

The non-functional requirements of F, instead, include extensibility, formal precision (axiomatization), modularity, and reusability.

Among the proposed Event-Centric is the one proposed in [GIU]. This methodology adopts the slogan "Aggregation via Media Events". In this proposal, the events are the reference structures for aggregating the media. In [GIU] an implicit model of complex events (without explicit constructs of representation) and a simple mechanism to determine the "where" a complex event happens, starting from the "where" of the components' events, are introduced.

To represent the events, some formalisms were inspired from a model that has its roots in journalism. This model called "W's and one H" adopts six attributes for the representation of events: Who, When, Where, What, Why, and How. The project Eventory [WAN] adopts a model "W's and one H". Eventory has a particular structure of the "When" attribute, having two references for time: the first referring to the chronological time of "real events", the second, to the temporal attributes of some metadata (such as the length of a movie or the time during which a picture must be shown).

The decision to include the knowledge of the media in the attributes of the events, violates the constraint that characterizes the Event-Centric models, whereby the independency between the event representation and that of media is fixed. In fact, in the case of Eventory, the description contains information about the execution time of the media.

In this work, in relation to the representation of temporal intervals, the classification given in [MAJ], where all the possible combinations that exist between instants or time intervals are shown, when they represent a temporal relationship between two events, has been taken into consideration.

2 The representation of the events

In this work, an ontology for complex events has been defined: OntoCE. OntoCE has an abstract superclass (`AnythingInTime`) common to all entities that happen over

time. Two subclasses are specializations of `AnythingInTime`: `Event`, that represents the class of simple events, and `ComplexEvent`, that represents the class of complex events. In Fig. 2 a sketch representation is given.

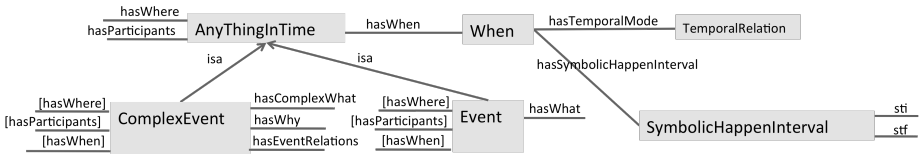


Fig. 2. The taxonomy of simple and complex events

In Fig.2, in brackets, the attributes that are inherited from their respective superclasses are reported. Formally, they are represented in Flora2[1][FLO](this formalism combines the advantages of conceptual modeling with object-oriented, owns a declarative syntax, allowing to build complex inferential apparatus in simple manner):

```
Event::AnyThingInTime.
ComplexEvent::AnyThingInTime.
AnyThingInTime[
   hasWhen*=>When, hasWhere*=>Where,
   hasParticipants*=>Participant].
Event[hasWhat*=> Action_Property].
ComplexEvent[     hasComplexWhat*=> AnyThingInTime,

      hasEventRelations*=>EventRelation].
```

`AnyThingInTime` is an abstract class (without instances) which is the superclass of the concrete classes: `Event` and `ComplexEvent`. The latter classes are the key concepts of the formalism OntoCE. The `Event` class has the descriptor `hasWhat`, which is associated to the class What. Generally, this class describes the action (which happens over time) that characterizes the event or describes a property that is true in a specific time interval. The attribute `hasComplexWhat` is a specific descriptor of `ComplexEvent`. The latter also has the attribute `hasWhy` that describes the causal relations between events. `hasWhy` is a relation between two events, so it can only be the attribute of a complex event. Attributes `hasWhen`, `hasWhere`, `hasParticipants`, `hasWhat` (or `hasComplexWhat`), and `hasWhy` correspond to the descriptors with which journalists describe their articles.

[1] To make reading simpler, some key constructs of Flora2 language are here reported. X:: Y (class X is a subclass of class Y), X: Y (X is an instance of class Y), X => Y (X is an attribute of type Y), X-> Y (Y is the value of X), X *=> Y (as X => Y, but the attribute is inherited by subclasses). In Flora2, chain of alphanumeric literal, starting with a capital letter are variables. The symbols ":-", the comma (",") and semicolon (";") have the same interpretation as the homologue constructs of Prolog language.

2.1 Instant and interval representations

The representation of time that has been adopted is mixed and based on points and time intervals. In OntoCE all temporal entities are represented as classes. `Time` is the main class and has several specializations: date or partial dates (`DateValue`), time instants or combinations of them with date (`TimeValue`), symbolic times (`Symbolic`), and time intervals (`Interval`). The definition is as follows:

```
DateCalendar[
  day*=>Integer, month*=>Integer, year*=>Integer].
DateWeek[week*=>Integer, year*=>Integer].
DateQualitative[value*=>String].
Clock[
  hour*=>Integer,minute*=>Integer,date*=>DateCalendar].
TimeQualitative[value*=>String].
Interval[
  hasBeginTime*=>(DateValue;TimeValue;SymbolicTime),
  hasEndTime*=>(DateValue; TimeValue; SymbolicTime)].
SymbolicHappenInterval[
  sti*=>SymbolicTime,stf*=> SymbolicTime].
```

2.2 The When class

In OntoCE there is a particular structure, the class `When` (see Fig. 2), which describes when an event happens using the effective symbolic interval (ESI) in which the event happens, and a temporal modality of happening, described by one (or more) temporal order relations (before, after, during, etc.) between ESI and some temporal interval of reference (or also another event). These relations have the objective of anchoring an event on the chronological axis or with another event, through a temporal order relation. The approach requires, therefore, that when an event (simple or complex) is created, it automatically generates a type identifier ESI, represented by two attributes: `sti`, the (effective) symbolic time in which the event starts (or in which the property is true), and `stf`, the time in which the event ends. The choice of having an effective time when an event happens and a temporal modality of happening, for it is motivated by the fact that often the effective time in which an event happens is not known, but one can easily know one or more relations (modality of happening) for it (after a time tx, dtx before a certain range, etc.). Thus, even if one does not exactly know the exact value of the start and end/or of an event, one can annotate (or, automatically discover) relationships with other time intervals, as soon as they become available. The class `When` has the following definition:

```
When[
  hasSymbolicHappenInterval*=>SymbolicHappenInterval,
  hasTemporalMode*=>TemporalRelation].
SymbolicHappenInterval[sti*=> SymbolicTime,
                       stf*=> SymbolicTime].
```

2.3 Simple Events

In OntoCE, the class `Event` represents simple events. This class inherits the attributes of the superclass `AnyThingInTime` and has the attribute `hasWhat`, which describes an action that happens or a property that is true in a temporal interval.

```
Event[hasWhat*=>Action-Property].
```

The attribute `hasWhat` has values in the `Property_Action` domain and describes exactly what happens (action) or what is true (property) in a temporal interval.

2.4 Complex events

`ComplexEvent` is a class defined as a set of events (simple or complex) described by `ComplexWhat` slot, and a set of relationships between events described by `hasEventRelations` slot. Also, the class is described by the method `hasWhy(AnyThingInThing)`, a function that given an input event belonging to a complex event, returns a set of causal relations that are the justification of why the event occurred.

```
ComplexEvent[ hasComplexWhat{2:*}*=>AnythingInTime,
              hasWhy(AnyThingInThing)*=> CausalRelation,
              hasEventRelations*=>EventRelation].
```

Complex events are defined by a temporal mode, described by the descriptor `hasWhen`, the same attribute used for the description of simple events. Fig. 3 shows the taxonomy of the complex events of the OntoCE ontology, where the narrative events, the causal events, the intentional events, and the perceptual events have been labeled and represented as complex events.

Narrative Events
The complex narrative events (`NarrativeEvent`) are represented as a set of events and temporal relations between events. The components of `NarrativeEvent` are simple events that describe actions that occur over time, or properties that are true in a temporal interval, or other complex events such as causal events (`CausalEvent`) or intentional events (`MentalEvent`).

`NarrativeEvent`, like all subclasses of `AnyThingInTime` inherits the attributes When, Where, and Who, and like all the subclasses of `ComplexEvent`, inherits the method `hasWhy(AnyThingInThing)`. The characterization of `NarrativeEvent` is given by the restriction of the slot `hasEventRelations`, which can only have instances of temporal relations as a value.

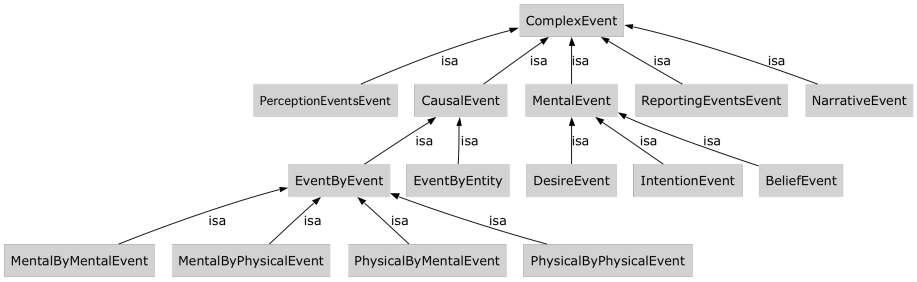


Fig. 3 The taxonomy (subclasses) of complex events.

Mental events

The class `MentalEvent` represents mental events of an agent (participant at the event), i.e., beliefs, desires, and intentions that occur over time. These entities allow for the representation as a causal events such as "*The church and monastery of Santa Chiara was built between 1310 and 1340 for desire of Roberto d'Angiò and Queen Sancha of Aragon*". Mental events are defined, through to the attributes When, What, Where, and Participants (inherited by `AnyThingInTime)` by a slot that describes a relation (`MentalRelation`) between a mental event and a physical event.

Causal events

The complex event `CausalEvent` describes events that relate to a cause-effect relation: the occurrence of an event (event cause) caused the occurrence of another event (event effect). In OntoCE a classification of causal events has been defined in accordance with the nature of the events involved, or that is, if the cause-effect relationship is defined by physical and/or mental events. We report some examples of the categories:

- *"I think it's a good book, I'll buy it"*, and "*I would like something hot, I'll take a cup of tea"*, are examples of causal relations of a mental event that causes a physical event (classified in OntoCE as a causal `PhysicalByMentalEvent`);
- "*He laughed and I thought he was joking*" is an example of causal event where a physical event (perception) caused a mental event (in OntoCE labeled as `MentalEventByPhysical`);
- "*He bumped the glass with his elbow and broke it*", "*It's raining and the road is wet*" (causal events labeled in OntoCE as `PhysicalEventByPhysical`); and,
- "*I think it's the best team and I think it will win the championship*" (causal events labeled in OntoCE as `MentalEventByMental`).

Causal events are defined by a causal relationship between events and like all events subclasses of `AnyThingInTime` inherit the attributes When, What, Where, and Participant.

For causal events, a widely shared relation (axiom), that brings together the causal relations with temporal relations, has been defined:

```
BeforeEE[Ex, Ey]:- CausalRelation[Ex, Ey].          (1)
```

If Ex is the cause of Ey, then the event Ex precedes temporally the event Ey.

2.5 The When Attribute of a complex event

For the simple event, the time of happening (hasSymbolicHappenInterval) defines the temporal interval in which the action occurs, while the time interval of occurrence of the complex event defines the minimum time interval in which all events belong to the complex event occurring. Therefore, the occurrence interval of complex events is not continuous, or that is, not in all temporal subintervals is there an event that happens. In addition, the temporal mode of a complex event is represented as the union of the temporal mode of event's components. It is obvious that, starting from the all temporal mode, one can define various algorithms that can determine, for example, the entire period of a complex event or the frequency of a particular action or category of action, etc.

The descriptor When can be calculated according to the descriptors of the component events, or it is instantiated interactively. In the latter case the compatibility checks (defined by constraints), with respect to the attributes When of the components events, must be run .

Informally, the interval of occurrence of a complex event is made up of the intervals of occurrence of the events' components and a set of temporal relations between these intervals. The rule for determining the minimum time of occurrence of a complex event, in accordance with the event components is reported. Let Ec be a complex event (a narrative), then its time of occurrence (defined as an instance of the class When) is calculated using the following rule:

```
01. Ec:NarrativeEvent[ hasComplexWhat->{E1, E2},
                       hasWhen->Wx]:-
02.  newId(Ec,E1,E2,Wx),
03.  genSymbolicInterval(I,T1,T2),
04.  insert{I:SymbolicHappenInterval[sti->T1,stf->T2]},
05.  E1:AnythingInTime[hasWhen->W1],
06.  E2:AnythingInTime[hasWhen->W2],
07.  minimun(W1, W2,Tmin), maximum(W1, W2,Tmax),
08.  tm_union(W1,W2,
          [EqualTT[T1,Tmin],EqualTT[T2,Tmax]], Tmx),
09.  insert{Wx:When[ hasSymbolicHappenInterval->I,
                     hasTemporalMode->Tmx]}.
```

To establish the time of occurrence of the narrative event Ec (01), the rule generates the structure Wx of the When attribute (02) and defines the symbolic interval of occurrence of the complex event (03-04). Then, it identifies the time of occurrence of the component events (05-06), and calculates the minimum and maximum time of occurrence of these events (07). The minimum and maximum calculated are correlated with the symbols' times with relations EqualTT[T1,Tmin] and EqualTT[T2,Tmax], which together with the temporal mode of the components (08) define the temporal mode of the complex event (09).

Others rules (not reported here) have been defined that consider the cases where the minimum and/or maximum of events cannot be determined, for example, because the annotation of the temporal mode is not complete.

In these cases, to overcome the inability to calculate the minimum and/or maximum, the rule defines temporal relations between the component events, `E1` and `E2`, and the occurrence interval `I` of complex event. The relations are `DuringEI[E1,I]` and `DuringEI[E2,I]`, which, together with the temporal mode pattern of the components, define the temporal mode of the complex event. Similar rules relating to the When for all types of complex events have been implemented.

3 Temporal reasoning for complex events

The classes of events that have been defined are appropriate for applying axiomatizations (expressed in the form of Horn clauses) to temporal reasoning. However, to apply these algorithms, one must follow a specific methodological statute that is associated with the ontological approach implemented.

Let `Ei` be a knowledge base of events, the algorithms (to check consistency and connectivity, and to discover new temporal relations) are applicable to all events that are not related to mental events: beliefs, desires and goals. This is because a mental event can be a component of a narrative, but the events that are among its arguments should not be involved in the analysis of the consistency and connectivity of a narrative.

Consider the following example: [*during the summer of 2010 a fan (Px) wants Inter to win the next season (2010-11)*]. The mental event of Px belongs to a narrative like this: 1.[*in summer 2010, a fan Px wants Inter to win the league*], 2.[*in summer 2011, Milan won the championship*]. Of course, there is no contradiction between the two events, because the event "*Inter won the league*" is something that belongs to the mind of a person, and does belong to events of real history. However, it is necessary to ensure a consistency in the set of events believed by a person. It is clear that the events to which we apply the algorithms to check consistency must belong to appropriate categories, and it's believed that this approach (to define events as classes) is appropriate for this purpose.

3.1 How to check the consistency of events

For consistency checks of temporal relations, axioms of Russell and Kramp [LAM] (RK) have been adopted, which allows for performance of reasoning about the events through the relations between the events $\texttt{prec}_{\texttt{ev}}\texttt{(E1,E2)}$ and $\texttt{over}_{\texttt{ev}}\texttt{(E1,E2)}$, where $\texttt{prec}_{\texttt{ev}}\texttt{(E1,E2)}$ means that the event `E1` precedes the event `E2`, and $\texttt{over}_{\texttt{ev}}\texttt{(E1,E2)}$ means that the event `E1` overlaps event `E2`.

The RK language, with the primitive $\texttt{prec}_{\texttt{ev}}\texttt{(E1,E2)}$ and $\texttt{over}_{\texttt{ev}}\texttt{(E1,E2)}$, is not expressive enough to represent the temporal relations in our ontological representation (`DuringEE[E1,E2]`, `StartEE[E1,E2]`, etc.). In fact, the

relations like `DuringEE[E1, E2]` cannot be defined only through the primitive $prec_{ev}$ and $over_{ev}$, because they also require the conditions between time instants. For this reason, a core with relations between time instants using the primitive $prec_t$`(T1,T2)` (the time instant `T1` precedes the time instant `T2`) and eq_t`(T1,T2)` (the time instant `T1` is equal to the time instant `T2`) has been defined. Through such primitives all temporal relations of OntoCE (between events, between events and time instant, etc.) have beeen defined and applied to the consistency analysis redefining the relations RK. Therefore, a set of axioms have been defined, that allow one to discover inconsistencies, not in order relations between events, but in order relations between time instants: $prec_t$`(T1,T2)` and eq_t`(T1,T2)`. The following axioms for the verification of the inconsistencies is provided[2]:

Axiom schema

1: $prec_t(T1,T3)$ <- $prec_t(T1,T2) \wedge prec_t(T2,T3)$.
2: $eq_t(Tx,Tx)$.
3: $eq_t(T1,T2)$ <- $eq_t(T2,T1)$.
4: $eq_t(T1,T2)$ <- $eq_t(T1,T2) \wedge eq_t(T2,T3)$.
5: $prec_t(T1,T3)$ <- $prec_t(T1,T2) \wedge eq_t(T1,T3)$.
6: $\neg prec_t(T2,T1)$ <- $prec_t(T1,T2)$.
7: $\neg eq_t(T1,T2)$ <- $prec_t(T1,T2)$.

Implementation using Stable Models

```
prect(T1,T2) :- prectD(T1,T2), not prectN(T1,T2).
prectN(T1,T2) :- time(T1), time(T2), prect(T2,T1).
prect(T1,T3) :- prect(T1,T2), prect(T2,T3),
   not prectN(T1,T3), time(T1),
   time(T2), time(T3).
prectN(T1,T2) :- time(T1), time(T2), eqt(T1,T2).
eqt(T1,T2) :- eqtD(T1,T2), not eqtN(T1,T2).
eqt(T1,T3) :- eqt(T1,T2), eqt(T2,T3),
   not eqtN(T1,T3), time(T1), time(T2), time(T3).
eqt(T1,T1) :- time(T1).
eqt(T2,T1) :- eqt(T1,T2), time(T1), time(T2).
eqtN(T1,T2) :- time(T1), time(T2), prect(T1,T2).
eqtN(T1,T2) :- time(T1), time(T2), prect(T2,T1).
prect(T1,T3) :- prect(T1,T2), eqt(T2,T3),
    not prectN(T1,T3),
    time(T1), time(T2),time(T3).
```

Fig. 4: Axiom schema and relative implementation for inconsistency checking

Axioms 1-4 define the properties of order relations: the transitivity of relation $prec_t$`(T1,T2)` and eq_t`(T1,T2)`, the reflexivity and symmetry of the relation eq_t`(T1,T2)`. As one can see, axioms 1-7 at the heads of the clauses are negated, so it is not possbile to implement them through monotonic logic programs and then through the traditional Horn clauses. For this reason a logic program for the axioms 1-7, through stable model semantics [GEL] by using the SModels system [SYR], has been implemented (in section 4 we report a running example of Stable Model Program of Fig. 4). In addition, there is the axiom:

```
prect(T1,T2) ∨ prect(T2,T1) ∨ eqt(T1,T2)
```

which translates the timeline by forcing every time point to have a relation of precedence or equality with another time. This axiom has not been implemented because it is not a useful consistency check. The axiom, however, would be useful to ensure the full connection of the time points of a narrative. For this purpose, an algorithm (shown in [MEL]) for controlling the connection has been defined. The definition of temporal relations between events in terms of primitive $prec_t$`(T1,T2)` and eq_t`(T1,T2)` is shown:

[2] In this paper we will use the symbols "∧" and "<-" to denote the conjunction and implication in generic expressions of Horn Clauses, we will use instead the symbols "," and ":-" in the corresponding expressions Prolog and Flora2.

```
BeforeEE[3][E1,E2]   <-> dt(E1,T1,T2), dt(E2,T3,T4),
                         prec_t(T2,T3).
AfterEE[E1,E2]       <->  BeforeEE[E2,E1].
MeetsEE[E1,E2]       <->  dt(E1,T1,T2), dt(E2,T3,T4),
                         eq_t(T2,T3).
Meet_byEE[E1,E2]     <->  MeetsEE[E2,E1].
EqualsEE[E1,E2]      <->  dt(E1,T1,T2), dt(E2,T3,T4),
                         eq_t(T1,T3), eq_t(T2,T4).
OverlapsEE[E1,E2]    <->  dt(E1,T1,T2), dt(E2,T3,T4),
                         prec_t(T1,T3), prec_t(T3,T2),
                         prec_t(T2,T4).
Overlapped_byEE[E1,E2] <->  OverlapsEE[E2,E1].
DuringEE[E1,E2]      <-> dt(E1,T1,T2), dt(E2,T3,T4),
                          prec_t(T2,T4), prec_t(T3,T1).
ContainsEE[E1,E2]    <-> DuringEE[E2,E1].
StartsEE[E1,E2]      <-> dt(E1,T1,T2), dt(E2,T3,T4),
                         eq_t(T1,T3), prec_t(T2,T4).
Started_byEE[E1,E2]  <-> StartsEE[E2,E1].
FinishesEE[E1,E2]    <-> dt(E1,T1,T2), dt(E2,T3,T4),
                         eq_t(T2,T4), prec_t(T3,T1).
Finished_byEE[E1,E2]  <-> FinishesEE[E2,E1].
```

`dt(E,T1,T2)` provides the value of the start time `T1` and end time `T2` of the event `E`. In addition, for each occurrence interval of event `E` there is the following relation:

```
prect(T1, T2) <- dt(E,T1,T2).
```

Similary, all temporal relations have been defined.

Once the temporal relations through the primitive $prec_t$`(T1,T2)` and eq_t`(T1,T2)` have been redefined, one can check the inconsistencies of a set of temporal relations expressed by the relationship between time points, by using the axioms 1-7.

The program, given a knowledge base defined through the relations between time points (the input of the program), calculates the stable models, or rather, groups of consistent sets of relations (satisfying the axioms 1-7). If the program returns more than one stable model, then the relations are inconsistent.

In addition, among the events causal relation can be annotated and this produces new temporal relation between events.

Thus, to have a consistency check for all the relations annotated in a narrative, the temporal relation derived/obtained from the causal relations must be definend through the relations $prec_t$`(T1,T2)` and eq_t`(T1,T2)`. In this way, it is possible to check

[3] In order to ease reading (so there is no ambiguity), simplified Flora2 notation, omitting the names of attributes, i.e., instead of the notation of instances `id:ClassName[attr1->val1,attr2->val2, ..,attrn->valn]`, the following notation `id:ClassName[val1,val2,..,valn]` has been adopted.

the consistency of a narrative, according to the methodological approach represented in Fig. 5.

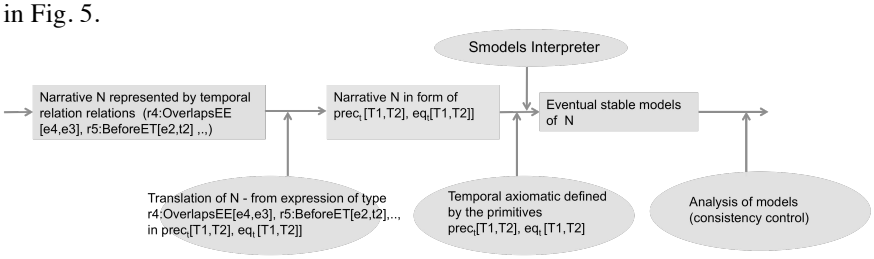


Fig. 5: Transformation of temporal relations in the primitive $prec_t$, eq_t.

3.2 How to discover new temporal relations between events

Through the formalism defined for the representation of annotations, a new technique to discover new temporal relations between events has been developed. The methodology uses the results of the process of the consistency check. The axiomatization checks the consistency of the annotations and for the same model, provides all consistent derivations of set relations. Then, once the consistency of temporal relationships has been evaluated and (under the assumption of consistency) the only possible stable model (`SM`) has been identified, the `SM` can be used to identify temporal relations between events, using the equivalence relation between the temporal relations and order relations defined on time points showed in section 3. For this purpose, a rule for each temporal relation among events (all subclasses of `TemporalRelationEE`) has been defined. Each rule tries to find a particular temporal relationship (before, after, meets, etc.). Each rule has in input two events (`E1` and `E2`) and the totality of relations between time points (stable model `SM`). The rule for finding `BeforeEE` relations between two events is shown:

```
1 findRel(E1,E2,SM) :-
2  not BeforeEE[E1,E2],
3  dt(E1,T1,T2), dt(E2,T3,T4),
4  subset({prec_t(T1,T2),prec_t(T3,T4),prec_t(T2,T4)},SM),
5  insert{BeforeEE[E1,E2]}.
```

In this case, the rule checks if there already exists a `BeforeEE` relation between two events `E1` and `E2` (2), then it reads the end points of occurrence intervals of events `E1` and `E2` (3), and verify the existence of conditions to discover a `BeforeEE` relation, or rather, if the relations on time points, which describe the condition, are a subset of the stable `SM` (4). Finally, the rule asserts the relation discovered in the knowledge base (5).

The above rules have been defined for all temporal relations between events: `AfterEE[E1,E2]`, `MeetsEE [E1,E2]`, etc.

3.3 Causal reasoning

For causal reasoning, an axiomatization (a variant of axiomatization defined in [BOC]) has been defined. The axioms that have been expressed (in a simplification of Flora2, see note 2) are reported as follows:

```
Id1:CausalRelation[A, B]:-                    Strengthening
    Id2:BeforeEE[A, B], demo(A,B),
    Id3:CausalRelation[B,C], newId(Id1, Id2, Id3).
Id1:CausalRelation[A, C]:-                    Weakening
    Id2:CausalRelation[A,B], Id3:BeforeEE[B, C],
    demo(B,C), newId(Id1,Id2,Id3).
Id1:CausalRelation[A,B∧C]:-                   And
    Id2:CausalRelation[A,B], Id3:CausalRelation[A,C],
    newId(Id1,Id2,Id3).
Id1:CausalRelation[A∨B,C]:-                   Or
    Id2:CausalRelation[A,C], Id3:CausalRelation[B,C],
    newId(Id1,Id2,Id3).
Id1:CausalRelation[A, C]:-                    Cut
    Id2:CausalRelation[A,B], Id3:CausalRelation[A∧B,C],
    newId(Id1,Id2,Id3).
Id1:CausalRelation[A∧C,B]:-           Left Monotonicity
    Id2:CausalRelation[A, B], C:Event,
    Id3:BeforeEE[C,B], newId(Id1, Id2, Id3).
Id1:CausalRelation[A,B∨C]:-           Right Monotonicity
    Id2:CausalRelation[A,B], C:Event,
    Id3:BeforeEE[A,C], newId(Id1, Id2, Id3).
```

The predicate `newId(Id1, Id2, Id3)` generates a new id `Id1` depending on `Id2` and `Id32`.

In the axiom Left Monotonicity the condition BeforeEE[C,B] has been included, because C is an event that cause B, and for this reason, C must precede B; otherwise it generates a contradiction.

For the axioms of *Weakening* and *Strengthening*, the meta-predicate `demo(A,B)` (implements the relation "B is deducible from A" [BOC]) have been defined, which was implemented as a variant of the meta-interpreter [BAT].

The axioms for causal relationships shown above have been defined for the class of causal events and are applied to all subclasses of that class.

4 An example of the application of the axiomatization for checking the temporal consistency

In this section, an example of the application of reasoning provided above is presented. In particular, as an example, a generic consistent narrative (without specifying in detail the actions of the events) has been defined.

Let e1, e2, e3 and e4 be four events and r1:DuringEI[e1,i1], r2:AfterET[e3,t2], r3:MeetEE [e1,e2], r4:OverlapsEE[e4,e3], and r5:BeforeET[e2,t2] be the temporal relations annotated.

t1 and t2 are the endpoints of interval i1; [st1, st2] is the symbolic interval of occurrence of e1, [st3, st4] is the symbolic interval of occurrence of e2, [st5, st6] is the symbolic interval of occurrence of e3, [st7, st8] is symbolic interval of e4.

Before performing the consistency check, the temporal relations were translated into temporal relations between time points as described in paragraph 3, obtaining the following: e $prec_t$(t1,t2),$prec_t$(st1,st2),$prec_t$(st3,st4), $prec_t$(st5,st6),$prec_t$(st7,st8),$prec_t$(st7,st5),$prec_t$(st5,st8), $prec_t$(st8,st6),$prec_t$(st2,t2),$prec_t$(t1,st1),$prec_t$(st4,t2), $prec_t$(t2,st5),$prec_t$(st2,st3).

In this set of relations the algorithm for checking consistency(paragraph 3) has been applied. Because the relations are consistent, the program returns to output only one stable model: **stableModelSet**([prect(st3,st4), prect(t2,st5), …, eqt(st3,st2), eqt(st2,st2), eqt(st1,st1), eqt(t2,t2)]).

From this result, new temporal relations through the axioms for discovering new temporal relations (paragraph 3.2) have been identified, in particular, the rules identified the temporal relations beforeEE[e1,e3], and beforeEE[e2,e3].

If we add the relation beforeEI[e2,i1], intentionally making the knowledge base inconsistent, applying the algorithm for the consistency check obtains a more stable model. This response highlights the presence of inconsistencies.

Conclusions

In this paper a formalism for the representation of complex events has been proposed. The formalism for the representation of events is based both on time points and temporal intervals. Associated with this representation, temporal reasoning for checking the consistency and discovering new temporal relations has been constructed.

In this approach, unlike [MAN], the phase for checking the consistency is processed separately from the process for discovering the temporal relations. For this purpose, the axioms of Russell-Kamp [LAM] have been used to discover inconsistencies, and then evaluate the possible extensions of temporal relations. Furthermore, the algorithms must be applied separately, because they relate to different stages of the process of annotation and its use. Consistency checking is related to the content resulting from a process of semantic annotation, in which temporal inconsistencies could arise, while the discovery of temporal relations is a process that can be activated only after the consistency check.

In this approach, then, the process for checking the consistency of the annotation and discovering temporal relations have been separated. The motivation for this choice is so that one can apply different axiomatizations separately and divide complex problems into simple problems.

References

[ALL] J. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM vol. 26 (11), p. 832-843, 1983.

[BAR] C. Baral, M. Gelfond, A. Provetti, *Representing Actions: Laws, Observations and Hypotheses*. J. Log. Program. 31(1-3): 201-243. 1997

[BAT] R. Barták, P. Stepánek, *Extendible Meta-Interpreters*, in: Journal KYBERNETIKA, Volume 33, Number 3, pages 291-310. 1977

[CAS] R. Casati, A. Varzi, "*Events*", The Stanford Encyclopedia of Philosophy (Spring 2010 Edition), E. N. Zalta (ed.),http://plato.stanford.edu/archives/spr2010/entries/events

[DAV] Davidson D., *Essay on Actions and Events*, New York, Oxford university Pers, 1980

[FLO] FLORA-2: *An Object-Oriented Knowledge Base Language* http://flora.sourceforge.net/

[GEL] M. Gelfond, Vladimir Lifschitz, *The Stable Model Semantics For Logic Programming*. In Proceedings of the Fifth International Conference on Logic Programming, pages 1070-1080, Seattle USA, August 1988.

[GIU] F. Giunchiglia, P. Andrews, G. Trecarichi, and R. Chenu-Abente, *Media Aggregation via Events*, Proceedings of the Workshop on Recognising and Tracking Events on the Web and in Real Life, Athens, Greece. 2010.

[GON] G. Gonzalez, C. Baral, M. Gelfond, *Alan: An Action Language For Modelling Non-Markovian Domains*. Studia Logica 79(1): 115-134 (2005)

[LAM] M.V. Lambalgen, F. Hamm. *The Proper Treatment of Events*. Blackwell, Oxford and Boston, 2004.

[LEV] Levesque H. J. , Pirri F., Reiter R., *Foundations for the Situation Calculus*. Electron. Trans. Artif. Intell. 2: 159-178 (1998)

[LIN] Lin F., Reiter R, *Rules as Actions: A Situation Calculus Semantics for Logic Programs*. J. Log. Program. 31(1-3): 299-330 (1997)

[MAJ] J. Ma, *Ontological considerations of time, meta-predicates and temporal propositions*. Applied Ontology, vol. 2, pp. 37-66, 2007.

[MAN] I. Mani, B. Wellner, M. Verhagen, and J. Pustejovsky, *Three approaches to learning Tlinks in timeml*, Technical Report, 2007.

[MEL] Francesco Mele, Antonio Sorgente, Giuseppe Vettigli, *Designing and Building Multimedia Cultural Stories Using Concepts of Film Theories and Logic Programming*. In Proceedings of Cognitive and Metacognitive Educational Systems (MCES) symposium. pag 57-63. ISSN 1613-0073 Vol-598. 2010.

[MIL] R., Miller, M. Shanahan, *The event-calculus in classical logic – alternative axiomatizations*. In Electronic Transactions on AI 3(1): 77-105. 1999

[PUS03] Pustejovsky, J.; Castaño, J.; Ingria, R.; Saurí, R.; Gaizauskas, R.; Setzer, A; G. Katz, *TimeML: A Specification Language for Temporal and Event Expressions*. Netherlands, Kluwer Academic Publishers. 2003.

[PUS08] J. Pustejovsky, K. Lee, H.B. Harry, B. Boguraev, and N. Ide, *Language Resource Management—Semantic Annotation Framework (SemAF)—Part 1: Time and events*, International Organization, 2008.

[SHO] Shoham, Y. *Artificial Intelligence Techniques in Prolog*. Morgan Kaufmann Publishers.

[SHE] Scherp, A. Franz, T.Saathoff, C., Staab, S., *F--a model of events based on the foundational ontology dolce+DnS ultralight*, Proceedings of the fifth international conference on Knowledge capture - K-CAP '09

[SYR] T. Syrjänen, *Lparse 1.0 User's Manual*, http://www.tcs.hut.fi/Software/smodels/

[WAN] H.S. X. Wang, S. Mamadgi, A.Thekdi, A. Kelliher, *Eventory - An Event Based Media Repository*, International Conference on Semantic Computing, 2007, pp. 95-102.

[WIN] T. Winkler, A. Artikis, Y. Kompatsiaris, and P. Milonas, eds., *Workshop Recognising and Tracking Events on the Web and in Real Life,* Athens, Greece: 2010.

Solving XCSP problems by using Gecode

Massimo Morara, Jacopo Mauro, and Maurizio Gabbrielli

University of Bologna.
morara | jmauro | gabbri@cs.unibo.it

Abstract. Gecode is one of the most efficient libraries that can be used for constraint solving. However, using it requires dealing with C++ programming details. On the other hand several formats for representing constraint networks have been proposed. Among them, XCSP has been proposed as a format based on XML which allows us to represent constraints defined either extensionally or intensionally, permits global constraints and has been the standard format of the international competition of constraint satisfaction problems solvers. In this paper we present a plug-in for solving problems specified in XCSP by exploiting the Gecode solver. This is done by dynamically translating constraints into Gecode library calls, thus avoiding the need to interact with C++.

1 Introduction

Constraint Programming [13] has attracted high attention among experts from many areas because of its potential for solving hard real life problems and because it is based on a strong theoretical foundation. The success of Constraint Programming (CP) derives from the fact that on one hand it allows to model a problem in a simple way and on the other hand it provides an efficient problem solving algorithms. However, the CP community lacks a standardized representation of problem instances and this still limits the acceptance of CP by the business world. One attempt to overcome this problem was taken by the Association for Constraint Programming with the proposal of Java Specification Request JSR-331 "Constraint Programming API" [8, 5]. The goal of this specification is the creation of a powerful API for specifying CP problems. In the last five years other approaches focusing on more low level languages emerged. The aim of these approaches is to define a minimal domain dependent language that supports all the major constraint features and requires, at the same time, a minimal implementation effort to be supported by constraint solvers. Two languages following this goal are worth mentioning: FlatZinc [9] and XCSP [12]. The former was originally created to be the target language into which a higher level CSP instance (e.g. a CSP modeled with MiniZinc [11]) is translated. Today FlatZinc is also used as a low level "lingua franca" for solver evaluation and testing. For instance, since 2008 FlatZinc has been used in the MiniZinc Challenge [7, 14], a competition where different solvers are compared by using a benchmark of MiniZinc instances that are compiled into FlatZinc.

XCSP is a language structurally very similar to FlatZinc. XCSP was defined with the purpose of being a unique constraint model that could be used by all

the CP solvers. It was first proposed in 2005 for the solvers competing in the International CSP Solver Competition [1], and has then been used in other contexts and extended. In this paper, we focus on XCSP. In particular, we consider its current version, i.e. XCSP version 2.1.

The need of a standard is also caused by the huge number and diversity of solvers. Today only few solvers support natively FlatZinc or XCSP. Unfortunately, Gecode [3], one of the most well known and used solvers, support FlatZinc only. For this reason we have created *x4g*, a plug-in that allows us to solve problems defined in XCSP by using the Gecode solver. The goal of x4g is twofold. Firstly, we want to exploit Gecode for solving problems that are specified in XCSP without considering low level implementation details or writing a single line of code in C++. Secondly, we want to provide a tool that can be used to evaluate the performances of the Gecode solver with respect to the other entries of the International Solver Competition. This could be very interesting since, to the best of our knowledge, the benchmark used in the International Solver Competition is the biggest available to the CP community.[1]

In the reset of this paper, we present in Section 2 a brief overview of the XCSP language and of the Gecode solver. In Section 3 we describe the idea behind the x4g plug-in. Section 4 concludes by mentioning some directions for future works.

2 Background

An extensive presentation of the XCSP format and the Gecode solver is beyond the scope of this paper. Here we just give a short overview of XCSP and Gecode.

2.1 XCSP

XSCP is an extended format to represent constraint networks using XML. The Extensible Markup Language (XML) is a simple and flexible text format playing an increasingly important role in the exchange of a wide variety of data on the Web. The objective of the XML representation (in XCSP) is to ease the effort required to test and compare different algorithms by providing a common testbed of constraint satisfaction instances. The proposed representation is low-level: for each instance the domains, variables, relations (if any), predicates (if any) and constraints are exhaustively defined. No control flow constructs like"for" cycles or "if then else" statements can be used.

Roughly speaking, there exist two variants of this format: a fully-tagged representation and an abridged representation. The first one is a full XML, completely structured representation which is suitable for using generic XML tools but is quite verbose and tedious to use for a human being. The second representation is just a shorthand notation of the first one and it is easier to read and to write for a human being, but less suitable for generic XML tools.

[1] The MiniZinc Challenge has a smaller benchmark of instances and fewer participants than the International Solver Competition.

As an example of an XCSP program consider the following one where the well known "all different" constraint is applied to two variables $A1$ and $A2$ which can assume values only in the domain $[1, 2]$.

```
<domains nbDomains="1">
  <domain name="d0" nbValues="2">1..2</domain>
</domains>
<variables nbVariables="2">
  <variable name="A1" domain="d0"/>
  <variable name="A2" domain="d0"/>
</variables>
<constraint name="c0"  arity="2"
     scope="A1 A2"
     reference="global:alldifferent"/>
```

2.2 Gecode

Gecode (Generic Constraint Development Environment) provides a constraint solver with state-of-the-art performance while being modular and extensible. It supports the programming of new propagators, branching strategies, and search engines. New domains can be programmed at the same level of efficiency as finite domain and integer set variables that are already predefined. Furthermore Gecode is distributed under a very permissive license, it is portable and well documented, and comes with a complete tutorial. All these features have made Gecode one of the preferred choices for solving CSPs.

Gecode is written in C++ and supports the FlatZinc format through an external plug-in that is able to parse a FlatZinc instance and solve it using Gecode library calls. The use of this plug-in allowed Gecode to participate to the 2010 MiniZinc challenge [7], where it won all the tracks of the competition.

3 x4g

In principle the translation of XCSP instances into Gecode is similar to the task performed by the Gecode/FlatZinc plug-in [2] that parses a FlatZinc instance and produces an internal data structure (Gecode Space Object) that can be later used to retrieve the solution of the CSP problem.

To use Gecode in order to solve CSPs defined in XCSP, one could try implement a XCSP to FlatZinc compiler. However, in order to avoid potential loss of information, to be more flexible and less dependent on the Gecode/FlatZinc plug-in, we chose to provide a direct translation from XCSP into Gecode. Hence we have developed x4g; a plug-in that parses an XCSP instance and, for every constraint defined within the XCSP file, generates an equivalent number of Gecode constraints. When all the XCSP constraints are translated into Gecode constraints, a Gecode Space Object is returned. This object can later be used to

get a solution to the CSP problem by using one of the many predefined search strategies provided by Gecode or local search strategies following [10].[2]

To develop the x4g plug-in we used the XCSP parser provided by the International CSP competition organizers. This parser, developed in particular to support the abridged notation, is written in C++ by using the well known libxml2 libraries [6]. Unfortunately, the parser supports only a limited number of global constraints. Therefore, we modified it in order to support additional ones.

The XCSP format is mainly used to specify CSPs but it also supports extensions to define weighted constraints or quantifiers over constraints. x4g was designed to target only CSPs, hence it does not support these additional features.

XCSP supports only finite domains. Since Gecode supports finite domains too, the domain encoding from XCSP to Gecode was straightforward. XCSP provides a construct which allows one to define relations over variables. These could be seen as constraints listing all the admissible values that some variables can take. XCSP relations are mapped into Gecode by using extensional constraints. In XSCP the semantics of a relation can also be given by stating all the non admissible values of the variables (i.e. conflicts between variables). We used inequality constraints for translating these relations into Gecode. Another construct of XCSP is predicate, a boolean parametric expression that is considered satisfied if and only if it evaluates to true. The number of parameters in a predicate is fixed and, differently from relations, predicates allow integer parameters. The mapping of the predicates was also straightforward because, with only few exceptions, Gecode has for every predicate an API for posting an equivalent constraint. As far as global constraints are concerned, XCSP supports the majority of the global constraints defined in the Global Constraint Catalog [4]. Since in this catalog there are hundreds of global constraints, a full XCSP support means to provide an encoding for a huge number of global constraints. This was out of the scope of the project, we have just chosen to support a subset of the most used global constraints. Currently x4g supports the following global constraints: `alldifferent`, `among`, `atleast`, `atmost`, `cumulative`, `diffn`, `disjunctive`, `element`, `global_cardinality`, `lex_less`, `lex_lesseq`, `not_all_equal`, `weightedSum`.[3]

Finally, to give an example of how x4g could be used, we developed a program that takes as input a XCSP instance and, by using x4g, allows us to find a solution by using the deep first strategy natively implemented in Gecode. Since the output of this program follows the output rules of the International Solver Competition, it could be used to let Gecode enter the next International CSP Competition.

[2] Note that the XCSP format specifies only the constraints, the choice of the search strategy is left to the user.

[3] For a precise definition of these constraints see [4].

4 Conclusion and Future Work

In this paper we described x4g, a plug-in that allows the use of Gecode for solving a CSP instance defined using the XCSP format. The source code of x4g and of the above mentioned program can be found at `http://www.cs.unibo.it/~jmauro/cilc_2011.html`.

This work has to be considered as a first step in the direction of providing a full translation of XCSP constraints into Gecode constraints. As a future work, we would like to support more global constraints and to define efficient ways of decomposing them by using Gecode constraints. We also would like to extend our tool in order to use Gecode for solving the optimization problems that can be defined in XCSP exploiting weighted constraints.

Moreover, we are interested in the development of a FlatZinc/XCSP conveyer that will allow us to add FlatZinc instances into the benchmarks that could be used for comparing Gecode with other constraint solvers. Furthermore, with such a converter it could be also possible to compare the efficiency of our x4g translation with the one of the Gecode/FlatZinc plug-in.

References

1. Fourth International CSP Solver Competition website: `http://www.cril.univ-artois.fr/CPAI09/`.
2. Gecode/FlatZinc plugin website: `http://www.gecode.org/flatzinc.html`.
3. Generic constraint development environment website: `http://www.gecode.org/`.
4. Global Constraint Catalog website: `http://www.emn.fr/x-info/sdemasse/gccat`.
5. JSR 331: Constraint Programming API website: `http://jcp.org/en/jsr/summary?id=331`.
6. libxml2 libraries. Available at: `http://xmlsoft.org/`.
7. MiniZinc Challenge 2011 website: `http://www.g12.csse.unimelb.edu.au/minizinc/challenge2011/challenge.html`.
8. Standardization of Constraint Programming website: `http://4c110.ucc.ie/cpstandards/`.
9. Ralph Becket. Specification of FlatZinc. version 1.3. Available at `http://www.g12.csse.unimelb.edu.au/minizinc/downloads/doc-1.3/flatzinc-spec.pdf`.
10. Raffaele Cipriano, Luca Di Gaspero, and Agostino Dovier. A Hybrid Solver for Large Neighborhood Search: Mixing Gecode and EasyLocal^{++}. In *Hybrid Metaheuristics*, pages 141–155, 2009.
11. Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. MiniZinc: Towards a Standard CP Modelling Language. In *CP Proceeding*, pages 529–543, 2007.
12. Organising Committee of the Third International Competition of CSP Solvers. XML Representation of Constraint Networks Format XCSP 2.1, 2009. Available at `http://www.cril.univ-artois.fr/CPAI08/XCSP2_1.pdf`.
13. Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
14. Peter J. Stuckey, Ralph Becket, and Julien Fischer. Philosophy of the MiniZinc challenge. *Constraints*, 15(3):307–316, 2010.

Formalization and Automated reasoning about a Complex Signalling Network

Annamaria Basile[1], Maria Rosa Felice[2], Alessandro Provetti[1]

[1] Dept. of Physics - Informatics section, Univ. of Messina. Messina, Italy.
[2] Dept. of Life Sciencies, Univ. of Messina. Messina, Italy.

Abstract. Tran and Baral have proposed an action language (BioSigNet-RR) that is specific for the modeling of signalling networks from Biology and for answering queries relative to the expected response to a stimulus. Translation of their action language to logic programs under Answer Set semantics yields a reasoning mechanisms that gracefully handles incomplete/partial information, updates etc. Those features are extremely important since existing regulatory networks often contain missing or suspected interaction links, or proven interactions whose outputs are uncertain. We present our application experience in developing a BioSigNet-RR formalization of the Signalling network for Arabidopsis Brassinosteroid, a complex interaction that is at the base of growth in some plant species. Such modeling exercise has involved 'filling the gaps' between the terse graphical language of signalling networks literature and the precise specification of the triggering conditions required by BioSigNet-RR. This application experience leads us to propose a new formalization style for action theories representing signaling networks that allows for the description of non-immediate effects of actions. Empirical evaluation of our declarative model has involved formulating and testing several 'what if' queries and checking the quality of the answer with domain experts.

1 Introduction

In Biology, *signalling networks* (also referred to as *signalling pathways)* are specific collections of interactions with a common theme. They are used to provide a summary, working model of the complex interactions that explain how a living cell receives and responds to signals from its environment. Modeling signalling networks is sometimes essential for understanding how cells function and it may lead to effective therapeutic strategies that correct or alter abnormal cell behaviors.

From the point of view of Artificial Intelligence signalling networks represent an interesting form of semi-formal knowledge representation: relevant cellular interactions are to be explicitly described, in a simple graphical language. However, two main issues make modeling signalling networks with action languages challenging:

1. *inhibition,* which we see as a special form of constraint, needs to be explicitly represented and reasoned about,
2. several unspoken assumptions lie in the background as they are assumed to be known by the (expert) reader. For instance, the time element, i.e., a description of the delay between stimulus and reaction, is not explicitly represented yet it is an essential element in reasoning about the long-term evolution of the cell.

This extended abstract reports the results of our work on formalization and automated reasoning with a signalling network that is considered[3]representative of the size, level of detail and complexity of such networks in the Biology literature. The chosen pathway, which is depicted in Figure 1, represents the current knowledge on the interaction that makes the arabdiposis of Brassinosteroids (represented by the lone diamond-shaped node in the pathway) stimulate growth in plants.

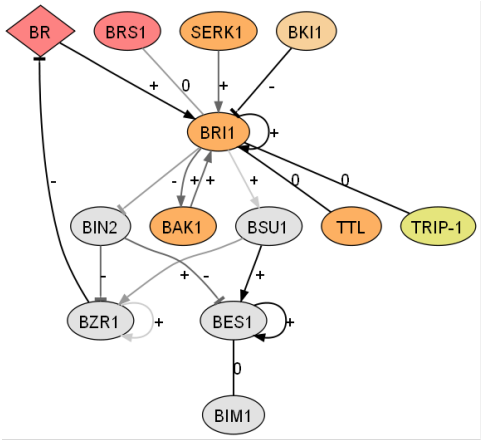


Fig. 1. Signalling network for BR, from the Web site of *Science Signaling*.

Chory et al. [3] presents the pathway and comments in detail each interaction; at this level, we can identify each arc with one relevant interaction. The nodes of the pathway represents disparate physical elements: extra-cellular signals, receptors on the cell's surface or intracellular elements able to *trasduce* signals to the nucleus. Color ans shape of each node guide the expert reader on the actual nature of the element being represented.

From the point of view of knowledge representation and reasoning, the following scientific hypotheses motivate our work.

First, there is a question about the adequacy of BioSigNet-RR [2,8] to support the formalization and reasoning about this specific signalling pathway. As we will see, some specific aspect of the cellular organization is not easily described in terms of BioSigNet-RR primitives.

Second, there is a question of adequacy of our action languages with respect to reasoning about pathway interactions. We consider the following informal test: can we apply BioSigNet-RR to formalize the type of 'what if' questions that an examiner would ask to check a student's assimilation of the material. Hence, we proceed to formulate some easy sample questions and see how the query language part of BioSigNet-RR allows to formulate it.

[3] Paccanaro and Bogre, personal communications.

Finally, our overall hypothesis is that, in the middle term, we should be able to design and implement a vertical solution by which the domain knowledge synthesized in a signalling pathway can be accessed and reasoned about automatically.

2 The representation language

Our modeling effort has adopted the BioSigNet-RR [2,8] language as it now considered the language of reference for reasoning about actions in the Biological domain. Essentially, BioSigNet-RR is an extension of the family of action languages developed by Gelfond and Lifschitz in the 90s; we refer the interested reader to [7] for a survey of the approach. Gelfond and Lifschitz proposed a sorted language, where sorts are actions and fluents, where primitives are the well-known *initially* and *causes* statements. A set of those statements is called an action theory. State is defined in terms of a set of fluents that are deemed true thereof. A declarative semantics is assigned to action theories in terms of trajectories, i.e., an iteration throughout states that the domain in undergoing. At the same time, action languages receive a semantics thanks to translation of action statements to logic programs under answer set semantics [6,1].

When the initial situation is only partially defined, or actions are unknown or even may have non-deterministic effects, alternative answer sets account for the alternative evolutions of the domain. The translation from action languages, including BioSigNet-RR, to logic programs is *modular,* in the sense that it can be done line-by-line by a parser and generator. The translation, which is described in detail in [] has been adopted as is and implemented by a Python-language program derived from Gregory Gelfond's Al2ASP project [4] (see [5] for another application project on the same guidelines).

3 Formalizing the Background knowledge

Signalling pathways are a graphical, synthetic representation of knowledge. However, to fully grasp the dynamics represented by the pathway one often needs to read attentively a natural-language background description that comes with the network.

In our project we have spent a great amount of time to understand and organize the background description of the Arabidopsis Brassinosteroid process given by Chory et al. [3]. The following phrases have been singled out and analyzed separately.

1. In the absence of steroid, BKI1, a plasma membrane-associated protein, interacts directly with the kinase domain of BRI1 to negatively regulate the signalling pathway
2. Binding of BRs to preformed BRI1 homo-oligomers leads to the dissociation of BKI1 from the plasma membrane.
3. It has been proposed that the physical interaction between BRI1 and BAK1 leads to the formation of a signalling-competent hetero-oligomer.
4. The signals transmitted from the plasma membrane-localized BRI1-BAK1 hetero-oligomer negatively regulate the activity of a glycogen synthase kinase 3 (GSK-3), called BIN2.

5. Although the mechanism is as of yet uncharacterized, inactivation of BIN2 leads to the dephosphorylation of BES1 and BZR1, members of a new family of plant-specific transcription factors.
6. BES1, and likely other family members, are further dephosphorylated through the activity of a nuclear-localized, kelch-containing protein phosphatase BSU1.
7. Current data suggest that dephosphorylated BES1 is then able to form homo or heterodimers with the basic helix -loop-helix (bHLH) transcription factor BIM1, to bind to E-box elements in the promoters of BR-regulated genes.
8. Dephosphorylated BZR1 binds to a novel element in the promoters of BR biosynthetic genes to repress their expression.
9. Because BES1 is identical to BZR1, it is expected that BES1 and BZR1 will have both activating and repressing activities.
10. Other proteins have been identified that interact genetically or physically with BRI1, but their precise functions are currently unknown.
11. In vitro and in vivo, BRI1 associates with TTL, transthyretin-like protein.
12. Overexpression of TTL causes slight dwarfing, suggesting that it may play a negative role early in the BR signalling pathway.
13. A suppressor screen using a weak allele of bri1 identified a secreted and active carboxypeptidase, called BRS1, although its molecular target in the BR signalling pathway is unknown.
14. Biochemical studies identified TRIP-1 as a BRI1 interactor; however, the knockdown in expression of TRIP family members yields a pleiotropic phenotype that is slightly reminiscent of those observed for BR biosynthetic or signalling mutants.

It relatively easy to associate each phrase to one of the arcs represented by the signalling pathway. Such association, however, is not always straightforward and will be further commented upon.

It must be pointed out that in our analysis we have discovered that the interaction which is represented by arc:

$$BR1 \stackrel{+/-}{\Longrightarrow} BAK1$$

is not found in the pathway depicted in Chory et al. [3], from where our work started, but is in Figure 1, which was later found on the Web site of the *Science Signaling*[4] journal.

3.1 Formalization of the Signalling Pathway for BR

The formalization of the pathway proceeds as follows. For each named cellular component, e.g., BR, we introduce two fluents[5]: *high(br)* e *low(br)*. Then, we introduce two *activation* actions: *activate(br)* and *inactivate(br)*, where the latter is an *inhibition* action that, in some sense, depresses BR.

[4] `http://www.sciencemag.org`

[5] The labels used in the signalling pathway are in lowercase, since they are constant names in the domain description.

Next, the remaining arcs are formalized, by coupling each arc to the illustrative phrase found in the description. For instance, the arc connecting BRI1 to BAK1 is connected to the phrase:

"BRI1 interacts directly with BAK1 [through a phosphorylation process]."

$$activate(bak1) \textbf{ causes } up(bri1) \tag{1}$$

"BKI1 interacts directly with the kinase domain of BRI1 to negatively regulate the signalling pathway."

$$high(bki1) \textbf{ inhibits } activate(bri1) \tag{2}$$

"Binding of BR to preformed BRI1 homo-oligomers lead to the dissociation of BKI1 from the plasma membrane."

$$binding(br, bki1) \textbf{ causes } dissociated(bki1) \textbf{ if } high(bri1) \tag{3}$$

Even though it looks like the description of a local, direct interaction, this formalization may be the most effective in capturing the rationale of the pathway. An alternative formulation, which has hitherto not been tested is the following:

$$high(br)\ high(bri1) \textbf{ triggers } dissociated(bki1)$$

"A suppressor screen using a weak allele of BRI1 identified a secreted and active carboxypeptidase, called BRS1, although its molecular target in the BR signaling pathway is unknown. It has been proposed that the physical interaction between BRI1 and BRS1 leads to the formation of a signaling-competent hetero-oligomer."

$$high(brs1) \textbf{ triggers } activate(bri1) \tag{4}$$

"In vitro and in vivo, BRI1 associates with TTL, transthyretin-like protein. Overexpression of TTL causes slight dwarfing, suggesting that it may play a negative role early in the BR signaling pathway."

$$high(bri1) \textbf{ triggers } downregulate(ttl) \tag{5}$$

"Biochemical studies identified TRIP-1 as a BRI1 interactor."

$$high(bri1) \textbf{ triggers } activate(trip1) \tag{6}$$

"The signals transmitted from the plasma membrane-localized BRI1-BAK1 heterooligomer negatively regulate the activity of a glycogen synthase kinase 3 (GSK-3), called BIN2."

$$high(bri1),\ high(bak1) \textbf{ inhibits } activate(bin2) \tag{7}$$

"Although the mechanism is as yet uncharacterized, inactivation of BIN2 leads to the dephosphorylation of BES1 and BZR1, members of a new family of plant-specific transcription factors."

$$inactivate(bin2) \textbf{ causes } low(bzr1) \tag{8}$$

$$inactivate(bin2) \textbf{ causes } low(bes1) \tag{9}$$

"Dephosphorylated BZR1 binds to a novel element in the promoters of BR biosynthetic genes to repress their expression."

$$high(bzr1) \textbf{ triggers } activate(br) \tag{10}$$

"Current data suggest that dephosphorylated BES1 is then able to form homo-or hetero-dimers with the basic helix-loop-helix (bHLH) transcription factor BIM1, to bind to E-box elements in the promoters of br-regulated genes."

$$high(bes1) \textbf{ triggers } activate(bim1) \tag{11}$$

$$high(bsu1) \textbf{ triggers } activate(bes1) \tag{12}$$

$$low(bsu1) \textbf{ inhibits } activate(bes1) \tag{13}$$

$$high(bri1) \textbf{ triggers } activate(bsu1) \tag{14}$$

$$high(serk1) \textbf{ triggers } activate(bri1) \tag{15}$$

It should be noticed again that in Chory et al. [3] the textual description seems not aligned to the graphics of the pathway. As a result, we tentatively interpret the inhibition from BRI1 to BIN2 with:

$$high(bri1) \textbf{ inhibits } (bin2) \tag{16}$$

3.2 Connecting actions to fluents

For each fluent we have had to introduce a couple of actions that represent the *upregulation* and *downregulation* of the fluent itself. Hence, we need to introduce the following two schematic rules, to be instantiated to each fluent:

$$activate(C) \textbf{ causes } high(C) \tag{17}$$

$$downregulate(C) \textbf{ causes } low(C) \tag{18}$$

In a sense, these axiom schemata naturally complement the action theory by capturing the essence of the $+/-$ labeling of the arcs. However, they introduce an extra level of complexity in the representation, since new conditions must be devised to disallow these definitions for specific values of *C,* e.g., *activate(bak1),* that do not have a direct Biological interpretation.

4 Query Formulation and informal validation of the model

To assess the adequacy of the representation language and of our specific action theory we have considered the following classroom scenario: questions about the Arabdiposis brassinosteroid process that a teacher would use to check whether her students have properly learn the material and are now able to reason about Brassinosteroids and their effects. Such questions were formulated with the goal of stressing the connections between the several cellular components of the cell.
To illustrate how natural BioSigNet-RR queries are, we now list, for each question, the expected answer in English, paired with BioSigNet-RR that captures, to some extent, the question itself.

Q: Looking at the pathway, how does *BR* affect the cell?
A: BR causes the activation of *BRI1* and *BAK1*, which, in turn, inhibit the activation of *BIN2*.

Our formalization is based on three distinct queries:

$$? - high(bri1) \textbf{ after } activate(br) \tag{19}$$

$$? - high(bak1) \textbf{ after } activate(br) \tag{20}$$

$$? - low(bin2) \textbf{ after } activate(br) \tag{21}$$

Q: What are the effects of activation of *BAK1*?
A: BAK1 brings about activation of *BR1*; subsequently, *BR1* shall affect the whole cell network.

This question can be translated directly in the following formula (query):

$$? - high(bri1) \textbf{ after } activate(bak1) \tag{22}$$

Q: What are the effects of inactivation of *BIN2*?
A: inactivation of *BIN2* shall cause the inhibition of *BZR1* and *BES1*.

Again, me must resort to two separated queries:

$$? - low(bzr1) \textbf{ after } activate(bin2) \tag{23}$$

$$? - low(bes1) \textbf{ after } activate(bin2) \tag{24}$$

5 Conclusions

The formalization and deployment project described in this article can be considered successful from the point of view of assessing what can be done with action languages (and Logic Programming in general) in the context of Biological knowledge representation and automated reasoning. The overall Artificial intelligence goal, i.e. to have computers process the meaning synthesized in a signalling pathway without human intervention, is yet to be achieved, as our formalization had to deal with a time-consuming human analysis of the accompanying textual explanation, often a heavy-going technical explanation.

BioSigNet-RR has shown to be an ideal platform for formalization in this domain. However, we believe that more research is needed in order to have the action theory match the pathway. One problem that was, in our opinion, only partially solved here is that *action* is understood in two ways. The first understanding, which is probably what Gelfond-Lifschitz meant, is that of *external intervention,* i.e., alteration of the state of affairs. The second understanding, which we suspect is more frequent in Biology, is that an action may also be an alteration, called *upregulation* or downregulation, of a component of the cell. For this second understanding, a specific action sort should be devised and introduced to the formalization language.

Acknowledgments

We are grateful to Alberto Paccanaro and Laszlo Bogre at Royal Holloway, University of London for providing us careful advice and guidance in the Biological and Bioinformatics aspects of this work. Many thanks to Gregory Gelfond for providing us constant advice and support on our work to extend his AL2ASP program.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Baral, C., Chancellor, K., Tran, N., Joy, A.M., Berens, M.E.: A knowledge based approach for representing and reasoning about signalling networks. In: ISMB/ECCB (Supplement of Bioinformatics). pp. 15–22 (2004)
3. Chory, J., Belkhadir, Y., Wang, X.: Arabidopsis brassinosteroid signalling pathway. Science Signaling 364, cm5 (2006)
4. Gelfold, G.: From AL to ASP - the system al2asp. Tech. rep., Arizona State University (2011)
5. Gelfold, M., Inclezan, D.: Yet another modular action language. In: Proc. of Int'l Workshp on Software Engineering for Answer Set Programming. pp. 64–78. CEUR WS Proceedings, vol. 546. (2009)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
7. Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. Artif. Intell. 2, 193–210 (1998)
8. Tran, N., Baral, C.: Hypothesizing about signalling networks. Journal of Applied Logic 7(3), 253–274 (2009)