Domenico Cantone, Marianna Nicolosi Asmundo (Eds.)

# CILC 2013

28th Italian Conference on Computational Logic
Catania, Italy, September 25 - 27, 2013
Proceedings

*Editors's address:*
Domenico Cantone, Marianna Nicolosi Asmundo
University of Catania
Dipartimento di Matematica e Informatica
Città universitaria Viale A. Doria 6
95125 Catania, Italy

cantone@dmi.unict.it, nicolosi@dmi.unict.it

## Foreword

The $28^{th}$ Italian Conference on Computational Logic, CILC 2013, was hosted by the University of Catania from September $25^{th}$ to September $27^{th}$ 2013. The event was the $28^{th}$ edition of the annual meeting organized by GULP (Gruppo ricercatori e Utenti Logic Programming). Since the first conference, which took place in Genoa in 1986, the annual conference organized by GULP is the most important occasion for meeting and exchanging ideas and experiences among Italian users, researchers and developers, who work in the field of computational logic.

The program included 24 technical papers accepted for presentation (17 for long presentation and 7 for short presentation). Authors were mainly affiliated to Italian universities, but some of them belonged to universities of other countries (Mexico, France, Argentina, Finland, Iceland, Spain, and United States). Paper selection was made by peer reviewing: each submitted paper was assigned to at least three members of the Program Committee, who in many cases availed themselves of the help of external referees.

Technical presentations concerned several different topics related to computational logic, including verification of logic programs, answer set programming, proof and decision systems for several non-classical logics, computable set theory, machine learning. The quality of the technical contributions confirms that the Italian community of computational logic is lively and active.

The program included also three invited talks and a tutorial. The invited talks were given by Maria Paola Bonacina, who reviewed recent trends and current developments on model-based reasoning; by Eugenio G. Omodeo, who illustrated the state-of-the-art of proof-verification technology based on set theory and surveyed the proof checker ÆtnaNova/Referee; and by Alberto Policriti, who presented the result on the decidability of the satisfiability problem for the class of purely universal formulae in set theory. The tutorial was given by Joanna Golińska-Pilarek, who presented specific methodological principles of constructing relational dual tableaux, also illustrating their applications to non-classical logics.

A selection of the accepted papers will appear in a special issue of a scientific journal. The complete program, with links to full papers and presentation slides, is available at `http://www.dmi.unict.it/~cilc2013/en/programma.html`.

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

Domenico Cantone, Marianna Nicolosi Asmundo
October 2013

## Program Committee

Domenico Cantone (co-chair), University of Catania, Italy
Simona Colucci, Tuscia University, Viterbo, Italy
Stefania Costantini, University of Aquila, Italy
Alessandro Dal Palù, University of Parma, Italy
Wolfgang Faber, University of Calabria, Italy
Fabio Fioravanti, University of Chieti-Pescara, Italy
Camillo Fiorentini, University of Milano, Italy
Andrea Formisano, University of Perugia, Italy
Marco Gavanelli, University of Ferrara, Italy
Laura Giordano, University of Piemonte Orientale, Italy
Francesca Alessandra Lisi, University of Bari, Italy
Cristiano Longo, Network Consulting Engineering Srl (Valverde, Catania), Italy
Viviana Mascardi, University of Genova, Italy
Marianna Nicolosi Asmundo (co-chair), University of Catania, Italy
Eugenio G. Omodeo, University of Trieste, Italy
Mimmo Parente, University of Salerno, Italy
Alberto Pettorossi, University of Roma Tor Vergata, Italy
Carla Piazza, University of Udine, Italy
Enrico Pontelli, New Mexico State University, USA
Gian Luca Pozzato, University of Torino, Italy
Maurizio Proietti, IASI-CNR (Roma), Italy
Alessandro Provetti, University of Messina, Italy
Luca Pulina, University of Sassari, Italy
Silvio Ranise, Fondazione Bruno Kessler (Trento), Italy
Francesco Ricca, University of Calabria, Italy
Fabrizio Riguzzi, University of Ferrara, Italy
Giuseppe Scollo, University of Catania, Italy
Valerio Senni, IMT Institute for Advanced Studies (Lucca), Italy
Fausto Spoto, University of Verona, Italy
Cesare Tinelli, University of Iowa, USA
Paolo Torroni, University of Bologna, Italy
Alexandru Tomescu, University of Helsinki, Finland

## Organizing Committee

Domenico Cantone, University of Catania, Italy
Salvatore Cristofaro, University of Catania, Italy
Cristiano Longo, Network Consulting Engineering Srl (Valverde, Catania), Italy
Marianna Nicolosi Asmundo, University of Catania, Italy

# Contents

## INVITED TALKS AND TUTORIAL



## LONG PRESENTATIONS

## SHORT PRESENTATIONS

The following papers were presented at the CILC 2013 conference but have not been included in this volume because they appear elsewhere.

The paper *A foundational view of co-LP* (by Davide Ancona and Agostino Dovier) is a revised version of the paper *co-LP: Back to the Roots,* to appear in the technical communications of the 29th International Conference on Logic Programming (ICLP 2013).

The paper *A Tableau System for Right Propositional Neighborhood Logic over Finite Linear Orders: an Implementation* (by Davide Bresolin, Dario Della Monica, Angelo Montanari, and Guido Sciavicco) is an extended version of a paper with the same title appeared in the *Proceedings of TABLEAUX 2013*, Lecture Notes in Computer Science 8123, pp. 74–80, 2013.

The paper *Using log as a Test Case Generator for Z Specifications* (by Maximiliano Cristià and Gianfranco Rossi) is a revised version of the paper *log as a Test Case Generator for the Test Template Framework* (by M. Cristià, G. Rossi, C. Frydman) to appear in R. M. Hierons, M. Merayo, and M. Bravetti (Eds.), SEFM 2013: 11th International Conference on Software Engineering and Formal Methods, vol. 8137 of LNCS, Springer-Verlag, 229-243, 2013.

The paper *A Declarative Modeling Language for Concept Learning in Description Logics* (by Francesca Alessandra Lisi) already appeared in: F. Riguzzi, F. Zelezny (Eds.). Inductive Logic Programming, 22nd International Conference, ILP 2012, Dubrovnik, Croatia, September 17-19, 2012, Revised Selected Papers. Series: Lecture Notes in Computer Science, Vol. 7842, 151-165, Springer (2013).

# On Model-Based Reasoning: Recent Trends and Current Developments

Maria Paola Bonacina

Dipartimento di Informatica Università degli Studi di Verona

**Abstract.** Proofs and models are the mainstay of automated reasoning. Traditionally, proofs have taken center stage, because in first-order logic theorem proving is semi-decidable, and model building is not. The growth of algorithmic reasoning and of its applications to software has changed the balance, because if satisfiability is decidable, symmetry is restored, and models are useful for applications and intuitive for users. While first-order provers search for a proof and may use an interpretation to guide the search, algorithmic reasoners search for a model, use deduction to drive the search, and the candidate model to guide the deduction. Thus, the symmetry is also in the reasoner's operations. However, decidability comes at the expense of expressivity. After some historical perspective on the evolution from proof to model oriented reasoning, we present a method, named DPLL(Gamma+T), which integrates algorithmic reasoner and first-order prover, in such a way that each does what it is best at, and the prover also makes use of the candidate model.

# Proof verification within set theory

Eugenio G. Omodeo

Dipartimento Matematica e Geoscienze sez. Matematica e Informatica,
Università degli Studi di Trieste

**Abstract.** The proof-checker ÆtnaNova, aka Ref, processes proof scenarios to establish whether or not they are formally correct. A scenario, typically written by a working mathematician or computer scientist, consists of definitions, theorem statements and proofs of the theorems. There is a construct enabling one to package definitions and theorems into reusable proofware components. The deductive system underlying Ref mainly first-order, but with an important second-order feature: the packaging construct just mentioned is a variant of the Zermelo-Fraenkel set theory, ZFC, with axioms of regularity and global choice. This is apparent from the very syntax of the language, borrowing from the set-theoretic tradition many constructs, e.g. abstraction terms. Much of Ref's naturalness, comprehensiveness, and readability, stems from this foundation; much of its effectiveness, from the fifteen or so built-in mechanisms, tailored on ZFC, which constitute its inferential armory. Rather peculiar aspects of Ref, in comparison to other proof-assistants (Mizar to mention one), are that Ref relies only marginally on predicate calculus and that types play no significant role, in it, as a foundation.
This talk illustrates the state-of-the-art of proof-verification technology based on set theory, by reporting on 'proof-pearl' scenarios currently under development and by examining some small-scale, yet significant, examples of use of Ref. The choice of examples will reflect today's tendency to bring Ref's scenarios closer to algorithm-correctness verification, mainly referred to graphs. The infinity axiom rarely plays a role in applications to algorithms; nevertheless the availability of all resources of ZFC is important in general: for example, relatively unsophisticated argumentations enter into the proof that the Davis-Putnam-Logemann-Loveland satisfiability test is correct, but in order to prove the compactness of propositional logic or Stone's representation theorem for Boolean algebras one can fruitfully resort to Zorn's lemma.

# On the decidability of the $\exists^*\forall^*$ prefix class in Set Theory

Alberto Policriti

Dipartimento di Matematica e Informatica, Universit degli Studi di Udine

**Abstract.** In this talk I will describe the set-theoretic version of the Classical Decision Problem for First Order Logic. I will then illustrate the result on the decidability of the satisfiability problem class of purely universal formulae ($\exists^*\forall^*$-sentences) on the unquantified language whose relational symbols are membership and equality. The class we studied is, in the classical (first order) case, the so-called Bernays-Schoenfinkel-Ramsey (BSR) class. The set-theoretic decision problem calls for the existence of an algorithm that, given a purely universal formula in membership and equality, establishes whether there exist sets that substituted for the free variables will satisfy the formula. The sets to be used are pure sets, namely sets whose only possible elements are themselves sets. Much of the difficulties in solving the decision problem for the BSR class in Set Theory came from the ability to express infinity in it, a property not shared by the classical BSR class. The result makes use of a set-theoretic version of the argument Ramsey used to characterize the spectrum of the BSR class in the classical case. This characterization was the result that motivated Ramsey celebrated combinatorial theorem.

# Relational Dual Tableaux: Foundations and Applications

Joanna Golińska-Pilarek

Institute of Philosophy, University of Warsaw

The origin of dual tableaux goes back to the paper [RAS60] of Rasiowa and Sikorski, where a cut-free deduction system for the classical first-order logic has been presented. Systems in the Rasiowa-Sikorski style are top-down validity checkers and they are dual to the well known tableau systems. The common language of most of relational dual tableaux is the logic of binary relations which was introduced in [ORL88] as a logical counterpart to the class of representable relation algebras given by Tarski in [TAR41].

The relational methodology enables us to represent within a uniform formalism the three basic components of formal systems: syntax, semantics, and deduction apparatus. Hence, the relational approach can be seen as a general framework for representing, investigating, implementing, and comparing theories with incompatible languages and/or semantics. Relational dual tableaux are powerful tools which perform not only verification of validity (i.e., verification of truth of the statements in all the models of a theory) but often they can also be used for proving entailment (i.e., verification that truth of a finite number of statements implies truth of some other statement), model checking (i.e., verification of truth of a statement in a particular fixed model), and finite satisfaction (i.e., verification that a statement is satisfied by some fixed objects of a finite model).

This presentation is an introductory overview on specific methodological principles of constructing relational dual tableaux and their applications to non-classical logics. In particular, we will present relational dual tableaux for standard non-classical logics (modal, intuitionistic, relevant, many-valued) and for various applied theories of computational logic (fuzzy-set-based reasoning, qualitative reasoning, reasoning about programs, among others). Furthermore, we will discuss decision procedures in dual tableaux style. By way of example, we will show how to modify the classical dual tableau systems to obtain decision procedures for some modal and intuitionistic logics.

### References

[GHM13] J. Golińska-Pilarek, T. Huuskonen, E. Munoz-Velasco, *Relational dual tableau decision procedures and their applications to modal and intuitionistic logics*, forthcoming in Annals of Pure and Applied Logics, doi: 10.1016/j.apal.2013.06.003

[ORG11] E. Orłowska, J. Golińska-Pilarek, *Dual Tableaux: Foundations, Methodology, Case Studies*, Springer, 2011.

[ORL88] E. Orłowska, *Relational interpretation of modal logics*, in: H. Andreka, D. Monk, I. Nemeti (eds.), Algebraic Logic, Colloquia Mathematica Societatis Janos Bolyai 54, North Holland, Amsterdam, 1988, 443-471.

[RAS60] H. Rasiowa, R. Sikorski, *On Gentzen theorem*, Fundamenta Mathematicae 48 (1960), 57-69.

[TAR41] A. Tarski, *On the calculus of relations*, Journal of Symbolic Logic 6 (1941), 73- 89.

# Negation as a Resource: a novel view on Answer Set Semantics*

Stefania Costantini[1] and Andrea Formisano[2]

[1] DISIM, Università di L'Aquila, Italy stefania.costantini@univaq.it
[2] DMI, Università di Perugia, Italy formis@dmi.unipg.it

**Abstract.** In recent work, we provided a formulation of ASP programs in terms of linear logic theories. Answer sets were characterized in terms of maximal tensor conjunctions provable from such theories. In this paper, we propose a full comparison between Answer Set Semantics and its variation obtained by interpreting literals (including negative literals) as resources, which leads to a different interpretation of negation. We argue that this novel view can be of both theoretical and practical interest, and we propose a modified Answer Set Semantics that we call Resource-based Answer Set Semantics. One advantage is that of avoiding inconsistencies, so every program has a (possibly empty) resource-based answer set. This implies however the introduction of a different way of representing constraints.

**Keywords:** Answer Set Programming, Linear Logic, Default Negation

## 1 Introduction

In [1], we proposed a comparison between RASP and linear logic [2], where RASP [3, 4, 5] is a recent extension of the Answer Set Programming (ASP) framework obtained by explicitly introducing the notion of *resource*. As it is well-known, ASP is nowadays a well-established programming paradigm, with applications in many areas (see among many [6, 7, 8] and the references therein). RASP is a significant extension, supporting both formalization and quantitative reasoning on consumption and production of amounts of resources.

We proved in particular that RASP (and thus, ASP as a particular case) corresponds to a fragment of linear logic. This was done by providing a two-ways translation of RASP programs into a linear logic theory. The result implies that a RASP inference engine (such as Raspberry [5]) can be used for reasoning in this fragment. In defining the correspondence, we introduced a RASP and linear-logic modeling of default negation as understood under the answer set semantics. We meant in some sense to propose "yet another definition of answer set", in addition to those reported in [9].

In the present paper, we show that understanding default negation as a resource goes beyond, and leads to the definition of a generalization of the answers set semantics (for short AS, on which ASP is based), with some potential advantages. We provide a

* A short version of this paper will appear in the Proc. of LPNMR 2013. This research is partially supported by GNCS-13 project.

model-theoretic definition of the new semantics, that we call *Resource-based Answer Set Semantics*. In the new setting, there are no inconsistent programs, and basic odd cycles (similarly to basic even cycles in AS) are interpreted as exclusive disjunctions. Constraints must then be represented explicitly (while in ASP they are "simulated" via unary odd cycles). Therefore, what was before programs with constraints becomes a plain ASP program (under the extended semantics) augmented with a set of explicitly represented constraints. We argue that representing constraints separately can lead to more generality and to an improved elaboration-tolerance (in the sense of [10]). In the proposed approach, the "practical expressivity" in terms of knowledge representation is improved (as we demonstrate by means of significant examples), though unfortunately also the computational complexity increases.

The paper is organized as follows. In Sections 2 and 3, we provide the necessary background on linear logic and ASP. In Section 4, we specialize the method defined in [1] for RASP, so as to show that ASP can be defined as a fragment of linear logic. It is relevant to recall this formalization, because it makes it clear which is the motivation of the new notion of negation and of the generalized answer set semantics that we then propose. In Sections 5 and 6 the semantic extension is described, formalized and discussed. Finally, in Section 7 we conclude.

## 2 Background on Linear Logic

Linear logic [2] can be considered as a *resource sensitive* refinement of classical logic, since it intrinsically supports a natural accounting of resources. Intuitively speaking, in linear logic, two assumptions of a formula $P$ are distinguished from a single assumption of it. Below we briefly review the basic traits of (a fragment of) linear logic, by recalling only the notions that will be used in the remaining part of the paper. For a comprehensive treatment we refer the reader to [11] and to the references therein.

In linear logic, contraction and weakening rules are not allowed: hence, while a statement of the form $P \rightarrow P \wedge P$ is valid in classical logic, this is not the case in linear logic. The point here can be explained by observing that in classical logic statements are assumed to express "static" properties, unchanging facts about the world. On the contrary, linear propositions are concerned with dynamic properties of finite amounts of resources (and the processes that use them). An example well-known in the literature [11, 12] may further clarify this point. Consider the following propositions/resources:

$P$ : "*One dollar*"
$Q$ : "*One pack of Camel*"
$R$ : "*One pack of Marlboro*"

and the following axiomatization of a vending machine:

$$P \rightarrow Q$$
$$P \rightarrow R$$

In classical logic, one can derive that $P \rightarrow Q \wedge R$, but this makes little sense if we are assuming the mentioned interpretation of propositions as resources (and of implications as transformation processes, very much like in RASP).

One of the crucial features of linear logic is that it makes a neat distinction between two forms of conjunction that are not distinguished by classical logic. Namely, one of them intuitively means "I have both". This is said *multiplicative conjunction* and is written as $\otimes$. The other, the *additive conjunction* means "I have a choice" (and is written as &). Dually, there are two disjunctions. The multiplicative one, written $P \parr Q$ can be read as "if not $P$, then $Q$", and the additive disjunction $P \oplus Q$, that intuitively stands for the possibility of either $P$ or $Q$, but we do not know which of the two. That is, it involves "someone else's choice".

Finally, we have linear implication $P \multimap Q$. It encodes a form of production process: it can be read as "$Q$ can be derived using $P$ exactly once". (Notice that, in such a process $P$ is "consumed", so it cannot be used again.)

Linear negation $^\perp$ is the only negative operation in the logic. It is involutive (namely, $(P^\perp)^\perp$ and $P$ can be safely identified) and, at the same time, it retains a constructive character. Notice that it acts as a sort of *transposition*: $P \multimap Q$ coincides with $Q^\perp \multimap P^\perp$. Moreover, the linear implication $P \multimap Q$ can be rewritten as $P^\perp \parr Q$.

In order to re-gain the full power of classical logic *exponential* operators, namely ! and its dual ?, are introduced. Intuitively, $!P$ means that we have how many $P$ we want. These connectives reintroduce, in a more controllable way, contraction and weakening in the logical framework.

To better illustrate all these connectives, let us recall another example (taken from [12]). Suppose that for a fixed price of 5 Dollars a restaurant will provide a hamburger, a Coke, as many french fries as you like, onion soup or salad (your choice), and pie or ice cream (depending on availability, hence by someone else's choice). This is the menu:

For a fixed-Price Menu: 5 Dollars (D) you can have:
Hamburger (H)
Coke (C)
All the french fries (F) you can eat
One between Onion-Soup (O) or Salad (S)
Pie (P) or Ice-Cream (I) depending on availability

and its encoding in a linear logic formula:

$$(D \otimes D \otimes D \otimes D \otimes D) \multimap \big(H \otimes C \otimes !F \otimes (O \,\&\, S) \otimes (P \oplus I)\big)$$

Some further notions will be used in what follows. Let $X$s and $Y$s denote *tensor products* of positive literals, e.g. formulas of the form $(P_1 \otimes \cdots \otimes P_n)$ (for $n > 0$). Then, *generalized Horn implications* are defined as follows:

- an Horn implication has the form: $X \multimap Y$.
- An $\oplus$-Horn implication has the form: $\big(X_1 \multimap (Y_1 \oplus Y_2)\big)$.
- An &-Horn implication has the form: $\big((X_1 \multimap Y_1) \,\&\, (X_2 \multimap Y_2)\big)$.

Notice that a formula of the last form, say $(P_1 \multimap Q_1) \,\&\, (P_2 \multimap Q_2)$, encodes a nondeterministical process where a choice is made between the two disjuncts (say $P_2 \multimap Q_2$) and then the (sub-)process encoded by the selected option is executed (in our case $Q_2$ is produced using $P_2$).

A formal proof system for linear logic can be formulated in terms of a Gentzen-style sequent calculus. A *sequent* is composed of two sequences of formulas separated by a turnstile ($\vdash$) symbol. The sequent $\Delta \vdash \Gamma$ asserts that the multiplicative conjunction of the formulas in $\Delta$ together imply the multiplicative disjunction of the formulas in $\Gamma$. In general, a *sequent calculus proof rule* consists of a set of hypothesis sequents and a single conclusion sequent. A full set of Gentzen-style sequent rules for linear logic can be found, for instance, in [13].

## 3 Background on Answer Set Semantics

In the answer set semantics (originally named "stable model semantics"), a (logic) program $\Pi$ (cf., [14, 15]) is a collection of *rules* of the form $H \leftarrow L_1, \ldots, L_n.$ where $H$ is an atom, $n \geqslant 0$ and each literal $L_i$ is either an atom $A_i$ or its *default negation not* $A_i$. The left-hand side and the right-hand side of rules are called *head* and *body*, respectively. A rule can be rephrased as $H \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n.$ where $A_1, \ldots, A_m$ can be called *positive body* and *not* $A_{m+1}, \ldots, not\ A_n$ can be called *negative body*. A rule with empty body ($n = 0$ is called a *fact*. A rule with empty head is a *constraint*, where a constraint is of the form $\leftarrow L_1, \ldots, L_n.$ and states that literals $L_1, \ldots, L_n$ cannot be simultaneously true.

Various extensions to the basic paradigm exist, that we do not consider here as they are not essential in the present context. We do not even consider "classical negation" (cf., [15]).

In the rest of the paper, whenever it is clear from the context, by "a (logic) program $\Pi$" we mean an answer set program (ASP program) $\Pi$, and we will implicitly refer to the "ground" version of $\Pi$. The ground version of $\Pi$ is obtained by replacing in all possible ways the variables occurring in $\Pi$ with the constants occurring in $\Pi$ itself, and is thus composed of ground atoms, i.e., atoms which contain no variables. By "minimal model of $\Pi$" we mean a minimal model of $\Pi$ intended as a classical logic theory, where $\leftarrow$ is intended as implication and *not* as negation in classical logic terms.

The answer sets semantics [14, 15] is a view of a logic program as a set of inference rules (more precisely, default inference rules), or, equivalently, a set of constraints on the solution of a problem: each answer set represents a solution compatible with the constraints expressed by the program. Consider simple program $\{q \leftarrow not\ p.\quad p \leftarrow not\ q.\}$. For instance, the first rule is read as "assuming that $p$ is false, we can *conclude* that $q$ is true." This program has two answer sets. In the first one, $q$ is true while $p$ is false; in the second one, $p$ is true while $q$ is false.

Unlike other semantics, a program may have several answer sets, or may have no answer set. Whenever a program has no answer sets, we will say that the program is *inconsistent*. Correspondingly, checking for consistency means checking for the existence of answer sets. The following program has no answer set: $\{a \leftarrow not\ b.\ b \leftarrow not\ c.\ c \leftarrow not\ a.\}$. The reason is that in every minimal model of this program there is a true atom that depends (in the program) on the negation of another true atom, which is strictly forbidden in this semantics, where every answer set can be considered as a self-consistent and self-supporting set of consequences of given program. The program $\{p \leftarrow not\ p.\}$ has no answer sets either as it is contradictory. Constraints of the form defined above can

be simulated by plain rules of the form $p \leftarrow not\ p, L_1, \ldots, L_n.$ where $p$ is a fresh atom. Thus, consistency is related (as discussed at length in [16, 17]) to the occurrence of "odd cycles" (of which $p \leftarrow not\ p$ is the basic case, though odd cycles may involve any odd number of atoms) and how they are connected to other parts of the program. The reason is that, in principle, the negation *not A* of atom $A$ is an assumption, that must be dropped whenever $A$ can be proved, as answer sets are by definition non-contradictory.

Below is the specification of the Answer Set Semantics, reported from [14].

**Definition 1 (The Gelfond-Lifschitz Operator).** *Let $I$ be a set of atoms and $\Pi$ a program. A GL-transformation of $\Pi$ modulo $I$ is a new program $\Pi/I$ obtained from $\Pi$ by performing the following two reductions:*

1. *removing from $\Pi$ all rules which contain a negative premise not A such that $A \in I$;*
2. *removing from the remaining rules those negative premises not A such that $A \notin I$.*

*$\Pi/I$ is a positive logic program, with Least Herbrand Model*[1] *$J$. Let $\Gamma_\Pi(I) = J$.*

Answer sets are defined as follows.

**Definition 2.** *Let $I$ be a set of atoms and $\Pi$ a program. $I$ is an answer set of $\Pi$ iff $\Gamma_\Pi(I) = I$.*

It will be useful in what follows to report from [16] a simple property of $\Gamma_\Pi$.

**Proposition 1.** *Let $M$ be a minimal model*[2] *of $\Pi$. Then, $\Gamma_\Pi(M) \subseteq M$.*

Answer sets are in fact minimal supported models, and non-empty answer sets form an anti-chain with respect to set inclusion.

In the ASP (Answer Set Programming) paradigm, each answer set is seen as a solution of given problem, encoded as an ASP program. To find these solutions, an ASP-solver is used. Several solvers have became available, see [19], each of them being characterized by its own prominent valuable features. The expressive power of ASP, as well as, its computational complexity have been deeply investigated (cf. e.g., [20]).

## 4 ASP and Linear Logic

In this section, we specialize the method defined in [1] for RASP, so as to show that ASP can be defined as a fragment of linear logic. In particular, we define a translation of ASP programs into a linear logic theory employing as connectives tensor product $\otimes$ (to express concomitant use/production of different resources), linear implication $\multimap$ (to model production processes), and additive conjunction & (to represent alternative/exclusive resource allocation). In well-known terminology, we adopt formulas belonging to the so-called Horn-fragment of linear logic. In [1] we treat the more general case of RASP, which manages resource production and consumption.

[1] Cf. [18] for the definition of Least Herbrand Model of a Horn logic program, due to Van Emden and Kowalski.

[2] The property holds for models in general, but minimal ones are those of interest here.

A positive ASP program $\Pi$ (i.e., a program without default negation) can be transformed into a corresponding Linear Logic RASP Theory as follows (notice that the reverse translation is also possible, i.e., to transform a Linear Logic RASP Theory into a (R)ASP program). In particular, in the definition below each atom $q$ in the body of the j-th rule of given program is renamed as $q^j$, where the $q^j$'s are called the standardized-apart versions of $q$. Moreover, since the formalization passes through RASP, which considers atoms as resources, each standardized-apart atom $q^j$ will stand for $q^j{:}1$ (In RASP terminology, a writing of the form $q{:}a$ denotes an *amount* $a$ of the *resource* $q$.). The meaning is that, when using the body of a rule to derive the head, one uses one unit of each atom (seen as a resource) in the body[3]. Notice that, in $\Pi$, the truth of an atom might be used to prove several consequences (through different rules). As we mentioned before, linear logic provides the exponential connective $!A$, intuitively meaning that we can use as many occurrences of $A$ as we want. However, exploiting this connective would bring us outside the finite propositional fragment of linear logic at hand. The devised method remains within the propositional fragment.

**Definition 3.** *Given a positive ASP program $\Pi$, the corresponding Linear Logic RASP Theory $\Sigma_\Pi$ is obtained by applying, in sequence, the following rewritings.*

- *Standardize apart the atoms in the bodies of rules of $\Pi$. Namely, each occurrence of an atom $A$ in the body of the $j$-th rule is replaced by $A^j{:}1$.*
- *For every atom $A$ occurring as head of $h > 0$ rules in (the standardize apart version of) $\Pi$, let $A \leftarrow B_{i,1},\ldots,B_{i,\ell_i}$, for $i = 1,\ldots,h$, be such rules (with $\ell_i$ possibly null, if the corresponding rule is a fact). Replace these rules by the following linear implications (where the $A_i$s are fresh atoms):*

$$\begin{array}{l} B_{i,1} \otimes \ldots \otimes B_{i,\ell_i} \multimap A_i \quad \textit{for } i = 1,\ldots,h \\ A_1 \,\&\, \ldots \,\&\, A_h \multimap A \end{array}$$

- *For each atom $A$, let $A^1{:}1, \ldots, A^m{:}1$ be its standardized apart versions, introduced as described earlier. Add to $\Sigma_\Pi$ the linear implication $A \multimap A^1{:}1 \otimes \ldots \otimes A^m{:}1$.*
- *Replace in $\Sigma_\Pi$ any linear implication $B_1 \otimes \ldots \otimes B_n \multimap H$ with the implication $B_1 \otimes \ldots \otimes B_n \multimap H \otimes B_{1R} \otimes \ldots \otimes B_{nR}$.*

Let us remark some aspects of the previous definition. Notice that through the second step of the translation, the body of each rule in $\Pi$, which is a conjunction of atoms, is turned into a tensor conjunction of atoms. The purpose of the linear implication $A_1 \,\&\, \ldots \,\&\, A_h \multimap A$ is that of enabling the derivation of $A$ by either of the (translations of the) rules defining it. Clearly, the introduction of such an implication can be avoided in case $A$ occurs as head of a single rule (in this case $h = 1$ and we can simply replace $A_1$ by $A$ in the first linear implication). In what follows we will adhere to this convention whenever possible.

The linear implication $A \multimap A^1{:}1 \otimes \ldots \otimes A^m{:}1$ can be seen as an &-Horn implications with a unique conjunct. It models the fact that $A$ is a resource available to any rule that may need to use it.

[3] RASP allows for arbitrary quantities, not needed here.

Notice, moreover, the introduction of a fresh atom $B_{iR}$s corresponding to each atom $B_i$, in the last step of the translation. These fresh atoms are called the *r-copies* of the $B_i$s. They are produced just in order to keep a *record* of those resources that have been consumed. R-copies allow us to establish a correspondence between answer sets of $\Pi$ and maximal tensor conjunctions provable from $\Sigma_\Pi$, where:

**Definition 4.** *Given linear logic theory $\Sigma$, a tensor conjunction of atoms $A_1 \otimes \ldots \otimes A_n$ ($n \geq 0$), is called* maximally provable *if it is provable from $\Sigma$, and for any atom $B$, the tensor conjunction $A_1 \otimes \ldots \otimes A_n \otimes B$ is not provable from $\Sigma$ (we equivalently talk about a maximal tensor conjunction provable from $\Sigma$).*

**Lemma 1.** *Let $\Pi$ be a positive ASP program $\Pi$, and $\Sigma_\Pi$ be the corresponding* Linear Logic RASP Theory. *Every maximal tensor conjunction $\mathscr{A}$ provable from $\Sigma_\Pi$ includes all the r-copies of facts of $\Sigma_\Pi$ and of standardized-apart atoms occurring in the body of linear implications of $\Sigma_\Pi$ that have been used for proving atoms in $\mathscr{A}$.*

As mentioned, the role of r-copies is to keep records of facts (intended as resources originally present in the program) and of intermediate conclusions used (as resources) in further inference. In a linear-logic setting in fact, resources which are consumed "disappear", thus we would not be able to establish a relation between provable tensor conjunctions and answer sets. Now in fact, we are able to state (neglecting, by abuse of notation, the syntactic distinction between an atom $A$ and its r-copy $A_R$):

**Theorem 1.** *Let $\Pi$ be a positive ASP program $\Pi$, and $\Sigma_\Pi$ be the corresponding* Linear Logic RASP Theory. *$A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from $\Sigma_\Pi$ if and only if $\{A_1, \ldots, A_n\}$ is an answer set for $\Pi$.*

Let us now consider full ASP, where rule bodies involve negative literals. Assume there are $n$ occurrences of *not A* in the body of rules of given program $\Pi$. To represent full RASP (and thus full ASP) we improve the transformation devised in Definition 3:

**Definition 5.** *Given ASP program $\Pi$, the corresponding Linear Logic RASP Theory $\Sigma_\Pi$ is obtained by applying, in sequence, the following rewritings.*

- *For each atom A occurring negated in rules of $\Pi$, standardize apart each of its negated occurrences by replacing not A with not $A^j$:1, in the $j$-th rule.*
  *Being not $A^{j_1}$:1, …, not $A^{j_n}$:1 all the occurrences introduced in this manner, add the (linear) fact not A:n to the translation of $\Pi$.*
- *For each rule $A \leftarrow B_1, \ldots, B_\ell$ of $\Pi$. Let such rule be the $j$-th one; rewrite it as $A \leftarrow B_1, \ldots, B_\ell, not\ A^j{:}n$.*
  *Let us denote by not $A^{k_1}{:}n, \ldots, not\ A^{k_s}{:}n$ all the atoms introduced in this manner.*[4]
- *Apply the rewriting indicated in Definition 3 to the result of the previous steps.*
- *Finally, for each linear fact not A:n added to $\Sigma_\Pi$ (cf., the first two steps), also add the &-Horn implications to the translation of $\Pi$:*

$$(not\ A{:}n \multimap not\ A^{k_1}{:}n)\,\&\,\ldots\,\&\,(not\ A{:}n \multimap not\ A^{k_s}{:}n)\,\&$$
$$(not\ A{:}n \multimap not\ A^{j_1}{:}1 \otimes \ldots \otimes not\ A^{j_n}{:}1)$$

[4] In case identical atoms would be introduced in the body in consequence of different steps of the translation, e.g., *not* $A^k$:1 and *not* $A^k$:1 might occur in the same rule if $n$ equals 1 in the first step and *not A* already appeared in the ASP rule body, then further standardize apart these occurrences, e.g., as *not* $A^{k1}$:1 and *not* $A^{k2}$:1.

The intuitive meaning behind this translation is that the assumption *not A* is made available to every rule that intends to adopt it, unless $A$ itself is provable. In which case the assumption becomes totally unavailable (as proving $A$ consumes the full available quantity of the "resource" *not A*).

The transformation of Definitions 3 and 5 is clearly polynomial, as we add: (i) a new conjunct in the body (*not A* if the rule head is $A$) and new elements (r-copies) in the head of rules ; (ii) one &-Horn implication for each $A$ occurring in the head of some rule; (iii) one linear implication for each atom defined via several rules; (iv) one &-Horn implication for each $A$ occurring negatively in the body of some rule. Hence, we have:

**Theorem 2.** *Let $\Pi$ be an ASP program, and $\Sigma_\Pi$ the corresponding Linear Logic RASP Theory, obtained according to Definitions 3 and 5. Let $M = \{A_1, \ldots, A_n\}$ be an answer set for $\Pi$. Then, $A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from $\Sigma_\Pi$.*

Note that the reverse result does not necessarily hold, because there are maximal tensor conjunctions that are not answer sets but are provable from $\Sigma_\Pi$. This is due (as discussed in [1]) to the lack of relevance of the answer set semantics (cf., [21]), but also to the locality of a proof-based system such as linear logic.

## 5 Negation as a Resource: a novel view on Answer Set Semantics

It is interesting to notice that the linear logic formulation we summarized in the previous section prevents contradictions. Consider for example the program $\Pi_1 = \{p \leftarrow not\ p.\}$. It is transformed into:

$$
\begin{array}{l}
not\ p^{11}{:}1 \otimes not\ p^{12}{:}1 \ \multimap p, \\
not\ p{:}1, \\
(not\ p{:}1 \ \multimap not\ p^{11}{:}1) \,\&\, (not\ p{:}1 \ \multimap not\ p^{12}{:}1)
\end{array}
$$

In the first rule, one occurrence of *not p* corresponds to the one originally present, the other one has been added as for proving $p$ it is necessary to "absorb" the whole available quantity of *not p* (consider $n = 1$ in Definition 5). We can in fact verify that the singleton tensor conjunction $p$ is by no means provable: in fact, it would require two units of *not p*, while just one is available. This does not lead to inconsistency, but simply to the impossibility to prove $p$.

Consider again program $\Pi = \{a \leftarrow not\ b.\ b \leftarrow not\ c.\ c \leftarrow not\ a.\}$ which is an "odd cycle" involving three atoms. In our formulation, $\Sigma_\Pi$ is the following:

$$
\begin{array}{l}
not\ a^1{:}1 \otimes not\ b^1{:}1 \ \multimap a \\
not\ c^2{:}1 \otimes not\ b^2{:}1 \ \multimap b \\
not\ a^3{:}1 \otimes not\ c^3{:}1 \ \multimap c \\
not\ a{:}1 \\
not\ b{:}1 \\
not\ c{:}1 \\
(not\ a{:}1 \ \multimap not\ a^1{:}1) \,\&\, (not\ a{:}1 \ \multimap not\ a^3{:}1) \\
(not\ b{:}1 \ \multimap not\ b^1{:}1) \,\&\, (not\ b{:}1 \ \multimap not\ b^2{:}1) \\
(not\ c{:}1 \ \multimap not\ c^2{:}1) \,\&\, (not\ c{:}1 \ \multimap not\ c^3{:}1)
\end{array}
$$

From this linear logic theory we can prove the three maximal tensor conjunctions, namely, $a$, $b$ and $c$. Assume, in fact, to try to prove $a$ (the cases of $b$ and $c$ are of course analogous). Proving $a$ uses resources *not* $a^1$:1 and *not* $b^1$:1. Therefore, after proving $a$, $b$ cannot be proved because its own negation (i.e., *not* $b^2$:1) is not available: in fact, the &-Horn implication related to $b$ generates (indifferently) only one of the two items, and has already been requested to produce *not* $b^1$:1 for proving $a$. In turn, $c$ cannot be proved because *not* $a^3$:1 is not available, as the &-Horn implication related to $a$ generates (indifferently) only one of the two items, and has already been requested to produce *not* $a^1$:1 for proving $a$. Then, $\Sigma_\Pi$ behaves analogously to the GL-reduct as far as $c$ is concerned, being *not a* unavailable once $a$ has been proved. But it behaves in a more uniform way on $b$, in the sense that once *not b* has been used to prove $a$, it is no longer possible to prove $b$.

This means that the 3-atoms odd cycles is interpreted as an exclusive disjunction, exactly like the 2-atoms even cycle (such as $\{q \leftarrow not\ p.\ \ p \leftarrow not\ q.\}$) in AS. Therefore, in the generate-and-test perspective which is at the basis of the ASP programming methodology, our new view provides a new mean of easily generating the search space.

We call $\{a\}$, $\{b\}$, and $\{c\}$ *resource-based answer sets*, for which we provide below a logic programming characterization. The resource-based answers set for program $\{p \leftarrow not\ p.\}$ is the empty set.

The ternary cycle has many well-known interpretations in terms of knowledge representation, among which the following is an example:

$$\begin{array}{l}\{beach \leftarrow not\ mountain.\\ \ mountain \leftarrow not\ travel.\\ \ travel \leftarrow not\ beach.\}\end{array}$$

In our approach we would have exactly one of (indifferently) *beach*, *mountain*, or *travel*. Similarly for the program $\{work \leftarrow not\ tired.\ tired \leftarrow not\ sleep.\ sleep \leftarrow not\ work.\}$. Note that, in answer set programming, for defining the exclusive disjunction of three atoms one has to resort to the *extremal program* [22] $\{a \leftarrow not\ b, not\ c.\ b \leftarrow not\ c, not\ a.\ c \leftarrow not\ a, not\ b.\}$

There are other semantic approaches to managing odd cycles, such as for instance [23, 24] and [25, 26], with their own sound theoretical foundations, that can however be distinguished from the present one: in fact, the former proposals basically choose (variants of) the classical models, and the latter ones treat differently the unary and ternary cycles.

Below we provide a variation of the answer set semantics that defines resource-based answer sets.

**Definition 6.** *Let $\Pi$ be a program and $I$ a minimal model of $\Pi$. $I$ is called a $\Pi$-based minimal model iff $\forall A \in I$, there exists a rule in $\Pi$ with head $A$ and positive body $C_1, \ldots, C_m$, $m \geq 0$, where $\{C_1, \ldots, C_m\} \subseteq I$.*

**Definition 7.** *Let $\Pi$ be a program. $M$ is a resource-based answer set of $\Pi$ iff $M = \Gamma_\Pi(I)$, where $I$ is a $\Pi$-based minimal model of $\Pi$.*

By Definition 7, there is a resource-based answer set for each $\Pi$-based classical minimal model. It is clear that answer sets are among resource-based answer sets. In

fact, as stated in Section 3 (Proposition 1), for any minimal model $I$ it holds $\Gamma_\Pi(I) \subseteq I$: thus any answer set $S$, being a minimal model which is equal to $\Gamma_\Pi(S)$, fits as a particular case in the above definition. Therefore, some of the resource-based answer sets of $\Pi$ are classical models (coinciding with its answer sets), while the others are subsets of the remaining $\Pi$-based minimal models (if any). Non-empty resource-based answer sets still form an anti-chain w.r.t. set inclusion.

We call the new semantics RAS semantics (Resource-Based Answer Set semantics), w.r.t. AS (Answer Set) semantics. Differently from answer sets, a (possibly empty) resource-based answer set always exists. Complexity of RAS semantics is however higher than complexity of AS semantics: in fact, [27] proves that deciding whether a set of formulas is a minimal model of a propositional theory is co-NP-complete. Clearly, checking whether a minimal model $I$ is $\Pi$-based and computing $\Gamma_\Pi(I)$ has polynomial complexity. Then:

**Proposition 2.** *Given program $\Pi$, deciding whether a set of atom $I$ is a resource-based answer set of $\Pi$ is co-NP-complete.*

The previous result about the relation with linear logic (Theorem 2) extends to the new semantics. The proof, reported in [1] in the context of full RASP programs, remains essentially the same. The difference is that where in previous case one referred to answer sets, which implies that given program $\Pi$ was supposed to be consistent, we are now able to refer to any ASP program. Then we have:

**Theorem 3.** *Let $\Pi$ be an ASP program, and $\Sigma_\Pi$ the corresponding Linear Logic RASP Theory, obtained according to Definitions 3 and 5. If $M = \{A_1, \ldots, A_n\}$ is a resource-based answer set for $\Pi$, then $A_1 \otimes \ldots \otimes A_n$ is a maximal tensor conjunction provable from $\Sigma_\Pi$.*

It remains to be explained why the new definition models the intuition, and how it applies to practical cases. In particular, given minimal model $I$ of $\Pi$, it may be that $\Gamma_\Pi(I) \subset I$, i.e., $\Gamma_\Pi(I)$ is a proper subset of $I$ and thus $I$ is not an answer set, for only one reason. For atom $A$ to belong to a $\Pi$-based minimal model $I$, there exists some rule in $\Pi$ with head $A$. For $A$ not to belong to $\Gamma_\Pi(I)$, so that $\Gamma_\Pi(I) \subset I$, each of the rules that could cause $A$ to be in the model must have been canceled by step (1) of $\Gamma_\Pi$, as they include literal *not B* in their body, $B \in I$. Atoms belonging to $\Gamma_\Pi(I)$ are therefore those atoms in $I$ that can be derived without such contradictions. As widely discussed in [16, 17], contradictions only arise in program fragments corresponding to *unbounded odd cycles*, i.e., odd cycles where no atom is bounded to be true/false (thus resolving the contradiction) by links with other parts of the program. Starting from $\Pi$-based minimal models however, $\Gamma_\Pi(I)$ provides for these cycles the "exclusive or" interpretation that we have proposed above.

Regarding general odd cycles involving $k$ atoms, of the form $\{a_1 \leftarrow not\ a_2.\ a_2 \leftarrow not\ a_3 \ldots a_k \leftarrow not\ a_1.\}$, it is easy to see that each such cycle has $k$ classical minimal $\Pi$-based models (it admits $k$ classical minimal models, all of them $\Pi$-based as there are no positive conditions). Correspondingly, we obtain $k$ resource-based answer sets, where we have $M_i = \{a_{i+d},$ with $d$ even, $0 \leq d < k-1\}$. This fact can be verified by producing a translation into the corresponding Linear Logic RASP Theory analogous to the one performed above for unary and ternary cycles. Then, unfortunately, odd cycles

no longer model disjunction if $k > 3$, similarly to even cycles, which do not model disjunction if $k > 2$.

In resource-based answer set semantics, there are no inconsistent programs. Nevertheless, the new semantics is useful in knowledge representation not just to fix inconsistencies: rather, it depicts a more general scenario in many reasonable examples. Consider for instance the variation of the above program (inspired to examples proposed in [23, 24]):

$$\begin{array}{l} \mathit{beach} \leftarrow \mathit{not\ mountain}. \\ \mathit{mountain} \leftarrow \mathit{not\ travel}. \\ \mathit{travel} \leftarrow \mathit{not\ beach}, \mathit{passport\_ok}. \\ \mathit{passport\_ok} \leftarrow \mathit{not\ forgot\_renew}. \\ \mathit{forgot\_renew} \leftarrow \mathit{not\ passport\_ok}. \end{array}$$

This program has answer set $M_1 = \{\mathit{forgot\_renew}, \mathit{mountain}\}$, as *passport_ok* being false forces *travel* to be false, which in turn makes *mountain* true. The answer set semantics cannot cope with the case of the passport being ok, which is in fact excluded as this option determines no answer set. Instead, in resource-based answer set semantics we have, in addition to $M_1$, three other answer sets stating that, if the passport is ok, any choice is possible, namely we have $M_2 = \{\mathit{passport\_ok}, \mathit{mountain}\}$, $M_3 = \{\mathit{passport\_ok}, \mathit{beach}\}$, and $M_4 = \{\mathit{passport\_ok}, \mathit{travel}\}$. We may notice that the semantics is still a bit strong on this example on the side of the answer set, as one would say that not having *passport_ok* prevents traveling, but any other choice should be possible, while instead the *mountain* choice is forced.

A better formalization of the above example would be by means of the plain odd cycle, plus the even cycle concerning passport, plus the constraint

$$\leftarrow \mathit{not\ passport\_ok}, \mathit{travel}.$$

In the next section we will discuss how to introduce such a constraint, as a unary odd cycle is no longer usable to this purpose.

## 6 Modeling Constraints

In resource-based answer set semantics, constraints cannot be modeled in terms of odd cycles. Therefore, they have to be modeled explicitly. In particular, let assume a constraint $\mathscr{C}$ to be of the form $\leftarrow E_1, \ldots, E_n$. where the $E_i$s are atoms[5]. This is with no loss of generality, as a constraint such as, for instance, $\leftarrow A, \mathit{not}\ B.$ can be reformulated as the program fragment $\leftarrow A, B'.$ $B' \leftarrow \mathit{not}\ B.$ Thus, an overall program $\Pi_{\mathscr{C}}$ can be seen as composed of answer set program $\Pi$ plus a set $\{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$ of constraints, and, possibly, an auxiliary program $\Pi_{\mathscr{C}}$, so that constraints can be defined on atoms belonging to either $\Pi$ or $\Pi_{\mathscr{C}}$. We assume however that $\Pi_{\mathscr{C}}$ is stratified (i.e., it contains no cycles, cf. e.g., [28] for a formal definition) and that atoms of $\Pi$ may occur in $\Pi_{\mathscr{C}}$ only in the body of rules (in the terminology of [16, 29], $\Pi_{\mathscr{C}}$ is a *top program* of $\Pi$).

[5] This limitation will be useful for the linear logic formulation (provided below).

Consider for instance $\Pi_{\mathscr{O}}$ to be composed of the following $\Pi$:

$$\{beach \leftarrow not\ mountain.\\ mountain \leftarrow not\ travel.\\ travel \leftarrow not\ beach.\\ hyperthyroidism.\}$$

plus the following $\Pi_{\mathscr{C}}$:

$$\{unhealthy \leftarrow beach, hyperthyroidism.\}$$

plus the constraint $\leftarrow unhealthy$.

The resulting theory will have resource-based answer sets $\{mountain, hyperthyroidism\}$, and $\{travel, hyperthyroidism\}$, while $\{beach, hyperthyroidism, unhealthy\}$ is excluded by the constraint. We now proceed to the formal definition.

**Definition 8.** *An Answer Set Theory $\mathscr{T}$ is a couple $\langle \Pi_{\mathscr{O}}, Constr \rangle$, with $\Pi_{\mathscr{O}} = \Pi \cup \Pi_{\mathscr{C}}$, where $\Pi_{\mathscr{C}}$ is a top program for $\Pi$, and where Constr is a set $\{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$, $v \geq 0$, of constraints.*

**Definition 9.** *Given Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$, a resource-based Answer Set $M$ for $\Pi$ fulfills the constraints in Constr iff the answer set program $\Pi'$ is consistent (in the sense of traditional answer set semantics), where $\Pi'$ is obtained from $\Pi_{\mathscr{C}}$ by adding all atoms in $M$ as facts, and all constraints in Constr as rules.*

**Definition 10.** *A* Resource-based Answer Set *$M$ of Answer Set Theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$ is a resource-based answer set for $\Pi$ that fulfills all constraints in Constr.*

It is easy to see that, in order to check that resource-based Answer Set $M$ for $\Pi$ fulfills the constraints, one can check consistency of $\Pi'$ in a simple way, by: (i) computing (in polynomial time, cf., e.g., [20]) the unique answer set $M''$ of the stratified program $\Pi''$ obtained from $\Pi_{\mathscr{C}}$ by adding all atoms in $M$ as facts, and then (ii) checking constraints on $M''$ by pattern-matching. Then, for constraints of the above simple form, we can conclude that:

**Proposition 3.** *Given Answer Set Theory $\mathscr{T}$, deciding about the existence of a resource-based answer set is a co-NP-complete problem.*

The partition of $\Pi_{\mathscr{O}}$ into $\Pi$ and $\Pi_{\mathscr{C}}$ is not strictly necessary in the present context. In fact, one might simply check the constraints on $\Pi \cup \Pi_{\mathscr{C}}$. However, we choose to introduce the distinction because we believe that it may have a significance in terms of knowledge representation and elaboration-tolerance, in the sense of [10]. In fact, the same "generate" part ($\Pi$) can be customized by adding on top, as an independent layer, different "test" parts ($\Pi_{\mathscr{C}}$). Moreover, constraints might be generalized with respect to the simple form proposed above, for instance drawing inspiration from the discussion in [30, 31, 32], or also following the approach of *Answer Set Optimization* (cf. [33] and the references therein), which proposes constraints expressing complex preferences for choosing among answer sets.

For the sake of completeness, it may be interesting to illustrate the linear logic formalization of the full approach. To this aim, we have to resort to linear logic negation. A constraint $\mathscr{C} = \leftarrow E_1, \ldots, E_n.$ can in fact be represented in linear logic as $\mathscr{C}^L = {E_1}^{\perp} \parr \ldots \parr {E_n}^{\perp}$ where $\parr$ is the multiplicative disjunction, and $^{\perp}$ is linear logic negation, $A^{\perp}$ meaning "there is no proof for $A$".[6]

Thus, the overall linear logic theory would be $\Sigma_{\Pi_{\mathscr{O}}}$, and its resource-based answer sets should be matched against the constraints. Formally:

**Definition 11.** *Given resource-based answer set $M = \{A_1, \ldots, A_n\}$ for $\Pi_{\mathscr{O}}$, $M$ is a resource-answer set for answer set theory $\mathscr{T} = \langle \Pi_{\mathscr{O}}, Constr \rangle$ where $Constr = \{\mathscr{C}_1, \ldots, \mathscr{C}_v\}$ iff tensor conjunction $A_1 \otimes \ldots \otimes A_n \otimes \mathscr{C}_1^L \otimes \ldots \otimes \mathscr{C}_v^L$ is provable from $\Sigma_{\Pi_{\mathscr{O}}}$.*

Notice that each constraint is provable whenever at least one of its disjuncts is not one of the $A_i$'s. Then, in terms of equivalence between the logic programming and linear logic formulation, nothing really changes w.r.t. Theorem 3.

## 7 Concluding Remarks

In this paper, we have proposed an extension of the answer set semantics where ternary odd cycles are understood as exclusive disjunctions, similarly to binary even cycles. This extension stems from the interpretation of an answer set program as a linear logic theory, where default negation is considered to be a resource. The practical advantage is that there is more freedom in defining a search space, where constraints must however be defined in a separate "module" to be added to given answer set program.

Concerning implementation, which is of course a main future issue for this research, answer set solvers based on SAT appear to be good candidates for extension to the new setting. In fact, apart from checking for minimality of models (which is the part responsible for the additional complexity), they do not seem to need substantial modifications in order to cope with the new semantics, that thus might in principle be easily and quickly implemented.

## References

[1] S. Costantini and A. Formisano, "RASP and ASP as a fragment of linear logic," *Journal of Applied Non-Classical Logics (JANCL)*, vol. 23, no. 1-2, pp. 49–74, 2013.

[2] J.-Y. Girard, "Linear logic," *Theoretical Computer Science*, vol. 50, pp. 1–102, 1987.

[3] S. Costantini and A. Formisano, "Answer set programming with resources," *Journal of Logic and Computation*, vol. 20, no. 2, pp. 533–571, 2010.

[4] S. Costantini and A. Formisano, "Modeling preferences and conditional preferences on resource consumption and production in ASP," *Journal of Algorithms in Cognition, Informatics and Logic*, vol. 64, no. 1, pp. 3–15, 2009.

[5] S. Costantini, A. Formisano, and D. Petturiti, "Extending and implementing RASP," *Fundamenta Informaticae*, vol. 105, no. 1-2, pp. 1–33, 2010.

[6] In fact, linear logic negation as such was used in [34] to model negation-as-failure in Prolog.

[6] C. Baral, *Knowledge representation, reasoning and declarative problem solving*. Cambridge University Press, 2003.
[7] M. Truszczyński, "Logic programming for knowledge representation," in *Logic Programming, 23rd Intl. Conference, ICLP 2007* (V. Dahl and I. Niemelä, eds.), pp. 76–88, 2007.
[8] M. Gelfond, "Answer sets," in *Handbook of Knowledge Representation. Chapter 7*, Elsevier, 2007.
[9] V. Lifschitz, "Twelve definitions of a stable model," in *Proc. of the 24th Intl. Conference on Logic Programming* (M. Garcia de la Banda and E. Pontelli, eds.), vol. 5366 of *LNCS*, pp. 37–51, Springer, 2008.
[10] J. McCarthy, "Elaboration tolerance," in *Proc. of Common Sense'98*, 1998. Available at http://www-formal.stanford.edu/jmc/ elaboration.html.
[11] J.-Y. Girard, "Linear logic: Its syntax and semantics," in *Advances in Linear Logic* (J.-Y. Girard, Y. Lafont, and L. Regnier, eds.), Proc. of the 1993 Workshop on Linear Logic, pp. 1–42, Cambridge Univ. Press, 1995.
[12] P. Lincoln, "Linear logic," *ACM SIGACT News*, vol. 23, no. 2, pp. 29–37, 1992.
[13] M. I. Kanovich, "The complexity of horn fragments of linear logic," *Ann. Pure Appl. Logic*, vol. 69, no. 2-3, pp. 195–241, 1994.
[14] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proc. of the 5th Intl. Conference and Symposium on Logic Programming* (R. Kowalski and K. Bowen, eds.), pp. 1070–1080, The MIT Press, 1988.
[15] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, pp. 365–385, 1991.
[16] S. Costantini, "Contributions to the stable model semantics of logic programs with negation," *Theoretical Computer Science*, vol. 149, no. 2, pp. 231–255, 1995.
[17] S. Costantini, "On the existence of stable models of non-stratified logic programs," *Theory and Practice of Logic Programming*, vol. 6, no. 1-2, 2006.
[18] J. W. Lloyd, *Foundations of Logic Programming*. Springer-Verlag, 1987.
[19] Web-references, "Some ASP solvers." Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.
[20] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Computing Surveys*, vol. 33, no. 3, pp. 374–425, 2001.
[21] J. Dix, "A classification theory of semantics of normal logic programs I-II.," *Fundam. Inform.*, vol. 22, no. 3, pp. 227–255 and 257–288, 1995.
[22] P. Cholewinski and M. Truszczynski, "Extremal problems in logic programming and stable model computation," in *JICSLP*, pp. 408–422, 1996.
[23] L. M. Pereira and A. M. Pinto, "Revised stable models - a semantics for logic programs," in *Progress in Artificial Intelligence, Proc. of EPIA 2005* (C. Bento, A. Cardoso, and G. Dias, eds.), vol. 3808 of *LNCS*, Springer, 2005.
[24] L. M. Pereira and A. M. Pinto, "Tight semantics for logic programs," in *Tech. Comm. of the 26th Intl. Conference on Logic Programming, ICLP 2010* (M. V. Hermenegildo and T. Schaub, eds.), vol. 7 of *LIPIcs*, pp. 134–143, 2010.
[25] M. Osorio and A. López, "Expressing the stable semantics in terms of the pstable semantics," in *Proc. of the LoLaCOM06 Workshop*, vol. 220 of *CEUR Workshop Proc.* , CEUR-WS.org, 2006.
[26] M. Osorio, J. A. N. Pérez, J. R. A. Ramírez, and V. B. Macías, "Logics with common weak completions," *J. Log. Comput.*, vol. 16, no. 6, pp. 867–890, 2006.
[27] M. Cadoli, "The complexity of model checking for circumscriptive formulae," *Inf. Process. Lett.*, vol. 44, no. 3, pp. 113–118, 1992.
[28] K. R. Apt and R. N. Bol, "Logic programming and negation: A survey," *J. Log. Program.*, vol. 19/20, pp. 9–71, 1994.

[29] V. Lifschitz and H. Turner, "Splitting a logic program," in *Proc. of ICLP'94, Intl. Conference on Logic Programming*, pp. 23–37, 1994.

[30] S. Costantini and A. Formisano, "Weight constraints with preferences in ASP," in *Proc. of Intl. Conf. on Logic Programming and Nonmonotonic Reasoning LPNMR'11*, vol. 6645 of *LNCS*, Springer-Verlag, 2011.

[31] S. Costantini and A. Formisano, "Nested weight constraints in ASP," *Fundamenta Informaticae*, vol. 124, no. 4, pp. 449–464, 2013.

[32] S. Costantini and A. Formisano, "Augmenting weight constraints with complex preferences," in *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, AAAI Press, 2011. Technical report SS-11-06.

[33] G. Brewka, I. Niemelä, and M. Truszczynski, "Answer set optimization," in *IJCAI-03, Proc. of the Eighteenth Intl. Joint Conference on Artificial Intelligence* (G. Gottlob and T. Walsh, eds.), pp. 867–872, Morgan Kaufmann, 2003.

[34] S. Cerrito, "A linear axiomatization of negation as failure," *Journal of Logic Programming*, vol. 12, no. 1&2, pp. 1–24, 1992.

# A Description Logics Tableau Reasoner in Prolog

Riccardo Zese[1], Elena Bellodi[1], Evelina Lamma[1], and Fabrizio Riguzzi[2]

[1] Dipartimento di Ingegneria – University of Ferrara
[2] Dipartimento di Matematica e Informatica – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
[riccardo.zese,elena.bellodi,evelina.lamma,fabrizio.riguzzi]@unife.it

**Abstract.** Description Logics (DLs) are gaining a widespread adoption as the popularity of the Semantic Web increases. Traditionally, reasoning algorithms for DLs have been implemented in procedural languages such as Java or C++. In this paper, we present the system TRILL for "Tableau Reasoner for descrIption Logics in proLog". TRILL answers queries to $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases using a tableau algorithm. Prolog non-determinism is used for easily handling non-deterministic expansion rules that produce more than one tableau. Moreover, given a query, TRILL is able to return instantiated explanations for the query, i.e., instantiated minimal sets of axioms that allow the entailment of the query. The Thea2 library is exploited by TRILL for parsing ontologies and for the internal Prolog representation of DL axioms.

**Keywords:** Description Logics, Tableau, Prolog, Semantic Web

## 1 Introduction

The Semantic Web aims at making information available in a form that is understandable by machines [9]. In order to realize this vision, the World Wide Web Consortium has supported the development of the Web Ontology Language (OWL), a family of knowledge representation formalisms for defining ontologies. OWL is based on Description Logics (DLs), a set of languages that are restrictions of first order logic (FOL) with decidability and for some of them low complexity. For example, the OWL DL sublanguage is based on the expressive $\mathcal{SHOIN}(\mathbf{D})$ DL while OWL 2 corresponds to the $\mathcal{SROIQ}(\mathbf{D})$ DL [9].

In order to fully support the development of the Semantic Web, efficient DL reasoners, such us Pellet, RacerPro, FaCT++ and HermiT, must be available to extract implicit information from the modeled ontologies. Most DL reasoners implement a tableau algorithm in a procedural language. However, some tableau expansion rules are non-deterministic, requiring the developers to implement a search strategy in an or-branching search space. Moreover, in some cases we want to compute all explanations for a query, thus requiring the exploration of all the non-deterministic choices done by the tableau algorithm.

In this paper, we present the system TRILL for "Tableau Reasoner for descrIption Logics in proLog", a tableau reasoner for the $\mathcal{SHOIN}(\mathbf{D})$ DL implemented in Prolog. Prolog's search strategy is exploited for taking into account

non-determinism of the tableau rules. TRILL uses the Thea2 library [27] for parsing OWL in its various dialects. Thea2 translates OWL files into a Prolog representation in which each axiom is mapped into a fact.

TRILL can check the consistency of a concept and the entailment of an axiom from an ontology and return "instantiated explanations" for queries, a non-standard reasoning service that is useful for debugging ontologies and for performing probabilistic reasoning. Instantiated explanations record, besides the axioms necessary to entail the query, also the individuals involved in the application of the axioms. This service was used in [21] for doing inference from DL knowledge bases under the probabilistic DISPONTE semantics [20].

Our ultimate aim is to use TRILL for performing probabilistic reasoning. The availability of a Prolog implementation of a DL reasoner will also facilitate the development of a probabilistic reasoner for integrations of probabilistic logic programming [23] with probabilistic DLs.

In the following, section 2 briefly introduces $\mathcal{SHOIN}(\mathbf{D})$ and its translation into predicate logic. Section 3 defines the problem we are trying to solve while Section 4 illustrates related work. Section 5 discusses the tableau algorithm used by TRILL and Section 6 describes TRILL's implementation. Section 7 shows preliminary experiments and Section 8 concludes the paper.

## 2 Description Logics

Description Logics (DLs) are knowledge representation formalisms that possess nice computational properties such as decidability and/or low complexity, see [1, 2] for excellent introductions. DLs are particularly useful for representing ontologies and have been adopted as the basis of the Semantic Web.

While DLs can be translated into FOL, they are usually represented using a syntax based on concepts and roles. A concept corresponds to a set of individuals of the domain while a role corresponds to a set of couples of individuals of the domain. We now briefly describe $\mathcal{SHOIN}(\mathbf{D})$.

Let $\mathbf{A}$, $\mathbf{R}$ and $\mathbf{I}$ be sets of *atomic concepts*, *roles* and *individuals*, respectively. A *role* is either an atomic role $R \in \mathbf{R}$ or the inverse $R^-$ of an atomic role $R \in \mathbf{R}$. We use $\mathbf{R}^-$ to denote the set of all inverses of roles in $\mathbf{R}$. An *RBox* $\mathcal{R}$ consists of a finite set of *transitivity axioms* $Trans(R)$, where $R \in \mathbf{R}$, and *role inclusion axioms* $R \sqsubseteq S$, where $R, S \in \mathbf{R} \cup \mathbf{R}^-$. *Concepts* are defined by induction as follows. Each $C \in \mathbf{A}$ is a concept, $\bot$ and $\top$ are concepts, and if $a \in \mathbf{I}$, then $\{a\}$ is a concept called *nominal*. If $C$, $C_1$ and $C_2$ are concepts and $R \in \mathbf{R} \cup \mathbf{R}^-$, then $(C_1 \sqcap C_2)$, $(C_1 \sqcup C_2)$, and $\neg C$ are concepts, as well as $\exists R.C$, $\forall R.C$, $\geq nR$ and $\leq nR$ for an integer $n \geq 0$. A *TBox* $\mathcal{T}$ is a finite set of *concept inclusion axioms* $C \sqsubseteq D$, where $C$ and $D$ are concepts. We use $C \equiv D$ to abbreviate the conjunction of $C \sqsubseteq D$ and $D \sqsubseteq C$. An *ABox* $\mathcal{A}$ is a finite set of *concept membership axioms* $a : C$, *role membership axioms* $(a, b) : R$, *equality axioms* $a = b$ and *inequality axioms* $a \neq b$, where $C$ is a concept, $R \in \mathbf{R}$ and $a, b \in \mathbf{I}$. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a TBox $\mathcal{T}$, an RBox $\mathcal{R}$ and an ABox $\mathcal{A}$. A knowledge base $\mathcal{K}$ is usually assigned a semantics in terms of

set-theoretic interpretations and models of the form $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty *domain* and $\cdot^{\mathcal{I}}$ is the *interpretation function* that assigns an element in $\Delta^{\mathcal{I}}$ to each $a \in \mathbf{I}$, a subset of $\Delta^{\mathcal{I}}$ to each $C \in \mathbf{A}$ and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each $R \in \mathbf{R}$.

The semantics of DLs can be given equivalently by converting a KB into a predicate logic theory and then using the model-theoretic semantics of the resulting theory. A translation of $\mathcal{SHOIN}$ into First-Order Logic with Counting Quantifiers is given in the following as an extension of the one given in [24]. We assume basic knowledge of logic. In predicate logic, a concept is a unary predicate symbol while a role is a binary predicate symbol. The translation uses two functions $\pi_x$ and $\pi_y$ that map concept expressions to logical formulas, where $\pi_x$ is given by

$$\begin{aligned}
\pi_x(A) &= A(x) & \pi_x(\neg C) &= \neg\pi_x(C)\\
\pi_x(\{a\}) &= (x = a) & \pi_x(C \sqcap D) &= \pi_x(C) \wedge \pi_x(D)\\
\pi_x(C \sqcup D) &= \pi_x(C) \vee \pi_x(D) & \pi_x(\exists R.C) &= \exists y.R(x,y) \wedge \pi_y(C)\\
\pi_x(\exists R^-.C) &= \exists y.R(y,x) \wedge \pi_y(C) & \pi_x(\forall R.C) &= \forall y.R(x,y) \rightarrow \pi_y(C)\\
\pi_x(\forall R^-.C) &= \forall y.R(y,x) \rightarrow \pi_y(C) & \pi_x(\geq nR) &= \exists^{\geq n} y.R(x,y)\\
\pi_x(\geq nR^-) &= \exists^{\geq n} y.R(y,x) & \pi_x(\leq nR) &= \exists^{\leq n} y.R(x,y)\\
\pi_x(\leq nR^-) &= \exists^{\leq n} y.R(y,x)
\end{aligned}$$

and $\pi_y$ is obtained from $\pi_x$ by replacing $x$ with $y$ and vice-versa. Table 1 shows the translation of each axiom of $\mathcal{SHOIN}$ knowledge bases into predicate logic.

| Axiom | Translation |
|---|---|
| $C \sqsubseteq D$ | $\forall x.\pi_x(C) \rightarrow \pi_x(D)$ |
| $R \sqsubseteq S$ | $\forall x, y.R(x,y) \rightarrow S(x,y)$ |
| $Trans(R)$ | $\forall x, y, z.R(x,y) \wedge R(y,z) \rightarrow R(x,z)$ |
| $a : C$ | $\pi_a(C)$ |
| $(a,b) : R$ | $R(a,b)$ |
| $a = b$ | $a = b$ |
| $a \neq b$ | $a \neq b$ |

**Table 1.** Translation of $\mathcal{SHOIN}$ axioms into predicate logic.

$\mathcal{SHOIN}(\mathbf{D})$ adds to $\mathcal{SHOIN}$ datatype roles, i.e., roles that map an individual to an element of a datatype such as integers, floats, etc. Then new concept definitions involving datatype roles are added that mirror those involving roles introduced above. We also assume that we have predicates over the datatypes.

A query $Q$ over a KB $\mathcal{K}$ is usually an axiom for which we want to test the entailment from the knowledge base, written $\mathcal{K} \models Q$. The entailment test may be reduced to checking the unsatisfiability of a concept in the knowledge base, i.e., the emptiness of the concept.

$\mathcal{SHOIN}(\mathbf{D})$ is decidable if there are no number restrictions on non-simple roles. A role is non-simple iff it is transitive or has transitive subroles.

Given a predicate logic formula $F$, a *substitution* $\theta$ is a set of pairs $x/a$, where $x$ is a variable universally quantified in the outermost quantifier in $F$ and $a \in \mathbf{I}$. The application of $\theta$ to $F$, indicated by $F\theta$, is called an *instantiation of* $F$ and is obtained by replacing $x$ with $a$ in $F$ and by removing $x$ from the external quantification for every pair $x/a$ in $\theta$. Formulas not containing variables are called *ground*. A substitution $\theta$ is *grounding* for a formula $F$ if $F\theta$ is ground.

*Example 1.* The following KB is inspired by the ontology `people+pets` [16]:
$\exists hasAnimal.Pet \sqsubseteq NatureLover \quad fluffy : Cat \quad tom : Cat \quad Cat \sqsubseteq Pet$
$(kevin, fluffy) : hasAnimal \quad (kevin, tom) : hasAnimal$
It states that individuals that own an animal which is a pet are nature lovers and that *kevin* owns the animals *fluffy* and *tom*. Moreover, *fluffy* and *tom* are cats and cats are pets. The predicate logic formulas equivalent to the axioms are $F_1 = \forall x.\exists y.hasAnimal(x, y) \wedge Pet(y) \rightarrow NatureLover(x)$, $F_2 = hasAnimal(kevin, fluffy)$, $F_3 = hasAnimal(kevin, tom)$, $F_4 = Cat(fluffy)$, $F_5 = Cat(tom)$ and $F_6 = \forall x.Cat(x) \rightarrow Pet(x)$. The query $Q = kevin : NatureLover$ is entailed by the KB.

## 3 Querying KBs in $\mathcal{SHOIN}$(D)

Traditionally, a reasoning algorithm decides whether an axiom is entailed or not by a KB by refutation: axiom $E$ is entailed if $\neg E$ has no model in the KB. Besides deciding whether an axiom is entailed by a KB, we want to find also *instantiated* explanations for the axiom.

The problem of finding explanations for a query has been investigated by various authors [25, 11, 13, 7, 12]. It was called *axiom pinpointing* in [25] and considered as a non-standard reasoning service useful for tracing derivations and debugging ontologies. In particular, Schlobach and Cornet [25] define *minimal axiom sets* or *MinAs* for short.

**Definition 1 (MinA).** *Let $\mathcal{K}$ be a knowledge base and $Q$ an axiom that follows from it, i.e., $\mathcal{K} \models Q$. We call a set $M \subseteq \mathcal{K}$ a* minimal axiom set *or* MinA *for $Q$ in $\mathcal{K}$ if $M \models Q$ and it is minimal w.r.t. set inclusion.*

The problem of enumerating all MinAs is called MIN-A-ENUM in [25]. ALL-MINAS$(Q, \mathcal{K})$ is the set of all MinAs for query $Q$ in knowledge base $\mathcal{K}$.

However, in some cases, besides ALL-MINAS$(Q, \mathcal{K})$, we may want to know also the individuals to which the axioms were applied. We call this problem *instantiated axiom pinpointing*.

In instantiated axiom pinpointing we are interested in instantiated minimal sets of axioms that entail an axiom. An *instantiated axiom set* is a finite set $\mathcal{F} = \{(F_1, \theta_1), \ldots, (F_n, \theta_n)\}$ where $F_1, \ldots, F_n$ are axioms contained in $\mathcal{K}$ and $\theta_1, \ldots, \theta_n$ are substitutions. Given two instantiated axiom sets $\mathcal{F} = \{(F_1, \theta_1), \ldots, (F_n, \theta_n)\}$ and $\mathcal{E} = \{(E_1, \delta_1), \ldots, (E_m, \delta_m)\}$, we say that $\mathcal{F}$ *precedes* $\mathcal{E}$, written $\mathcal{F} \preceq \mathcal{E}$, iff, for each $(F_i, \theta_i) \in \mathcal{F}$, there exists an $(E_j, \delta_j) \in \mathcal{E}$ and a substitution $\eta$ such that $F_j\theta_j = E_i\delta_i\eta$.

**Definition 2 (InstMinA).** *Let $\mathcal{K}$ be a knowledge base and $Q$ an axiom that follows from it, i.e., $\mathcal{K} \models Q$. We call $\mathcal{F} = \{(F_1, \theta_1), \ldots, (F_n, \theta_n)\}$ an* instantiated minimal axiom set *or* InstMinA *for $Q$ in $\mathcal{K}$ if $\{F_1\theta_1, \ldots, F_n\theta_n\} \models Q$ and $\mathcal{F}$ is minimal w.r.t. precedence.*

Minimality w.r.t. precedence means that axioms in an InstMinA are as instantiated as possible. We call INST-MIN-A-ENUM the problem of enumerating all InstMinAs. ALL-INSTMINAS$(Q, \mathcal{K})$ is the set of all InstMinAs for the query $Q$ in knowledge base $\mathcal{K}$.

*Example 2.* The query $Q = kevin : NatureLover$ of Example 1 has two MinAs (in predicate logic): $\{ hasAnimal(kevin, fluffy), Cat(fluffy), \forall x.Cat(x) \rightarrow Pet(x), \forall x.\exists y.hasAnimal(x, y) \wedge Pet(y) \rightarrow NatureLover(x)\}$ and $\{ hasAnimal(kevin, tom), Cat(tom), \forall x.Cat(x) \rightarrow Pet(x), \forall x.\exists y.hasAnimal(x, y) \wedge Pet(y) \rightarrow NatureLover(x)\}$. The corresponding InstMinAs are $\{hasAnimal(kevin, fluffy), Cat(fluffy) \rightarrow Pet(fluffy), Cat(fluffy), hasAnimal(kevin, fluffy) \wedge Pet(fluffy) \rightarrow NatureLover(kevin)\}$ and $\{ hasAnimal(kevin, tom), Cat(tom), Cat(tom) \rightarrow Pet(tom), hasAnimal(kevin, tom) \wedge Pet(tom) \rightarrow NatureLover(kevin)\}$.

Instantiated axiom pinpointing is useful for a more fine-grained debugging of the ontology: by highlighting the individuals to which axioms are applied, it may point to parts of the ABox to be modified for repairing the KB. INST-MIN-A-ENUM is also required to support reasoning in probabilistic DLs, in particular in those that follow the DISPONTE probabilistic semantics [20, 19].

## 4 Related Work

Usually, DL reasoners implement a tableau algorithm using a procedural language. Since some tableau expansion rules are non-determinsitic, the developers have to implement a search strategy from scratch. Moreover, in order to solve MIN-A-ENUM, all different ways of entailing an axiom must be found. For example, Pellet [26] is a tableau reasoner for OWL written in Java and able to solve MIN-A-ENUM. It computes ALL-MINAS$(Q, \mathcal{K})$ by finding a single MinA using the tableau algorithm and then applying the *hitting set algorithm* [17] to find all the other MinAs. This is a black box method: Pellet repeatedly removes an axiom from the KB and then computes again a MinA recording all the different MinAs so found. Recently, BUNDLE [21] was proposed for reasoning over KBs following the DISPONTE probabilistic semantics. BUNDLE computes the probability of queries by solving the INST-MIN-A-ENUM problem. BUNDLE is based on Pellet's source code and modifies it for recording the individuals to which the axioms are applied. As in Pellet, it uses a black box method to compute ALL-INSTMINAS$(Q, \mathcal{K})$.

Reasoners written in Prolog can exploit Prolog's backtracking facilities for performing the search. This has been observed in various works. In [3] the authors proposed a tableau reasoner in Prolog for FOL based on free-variable semantic

tableaux. However, the reasoner is not tailored to DLs. SWI Prolog [28] has an RDF and Semantic Web library but is more focused on storing and querying RDF triples, while it has limited support for OWL reasoning. Meissner [15] presented the implementation of a Prolog reasoner for the DL $\mathcal{ALCN}$. This work was the basis of [8], that considered $\mathcal{ALC}$ and improved [15] by implementing heuristic search techniques to reduce the running time. Faizi [6] added to [8] the possibility of returning explanations for queries but still handled only $\mathcal{ALC}$.

In [10] the authors presented the KAON2 algorithm that exploits basic superposition, a refutational theorem proving method for FOL with equality, and a new inference rule, called decomposition, to reduce a $\mathcal{SHIQ}$ KB into a disjunctive datalog program, while DLog [14] is an ABox reasoning algorithm for the $\mathcal{SHIQ}$ language that allows to store the content of the ABox externally in a database and to respond to instance check and instance retrieval queries by transforming the KB into a Prolog program. TRILL differs from these work for the considered DL and from DLog for the capability of answering general queries.

## 5 The Tableau Algorithm

A *tableau* is an ABox. It can also be seen as a graph $G$ where each node represents an individual $a$ and is labeled with the set of concepts $\mathcal{L}(a)$ it belongs to. Each edge $\langle a, b\rangle$ in the graph is labeled with the set of roles $\mathcal{L}(\langle a, b\rangle)$ to which the couple $(a, b)$ belongs. A tableau algorithm proves an axiom by refutation: it starts from a tableau that contains the negation of the axiom and applies the tableau expansion rules. For example, if the query is a class assertion, $C(a)$, we add $\neg C$ to the label of $a$. If we want to test the emptyness (inconsistency) of a concept $C$, we add a new anonymous node $a$ to the tableau and add $C$ to the label of $a$. The axiom $C \sqsubseteq D$ can be proved by showing that $C \sqcap \neg D$ is empty. A *tableau algorithm* repeatedly applies a set of consistency preserving *tableau expansion rules* until a clash (i.e., a contradiction) is detected or a clash-free graph is found to which no more rules are applicable. A clash is, for example, a concept $C$ and a node $a$ where $C$ and $\neg C$ are present in the label of $a$, i.e. $\{C, \neg C\} \subseteq \mathcal{L}(a)$. If no clashes are found, the tableau represents a model for the negation of the query, which is thus not entailed.

In TRILL we use the tableau expansion rules for $\mathcal{SHOIN}(\mathbf{D})$ shown in Figure 1 that are similar to those of Pellet [11]. Each expansion rule updates as well a *tracing function* $\tau$, which associates sets of axioms with labels of nodes and edges. It maps couples (concept, individual) or (role, couple of individuals) to a fragment of the knowledge base $\mathcal{K}$. $\tau$ is initialized as the empty set for all the elements of its domain except for $\tau(C, a)$ and $\tau(R, \langle a, b\rangle)$ to which the values $\{a : C\}$ and $\{(a, b) : R\}$ are assigned if $a : C$ and $(a, b) : R$ are in the ABox respectively. The output of the tableau algorithm is a set $S$ of axioms that is a fragment of $\mathcal{K}$ from which the query is entailed.

For ensuring the termination of the algorithm, TRILL, as Pellet, uses a special condition known as *blocking* [11]. In a tableau a node $x$ can be a *nominal*

node if its label $\mathcal{L}(x)$ contains a *nominal* or a *blockable* node otherwise. If there is an edge $e = \langle x, y\rangle$ then $y$ is a *successor* of $x$ and $x$ is a *predecessor* of $y$. *Ancestor* is the transitive closure of predecessor while *descendant* is the transitive closure of successor. A node $y$ is called an *R-neighbour* of a node $x$ if $y$ is a successor of $x$ and $R \in \mathcal{L}(\langle x, y\rangle)$, where $R \in \mathbf{R}$.

An R-neighbour $y$ of $x$ is *safe* if (i) $x$ is blockable or if (ii) $x$ is a nominal node and $y$ is not blocked. Finally, a node $x$ is *blocked* if it has ancestors $x_0$, $y$ and $y_0$ such that all the following conditions are true: (1) $x$ is a successor of $x_0$ and $y$ is a successor of $y_0$, (2) $y$, $x$ and all nodes on the path from $y$ to $x$ are blockable, (3) $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x_0) = \mathcal{L}(y_0)$, (4) $\mathcal{L}(\langle x_0, x\rangle) = \mathcal{L}(\langle y_0, y\rangle)$. In this case, we say that $y$ *blocks* $x$. A node is blocked also if it is blockable and all its predecessors are blocked; if the predecessor of a safe node $x$ is blocked, then we say that $x$ is indirectly blocked.

Since we want to solve also the INST-MIN-A-ENUM problem, we modified the tableau expansion rules of Pellet to return a set of pairs (axiom, substitution) instead of a set of axioms. The tracing function $\tau$ now stores, together with information regarding concepts and roles, also information concerning individuals involved in the expansion rules, which will be returned at the end of the derivation process together with the axioms. In Figure 1, $(A \sqsubseteq D, a)$ is the abbreviation of $(A \sqsubseteq D, \{x/a\})$, $(R \sqsubseteq S, a)$ of $(R \sqsubseteq S, \{x/a\})$, $(R \sqsubseteq S, a, b)$ of $(R \sqsubseteq S, \{x/a, y/b\})$, $(\mathit{Trans}(R), a, b)$ of $(\mathit{Trans}(R), \{x/a, y/b\})$ and $(\mathit{Trans}(R), a, b, c)$ of $(\mathit{Trans}(R), \{x/a, y/b, z/c\})$, with $a$, $b$, $c$ individuals and $x$, $y$, $z$ variables contained in the logical translation of the axioms (Table 1). The most important modifications of Pellet's tableau algorithm are in the rules $\rightarrow \forall^+$ and $\rightarrow \forall$. For rule $\rightarrow \forall^+$, we record in the explanation a transitivity axiom for the role $R$ in which only two individuals, those connected by the super role $S$, are involved. For rule $\rightarrow \forall$, we make a distinction between the case in which $\forall S_1.C$ was added to $\mathcal{L}(a_1)$ by a chain of applications of $\rightarrow \forall^+$ or not. In the first case, we fully instantiate the transitivity and subrole axioms. In the latter case, we simply obtain $\tau(C, b)$ by combining the explanation of $\forall S_1.C(a_1)$ with that of $(a_1, b) : S_1$. To clarify how the rules $\rightarrow \forall$ and $\rightarrow \forall^+$ work we now give two examples.

*Example 3.* Let us consider the query $Q = ann : Person$ for the following knowledge base:

$kevin : \forall kin.Person \quad (kevin, lara) : relative \quad (lara, eva) : ancestor$
$(eva, ann) : ancestor \quad Trans(ancestor) \quad Trans(relative)$
$relative \sqsubseteq kin \quad ancestor \sqsubseteq relative$

TRILL first applies the $\rightarrow \forall^+$ rule to *kevin*, adding $\forall relative.Person$ to the label of *lara*. The tracing function $\tau$ is (in predicate logic):

$$\tau(\forall relative.Person, lara) = \{\ \forall y.kin(kevin, y) \rightarrow Person(y),\ relative(kevin, lara),\ relative(kevin, lara) \rightarrow kin(kevin, lara),\ \forall z.relative(kevin, lara) \wedge relative(lara, z) \rightarrow relative(kevin, z)\}$$

Note that the transitivity axiom is not fully instantiated, the variable $z$ is still present. Then TRILL applies the $\rightarrow \forall^+$ rule to *lara* adding $\forall ancestor.Person$ to *eva*. The tracing function $\tau$ is (in predicate logic):

**Deterministic rules:**
→ *unfold*: **if** $A \in \mathcal{L}(a)$, $A$ atomic and $(A \sqsubseteq D) \in K$, **then**
  **if** $D \notin \mathcal{L}(a)$, **then**
    $\mathcal{L}(a) := \mathcal{L}(a) \cup \{D\}$
    $\tau(D, a) := \tau(A, a) \cup \{(A \sqsubseteq D, a)\}$
→ *CE*: **if** $(C \sqsubseteq D) \in K$, **then**
  **if** $(\neg C \sqcup D) \notin \mathcal{L}(a)$, **then**
    $\mathcal{L}(a) := \mathcal{L}(a) \cup \{\neg C \sqcup D\}$
    $\tau(\neg C \sqcup D, a) := \{(C \sqsubseteq D, a)\}$
→ $\sqcap$: **if** $(C_1 \sqcap C_2) \in \mathcal{L}(a)$, **then**
  **if** $\{C_1, C_2\} \not\subseteq \mathcal{L}(a)$, **then**
    $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_1, C_2\}$
    $\tau(C_i, a) := \tau((C_1 \sqcap C_2), a)$
→ $\exists$: **if** $\exists S.C \in \mathcal{L}(a)$, **then**
  **if** $a$ has no S-neighbor $b$ with $C \in \mathcal{L}(b)$, **then**
    create new node $b$, $\mathcal{L}(b) := \{C\}$, $\mathcal{L}(\langle a, b\rangle) := \{S\}$,
    $\tau(C, b) := \tau((\exists S.C), a)$, $\tau(S, \langle a, b\rangle) := \tau((\exists S.C), a)$
→ $\forall$: **if** $\forall S_1.C \in \mathcal{L}(a_1)$, $a_1$ is not indirectly blocked and there is an $S_1$-neighbor $b$ of $a_1$, **then**
  **if** $C \notin \mathcal{L}(b)$, **then** $\mathcal{L}(b) := \mathcal{L}(a) \cup \{C\}$
  **if** there is a chain of individuals $a_2, \ldots, a_n$ and roles $S_2, \ldots, S_n$ such that
    $\bigcup_{i=2}^{n} \{(Trans(S_{i-1}), a_i, a_{i-1}), (S_{i-1} \sqsubseteq S_i, a_i, a_{i-1})\} \subseteq \tau(\forall S_1.C, a_1)$
    and $\neg\exists a_{n+1} : \{(Trans(S_n), a_{n+1}, a_n), (S_n \sqsubseteq S_{n+1}, a_{n+1})\} \subseteq \tau(\forall S_1.C, a_1)$, **then**
    $\tau(C, b) := \tau(\forall S_1.C, a_1) \setminus \bigcup_{i=2}^{n} \{(Trans(S_{i-1}), a_i, a_{i-1}), (S_{i-1} \sqsubseteq S_i, a_i, a_{i-1})\} \cup$
    $\bigcup_{i=2}^{n} \{(Trans(S_{i-1}), a_i, a_{i-1}, b), (S_{i-1} \sqsubseteq S_i, a_i, b)\} \cup \tau(S_1, \langle a_1, b\rangle)$
  **else**
    $\tau(C, b) := \tau(\forall S_1.C, a_1) \cup \tau(S_1, \langle a_1, b\rangle)$
→ $\forall^+$: **if** $\forall(S.C) \in \mathcal{L}(a)$, $a$ is not indirectly blocked
    and there is an $R$-neighbor $b$ of $a$, $Trans(R)$ and $R \sqsubseteq S$, **then**
  **if** $\forall R.C \notin \mathcal{L}(b)$, **then** $\mathcal{L}(b) := \mathcal{L}(b) \cup \{\forall R.C\}$
    $\tau(\forall R.C, b) := \tau(\forall S.C, a) \cup \tau(R, \langle a, b\rangle) \cup \{(Trans(R), a, b), (R \sqsubseteq S, a, b)\}$
→$\geq$: **if** $(\geq nS) \in \mathcal{L}(a)$, $a$ is not blocked, **then**
  **if** there are no $n$ safe $S$-neighbors $b_1, ..., b_n$ of $a$ with $b_i \neq b_j$, **then**
    create $n$ new nodes $b_1, ..., b_n$; $\mathcal{L}(\langle a, b_i\rangle) := \{S\}$;
    add in the ABox $\neq (b_i, b_j)$
    $\tau(S, \langle a, b_i\rangle) := \tau((\geq nS), a)$
    $\tau(\neq (b_i, b_j)) := \tau((\geq nS), a)$
→ $O$: **if**, $\{o\} \in \mathcal{L}(a) \cap \mathcal{L}(b)$ and not $a \neq b$, **then**
  $Merge(a, b)$
  $\tau(Merge(a, b)) := \tau(\{o\}, a) \cup \tau(\{o\}, b)$
  For each concept $C_i$ in $\mathcal{L}(a)$, $\tau(C_i, b) := \tau(C_i, a) \cup \tau(Merge(a, b))$
  (similarly for roles merged, and correspondingly for concepts in $\mathcal{L}(b)$)

**Non-deterministic rules:**
→ $\sqcup$: **if** $(C_1 \sqcup C_2) \in \mathcal{L}(a)$ and $a$ is not indirectly blocked, **then**
  **if** $\{C_1, C_2\} \cap \mathcal{L}(a) = \emptyset$, **then**
    Generate graphs $G_i := G$ for each $i \in \{1, 2\}$, $\mathcal{L}(a) := \mathcal{L}(a) \cup \{C_i\}$ for each $i \in \{1, 2\}$
    $\tau(C_i, a) := \tau((C_1 \sqcup C_2), a)$
→$\leq$: **if** $(\leq nS) \in \mathcal{L}(a)$, $a$ is not indirectly blocked
    and there are $m$ $S$-neighbors $b_1, ..., b_m$ of $a$ with $m > n$, **then**
  For each possible pair $b_i, b_j$, $1 \leq i, j \leq m; i \neq j$ **then**
    Generate a graph $G_k := G$
    $\tau(Merge(b_i, b_j)) := (\tau((\leq nS), a) \cup \tau(S, \langle a, b_1\rangle) ... \cup \tau(S, \langle a, b_m\rangle))$
    For each concept $C_i$ in $\mathcal{L}(b_i)$, $\tau(C_i, b_j) := \tau(C_i, b_i) \cup \tau(Merge(b_i, b_j))$
    (similarly for roles merged, and correspondingly for concepts in $\mathcal{L}(b_j)$)

**Fig. 1.** TRILL tableau expansion rules for OWL DL.

$\tau(\forall ancestor.Person, eva) = \{\ \forall y.kin(kevin, y) \rightarrow Person(y),$
$relative(kevin, lara),\ \mathbf{ancestor(lara, eva)},$
$\forall z.relative(kevin, lara) \wedge relative(lara, z) \rightarrow relative(kevin, z),$
$\forall \mathbf{z}.\mathbf{ancestor(lara, eva)} \wedge \mathbf{ancestor(eva, z)} \rightarrow \mathbf{ancestor(lara, z)},$
$relative(kevin, lara) \rightarrow kin(kevin, lara),$
$\mathbf{ancestor(lara, eva)} \rightarrow \mathbf{relative(lara, eva)}\ \}$

Now TRILL applies the $\rightarrow \forall$ rule to *eva* adding $Person$ to the label of *ann*. The tracing function $\tau$ is (in predicate logic):

$\tau(Person, ann) = \{\ \forall y.kin(kevin, y) \rightarrow Person(y),$
$relative(kevin, lara),\ ancestor(lara, eva),\ \mathbf{ancestor(eva, ann)},$
$relative(kevin, lara) \wedge relative(lara, \mathbf{ann}) \rightarrow relative(kevin, \mathbf{ann}),$
$ancestor(lara, eva) \wedge ancestor(eva, \mathbf{ann}) \rightarrow ancestor(lara, \mathbf{ann}),$
$relative(kevin, \mathbf{ann}) \rightarrow kin(kevin, \mathbf{ann}),$
$ancestor(lara, \mathbf{ann}) \rightarrow relative(lara, \mathbf{ann})\ \}$

Here the chain of transitivity and subrole axioms becomes ground. At this point the tableau contains a clash so the algorithm stops and returns the explanation given by $\tau(Person, ann)$.

It is easy to see that the explanation entails the axiom represented by the arguments of $\tau$. In general, the following theorem holds.

**Theorem 1.** *Let $Q$ be an axiom entailed by $\mathcal{K}$ and let $S$ be the output of a reasoner with the tableau expansion rules of Figure 1, such as TRILL, with input $Q$ and $\mathcal{K}$. Then $S \in$ All-InstMinAs$(Q, \mathcal{K})$.*

*Proof.* The full details of the proof are given in [18], Theorem 5, with reference to the reasoner BUNDLE that implements the same tableau algorithm as TRILL. The proof proceeds by induction on the number of rule applications following the proof of Theorem 2 of [11].

## 6 TRILL

We use the Thea2 library [27] that converts OWL DL ontologies to Prolog by exploiting a direct translation of the OWL axioms into Prolog facts. For example, a simple subclass axiom between two named classes $Cat \sqsubseteq Pet$ is written using the `subClassOf/2` predicate as `subClassOf('Cat','Pet')`. For more complex axioms Thea2 exploits the list construct of Prolog, so the axiom $NatureLover \equiv PetOwner \sqcup GardenOwner$ becomes `equivalentClasses(['NatureLover', unionOf(['PetOwner', 'GardenOwner']))`.

In order to represent the tableau, we use a couple $Tableau = (A, T)$, where $A$ is a list containing all the class assertions of the individuals with the corresponding value of $\tau$ and the information about nominal individuals, while $T$ is a triple ($G$,$RBN$,$RBR$) in which $G$ is a directed graph that contains the structure of the tableau, $RBN$ is a red-black tree in which each key is a couple of individuals and the value associated to it is the set of the labels of the edge between the two individuals, and $RBR$ is a red-black tree in which each key is a role and the value associated to it is the set of couples of individuals that are linked by the role. This representation allows us to rapidly find the information

needed during the execution of the tableau algorithm. For managing the *blocking* system we use a predicate for each blocking state, so we have the following predicates: `nominal/2`, `blockable/2`, `blocked/2`, `indirectly_blocked/2` and `safe/3`. Each predicate takes as arguments the individual $Ind$ and the tableau, $(A, T)$. `safe/3` takes as input also the role $R$. For each nominal individual $Ind$ at the time of the creation of the ABox we add the atom $nominal(Ind)$ to $A$, then every time we have to check the blocking status of an individual we call the corresponding predicate that returns the status by checking the tableau.

In TRILL deterministic and non-deterministic tableau expansion rules are treated differently, see Figure 1 for the list of rules. Deterministic rules are implemented by a predicate $rule\_name(Tab, Tab1)$ that, given the current tableau $Tab$, returns the tableau $Tab1$ to which the rule was applied. Figure 2 shows the code of the deterministic rule $\rightarrow$ *unfold*. The predicate `unfold_rule/2` searches in $Tab$ for an individual to which the rule can be applied and calls the predicate `find_sub_sup_class/3` in order to find the class to be added to the label of the individual. `find/2` implements the search for a class assertion. Since the data structure that stores class assertions is currently a list, `find/2` simply calls `member/2`. `absent/3` checks if the class assertion axiom with the associated explanation is already present in $A$. Non-deterministic rules are implemented by a

```
unfold_rule((A,T),([(classAssertion(D,Ind),[(Ax,Ind)|Expl])|A],T)):-
  find((classAssertion(C,Ind),Expl),A),
  atomic(C),
  find_sub_sup_class(C,D,Ax),
  absent(classAssertion(D,Ind),[(Ax,Ind)|Expl],(A,T)).

find_sub_sup_class(C,D,subClassOf(C,D)):-
  subClassOf(C,D).

find_sub_sup_class(C,D,equivalentClasses(L)):-
  equivalentClasses(L),
  member(C,L),
  member(D,L),
  C\==D.
```

**Fig. 2.** Code of $\rightarrow$ *unfold* rules.

predicate $rule\_name(Tab, TabList)$ that, given the current tableau $Tab$, returns the list of tableaux $TabList$ obtained by applying the rule. Figure 3 shows the code of the non-deterministic rule $\rightarrow \sqcup$. The predicate `or_rule/2` searches in $Tab$ for an individual to which the rule can be applied and unifies $TabList$ with the list of new tableaux created by `scan_or_list/6`.

Expansion rules are applied in order by `apply_all_rules/2`, first the non-deterministic ones and then the deterministic ones. The predicate `apply_nondet_rules(RuleList,Tab,Tab1)` takes as input the list of

```
or_rule((A,T),L):-
  find((classAssertion(unionOf(LC),Ind),Expl),A),
  \+ indirectly_blocked(Ind,T0),
  findall((A1,T),scan_or_list(LC,Ind,Expl,A,T,A1),L),
  L\=[],!.

scan_or_list([],_Ind,_Expl,A,T,A).

scan_or_list([C|_T],Ind,Expl,A,T,[(classAssertion(C,Ind),Expl)|A]):-
  absent(classAssertion(C,Ind),Expl,(A,T)).

scan_or_list([_C|T],Ind,Expl,A0,T,A):-
  scan_or_list(T,Ind,Expl,A0,T,A).
```

**Fig. 3.** Code of $\rightarrow \sqcup$ rule.

non-deterministic rules and the current tableau and returns a tableau obtained by the application of one rule. `apply_nondet_rules/3` is called as `apply_nondet_rules([or_rule,max_rule],Tab,Tab1)` and is shown in Fig. 4. If a non-deterministic rule is applicable, the list of tableaux obtained by its

```
apply_all_rules(Tab,Tab2):-
  apply_nondet_rules([or_rule,max_rule],Tab,Tab1),
  (Tab=Tab1 -> Tab2=Tab1 ; apply_all_rules(Tab1,Tab2)).


apply_nondet_rules([],Tab,Tab1):-
  apply_det_rules([o_rule,and_rule,unfold_rule,add_exists_rule,
    forall_rule,forall_plus_rule,exists_rule,min_rule],Tab,Tab1).

apply_nondet_rules([H|T],Tab,Tab1):-
  C=..[H,Tab,L],
  call(C),!,
  member(Tab1,L),
  Tab \= Tab1.

apply_nondet_rules([_|T],Tab,Tab1):-
  apply_nondet_rules(T,Tab,Tab1).
```

**Fig. 4.** Code of the predicates `apply_all_rules/2` and `apply_nondet_rules/3`.

application is returned by the rule predicate, a cut is performed to avoid backtracking to other rule choices and a tableau from the list is non-deterministically chosen with the `member/2` predicate. If no non-deterministic rule is applicable, deterministic rules are tried sequentially with the predicate `apply_det_rules/3`,

shown in Figure 5, that is called as `apply_det_rules(RuleList, Tab,Tab1)`. It takes as input the list of deterministic rules and the current tableau and returns a tableau obtained by the application of one rule. After the application of a deterministic rule, a cut avoids backtracking to other possible choices for the deterministic rules. If no rule is applicable, the input tableau is returned and rule application stops, otherwise a new round of rule application is performed. In each rule application round, a rule is applied if its result is not already present

```
apply_det_rules([],Tab,Tab).

apply_det_rules([H|T],Tab,Tab1):-
  C=..[H,Tab,Tab1],
  call(C),!.

apply_det_rules([_|T],Tab,Tab1):-
  apply_det_rules(T,Tab,Tab1).
```

**Fig. 5.** Code of the predicates `apply_det_rules/3`.

in the tableau. This avoids both infinite loops in rule application and considering alternative rules when a rule is applicable. In fact, if a rule is applicable in a tableau, it will also be so in any tableaux obtained by its expansion, thus the choice of which expansion rule to apply introduces "don't care" non-determinism. Differently, non-deterministic rules introduce in the algorithm also "don't know" non-determinism, since a single tableau is expanded into a set of tableaux. We use Prolog search only to handle "don't know" non-determinism.

*Example 4.* Let us consider the knowledge base presented in Example 1 and the query $Q = kevin : NatureLover$. After the initialization of the tableau, TRILL can apply the $\rightarrow$ *unfold* rule to the individuals *tom* or *fluffy*. Suppose it selects *tom*. The tracing function $\tau$ becomes (in predicate logic):

$\tau(Pet, tom) = \{\ Cat(tom),\ Cat(tom) \rightarrow Pet(tom)\}$

At this point TRILL applies the $\rightarrow$ $CE$ rule to *kevin*, adding $\neg(\exists hasAnimal.Pet) \sqcup NatureLover = \forall hasAnimal.(\neg Pet) \sqcup NatureLover$ to $\mathcal{L}(kevin)$ with the following tracing function:

$\tau(\forall hasAnimal.(\neg Pet) \sqcup NatureLover, kevin) = \{$
$\exists y.hasAnimal(kevin, y) \wedge Pet(y) \rightarrow NatureLover(kevin)\}$

Then it applies the $\rightarrow \sqcup$ rule to *kevin* generating two tableaux. In this step we have a backtracking point because we have to choose which tableau to expand. In the first one TRILL adds $\forall hasAnimal.(\neg Pet)$ to the label of *kevin* with the tracing function

$\tau(\forall hasAnimal.(\neg Pet), kevin) = \{$
$\exists y.hasAnimal(kevin, y) \wedge Pet(y) \rightarrow NatureLover(kevin)\}$

Now it can apply the $\rightarrow \forall$ rule to *kevin*. In this step it can use either *tom* or *fluffy*, supposing it selects *tom* the tracing function will be:

$\tau(\neg(Pet), tom) = \{\ hasAnimal(kevin, tom),$
$hasAnimal(kevin, tom) \wedge Pet(tom) \rightarrow NatureLover(kevin)\}$

At this point this first tableau contains a clash for the individual *tom*, thus TRILL backtracks and expands the second tableau. The second tableau was created by applying the $\rightarrow CE$ rule that added $NatureLover$ to the label of *kevin*, so the second tableau contains a clash, too. Now TRILL joins the tracing functions of the two clashes to find the following InstMinA:

$\{hasAnimal(kevin, tom) \wedge Pet(tom) \rightarrow NatureLover(kevin),$
$hasAnimal(kevin, tom),\ Cat(tom),\ Cat(tom) \rightarrow Pet(tom)\}.$

The tableau algorithm returns a single InstMinA. The computation of All-InstMinAs$(Q, \mathcal{K})$ is performed by simply calling `findall/3` over the tableau predicate.

*Example 5.* Let us consider Example 4. Once the first InstMinA is found, TRILL performs backtracking. Supposing it applies the $\rightarrow$ *unfold* rule to the individual *fluffy* instead of *tom* and following the same steps used in Example 4 it finds a new InstMinA:

$\{hasAnimal(kevin, fluffy) \wedge Pet(fluffy) \rightarrow NatureLover(kevin),$
$hasAnimal(kevin, fluffy),\ Cat(fluffy),\ Cat(fluffy) \rightarrow Pet(fluffy)\}.$

## 7 Experiments

In this section, we evaluate TRILL performances when computing instantiated explanations by comparing it to BUNDLE that also solves the inst-min-a-enum problem. We consider four different knowledge bases of various complexity: the BRCA[3] that models the risk factor of breast cancer, an extract of the DBPedia[4] ontology that has been obtained from Wikipedia, the Biopax level 3[5] that models metabolic pathways and the Vicodi[6] that contains information on European history. For the tests, we used the DBPedia and the Biopax KBs without ABox while for BRCA and Vicodi we used a little ABox contaning 1 individual for the first one and 19 individuals for the second one. We ran two different subclass-of queries w.r.t. the DBPedia and the Biopax datasets and two different instance-of queries w.r.t. the other KBs. For each KB, we ran each query 50 times for a total of 100 executions of the reasoners. Table 2 shows, for each ontology, the number of axioms, the average number of explanations and the average time in milliseconds that TRILL and BUNDLE took for answering the queries. In particular, in order to stress the algorithm, the BRCA and the version of DBPedia that we used contain a large number of subclass axioms between complex concepts.These preliminary tests show that TRILL performance can sometimes be better than BUNDLE, even if it lacks all the optimizations that BUNDLE inherits from Pellet. This represents evidence that a Prolog implementation of a Semantic Web tableau reasoner is feasible and that may lead to a practical system.

[3] `http://www2.cs.man.ac.uk/~klinovp/pronto/brc/cancer_cc.owl`
[4] `http://dbpedia.org/`
[5] `http://www.biopax.org/`
[6] `http://www.vicodi.org/`

| Dataset | n. axioms | av. n. expl | TRILL time (ms) | BUNDLE time (ms) |
|---|---|---|---|---|
| BRCA | 322 | 6.5 | 95,691 | 10,210 |
| DBPedia | 535 | 16.0 | 80,804 | 28,040 |
| Biopax level 3 | 826 | 2.0 | 24 | 1,451 |
| Vicodi | 220 | 1.0 | 136 | 1,004 |

**Table 2.** Results of the experiments in terms of average times for inference.

## 8 Conclusions

In this paper we presented the algorithm TRILL for reasoning on $\mathcal{SHOIN}(\mathbf{D})$ knowledge bases and its Prolog implementation. The results we obtained show that Prolog is a viable language for implementing DL reasoning algorithms and that performances are comparable with those of a state of the art reasoner.

In the future we plan to apply various optimizations to TRILL in order to better manage the expansion of the tableau. In particular, we plan to carefully choose the rule and node application order. Moreover, we plan to exploit TRILL for performing reasoning on probabilistic ontologies and on integration of probabilistic logic programming with DLs and for implementing learning algorithms for such integration, along the lines of [4, 5, 22].

## References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Baader, F., Horrocks, I., Sattler, U.: Description logics. In: Handbook of knowledge representation, chap. 3, pp. 135–179. Elsevier (2008)
3. Beckert, B., Posegga, J.: leantap: Lean tableau-based deduction. J. Autom. Reasoning 15(3), 339–358 (1995)
4. Bellodi, E., Riguzzi, F.: Learning the structure of probabilistic logic programs. In: Muggleton, S.H., Tamaddoni-Nezhad, A., Lisi, F.A. (eds.) ILP 2011. LNCS, vol. 7207, pp. 61–75. Springer (2012)
5. Bellodi, E., Riguzzi, F.: Expectation Maximization over binary decision diagrams for probabilistic logic programs. Intel. Data Anal. 17(2), 343–363 (2013)
6. Faizi, I.: A Description Logic Prover in Prolog, Bachelor's thesis, Informatics Mathematical Modelling, Technical University of Denmark (2011)
7. Halaschek-Wiener, C., Kalyanpur, A., Parsia, B.: Extending tableau tracing for ABox updates. Tech. rep., University of Maryland (2006)
8. Herchenröder, T.: Lightweight Semantic Web Oriented Reasoning in Prolog: Tableaux Inference for Description Logics. Master's thesis, School of Informatics, University of Edinburgh (2006)
9. Hitzler, P., Krötzsch, M., Rudolph, S.: Foundations of Semantic Web Technologies. CRCPress (2009)
10. Hustadt, U., Motik, B., Sattler, U.: Deciding expressive description logics in the framework of resolution. Inf. Comput. 206(5), 579–601 (2008)

11. Kalyanpur, A.: Debugging and Repair of OWL Ontologies. Ph.D. thesis, The Graduate School of the University of Maryland (2006)
12. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: Aberer, K., et al. (eds.) ISWC/ASWC 2007. LNCS, vol. 4825, pp. 267–280. Springer (2007)
13. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.A.: Debugging unsatisfiable classes in OWL ontologies. J. Web Sem. 3(4), 268–293 (2005)
14. Lukácsy, G., Szeredi, P.: Efficient description logic reasoning in prolog: The dlog system. TPLP 9(3), 343–414 (2009)
15. Meissner, A.: An automated deduction system for description logic with alcn language. Studia z Automatyki i Informatyki 28-29, 91–110 (2004)
16. Patel-Schneider, P, F., Horrocks, I., Bechhofer, S.: Tutorial on OWL (2003)
17. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. 32(1), 57–95 (1987)
18. Riguzzi, F., Bellodi, E., Lamma, E., , Zese, R.: Probabilistic description logics under the distribution semantics. Tech. Rep. ML-01, University of Ferrara (2013), `http://sites.unife.it/ml/bundle`
19. Riguzzi, F., Bellodi, E., Lamma, E.: Probabilistic Datalog+/- under the distribution semantics. In: Kazakov, Y., Lembo, D., Wolter, F. (eds.) DL 2012. CEUR Workshop Proceedings, vol. 846. Sun SITE Central Europe (2012)
20. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Epistemic and statistical probabilistic ontologies. In: Bobillo, F., et al. (eds.) URSW 2012. CEUR Workshop Proceedings, vol. 900, pp. 3–14. Sun SITE Central Europe (2012)
21. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: BUNDLE: A reasoner for probabilistic ontologies. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 183–197. Springer (2013), `http://www.ing.unife.it/docenti/FabrizioRiguzzi/Papers/RigBelLam-RR13b.pdf`
22. Riguzzi, F., Bellodi, E., Lamma, E., Zese, R.: Parameter learning for probabilistic ontologies. In: Faber, W., Lembo, D. (eds.) RR 2013. LNCS, vol. 7994, pp. 265–270. Springer (2013), `http://www.ing.unife.it/docenti/FabrizioRiguzzi/Papers/RigBelLam-RR13a.pdf`
23. Sato, T.: A statistical learning method for logic programs with distribution semantics. In: ICLP 1995. pp. 715–729. MIT Press (1995)
24. Sattler, U., Calvanese, D., Molitor, R.: Relationships with other formalisms. In: Description Logic Handbook, chap. 4, pp. 137–177. Cambridge University Press (2003)
25. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Gottlob, G., Walsh, T. (eds.) IJCAI 2003. pp. 355–362. Morgan Kaufmann (2003)
26. Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
27. Vassiliadis, V., Wielemaker, J., Mungall, C.: Processing owl2 ontologies using thea: An application of logic programming. In: International Workshop on OWL: Experiences and Directions. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
28. Wielemaker, J., Schrijvers, T., Triska, M., Lager, T.: SWI-Prolog. Theory and Practice of Logic Programming 12(1-2), 67–96 (2012)

# Nested Sequent Calculi and Theorem Proving for Normal Conditional Logics

Nicola Olivetti[1] and Gian Luca Pozzato[2]

[1] Aix-Marseille Université, CNRS, LSIS UMR 7296 - France - nicola.olivetti@univ-amu.fr
[2] Dipartimento di Informatica - Università di Torino - Italy gianluca.pozzato@unito.it

**Abstract.** In this paper we focus on proof methods and theorem proving for normal conditional logics, by describing nested sequent calculi as well as a theorem prover for them. Nested sequent calculi are a useful generalization of ordinary sequent calculi, where sequents are allowed to occur within sequents. Nested sequent calculi have been profitably employed in the area of (multi)-modal logic to obtain analytic and modular proof systems for these logics. In this work, we describe nested sequent calculi recently introduced for the basic conditional logic CK and some of its significant extensions. We also provide a calculus for Kraus Lehman Magidor cumulative logic C. The calculi are internal (a sequent can be directly translated into a formula), cut-free and analytic. Moreover, they can be used to design (sometimes optimal) decision procedures for the respective logics, and to obtain complexity upper bounds. Our calculi are an argument in favour of nested sequent calculi for modal logics and alike, showing their versatility and power. We also describe NESCOND, a Prolog implementation of nested sequent calculi mentioned above. NESCOND (NESted sequent calculi for propositional CONDitional logics) is inspired by the methodolody of leanTAP. The paper also shows some experimental results, witnessing that the performances of NESCOND are promising. NESCOND is available at http://www.di.unito.it/∼pozzato/nescond/

## 1 Introduction

Conditional logics extend classical logic by means of a conditional operator $\Rightarrow$. They can be seen as a generalization of (multi)modal logics, where the modality $\Rightarrow$ is indexed by a formula of the same language. Conditional logics have a long history: Lewis [15, 16] introduced them in order to formalize a kind of hypothetical reasoning: the conditional formula $A \Rightarrow B$ is used to formalize a sentence like "if $A$ were the case then $B$" that cannot be captured by classical logic with material implication. One original motivation was to formalize *counterfactual sentences*, i.e. conditionals of the form "if $A$ were the case then $B$ would be the case", where $A$ is false.

Over the years, conditional logics firmly established themselves in various fields of artificial intelligence and knowledge representation. Just to mention a few, they have been used[3] to reason about prototypical properties [10] and to model belief change [12, 11]. Moreover, they can provide an axiomatic foundation of nonmonotonic reasoning [7, 14]: in detail, a conditional $A \Rightarrow B$ is read as "in normal circumstances, if $A$ then $B$". Recently, constructive conditional logics have been applied to reason about *access control* policies [9, 8]: the statement $A$ **says** $B$, intuitively meaning that a user/program

[3] We refer to [20, 1] for a complete bibliography about conditional logics.

$A$ *asserts* $B$ to hold in the system, can be naturally expressed by a conditional $A \Rightarrow B$. Finally, a kind of (multi)-conditional logics [3, 6] have been used to formalize epistemic change in a multi-agent setting and in some kind of epistemic "games", each conditional operator expresses the "conditional beliefs" of an agent.

All conditional logics enjoy a possible world semantics, with the intuition that a conditional $A \Rightarrow B$ is true in a world $x$ if $B$ is true in the set of worlds where $A$ is true and that are most similar to/closest to/"as normal as" $x$. Since there are different ways of formalizing "the set of worlds similar/closest/..." to a given world, there are expectedly rather different semantics for conditional logics, from the most general selection function semantics to the stronger sphere semantics.

However, from the point of view of proof-theory and automated deduction, conditional logics have not achieved a state of the art comparable with, say, the one of modal logics, where there are well-established calculi, whose proof-theoretical and computational properties are well-understood. In this work we first describe *nested sequent calculi*, called $\mathcal{NS}$, for propositional conditional logics, recently introduced in [1, 2]. Nested sequent calculi, introduced by Kashima in [13] for classical modal logics, are a natural generalization of ordinary sequent calculi where sequents are allowed to occur within sequents. However, a nested sequent always corresponds to a formula of the language, so that we can think of the rules as operating "inside a formula", combining subformulas rather than just combining outer occurrences of formulas as in ordinary sequent calculi.

We will consider the basic normal conditional logic CK and its extensions with ID, MP and CEM, as well as the cumulative logic **C** introduced in [14] which corresponds to the *flat* fragment (i.e., without nested conditionals) of CK+CSO+ID. The calculi are rather natural, all rules have a fixed number of premises. Completeness is established by cut-elimination, whose peculiarity is that it must take into account the substitution of equivalent antecedents of conditionals (a condition corresponding to *normality*). The calculi can be used to obtain a decision procedure for the respective logics by imposing some restrictions preventing redundant applications of rules. In all cases, we get a PSPACE upper bound, a bound that for CK and its extensions with ID and MP is optimal (but not for CK+CEM that is known to be CONP). For flat CK+CSO+ID = cumulative logic **C** we also get a PSPACE bound, we are not aware of a better upper bound for this logic (although we may suspect that it is not optimal).

Furthermore, we describe an implementation of $\mathcal{NS}$ calculi in Prolog. The program, called NESCOND, gives a PSPACE decision procedure for the respective logics, and it is inspired by the methodology introduced by the system leanTAP [4], even if it does not fit its style in a rigorous manner. The basic idea is that each axiom or rule of the nested sequent calculi is implemented by a Prolog clause of the program. The resulting code is therefore simple and compact: the implementation of NESCOND for CK consists of only 6 predicates, 24 clauses and 34 lines of code. We also provide some experimental results to show that the performances of NESCOND are promising, especially compared to the ones of CondLean [17, 18] and GOALD$\mathcal{U}$CK [19], to the best of our knowledge the only existing provers for conditional logics. This shows that nested sequent calculi are not only a proof theoretical tool, but they can be the basis of efficient theorem proving for conditional logics.

## 2 Normal Conditional Logics

We consider a propositional conditional language $\mathcal{L}$ over a set $ATM$ of propositional variables. Formulas of $\mathcal{L}$ are built as usual: $\bot$, $\top$ and the propositional variables in $ATM$ are *atomic formulas*; if $A$ and $B$ are formulas, then $\neg A$ and $A \otimes B$ are *compound formulae*, where $\otimes \in \{\wedge, \vee, \rightarrow, \Rightarrow\}$. We adopt the *selection function semantics*.

**Definition 1 (Selection function semantics).** *A model is a triple $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$:*
- *$\mathcal{W}$ is a non empty set of worlds;*
- *$f : \mathcal{W} \times 2^{\mathcal{W}} \longmapsto 2^{\mathcal{W}}$ is the* selection function*;*
- *$[\,]$ is the* evaluation function*, which assigns to an atom $P \in ATM$ the set of worlds where $P$ is true, and is extended to boolean formulas as follows:*
  - $[\top] = \mathcal{W}$;
  - $[\bot] = \emptyset$;
  - $[\neg A] = \mathcal{W} - [A]$;
  - $[A \wedge B] = [A] \cap [B]$;
  - $[A \vee B] = [A] \cup [B]$;
  - $[A \rightarrow B] = [B] \cup (\mathcal{W} - [A])$;
  - $[A \Rightarrow B] = \{w \in \mathcal{W} \mid f(w, [A]) \subseteq [B]\}$.

*A formula $F \in \mathcal{L}$ is valid in a model $\mathcal{M} = \langle \mathcal{W}, f, [\,] \rangle$, and we write $\mathcal{M} \models F$, if $[F] = \mathcal{W}$. A formula $F \in \mathcal{L}$ is valid, and we write $\models F$, if it is valid in every model, that is to say $\mathcal{M} \models F$ for every $\mathcal{M}$.*

The semantics above characterizes the *basic conditional system*, called CK, where no specific properties of the selection function are assumed. An axiomatization of CK is given by ($\vdash$ denotes provability in the axiom system):

- any axiomatization of the classical propositional calculus (prop)
- If $\vdash A$ and $\vdash A \rightarrow B$, then $\vdash B$ (Modus Ponens)
- If $\vdash A \leftrightarrow B$ then $\vdash (A \Rightarrow C) \leftrightarrow (B \Rightarrow C)$ (RCEA)
- If $\vdash (A_1 \wedge \cdots \wedge A_n) \rightarrow B$ then $\vdash (C \Rightarrow A_1 \wedge \cdots \wedge C \Rightarrow A_n) \rightarrow (C \Rightarrow B)$ (RCK)

Moreover, we consider the following standard extensions of the basic system CK:

| System | Axiom | Model condition |
|---|---|---|
| **ID** | $A \Rightarrow A$ | $f(w, [A]) \subseteq [A]$ |
| **CEM** | $(A \Rightarrow B) \vee (A \Rightarrow \neg B)$ | $\mid f(w, [A]) \mid \leq 1$ |
| **MP** | $(A \Rightarrow B) \rightarrow (A \rightarrow B)$ | $w \in [A]$ implies $w \in f(w, [A])$ |
| **CSO** | $(A \Rightarrow B) \wedge (B \Rightarrow A) \rightarrow ((A \Rightarrow C) \rightarrow (B \Rightarrow C))$ | $f(w, [A]) \subseteq [B]$ and $f(w, [B]) \subseteq [A]$ implies $f(w, [A]) = f(w, [B])$ |

## 3 Nested Sequent Calculi $\mathcal{N}S$ for Conditional Logics

In this section we recall nested sequent calculi $\mathcal{N}S$ introduced in [1, 2], where $S$ is an abbreviation for CK{+X}, with X $\in$ {CEM, ID, MP, ID+MP, CEM+ID}. We are able to deal with the basic normal conditional logic CK and its extensions with axioms ID, MP and CEM. We are also able to deal with some combinations of them, namely the systems allowing ID with either MP or CEM. The problem of extending $\mathcal{N}S$ to the conditional

$$\Gamma(P, \neg P)\ (AX)_{P \in ATM} \qquad \Gamma(\top)\ (AX_\top) \qquad \Gamma(\neg\bot)\ (AX_\bot) \qquad \frac{\Gamma(A)}{\Gamma(\neg\neg A)}(\neg)$$

$$\frac{\Gamma(A) \quad \Gamma(B)}{\Gamma(A \wedge B)}(\wedge^+) \qquad \frac{\Gamma(\neg A, \neg B)}{\Gamma(\neg(A \wedge B))}(\wedge^-) \qquad \frac{\Gamma(A, B)}{\Gamma(A \vee B)}(\vee^+) \qquad \frac{\Gamma(\neg A) \quad \Gamma(\neg B)}{\Gamma(\neg(A \vee B))}(\vee^-)$$

$$\frac{\Gamma(\neg A, B)}{\Gamma(A \to B)}(\to^+) \qquad \frac{\Gamma(A) \quad \Gamma(\neg B)}{\Gamma(\neg(A \to B))}(\to^-) \qquad \frac{\Gamma(\neg(A \Rightarrow B), [A' : \Delta, \neg B]) \quad A, \neg A' \quad A', \neg A}{\Gamma(\neg(A \Rightarrow B), [A' : \Delta])}(\Rightarrow^-)$$

$$\frac{\Gamma([A : B])}{\Gamma(A \Rightarrow B)}(\Rightarrow^+) \qquad \frac{\Gamma(\neg(A \Rightarrow B), A) \quad \Gamma(\neg(A \Rightarrow B), \neg B)}{\Gamma(\neg(A \Rightarrow B))}(MP) \qquad \frac{\Gamma([A : \Delta, \Sigma], [B : \Sigma]) \quad A, \neg B \quad B, \neg A}{\Gamma([A : \Delta], [B : \Sigma])}(CEM)$$

$$\frac{\Gamma([A : \Delta, \neg A])}{\Gamma([A : \Delta])}(ID) \qquad \frac{\Gamma, \neg(A \Rightarrow B), [A' : \Delta, \neg B] \quad \Gamma, \neg(A \Rightarrow B), [A : A'] \quad \Gamma, \neg(A \Rightarrow B), [A' : A]}{\Gamma, \neg(A \Rightarrow B), [A' : \Delta]}(CSO)$$

**Fig. 1.** The nested sequent calculi $\mathcal{N}S$.

logics allowing both MP and CEM is open at present. As usual, the completeness of the calculi is an easy consequence of the admissibility of cut. We are also able to turn $\mathcal{N}S$ into a terminating calculus, which gives us a decision procedure for the respective conditional logics.
A nested sequent $\Gamma$ is defined inductively as follows:

- a formula of $\mathcal{L}$ is a nested sequent;
- if $A$ is a formula and $\Gamma$ is a nested sequent, then $[A : \Gamma]$ is a nested sequent;
- a finite multiset of nested sequents is a nested sequent.

A nested sequent can be displayed as

$$A_1, \ldots, A_m, [B_1 : \Gamma_1], \ldots, [B_n : \Gamma_n],$$

where $n, m \geq 0$, $A_1, \ldots, A_m, B_1, \ldots, B_n$ are formulas and $\Gamma_1$, ..., $\Gamma_n$ are nested sequents.

A nested sequent can be directly interpreted as a formula, just replace "," by $\vee$ and ":" by $\Rightarrow$. More explicitly, the interpretation of a nested sequent $A_1, \ldots, A_m, [B_1 : \Gamma_1], \ldots, [B_n : \Gamma_n]$ is inductively defined by the formula

$$\mathcal{F}(\Gamma) = A_1 \vee \ldots \vee A_m \vee (B_1 \Rightarrow \mathcal{F}(\Gamma_1)) \vee \ldots \vee (B_n \Rightarrow \mathcal{F}(\Gamma_n)).$$

The calculi $\mathcal{N}S$ are shown in Figure 1. As usual, we say that a nested sequent $\Gamma$ is *derivable* in $\mathcal{N}S$ if it admits a *derivation*. A derivation is a tree whose nodes are nested sequents. A branch is a sequence of nodes $\Gamma_1, \Gamma_2, \ldots, \Gamma_n, \ldots$ such that each node $\Gamma_i$ is obtained from its immediate successor $\Gamma_{i-1}$ by applying *backward* a rule of $\mathcal{N}S$, having $\Gamma_{i-1}$ as the conclusion and $\Gamma_i$ as one of its premises. A branch is closed if one of its nodes is an instance of axioms $(AX)$, $(AX_\top)$, $(AX_\bot)$, otherwise it is open. We say that a tree is closed if all its branches are closed. A nested sequent $\Gamma$ has a derivation in $\mathcal{N}S$ if there is a closed tree having $\Gamma$ as the root. As an example, Figure 2 shows a derivation in the calculus $\mathcal{N}$CK+ID of an instance of the axiom ID.

$$\dfrac{\dfrac{\dfrac{}{[P : P, \neg P]}\,(AX)}{[P : P]}\,(ID)}{P \Rightarrow P}\,(\Rightarrow^{+})$$

**Fig. 2.** A derivation of an instance of the axiom ID in $\mathcal{N}$CK+ID.

We have also provided a nested sequent calculus for the flat fragment, i.e. without nested conditionals, of CK+CSO+ID, corresponding to KLM logic **C**. The rules of the calculus, called $\mathcal{N}\mathbf{C}_{\text{KLM}}$, are those ones of $\mathcal{N}$CK+ID (restricted to the flat fragment) where the rule $(\Rightarrow^{-})$ is replaced by the rule $(CSO)$.

The specificity of nested sequent calculi is to allow inferences that apply within sequence. In order to introduce the rules of the calculus, we need the notion of context. Intuitively a context denotes a "hole", a *unique* empty position, within a sequent that can be filled by a sequent. We use the symbol $(\ )$ to denote the empty context. A context is defined inductively as follows: $\Gamma(\ ) = \Lambda, (\ )$ is a context; if $\Sigma(\ )$ is a context, $\Gamma(\ ) = \Lambda, [A : \Sigma(\ )]$ is a context. Finally, we define the result of filling "the hole" of a context by a sequent. Let $\Gamma(\ )$ be a context and $\Delta$ be a sequent, then the sequent obtained by filling the context by $\Delta$, denoted by $\Gamma(\Delta)$ is defined as follows: if $\Gamma(\ ) = \Lambda, (\ )$, then $\Gamma(\Delta) = \Lambda, \Delta$; if $\Gamma(\ ) = \Lambda, [A : \Sigma(\ )]$ then $\Gamma(\Delta) = \Lambda, [A : \Sigma(\Delta)]$. The notions of derivation and of derivable sequent are defined as usual.

Nested sequent calculi $\mathcal{N}S$ are sound and complete with respect to the semantics for the respective logics.

**Theorem 1.** *The nested sequent calculi $\mathcal{N}S$ are sound and complete for the respective logics, i.e. a formula $F$ of $\mathcal{L}$ is valid in CK+X if and only if it is derivable in $\mathcal{N}$CK+X.*

*Proof.* (*Soundness*): If $\Gamma$ is derivable in $\mathcal{N}S$, then $\Gamma$ is valid. To improve readability, we slightly abuse the notation identifying a sequent $\Gamma$ with its interpreting formula $\mathcal{F}(\Gamma)$, thus we shall write $A \Rightarrow \Delta, \Gamma \wedge \Delta$, etc. instead of $A \Rightarrow \mathcal{F}(\Delta), \mathcal{F}(\Gamma) \wedge \mathcal{F}(\Delta)$. First, we prove that nested inference is sound, that is to say: let $\Gamma(\ )$ be any context. If the formula $A_1 \wedge \ldots \wedge A_n \rightarrow B$, with $n \geq 0$, is (CK{+X})-valid, then also $\Gamma(A_1) \wedge \ldots \wedge \Gamma(A_n) \rightarrow \Gamma(B)$ is (CK{+X}) valid. The proof is by induction on the depth of a context $\Gamma(\ )$, defined as follows:

- $\Gamma(\ ) = \Delta, (\ )$ is a context with depth $d(\Gamma(\ )) = 0$;
- if $\Sigma(\ )$ is a context, $\Gamma(\ ) = \Delta, [A : \Sigma(\ )]$ is a context with depth $d(\Gamma(\ )) = 1 + d(\Sigma(\ ))$.

Let $d(\Gamma(\ )) = 0$, then $\Gamma = \Lambda, (\ )$. Since $A_1 \wedge \ldots \wedge A_n \rightarrow B$ is valid, by propositional reasoning, we have that also $(\Lambda \vee A_1) \wedge \ldots (\Lambda \vee A_n) \rightarrow (\Lambda \vee B)$ is valid, that is $\Gamma(A_1) \wedge \ldots \wedge \Gamma(A_n) \rightarrow \Gamma(B)$ is valid. Let $d(\Gamma(\ )) > 0$, then $\Gamma(\ ) = \Delta, [C : \Sigma(\ )]$. By inductive hypothesis, we have that $\Sigma(A_1) \wedge \ldots \wedge \Sigma(A_n) \rightarrow \Sigma(B)$ is valid. By (RCK), we obtain that also $(C \Rightarrow \Sigma(A_1)) \wedge \ldots \wedge (C \Rightarrow \Sigma(A_n)) \rightarrow (C \Rightarrow \Sigma(B))$ is valid. Then, we get that $(\Lambda \vee (C \Rightarrow \Sigma(A_1))) \wedge \ldots \wedge (\Lambda \vee (C \Rightarrow \Sigma(A_n))) \rightarrow (\Lambda \vee (C \Rightarrow \Sigma(B)))$ is also valid, that is $\Gamma(A_1) \wedge \ldots \wedge \Gamma(A_n) \rightarrow \Gamma(B)$ is valid.

Now we can prove the soundness of the calculi, namely if $\Gamma$ is derivable in $\mathcal{N}S$, then $\Gamma$ is valid. By induction on the height of the derivation of $\Gamma$. In the base case, $\Gamma$ is an axiom, that is $\Gamma = \Gamma(P, \neg P)$; then, trivially $P \vee \neg P$ is valid, then also $\Gamma(P, \neg P)$ is valid by the fact that nested inference is sound. Similarly for $\Gamma(\top)$ and $\Gamma(\neg\bot)$. For the inductive step, in order to save space, we only show the most interesting case of $(\Rightarrow^-)$: $\Gamma = \Gamma(\neg(A \Rightarrow B), [A' : \Delta])$ is derived from $(i)$ $\Gamma(\neg(A \Rightarrow B), [A' : \Delta, \neg B])$, $(ii)$ $\neg A, A'$, $(iii)$ $\neg A', A$. By inductive hypothesis we have that $A \leftrightarrow A'$ is valid. We show that also $(*)$ $[\neg(A \Rightarrow B) \vee (A' \Rightarrow (\Delta \vee \neg B))] \rightarrow [\neg(A \Rightarrow B) \vee (A' \Rightarrow \Delta)]$ is valid, then we conclude since nested inference is sound and by applying the inductive hypothesis. To prove (*), by (RCK) we have that the following is valid: $[(A' \Rightarrow B) \wedge (A' \Rightarrow (\Delta \vee \neg B))] \rightarrow (A' \Rightarrow \Delta)$. Since $A \leftrightarrow A'$ is valid, by (RCEA) we get that $(A \Rightarrow B) \rightarrow (A' \Rightarrow B)$ is valid, so that also $(A \Rightarrow B) \rightarrow ((A' \Rightarrow (\Delta \vee \neg B)) \rightarrow (A' \Rightarrow \Delta))$ is valid, then we conclude by propositional reasoning.

(*Completeness*): If $\Gamma$ is valid, then $\Gamma$ has a derivation in $\mathcal{N}S$. First of all, it can be shown that weakening and contraction are height-preserving admissible in $\mathcal{N}S$, that is to say: (weakening) if $\Gamma(\Delta)$ is derivable in $\mathcal{N}S$ with a derivation of height $h$, then also $\Gamma(\Delta, \Sigma)$ is derivable in $\mathcal{N}S$ with a proof of height $h' \leq h$, where $\Delta$ and $\Sigma$ are nested sequents; (contraction) given a nested sequent $\Delta$, if $\Gamma(\Delta, \Delta)$ has a derivation of height $h$, then also $\Gamma(\Delta)$ has a derivation of height $h' \leq h$. Moreover, the derivation of the contracted sequent $\Gamma(\Delta)$ does not add any rule application to the initial derivation.

Completeness is an easy consequence of the admissibility of the following rule *cut*:

$$\frac{\Gamma(F) \qquad \Gamma(\neg F)}{\Gamma(\emptyset)}\,(cut)$$

where $F$ is a formula. The standard proof of admissibility of cut proceeds by a double induction over the complexity of $F$ and the sum of the heights of the derivations of the two premises of $(cut)$, in the sense that we replace one cut by one or several cuts on formulas of smaller complexity, or on sequents derived by shorter derivations. We only show the case of systems without MP and CSO, reminding to [2] for all the other details. However, in $\mathcal{N}S$ the standard proof does not work in the following case, in which the cut formula $F$ is a conditional formula $A \Rightarrow B$:

$$\dfrac{\dfrac{(1)\ \Gamma([A : B], [A' : \Delta])}{(3)\ \Gamma(A \Rightarrow B, [A' : \Delta])}\,(\Rightarrow^+) \quad \dfrac{(2)\ \Gamma(\neg(A \Rightarrow B), [A' : \Delta, \neg B]) \quad A, \neg A' \quad A', \neg A}{\Gamma(\neg(A \Rightarrow B), [A' : \Delta])}\,(\Rightarrow^-)}{\Gamma([A' : \Delta])}\,(cut)$$

Indeed, even if we apply the inductive hypothesis on the heights of the derivations of the premises to cut $(2)$ and $(3)$, obtaining (modulo weakening, which is admissible) a derivation of $(2')$ $\Gamma([A' : \Delta, \neg B], [A' : \Delta])$, we cannot apply the inductive hypothesis on the complexity of the cut formula to $(2')$ and $(1')$ $\Gamma([A : \Delta, B], [A' : \Delta])$ (obtained from $(1)$ again by weakening). Such an application would be needed in order to obtain a derivation of $\Gamma([A' : \Delta], [A' : \Delta])$ and then to conclude $\Gamma([A' : \Delta])$ since contraction is admissible: indeed, the two contexts are different, we have $[A' : \Delta, \neg B]$ in $(2')$ whereas we have $[A : \Delta, B]$ in $(1')$.

In order to tackle this problem, we need to prove another property, namely that if $A$ and $A'$ are equivalent, if a sequent $\Gamma([A : \Delta])$ is derivable in $\mathcal{N}S$, then also $\Gamma([A' : \Delta])$, obtained by replacing $A$ with the equivalent formula $A'$, is also derivable. In turn, we need $(cut)$ to prove this property, therefore we prove both the properties (admissibility of $(cut)$ and "substitution" of $A$ with $A'$) by mutual induction, namely:

In $\mathcal{N}S$, the following propositions hold:

- *(A)* If $\Gamma(F)$ and $\Gamma(\neg F)$ are derivable, so is $\Gamma(\emptyset)$, i.e. $(cut)$ is admissible in $\mathcal{N}S$;
- *(B)* if (I) $\Gamma([A : \Delta])$, (II) $A, \neg A'$ and (III) $A', \neg A$ are derivable, then $\Gamma([A' : \Delta])$ is derivable.

Let us first consider (A). We have the following cases:

- (at least) one of the premises of $(cut)$ is an instance of the axioms. Suppose that the left premise is an instance of $(AX)$. In case it has the form $\Gamma(P, \neg P, F)$, then also $\Gamma(P, \neg P)$ is an instance of $(AX)$ and we are done. Otherwise, we have $\Gamma(F, \neg F)$. In this case, the right premise of $(cut)$ has the form $\Gamma(\neg F, \neg F)$, whereas the conclusion is $\Gamma(\neg F)$: since contraction is admissible, we conclude that $\Gamma(\neg F)$ is derivable and we are done. The other cases are symmetric. The cases in which one premise of $(cut)$ is an instance of either $(AX_\top)$ or $(AX_\bot)$ are easy and left to the reader;
- the last step of *one* of the two premises is obtained by a rule (**R**) in which $F$ is *not* the principal formula. This case is standard, we can permute (**R**) over the cut, i.e. we cut the premise(s) of (**R**) and then we apply (**R**) to the result of cut.
- $F$ is the principal formula in the last step of *both* derivations of the premises of the cut inference. There are several subcases, to save space we only consider the above mentioned case in which the standard proof does not work, where the derivation is as follows:

$$\dfrac{\dfrac{(1)\ \Gamma(\neg(A \Rightarrow B), [A' : \Delta, \neg B]) \quad A, \neg A' \quad A', \neg A}{\Gamma(\neg(A \Rightarrow B), [A' : \Delta])}\,(\Rightarrow^-) \qquad \dfrac{(2)\ \Gamma([A : B], [A' : \Delta])}{(3)\ \Gamma(A \Rightarrow B, [A' : \Delta])}\,(\Rightarrow^+)}{\Gamma([A' : \Delta])}\,(cut)$$

First of all, since we have proofs for $A, \neg A'$ and for $A', \neg A$ and the complexity of $A$ is lower than the one of $A \Rightarrow B$, we can apply the inductive hypothesis for (B) to $(2)$, obtaining a derivation of $(2')$ $\Gamma([A' : B], [A' : \Delta])$. Since weakening is admissible, from $(3)$ we obtain a derivation of at most the same height of $(3')$ $\Gamma(A \Rightarrow B, [A' : \Delta, \neg B])$. We can then conclude as follows: we first apply the inductive hypothesis on the height for (A) to cut $(1)$ and $(3')$, obtaining a derivation of $(4)$ $\Gamma([A' : \Delta, \neg B])$. By weakening, we have also a derivation of $(4')$ $\Gamma([A' : \Delta, \neg B], [A' : \Delta])$. Again by weakening, from $(2')$ we obtain a derivation of $(2'')$ $\Gamma([A' : \Delta, B], [A' : \Delta])$. We then apply the inductive hypothesis on the complexity of the cut formula to cut $(2'')$ and $(4')$, obtaining a derivation of $\Gamma([A' : \Delta], [A' : \Delta])$, from which we conclude since contraction is admissible.

Concerning (B), we proceed by induction on the height $h$ of the premise (I), as follows:

- Base case: $\Gamma([A : \Delta])$ is an instance of the axioms, that is to say either $\Delta = \Lambda(P, \neg P)$ or $\Delta = \Lambda(\top)$ or $\Delta = \Lambda(\neg\bot)$: we immediately conclude that also $\Gamma([A' : \Delta])$ is an instance of the axioms, and we are done;
- Inductive step: we have to consider all possible rules ending (looking forward) the derivation of $\Gamma([A : \Delta])$. We only show the most interesting case, when $(\Rightarrow^-)$ is applied by using $[A : \Delta]$ as principal formula. The derivation ends as follows:

$$\frac{(1)\ \Gamma(\neg(C \Rightarrow D), [A : \Delta, \neg D]) \qquad (2)\ C, \neg A \qquad (3)\ A, \neg C}{\Gamma(\neg(C \Rightarrow D), [A : \Delta])}\ (\Rightarrow^-)$$

We can apply the inductive hypothesis to $(1)$ to obtain a derivation of $(1')\ \Gamma(\neg(C \Rightarrow D), [A' : \Delta, \neg D])$. Since weakening is admissible, from $(II)$ we obtain a derivation of $(II')\ C, A, \neg A'$, from $(III)$ we obtain a derivation of $(III')\ A', \neg A, \neg C$. Again by weakening, from $(2)$ and $(3)$ we obtain derivations of $(2')\ C, \neg A, \neg A'$ and $(3')\ A', A, \neg C$, respectively. We apply the inductive hypothesis of (A) that is that cut holds for the formula $A$ (of a given complexity $c$) and conclude as follows:

$$\frac{(1')\ \Gamma(\neg(C \Rightarrow D), [A' : \Delta, \neg D]) \quad \dfrac{(II')\ C, A, \neg A' \quad (2')\ C, \neg A, \neg A'}{C, \neg A'}\ (cut) \quad \dfrac{(III')\ A', \neg A, \neg C \quad (3')\ A', A, \neg C}{A', \neg C}\ (cut)}{\Gamma(\neg(C \Rightarrow D), [A' : \Delta])}\ (\Rightarrow^-)$$

With the admissibility of cut at hand, we can easily conclude the proof of completeness of $\mathcal{N}S$ by showing that the axioms are derivable and that the set of derivable formulas is closed under (Modus Ponens), (RCEA), and (RCK). A derivation of an instance of ID has been shown in Figure 2. A derivation of an instance of MP is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A \Rightarrow B), \neg A, B, A}\ (AX) \quad \dfrac{}{\neg(A \Rightarrow B), \neg A, B, \neg B}\ (AX)}{}\ (MP)}{\neg(A \Rightarrow B), \neg A, B}\ (\rightarrow^+)}{\neg(A \Rightarrow B), A \rightarrow B}\ (\rightarrow^+)}{(A \Rightarrow B) \rightarrow (A \rightarrow B)}$$

Here is a derivation of an instance of CEM:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[A : B, \neg B], [A : \neg B]}\ (AX) \quad \dfrac{}{A, \neg A}\ (AX) \quad \dfrac{}{\neg A, A}\ (AX)}{}\ (CEM)}{[A : B], [A : \neg B]}\ (\Rightarrow^+)}{[A : B], A \Rightarrow \neg B}\ (\Rightarrow^+)}{A \Rightarrow B, A \Rightarrow \neg B}\ (\vee^+)}{(A \Rightarrow B) \vee (A \Rightarrow \neg B)}$$

For (Modus Ponens), we have to show that, if $(1)\ A \rightarrow B$ ad $(2)\ A$ are derivable, then also $B$ is derivable. Since weakening is admissible, we have also derivations for $(1')\ A \rightarrow B, B, \neg A$ and $(2')\ A, B$. Furthermore, observe that $(3)\ A, B, \neg A$ and

$(4)$ $\neg B, B, \neg A$ are both instances of $(AX)$. Since cut is admissible (statement A above), the following derivation shows that $B$ is derivable:

$$\dfrac{(2')\ A, B \qquad \dfrac{(1')\ A \to B, B, \neg A \qquad \dfrac{(3)\ A, B, \neg A \qquad (4)\ \neg B, B, \neg A}{\neg(A \to B), B, \neg A}(\to^-)}{B, \neg A}(cut)}{B}(cut)$$

For (RCEA), we have to show that if $A \leftrightarrow B$ is derivable, then also $(A \Rightarrow C) \leftrightarrow (B \Rightarrow C)$ is derivable. As usual, $A \leftrightarrow B$ is an abbreviation for $(A \to B) \wedge (B \to A)$. Since $A \leftrightarrow B$ is derivable, and since $(\wedge^+)$ and $(\to^+)$ are invertible, we have a derivation for $A \to B$, then for $(1)$ $\neg A, B$, and for $B \to A$, then for $(2)$ $A, \neg B$. We derive $(A \Rightarrow C) \to (B \Rightarrow C)$ (the other half is symmetric) as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(A \Rightarrow C), [B : C, \neg C]}(AX) \quad (1)\ \neg A, B \qquad (2)\ A, \neg B}{\neg(A \Rightarrow C), [B : C]}(\Rightarrow^-)}{\neg(A \Rightarrow C), B \Rightarrow C}(\Rightarrow^+)}{(A \Rightarrow C) \to (B \Rightarrow C)}(\to^+)$$

For (RCK), suppose that we have a derivation in $\mathcal{N}S$ of $(A_1 \wedge \ldots \wedge A_n) \to B$. Since $(\to^+)$ is invertible, we have also a derivation of $B, \neg(A_1 \wedge \ldots \wedge A_n)$. Since $(\wedge^-)$ is also invertible, then we have a derivation of $B, \neg A_1, \ldots, \neg A_n$ and, by weakening, of $(1)$ $\neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B, \neg A_1, \neg A_2, \ldots, \neg A_n]$, from which we conclude as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(1)\ \neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B, \neg A_1, \neg A_2, \ldots, \neg A_n]}{\vdots}}{\neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B, \neg A_1, \neg A_2]}(\Rightarrow^-) \quad \dfrac{}{C, \neg C}(AX) \quad \dfrac{}{\neg C, C}(AX)}{\neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B, \neg A_1]}(\Rightarrow^-) \quad \dfrac{}{C, \neg C}(AX) \quad \dfrac{}{\neg C, C}(AX)}{\neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B]}(\Rightarrow^-)}{\neg(C \Rightarrow A_1), \ldots, \neg(C \Rightarrow A_n), [C : B]}(\wedge^-)}{\neg(C \Rightarrow A_1 \wedge \ldots \wedge C \Rightarrow A_n), [C : B]}(\Rightarrow^+)}{\neg(C \Rightarrow A_1 \wedge \ldots \wedge C \Rightarrow A_n), C \Rightarrow B}(\to^+)}{(C \Rightarrow A_1 \wedge \ldots \wedge C \Rightarrow A_n) \to (C \Rightarrow B)}(\to^+)$$

■

As usual, in order to obtain a decision procedure for the conditional logics under consideration, we have to control the application of the rules $(\Rightarrow^-)$/$(CSO)$, $(MP)$, $(CEM)$, and $(ID)$ that otherwise may be applied infinitely often in a backward proof search, since their principal formula is copied into the respective premise(s). In detail, we obtain a sound, complete and terminating calculus if we restrict the applications of these rules as follows [1, 2]: $(\Rightarrow^-)$ can be applied only once to each formula $\neg(A \Rightarrow B)$ with a

context $[A' : \Delta]$ in each branch - the same for $(CSO)$ in the system CK+CSO+ID; $(ID)$ can be applied only once to each context $[A : \Delta]$ in each branch; $(MP)$ can be applied only once to each formula $\neg(A \Rightarrow B)$ in each branch. For systems with $(CEM)$, we need a more complicated mechanism: due to space limitations, we refer to [1] for this case. These results give a PSPACE decision procedure for their respective logics.

## 4 Design of NESCOND

In this section we present a Prolog implementation of the nested sequent calculi $\mathcal{NS}$. The program, called NESCOND (NESted sequent calculi for CONDitional logics), is inspired by the "lean" methodology of leanTAP, even if it does not follow its style in a rigorous manner. The program comprises a set of clauses, each one of which implements a sequent rule or axiom of $\mathcal{NS}$. The proof search is provided for free by the mere depth-first search mechanism of Prolog, without any additional ad hoc mechanism.

NESCOND represents a nested sequent with a Prolog list of the form:

```
[F_1, F_2, ..., F_m,[[A_1,Gamma_1],AppliedConditionals_1],
[[A_2,Gamma_2],AppliedConditionals_2], ..., [[A_n,Gamma_n],AppliedConditionals_n]] ]
```

In detail, elements of a nested sequent can be either formulas `F` or contexts. A context is represented by a pair `[Context,AppliedConditionals]` where:

- `Context` is also a pair of the form `[F,Gamma]`, where `F` is a formula of $\mathcal{L}$ and `Gamma` is a Prolog list representing a nested sequent;
- `AppliedConditionals` is a Prolog list `[A_1=>B_1,A_2=>B_2,...,A_k=>B_k]`, keeping track of the negated conditionals to which the rule $(\Rightarrow^-)$ has been already applied by using `Context` in the current branch. This is used in order to implement the restriction on the application of the rule $(\Rightarrow^-)$ in order to ensure termination.

Symbols $\top$ and $\bot$ are represented by constants `true` and `false`, respectively, whereas connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\Rightarrow$ are represented by `!`, `^`, `v`, `->`, and `=>`.

As an example, the Prolog list

```
[p, q, !(p => q), [[p, [q v !p, [[p,[p => r]],[]], !r]],[p => q]], [[q, [p, !p]],[]]]
```

represents the nested sequent

$$P, Q, \neg(P \Rightarrow Q), [P : Q \vee \neg P, [P : P \Rightarrow R], \neg R], [Q : P, \neg P].$$

Furthermore, the list `[p => q]` in the leftmost context is used to represent the fact that, in a backward proof search, the rule $(\Rightarrow^-)$ has already been applied to $\neg(P \Rightarrow Q)$ by using $[P : Q \vee \neg P, [P : P \Rightarrow R], \neg R]$.

Let us now discuss more in detail the implementation of NESCOND, starting from the description of some auxiliary predicates, and then distinguishing among the different conditional logics considered.

### 4.1 Auxiliary predicates

In order to manipulate formulas "inside" a sequent, NESCOND makes use of the three following auxiliary predicates:

- `deepMember(+Formulas,+NS)` succeeds if and only if either (i) the nested sequent `NS` representing a nested sequent $\Gamma$ contains all the fomulas in the list `Formulas` or (ii) there exists a context `[[A,Delta],AppliedConditionals]` in `NS` such that `deepMember(Formulas,Delta)` succeeds, that is to say there is a nested sequent occurring in `NS` containing all the formulas of `Formulas`.
- `deepSelect(+Formulas,+NS,-NewNS)` operates exactly as `deepMember`, however it removes the formulas of the list `Formulas` by replacing them with a placeholder `hole`; the output term `NewNS` matches the resulting sequent.
- `fillTheHole(+NS,+Formulas,-NewNS)` replaces the placeholder `hole` in `NS` with the formulas in the list `Formulas`. The resulting sequent matches the output term `NewNS`.

### 4.2 NESCOND for CK

The calculi $\mathcal{NS}$ are implemented by the predicate

```
prove(+NS,-ProofTree).
```

This predicate succeeds if and only if the nested sequent represented by the list `NS` is derivable. When the predicate succeeds, then the output term `ProofTree` matches with a representation of the derivation found by the prover, used in order to display the proof tree. For instance, in order to prove that the formula $(A \Rightarrow (B \wedge C)) \rightarrow (A \Rightarrow B)$ is valid in CK, one queries NESCOND with the goal: `prove([(a => b ^ c) -> (a => b)],ProofTree).` Each clause of the `prove` predicate implements an axiom or rule of $\mathcal{NS}$. To search a derivation of a nested sequent $\Gamma$, NESCOND proceeds as follows. First of all, if $\Gamma$ is an axiom, the goal will succeed immediately by using one of the following clauses for the axioms:

```
prove(NS,tree(ax)):-deepMember([P,!P],NS),!.
prove(NS,tree(axt)):-deepMember([top],NS),!.
prove(NS,tree(axb)):-deepMember([!bot],NS),!.
```

implementing $(AX)$, $(AX_{\top})$ and $(AX_{\bot})$, respectively. If $\Gamma$ is not an instance of the axioms, then the first applicable rule will be chosen, e.g. if a nested sequent in $\Gamma$ contains a formula `A v B` then the clause implementing the $(\vee^{+})$ rule will be chosen, and NESCOND will be recursively invoked on the unique premise of $(\vee^{+})$. NESCOND proceeds in a similar way for the other rules. The ordering of the clauses is such that the application of the branching rules is postponed as much as possible.

As an example, the clause implementing $(\Rightarrow^{-})$ is as follows:

```
1.  prove(NS,tree(condn,A,B,Sub1,Sub2,Sub3)):-
2.     deepSelect([!(A => B),[[C,Delta],AppliedConditionals]],
           NS,NewNS),
```

```
3.      \+member(!(A => B),AppliedConditionals),!,
4.      prove([A,!C],Sub2),
5.      prove([C,!A],Sub3),
6.      fillTheHole(NewNS,[!(A => B),
            [[C,[!B|Delta]],[!(A => B)|AppliedConditionals]]],DefNS),
7.      prove(DefNS,Sub1).
```

In line 2, the auxiliary predicate `deepSelect` is invoked in order to find both a negated conditional $\neg(A \Rightarrow B)$ and a context $[C : \Delta]$ in the sequent (even in a nested subsequent). In this case, such formulas are replaced by the placeholder `hole`. Line 3 implements the restriction on the application of $(\Rightarrow^{-})$ in order to guarantee termination: the rule is applied only if $\neg(A \Rightarrow B)$ does not belong to the list `AppliedConditionals` of the selected context. Since the rules of $\mathcal{NS}$ are invertible[4], a cut `!` is introduced in order to avoid useless backtrackings in the choice of the rule to apply. In lines 4, 5 and 7, NESCOND is recursively invoked on the three premises of the rule. In line 7, NESCOND is invoked on the premise in which the context $[C : \Delta]$ is replaced by $[C : \Delta, \neg B]$. To this aim, in line 6 the auxiliary predicate `fillTheHole(+NewNS,+Formulas,-DefNS)` is invoked to replace the `hole` in `NewNS`, introduced by `deepSelect`, with the negated conditional $\neg(A \Rightarrow B)$, which is copied into the premise, and the context $[C : \Delta, \neg B]$, whose list of `AppliedConditionals` is updated by adding the formula $\neg(A \Rightarrow B)$ itself.

### 4.3 NESCOND for extensions of CK

The implementation of the calculi for extensions of CK with axioms ID and MP are very similar. For systems allowing ID, contexts are triples `[Context, AppliedConditionals, AllowID]`. The third element `AllowID` is a flag used in order to implement the restriction on the application of the rule $(ID)$, namely the rule is applied to a context only if `AllowID=true`:

```
prove(NS,tree(id,A,SubTree)):-
   deepSelect([[[A,Delta],AppliedConditionals,true]]],NS,NewNS),!,
   fillTheHole(NewNS,[[[A,[!A|Delta]],AppliedConditionals,false]]],DefNS),
   prove(DefNS,SubTree).
```

When $(ID)$ is applied to `[Context, AppliedConditionals, true]` then the predicate `prove` is invoked on the unique premise of the rule `DefNS`, and the flag is set to `false` in order to avoid multiple applications in a backward proof search.

The restriction on the application of the rule $(MP)$ is implemented by equipping the predicate `prove` by a third argument, `AppliedMP`, a Prolog list keeping track of the negated conditionals to which the rule has already been applied in the current branch. The clause of `prove` implementing $(MP)$ is as follows:

[4] The rule $(\Rightarrow^{-})$, as well as $(CEM)$, are only "weakly" invertible, in the sense that only the leftmost premise of these rules can be obtained by weakening from the respective conclusions (see [2] for details).

```
1.  prove(NS,AppliedMP,tree(mp,A,B,Sub1,Sub2)):-
2.     deepSelect([!(A => B)],NS,NewNS),
3.     \+member(A => B,AppliedMP),!,
4.     fillTheHole(NewNS,[A,!(A => B)],NS1),
5.     fillTheHole(NewNS,[!B,!(A => B)],NS2),
6.     prove(NS1,[A => B|AppliedMP],Sub1),
7.     prove(NS2,[A => B|AppliedMP],Sub2).
```

The rule is applicable to a formula $\neg(A \Rightarrow B)$ only if `[A => B]` does not belong to `AppliedMP` (line 3). When $(MP)$ is applied, then `[A => B]` is added to `AppliedMP` in the recursive calls of `prove` on the premises of the rule (lines 6 and 7).

The implemetation of the calculus for the flat fragment of CK+CSO+ID, corresponding to KLM cumulative logic **C**, is very similar to the one for CK+ID; the only difference is that the rule $(\Rightarrow^{-})$ is replaced by $(CSO)$. This does not make use of the predicate `deepSelect` to "look inside" a sequent to find the principal formulas $\neg(A \Rightarrow B)$ and $[C : \Delta]$: since the calculus only deals with the *flat* fragment of the logic under consideration, such principal formulas are directly selected from the current sequent by easy membership tests (standard Prolog predicates `member` and `select`), without searching inside other contexts.

As mentioned, NESCOND also deals with extensions with CEM; the clause implementing the rule $(CEM)$ is as follows:

```
1. prove(NS,tree(cem,A,B,Sub1,Sub2,Sub3)):-
2.    deepSelect([[[A,Delta],ApplCond1],[[B,Sigma],ApplCond2]],
                                                  NS,NewNS),
3.    notSequentIncluded(Delta,Sigma),!,
4.    prove([A,!B],Sub2),
5.    prove([B,!A],Sub3),
6.    append(Delta,Sigma,ResDelta),
7.    fillTheHole(NewNS,[[[A,ResDelta],ApplCond1],
                         [[B,Sigma],ApplCond2]],DefNS),
8.   prove(DefNS,Sub1).
```

In line 3 the predicate `notSequentIncluded` is invoked: this predicate implements the restriction on the application of the rule described in the previous section in order to ensure termination.

## 5 Performances of NESCOND

The performances of NESCOND are promising. We have tested it by running SICStus Prolog 4.0.2 on an Apple MacBook Pro, 3.06 GHz Intel Core 2 Duo, 4GB RAM machine. We have performed two kind of tests:

– we have compared the performances of NESCOND for CK with the ones of two other provers for conditional logics, namely CondLean, implementing labelled sequent calculi [17, 18], and the goal-directed procedure GOALD$\mathcal{U}$CK [19]. To this aim, we have tested the three theorem provers on randomly generated sequents, obtaining the

results shown in Figure 3. It is easy to conclude that the performances of NESCOND are better in all cases: in particular, concerning sequents containing formulas with a high level of conditional nesting (15), with a 1ms time limit, GOALD$\mathcal{U}$CK is not able to answer in the 41% of cases, whereas CondLean finds all the valid sequents but it is not able to find *any* non-valid ones. NESCOND answers correctly for the 100% of the generated formulas, discovering that all the missing answers of CondLean are non-valid ones;

– we have tested NESCOND over a set of 88 CK valid formulas obtained by translating K valid formulas[5] provided by Heuerding and used to test the theorem prover ModLeanTAP [5] by Beckert and Goré. Also in this case, the results, shown in Figure 4, are encouraging: NESCOND is not able to give an answer in less than 1 ms only in 16 cases over 88; the number of timeouts drops to 7 if we extend the time limit to 5 seconds.

| number of formulas : 30 | Prop. vars: 5 | Depth: 3 | Timeout: 1 ms |
|---|---|---|---|
| | yes | no | timeout |
| UCK | 73,00% | 21,00% | 6,00% |
| CondLean | 74,00% | 0,00% | 26,00% |
| Nested sequents | 74,00% | 26,00% | 0,00% |

| number of formulas: 10 | Prop. vars: 3 | Depth: 15 | Timeout: 1ms |
|---|---|---|---|
| | yes | no | timeout |
| UCK | 55,00% | 4,00% | 41,00% |
| CondLean | 69,00% | 0,00% | 31,00% |
| Nested sequents | 69,00% | 31,00% | 0,00% |

**Fig. 3.** NESCOND vs CondLean vs GOALD$\mathcal{U}$CK

| Time limit (ms) | 1 | 100 | 1000 | 5000 | 30000 |
|---|---|---|---|---|---|
| NESCOND | 16 | 13 | 10 | 7 | 5 |
| CondLean | 22 | 16 | 14 | 10 | 9 |

**Fig. 4.** Number of timeouts of NESCOND and CondLean over 88 CK valid formulas.

| Time limit (ms) | 1 | 50 | 100 | 5000 |
|---|---|---|---|---|
| Percentage of timeouts | 30% | 28% | 27% | 20% |

**Fig. 5.** Percentage of timeouts of NESCOND for extensions of CK.

These results show that the performances of NESCOND are encouraging, probably better than the ones of the other existing provers for conditional logics (we are currently completing the comparative statistics). Figure 5 shows that this also holds for extensions of CK: in the 70% of the tests (all of them are valid formulas), NESCOND gives an answer in less than 1ms. The remaining 30% concerns the case of CEM(+ID), where the performances worsen because of the overhead of the termination mechanism. We note in passim that it does not exist a set of acknowledged benchmarks for conditional logics. We are currently building a set of meaningful tests, they can be found at http://www.di.unito.it/∼pozzato/nescond/.

## 6 Conclusions and Future Issues

In this work we have presented NESCOND, a theorem prover for conditional logics implementing nested sequent calculi introduced in [1]. The performances described in the previous section show that nested sequent calculi do not only provide elegant and

[5] $\Box A$ is replaced by $\top \Rightarrow A$, whereas $\Diamond A$ is replaced by $\neg(\top \Rightarrow \neg A)$.

natural calculi for conditional logics, but they are also significant for developing efficient theorem provers for them. In future research we aim to extend NESCOND to other systems of conditional logics. To this regard, we strongly conjecture that adding a rule for (CS) will be enough to cover the whole cube of the extensions of CK generated by axioms (ID), (MP), (CEM) and (CS). This will be object of subsequent research.

## References


1. Alenda, R., Olivetti, N., Pozzato, G.L.: Nested Sequent Calculi for Conditional Logics. In: Luis Fariñas del Cerro, Andreas Herzig, J.M. (ed.) Logics in Artificial Intelligence - 13th European Conference, JELIA 2012. Lecture Notes in Artificial Intelligence (LNAI), vol. 7519, pp. 14–27. Springer-Verlag, Toulouse, France (Sptember 2012)
2. Alenda, R., Olivetti, N., Pozzato, G.L.: Nested Sequents Calculi for Normal Conditional Logics. Journal of Logic and Computation to appear, 1–48 (2013)
3. Baltag, A., Smets, S.: The logic of conditional doxastic actions. Texts in Logic and Games, Special Issue on New Perspectives on Games and Interaction 4, 9–31 (2008)
4. Beckert, B., Posegga, J.: lean$T^AP$: Lean tableau-based deduction. Journal of Automated Reasoning 15(3), 339–358 (1995)
5. Beckert, B., Goré, R.: Free variable tableaux for propositional modal logics. In: Proceedings of Tableaux 97. LNCS, vol. 1227, pp. 91–106. Springer, Pont-a-Mousson, France (1997)
6. Board, O.: Dynamic interactive epistemology. Games and Economic Behavior 49(1), 49–80 (2004)
7. Boutilier, C.: Conditional logics of normality: a modal approach. Artificial Intelligence 68(1), 87–154 (1994)
8. Genovese, V., Giordano, L., Gliozzi, V., Pozzato, G.L.: A conditional constructive logic for access control and its sequent calculus. In: Brünnler, K., Metcalfe, G. (eds.) Tableaux 2011. Lecture Notes in Artificial Intelligence (LNAI), vol. 6793, pp. 164–179. Springer-Verlag, Bern, Switzerland (July 2011)
9. Genovese, V., Giordano, L., Gliozzi, V., Pozzato, G.L.: Logics in access control: A conditional approach. Journal of Logic and Computation p. to appear (2012)
10. Ginsberg, M.L.: Counterfactuals. Artificial Intelligence 30(1), 35–79 (1986)
11. Giordano, L., Gliozzi, V., Olivetti, N.: Weak AGM postulates and strong ramsey test: A logical formalization. Artificial Intelligence 168(1-2), 1–37 (2005)
12. Grahne, G.: Updates and counterfactuals. Journal of Logic and Computation 8(1), 87–117 (1998)
13. Kashima, R.: Cut-free sequent calculi for some tense logics. Studia Logica 53(1), 119–136 (1994)
14. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
15. Lewis, D.: Counterfactuals. Basil Blackwell Ltd (1973)
16. Nute, D.: Topics in conditional logic. Reidel, Dordrecht (1980)
17. Olivetti, N., Pozzato, G.L.: CondLean: A Theorem Prover for Conditional Logics. In: Cialdea Meyer, M., Pirri, F. (eds.) Tableaux 2003. LNAI, vol. 2796, pp. 264–270. Springer, Roma, Italy (September 2003)
18. Olivetti, N., Pozzato, G.L.: CondLean 3.0: Improving Condlean for Stronger Conditional Logics. In: Beckert, B. (ed.) Proceedings of Tableaux 2005. LNAI, vol. 3702, pp. 328–332. Springer-Verlag, Koblenz, Germany (September 2005)
19. Olivetti, N., Pozzato, G.L.: Theorem Proving for Conditional Logics: CondLean and GoalDuck. Journal of Applied Non-Classical Logics (JANCL) 18(4), 427–473 (2008)
20. Olivetti, N., Pozzato, G.L., Schwind, C.B.: A Sequent Calculus and a Theorem Prover for Standard Conditional Logics. ACM Transactions on Computational Logic (ToCL) 8(4) (2007)

# Focusing on contraction

Alessandro Avellone[1], Camillo Fiorentini[2], Alberto Momigliano[2]

[1] DISMEQ, Università degli Studi di Milano-Bicocca
[2] DI, Università degli Studi di Milano

**Abstract.** Focusing [1] is a proof-theoretic device to structure proof search in the sequent calculus: it provides a normal form to cut-free proofs in which the application of invertible and non-invertible inference rules is structured in two separate and disjoint phases. It is commonly believed that every "reasonable" sequent calculus has a natural focused version. Although stemming from proof-search considerations, focusing has not been thoroughly investigated in actual theorem proving, in particular w.r.t. termination, if not for the folk observations that only negative formulas need to be duplicated (or contracted if seen from the top down) in the focusing phase. We present a contraction-free (and hence terminating) focused proof system for multi-succedent propositional intuitionistic logic, which refines the **G4ip** calculus of Vorob'ev, Hudelmeier and Dyckhoff. We prove the completeness of the approach semantically and argue that this offers a viable alternative to other more syntactical means.

## 1 Introduction and related work

Focusing [1] is a proof-theoretic device to structure proof search in the sequent calculus: it provides a normal form to cut-free proofs in which the application of invertible and non-invertible inference rules is structured in two separate and disjoint phases. In the first, called the negative or asynchronous phase, we apply (reading the proof bottom up) all invertible inference rules in whatever order, until none is left. The second phase, called the positive or synchronous phase, "focuses" on a formula, by selecting a not necessarily invertible inference rule. If after the (reverse) application of that introduction rule, a sub-formula of that focused formula appears that also requires a non-invertible inference rule, then the phase continues with that sub-formula as the new focus. The phase ends either with success or when only formulas with invertible inference rules are encountered and phase one is re-entered. Certain "structural" rules are used to recognize this switch. Compare this to standard presentation of proof search, such as [22], where Waaler and Wallen describe a search strategy for the intuitionistic multi-succedent calculus **LB** by dividing rules in groups to be applied following some priorities and a set of additional constraints. This without a proof of completeness. Focusing internalizes in the proof-theory a stringent strategy, and a provably complete one, from which many additional optimizations follow.

Contraction (or duplication, seen from the bottom up) is one of Gentzen's original structural rules permitting the reuse of some formula in the antecedent or succedent of a sequent:

$$\frac{\Gamma, A, A \vdash \Delta}{\Gamma, A \vdash \Delta}\ \text{Contr } L \qquad \frac{\Gamma \vdash A, A, \Delta}{\Gamma \vdash A, \Delta}\ \text{Contr } R$$

We are interested in proof search for propositional logics and from this standpoint contraction is a rather worrisome rule: it can be applied at any time making termination problematic even for decidable logics, thus forcing the use of potentially expensive and non-logical methods like loop detection. It is therefore valuable to ask whether contraction can be removed, in particular in the context of focused proofs.

As it emerged from linear logic, focusing naturally fits other logics with strong dualities, such as classical logic. As such, it is maybe not surprising that issue of contraction has not been fully investigated: in linear logic contraction (and weakening) are tagged by exponentials, while in classical logic duplication does not affect completeness. As far as intuitionistic logic, an important corollary of the completeness of focusing is that contraction is exactly located in between the asynchronous and synchronous phases and can be restricted to negative formulas[3]. This is a beginning, but it is well-known (see the system **G3ip** [21]) that the only propositional connective we *do* need to contract is *implication*.

There is a further element: Gentzen's presentation of intuitionistic logic is obtained from his classical system **LK** by means of a cardinality restriction imposed on the succedent of every sequent: at most one formula occurrence. This has been generalized by Maehara (see [15]), who retained a multiple-conclusion version, provided that the rules for right implication (and universal quantification) can only be performed if there is a single formula in the succedent of the premise to which these rules are applied. As these are the same connectives where in the Kripke semantics a world jump is required, this historically opened up a fecund link with tableaux systems. Moreover, Maehara's **LB** (following [22]'s terminology) has more symmetries from the permutation point of view and therefore may seem a better candidate for focusing than mono-succedent **LJ**. The two crucial rules are:

$$\frac{\Gamma, A \to B \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \to L \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B, \Delta} \to R$$

Interestingly here, in opposition to **LJ**, the $\to L$ rule *is* invertible, while $\to R$ is not. According to the focusing diktat, $\to L$ would be classified as left asynchronous and eagerly applied, and this makes the asynchronous phase endless. While techniques such as freezing [4] or some form of loop checking could be used, we exploit a well-known formulation of a contraction-free calculus, known as **G4ip** [21], following Vorob'ev, Hudelmeier and Dyckhoff, where the $\to L$ rule is replaced by a series of rules that originate from the analysis of the shape of

[3] Recall that in **LJ** a formula is negative (positive) if its right introduction rule is invertible (non-invertible).

the subformula $A$ of the main formula $A \to B$ of the rule. It is then routine that such a system is indeed terminating, in the sense that any bottom-up derivation of any given sequent is of finite length[4]. It is instead *not* routine to focalize such a system, called **G4ipf**, and this is the main result of the present paper.

As the focusing strategy severely restricts proofs construction, it is paramount to show that we do not lose any proof – in other terms that focusing is complete w.r.t. standard intuitionistic logic. There are in the literature several ways to prove that, all of them proof-theoretical and none of them completely satisfactory for our purposes:

1. The permutation-based approach, dating back to Andreoli [1], works by proving inversion properties of asynchronous connectives and postponement properties of synchronous ones. This is very brittle and particularly problematic for contraction-free calculi: in fact, it requires to prove at the same time that contraction is admissible and in the focusing setting this is far from trivial.
2. One can establish admissibility of the *cut* and of the non-atomic *initial* rule in the focused calculus and then show that all ordinary rules are admissible in the latter using cut. This has been championed in [8]. While a syntactic proof of cut-elimination is an interesting result per se, the sheer number of the judgments involved and hence of the cut reductions (principal, focus, blur, commutative and preserving cuts in the terminology of the cited paper) makes the well founded-ness of the inductive argument very delicate and hard to extend.
3. The so-called "grand-tour through linear logic" strategy of Miller and Liang [14]. Here, to show that a refinement of an intuitionistic proof system such as ours is complete, we have to provide an embedding into **LLF** (the canonical focused system for full linear logic) and then show that the latter translation is entailed by Miller and Liang's 1/0 translation. The trouble here is that contraction-free systems cannot be faithfully encoded in **LLF** [18]. While there are refinements of **LLF**, namely linear logic with sub-exponentials [20], which may be able to faithfully encode such systems, a "grand-tour" strategy in this context is uncharted territory. Furthermore, sub-exponential encodings of focused systems tend to be very, very prolix, which makes closing the grand-tour rather unlikely.
4. Finally, Miller and Saurin propose a direct proof of completeness of focusing in linear logic in [19] based on the notion of focalization graph. Again, this seems hard to extend to asymmetric calculi such as intutionism, let alone those contraction-free.

In this paper, instead, we prove completeness adapting the traditional Kripke semantic argument. While this is well-worn in tableaux-like systems, it is the first time that the model-theoretic semantics of focusing has been considered. The highlights of our proof are explained in Section 3.3.

[4] With some additional effort, one can prove that contraction is admissible in the contraction-free calculus [10].

Although stemming from proof-search considerations, focusing has still to make an impact in actual theorem proving. Exceptions are:

- Inverse-based systems such as Imogen [16] and LIFF [7]: because the inverse method is forward and saturation-based, the issue of contraction does not come into play – in fact it exhibits different issues w.r.t. termination (namely subsumption) and is in general not geared towards finite failure.
- TAC [5] is a prototype of a family of focused systems for automated inductive theorem proving, including one for **LJF**. Because the emphasis is on the automation of inductive proofs and the objective is to either succeed or quickly fail, most care is applied to limit the application of the induction rule by means of *freezing*. Contraction is handled heuristically, by letting the user set a bound for how many time an assumption can be duplicated for each initial goal; once the bound is reached, the system becomes essentially linear.
- Henriksen's [13] presents an analysis of contraction-free classical logic: here contraction has an impact only in the presence of two kinds of disjunction/conjunctions, namely positive vs. negative, as in linear logic. The author shows that contraction can be disposed of by viewing the introduction rule for positive disjunction as a *restart* rule, similar to Gabbay's [12]:

$$\frac{\vdash \Theta, pos(A) \Downarrow B}{\vdash \Theta \Downarrow A \vee^{+} B} \qquad \text{plus dual}$$

  where $pos(A) = A \wedge^{+} t^{+}$ delays the non-chosen branch if $A$ is negative ($\Theta$ is positive only), and the focus left rule does not make any contraction. This is neat, but not helpful as far as **LB** is concerned.

## 2 The proof system

We consider a standard propositional language based on a denumerable set of atoms, the constant $\bot$ and the connectives $\wedge$, $\vee$ and $\rightarrow$; $\neg A$ stands for $A \rightarrow \bot$. Our aim is to give a focalized version of the well-known contraction-free calculus **G4ip** of Vorob'ev, Hudelmeier and Dyckhoff [21]. To this end, one starts with a classification of formulas in the (a)synchronous categories. In focused versions of **LJ** such as **LJF** [14], an asynchronous formula has a right invertible rule and a non-invertible left one – and dually for synchronous. The contraction-free approach does not enjoy this symmetry – the idea is in fact to consider the possible shape that the antecedent of an implication can have and provide a specialized left (and here right[5]) introduction rule, yielding a finer view of implicational connectives, which now come in pairs. As we shall see shortly, formulas of the kind $(A \rightarrow B) \rightarrow C$ have non-invertible left and right rules, while the intro rules for $(A \wedge B) \rightarrow C$ and $(A \vee B) \rightarrow C$ are *both* invertible. Formulas

[5] And in this sense our calculus is reminiscent of Avron's decomposition proof systems [3].

$a \to B$, with $a$ an atom, have a peculiar behaviour: right rule is non-invertible, left rule is invertible, but can be applied only if the left context contains the atom $a$. This motivates the following, slight unusual, classification of formulas – we discuss the issue of *polarization* of atoms in Section 4.

$$\begin{array}{rcll} \text{Async Formula (AF)} & ::= & \bot \mid A \wedge B \mid A \vee B \mid \bot \to B \mid (A \wedge B) \to C \mid (A \vee B) \to C & \\ \text{Sync Formula (SF)} & ::= & a \mid a \to B \mid (A \to B) \to C & \text{where } a \text{ is an atom} \\ \mathrm{AF}^{+} & ::= & a \mid \mathrm{AF} & \\ \mathrm{SF}^{-} & ::= & \text{a non-atomic SF} & \end{array}$$

The calculus is based on the following judgments, whose rules are displayed in Figure 1:

- $\Theta; \Gamma \Longrightarrow \Delta; \Psi$. Active sequent;
- $\Theta; A \ll \Psi$. Left-focused sequent;
- $\Theta \gg A; \Psi$. Right-focused sequent.

$\Gamma$ and $\Delta$ denote multisets of formulas, while $\Theta$ and $\Psi$ denote multisets of SF. We use the standard notation of [21]; for instance, by $\Gamma, \Delta$ we mean multiset union of $\Gamma$ and $\Delta$.

Proof search alternates between an *asynchronous phase*, where asynchronous formulas are considered, and a *synchronous phase*, where synchronous ones are. The dotted lines highlights the rule that govern the phase change. In the asynchronous phase we eagerly apply the asynchronous rules to active sequents $\Theta; \Gamma \Longrightarrow \Delta; \Psi$. If the main formula is an AF, the formula is decomposed; otherwise, it is moved to one of the outer contexts $\Theta$ and $\Psi$ (rule $\mathrm{Act}^{\mathrm{L}}$ or $\mathrm{Act}^{\mathrm{R}}$). When the inner contexts are emptied (namely, we get a sequent of the form $\Theta; \cdot \Longrightarrow \cdot; \Psi$), no asynchronous rule can be applied and the synchronous phase starts by selecting a formula $H$ in $\Theta, \Psi$ for focus (rule $\mathrm{Focus}^{\mathrm{L}}$ or $\mathrm{Focus}^{\mathrm{R}}$). Differently from the asynchronous phase, the rules to be applied are determined by the formula under focus. Note that the choice of $H$ determines a backtracking point: if proof search yields a sequent where $\Theta$ only contains atoms and $\Psi$ is empty, no formula can be picked and the construction of the derivation fails; to continue proof search, one has to backtrack to the last applied $\mathrm{Focus}^{\mathrm{L}}$ or $\mathrm{Focus}^{\mathrm{R}}$ rule and select, if possible, a new formula for focus. The left-focused phase is started by the application of rule $\mathrm{Focus}^{\mathrm{L}}$ and involves left-focused sequents of the form $\Theta; A \ll \Psi$. Here we analyze implications whose antecedents are either $a$ or $A \to B$. In the first case (rule $\to\mathrm{at}$), we perform a sort of forward application of modus ponens, provided that $a \in \Theta$, otherwise we backtrack. The application of rule $\to\to L$ determines a transition to a new asynchronous phase in the left premise, while focus is maintained in the right premise. The phase terminates when an $\mathrm{AF}^{+}$ formula is produced with a call to rule $\mathrm{Blur}^{\mathrm{L}}$. Alternatively, a right-focused phase begins by selecting a formula $H$ in $\Psi$ (rule $\mathrm{Focus}^{\mathrm{R}}$). Let us assume that $H$ is an atom. If $H \in \Theta$, we apply the axiom-rule Init and the construction of a closed branch succeeds; otherwise, we get a failure and we have to backtrack. If $H = K \to B$, we apply $\to\mathrm{R}$, which ends the synchronous phase and starts a new asynchronous phase. This is similar to the **LJQ** system [9].

$$\frac{}{\Theta;\Gamma,\bot\Longrightarrow\Delta;\Psi}\ \bot L \qquad \frac{\Theta;\Gamma\Longrightarrow\Delta;\Psi}{\Theta;\Gamma\Longrightarrow\bot,\Delta;\Psi}\ \bot R$$

$$\frac{\Theta;\Gamma,A,B\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,A\wedge B\Longrightarrow\Delta;\Psi}\ \wedge L \qquad \frac{\Theta;\Gamma\Longrightarrow A,\Delta;\Psi \qquad \Theta;\Gamma\Longrightarrow B,\Delta;\Psi}{\Theta;\Gamma\Longrightarrow A\wedge B,\Delta;\Psi}\ \wedge R$$

$$\frac{\Theta;\Gamma,A\Longrightarrow\Delta;\Psi \qquad \Theta;\Gamma,B\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,A\vee B\Longrightarrow\Delta;\Psi}\ \vee L \qquad \frac{\Theta;\Gamma\Longrightarrow A,B,\Delta;\Psi}{\Theta;\Gamma\Longrightarrow A\vee B,\Delta;\Psi}\ \vee R$$

$$\frac{\Theta;\Gamma\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,\bot\to B\Longrightarrow\Delta;\Psi}\ \bot\to L \qquad \frac{}{\Theta;\Gamma\Longrightarrow\bot\to B,\Delta;\Psi}\ \bot\to R$$

$$\frac{\Theta;\Gamma,A\to B\to C\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,(A\wedge B)\to C\Longrightarrow\Delta;\Psi}\ \wedge\to L \qquad \frac{\Theta;\Gamma\Longrightarrow A\to B\to C,\Delta;\Psi}{\Theta;\Gamma\Longrightarrow(A\wedge B)\to C,\Delta;\Psi}\ \wedge\to R$$

$$\frac{\Theta;\Gamma,A\to C,B\to C\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,(A\vee B)\to C\Longrightarrow\Delta;\Psi}\ \vee\to L \qquad \frac{\Theta;\Gamma\Longrightarrow A\to C,\Delta;\Psi \qquad \Theta;\Gamma\Longrightarrow B\to C,\Delta;\Psi}{\Theta;\Gamma\Longrightarrow(A\vee B)\to C,\Delta;\Psi}\ \vee\to R$$

$$\frac{\Theta,S;\Gamma\Longrightarrow\Delta;\Psi}{\Theta;\Gamma,S\Longrightarrow\Delta;\Psi}\ \mathrm{Act^L} \qquad \frac{\Theta;\Gamma\Longrightarrow\Delta;S,\Psi}{\Theta;\Gamma\Longrightarrow S,\Delta;\Psi}\ \mathrm{Act^R}$$

$$\frac{\Theta;S^-\ll\Psi}{\Theta,S^-;\cdot\Longrightarrow\cdot;\Psi}\ \mathrm{Focus^L} \qquad \frac{\Theta\gg S;\Psi}{\Theta;\cdot\Longrightarrow\cdot;S,\Psi}\ \mathrm{Focus^R} \qquad \frac{\Theta;T\Longrightarrow\cdot;\Psi}{\Theta;T\ll\Psi}\ \mathrm{Blur^L}$$

$$\frac{}{\Theta,a\gg a;\Psi}\ \mathrm{Init} \qquad \frac{\Theta;K\Longrightarrow B;\cdot}{\Theta\gg K\to B;\Psi}\ \to\mathrm{R}$$

$$\frac{\Theta,a;B\ll\Psi}{\Theta,a;a\to B\ll\Psi}\ \to\mathrm{at} \qquad \frac{\Theta;A,B\to C\Longrightarrow B;\cdot \qquad \Theta;C\ll\Psi}{\Theta;(A\to B)\to C\ll\Psi}\ \to\!\to L$$

$A$, $B$ and $C$ are any formulas, $S$ is a SF, $S^-$ is a $\mathrm{SF}^-$, $T$ is a $\mathrm{AF}^+$ and $K\to B$ is a SF.

**Fig. 1.** The **G4ipf** calculus

We remark that the main difference between **G4ipf** and a standard focused calculus such as **LJF** is that the rule $\mathrm{Focus^L}$ does not require the contraction of the formula selected for focus. This is a crucial point to avoid the generation of branches of infinite length and to guarantee the termination of the proof search procedure outlined above (see Section 3.1).

A *derivation* $\mathcal{D}$ of a sequent $\sigma$ in **G4ipf** is a tree of sequents built bottom-up starting from $\sigma$ and applying backward the rules of **G4ipf**. A *branch* of $\mathcal{D}$ is a sequence of sequents corresponding to the path from the root $\sigma$ of $\mathcal{D}$ to a leaf $\sigma_l$ of $\mathcal{D}$. If $\sigma_l$ is the conclusion of one of the axiom-rules $\bot L$, $\bot\to R$ and Init (the rules with no premises), the branch is *closed*. A derivation is closed if all its branches are closed. A sequent $\sigma$ is *provable in* **G4ipf** if there exists a closed derivation of $\sigma$; a formula $A$ is provable if the active sequent $\cdot;\cdot\Longrightarrow A;\cdot$ with empty contexts $\Theta$, $\Gamma$ and $\Psi$ is provable.

*Example 1.* Here we provide an example of a **G4ipf**-derivation of the formula $\neg\neg(a \vee \neg a)$. Recall that a derivation of such a formula in the standard calculus requires an application of contraction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a; \bot \Longrightarrow \cdot;}\,\bot L}{a; \bot \ll \cdot}\,\mathrm{Blur^L}}{a; \neg a \ll \cdot}\,{\to}\mathrm{at}}{\neg a, a; \cdot \Longrightarrow \cdot; \cdot}\,\mathrm{Focus^L}}{\neg a; a, \bot \to \bot \Longrightarrow \bot; \cdot}\,[\bot R, \bot \to L, \mathrm{Act^L}] \qquad \dfrac{\dfrac{}{\neg a; \bot \Longrightarrow \cdot; \cdot}\,\bot L}{\neg a; \bot \ll \cdot}\,\mathrm{Blur^L}}{\neg a; \neg\neg a \ll \cdot}\,{\to}{\to}\,L}{\neg a, \neg\neg a; \cdot \Longrightarrow \cdot; \cdot}\,\mathrm{Focus^L}}{\cdot; \neg(a \vee \neg a) \Longrightarrow \bot; \cdot}\,[\bot R, \vee \to L, \mathrm{Act^L} \times 2]}{\cdot \gg \neg\neg(a \vee \neg a); \cdot}\,{\to}\mathrm{R}}{\cdot; \cdot \Longrightarrow \cdot; \neg\neg(a \vee \neg a)}\,\mathrm{Focus^R}}{\cdot; \cdot \Longrightarrow \neg\neg(a \vee \neg a); \cdot}\,\mathrm{Act^R}$$

The double line corresponds to an asynchronous phase where more than one rule is applied. The only backtracking point is the choice of the formula for left-focus in the active sequent $\neg a, \neg\neg a; \cdot \Longrightarrow \cdot; \cdot$. If we select $\neg a$ instead of $\neg\neg a$, we get the sequent $\neg\neg a; \neg a \ll \cdot$ and the construction of the derivation immediately fails.

## 3 Meta-theory

We show that proof search in **G4ipf** can be performed in finite time. We define a well-founded relation $\prec$ such that, if $\sigma$ is the conclusion of a rule $R$ of **G4ipf** and $\sigma'$ any of the premises of $R$, then $\sigma' \prec \sigma$. As a consequence, branches of infinite length cannot be generated in proof search and the provability of $\sigma$ in **G4ipf** can be decided in finite time.

### 3.1 Termination

We assign to any formula $A$ a *weight* $wg(A)$ following [21]:

$$\mathrm{wg}(a) = \mathrm{wg}(\bot) = 2 \qquad \mathrm{wg}(A \wedge B) = \mathrm{wg}(A) + \mathrm{wg}(A) \cdot \mathrm{wg}(B)$$
$$\mathrm{wg}(A \vee B) = 1 + \mathrm{wg}(A) + \mathrm{wg}(B) \qquad \mathrm{wg}(A \to B) = 1 + \mathrm{wg}(A) \cdot \mathrm{wg}(B)$$

The weight $\mathrm{wg}(\sigma)$ of a sequent $\sigma$ is the sum of $\mathrm{wg}(A)$, for every $A$ in $\sigma$. One can easily prove that the following properties hold:

- $\mathrm{wg}(A \to (B \to C)) < \mathrm{wg}((A \wedge B) \to C)$;
- $\mathrm{wg}(A \to C) + \mathrm{wg}(B \to C) < \mathrm{wg}((A \vee B) \to C)$;
- $\mathrm{wg}(A) + \mathrm{wg}(B \to C) + \mathrm{wg}(C) < \mathrm{wg}((A \to B) \to C)$.

The above properties suffice to prove that proof search in the calculus **G4ip** terminates. Indeed, if $R$ is a rule of **G4ip**, $\sigma_1$ the conclusion of $R$ and $\sigma_2$ any of the premises of $R$, it holds that $\mathrm{wg}(\sigma_2) < \mathrm{wg}(\sigma_1)$; since weights are positive numbers, we cannot generate branches of infinite length. On the other hand, in **G4ipf** we cannot use the weight of the whole sequent as a measure, since we have rules where the conclusion and the premise have the same weight (Focus, Act and Blur).

Let $\prec_s$ ($\prec_d$) be the smallest relation between two sequents related by a rule of the same (different) judgment such that $\sigma_1 \prec_s \sigma_2$ ($\sigma_1 \prec_d \sigma_2$) if there exists a rule $R$ of **G4ipf** such that $\sigma_2$ is the conclusion of $R$ and $\sigma_1$ is any of the premises of $R$. For instance:

$$(\Theta; \Gamma, A \Longrightarrow \Delta; \Psi) \prec_s (\Theta; \Gamma, A \vee B \Longrightarrow \Delta; \Psi) \quad (\Theta, a; B \ll \Psi) \prec_s (\Theta, a; a \to B \ll \Psi)$$
$$(\Theta; A \Longrightarrow B; \cdot) \prec_d (\Theta \gg A \to B; \Psi) \prec_d (\Theta; \cdot \Longrightarrow \cdot; A \to B, \Psi)$$

Note that $\sigma_1 \prec_s \sigma_2$ implies $\mathrm{wg}(\sigma_1) \leq \mathrm{wg}(\sigma_2)$; moreover, if $\sigma_1 \prec_d \sigma_2$ then $\mathrm{wg}(\sigma_1) = \mathrm{wg}(\sigma_2)$.

Using as a measure the lexicographic ordering of $\langle \mathrm{wg}(A), \mathrm{wg}(\Gamma), \mathrm{wg}(\Delta)\rangle$ we can show (see the proof in the Appendix):

**Lemma 1.** *$\prec_s$ is a well-founded relation.*

The relation $\prec_d$ corresponds to the application of a rule which starts or ends a synchronous phase. Note that a synchronous phase cannot start by selecting an atom (indeed, the formula $S^-$ chosen for focus by $\mathrm{Focus}^{\mathrm{L}}$ must be a $\mathrm{SF}^-$), otherwise we could generate an infinite loop where an atom $a$ is picked for focus by $\mathrm{Focus}^{\mathrm{L}}$ and immediately released by $\mathrm{Blur}^{\mathrm{L}}$. As a consequence, we cannot have chains of the form $\sigma_1 \prec_d \sigma_2 \prec_d \sigma_3$, but between two $\prec_d$ at least an $\prec_s$ must occur. In the following lemma we show that two active sequents immediately before and after a synchronous phase have decreasing weights.

**Lemma 2.** *Let $\sigma_a$ and $\sigma_b$ be two active sequents, let $\sigma_1, \ldots, \sigma_n$ be $n \geq 1$ focused sequents such that $\sigma_a \prec_d \sigma_1 \prec_s \cdots \prec_s \sigma_n \prec_d \sigma_b$. Then $wg(\sigma_a) < wg(\sigma_b)$.*

*Proof.* By definition of $\prec_d$, $\sigma_n$ is obtained by applying $\mathrm{Focus}^{\mathrm{L}}$ or $\mathrm{Focus}^{\mathrm{R}}$ to $\sigma_b$, $\sigma_a$ is obtained by applying $\mathrm{Blur}^{\mathrm{L}}$ or $\to\mathrm{R}$ to $\sigma_1$, while in $\sigma_1, \ldots, \sigma_n$ only synchronous rules are applied. If $n = 1$, we have two possible cases:

1. $\sigma_a = \Theta; A, B \to C \Longrightarrow B; \cdot$
   $\sigma_1 = \Theta; (A \to B) \to C \ll \Psi$
   $\sigma_b = \Theta, (A \to B) \to C; \cdot \Longrightarrow \cdot; \Psi$;
2. $\sigma_a = \Theta; A \Longrightarrow B; \cdot$
   $\sigma_1 = \Theta \gg A \to B; \Psi$
   $\sigma_b = \Theta; \cdot \Longrightarrow \cdot; A \to B, \Psi$ (where $A$ is an atom or an implication).

In both cases $\mathrm{wg}(\sigma_a) < \mathrm{wg}(\sigma_b)$. Let $n > 1$. We have:

$$\sigma_a = \Theta; H_1 \Longrightarrow \cdot; \Psi, \quad \sigma_1 = \Theta; H_1 \ll \Psi, \quad \ldots \quad \sigma_n = \Theta; H_n \ll \Psi$$
$$\sigma_b = \Theta, H_n; \cdot \Longrightarrow \cdot; \Psi$$

Since $\mathrm{wg}(H_1) < \mathrm{wg}(H_n)$, it holds that $\mathrm{wg}(\sigma_a) < \mathrm{wg}(\sigma_b)$. □

Let $\prec$ be the transitive closure of the relation $\prec_s \cup \prec_d$. Note that $\sigma_1 \prec \sigma_2$ implies $\mathrm{wg}(\sigma_1) \leq \mathrm{wg}(\sigma_2)$. Using lemmas 1 and 2, one can prove that (see the proof in the Appendix):

**Proposition 1.** *$\prec$ is a well-founded order relation.*

By Proposition 1, every branch of a derivation of **G4ipf** has finite length. Indeed, let $\mathcal{D}$ be a (possibly open) derivation of $\sigma_1$ and let $\sigma_1, \sigma_2, \ldots$ be a branch of $\mathcal{D}$. We have $\sigma_{i+1} \prec \sigma_i$ for every $i \geq 1$, hence the branch has finite length.

### 3.2 Semantics

A Kripke model is a structure $\mathcal{K} = \langle P, \leq, \rho, V\rangle$, where $\langle P, \leq, \rho\rangle$ is a finite poset with minimum element $\rho$; $V$ is a function mapping every $\alpha \in P$ to a subset of atoms such that $\alpha \leq \beta$ implies $V(\alpha) \subseteq V(\beta)$. We write $\alpha < \beta$ to mean $\alpha \leq \beta$ and $\alpha \neq \beta$. The *forcing relation* $\mathcal{K}, \alpha \Vdash H$ ($\alpha$ forces $H$ in $\mathcal{K}$) is defined as follows:

- $\mathcal{K}, \alpha \nVdash \bot$;
- for every atom $a$, $\mathcal{K}, \alpha \Vdash a$ iff $a \in V(\alpha)$;
- $\mathcal{K}, \alpha \Vdash A \wedge B$ iff $\mathcal{K}, \alpha \Vdash A$ and $\mathcal{K}, \alpha \Vdash B$;
- $\mathcal{K}, \alpha \Vdash A \vee B$ iff $\mathcal{K}, \alpha \Vdash A$ or $\mathcal{K}, \alpha \Vdash B$;
- $\mathcal{K}, \alpha \Vdash A \to B$ iff, for every $\beta \in P$ such that $\alpha \leq \beta$, $\mathcal{K}, \beta \nVdash A$ or $\mathcal{K}, \beta \Vdash B$.

*Monotonicity property* holds for arbitrary formulas, i.e.: $\mathcal{K}, \alpha \Vdash A$ and $\alpha \leq \beta$ imply $\mathcal{K}, \beta \Vdash A$. A formula $A$ *is valid in* $\mathcal{K}$ iff $\mathcal{K}, \rho \Vdash A$. It is well-known that intuitionistic propositional logic **Int** coincides with the set of formulas valid in all (finite) Kripke models [6].

Given a Kripke model $\mathcal{K} = \langle P, \leq, \rho, V\rangle$, a world $\alpha \in P$ and a sequent $\sigma$, the relation $\mathcal{K}, \alpha \rhd \sigma$ ($\mathcal{K}$ *realizes* $\sigma$ at $\alpha$) is defined as follows:

- $\mathcal{K}, \alpha \rhd \Theta; \Gamma \Longrightarrow \Delta; \Psi$ iff
  $\mathcal{K}, \alpha \Vdash A$ for every $A \in \Theta, \Gamma$ and $\mathcal{K}, \alpha \nVdash B$ for every $B \in \Delta, \Psi$.
- $\mathcal{K}, \alpha \rhd \Theta; A \ll \Psi$ iff $\mathcal{K}, \alpha \rhd \Theta; A \Longrightarrow \cdot; \Psi$.
- $\mathcal{K}, \alpha \rhd \Theta \gg A; \Psi$ iff $\mathcal{K}, \alpha \rhd \Theta; \cdot \Longrightarrow A; \Psi$.

A sequent $\sigma = \Theta; \Gamma \Longrightarrow \Delta; \Psi$ is *realizable* if there exists a model $\mathcal{K} = \langle P, \leq, \rho, V\rangle$ such that $\mathcal{K}, \rho \rhd \sigma$; in this case we say that $\mathcal{K}$ is a *model* of $\sigma$. We point out that $\sigma$ is realizable iff the formula $\bigwedge(\Theta, \Gamma) \to \bigvee(\Delta, \Psi)$ is not intuitionistically valid. Moreover, it is easy to check that, if $\sigma$ is the conclusion of one of the axiom-rules $\bot L$, $\bot \to R$ and Init, then $\sigma$ is not realizable. A rule $R$ is *sound* iff, if the conclusion of $R$ is realizable, then at least one of its premises is realizable. We can esaily proof that (see the Appendix):

**Proposition 2.** *The rules of* **G4ipf** *are sound.*

By Proposition 2 the soundness of **G4ipf** follows (see the proof in the Appendix):

**Theorem 1 (Soundness).** *If $\sigma$ is provable in* **G4ipf** *then $\sigma$ is not realizable.*

### 3.3 Completeness

We show that, if proof search for a sequent $\sigma$ fails, we can build a model $\mathcal{K}$ of $\sigma$, and this proves the completeness of **G4ipf**. Henceforth, by *unprovable* we mean 'not provable in **G4ipf**'.

A left-focused sequent $\Theta; H \ll \Psi$ is *strongly unprovable* iff one of the following conditions holds:

(i) $H$ is an $\mathrm{AF}^+$ and the sequent $\Theta; H \Longrightarrow \cdot; \Psi$ is unprovable;
(ii) $H = A \to B$ and $\Theta; B \ll \Psi$ is strongly unprovable.

By definition of the rules of **G4ipf**, we immediately get:

**Lemma 3.** *If $\sigma = \Theta; H \ll \Psi$ is strongly unprovable, then $\sigma$ is unprovable.*

Let $\sigma = \Theta; H \ll \Psi$ be a left-focused sequent.

- $\sigma$ is *at-unprovable w.r.t.* $a \to B$ iff, for some $m \geq 0$, it holds that $H = H_1 \to \cdots \to H_m \to a \to B$ and $a \notin \Theta$ (if $m = 0$, then $H = a \to B$);
- $\sigma$ is *at-unprovable* if, for some $a \to B$, $\sigma$ is at-unprovable w.r.t. $a \to B$;
- $\sigma$ is *$\to$-unprovable w.r.t.* $(A \to B) \to C$ iff, for some $m \geq 0$, it holds that $H = H_1 \to \cdots \to H_m \to (A \to B) \to C$ and $\Theta; A, B \to C \Longrightarrow B; \cdot$ is unprovable (if $m = 0$, then $H = (A \to B) \to C$);
- $\sigma$ is *$\to$-unprovable* if, for some $(A \to B) \to C$, $\sigma$ is $\to$-unprovable w.r.t. $(A \to B) \to C$.

Note that a sequent can match the above definitions in more than one way. For instance, let $\sigma = \cdot; a_1 \to (a_2 \to a_3) \to a_4 \to a_5 \ll a_6$; then:

- $\sigma$ is at-unprovable w.r.t. $a_1 \to (a_2 \to a_3) \to a_4 \to a_5$ and w.r.t. $a_4 \to a_5$;
- $\sigma$ is $\to$-unprovable w.r.t. $(a_2 \to a_3) \to a_4 \to a_5$.

**Lemma 4.** *Let $\sigma = \Theta; H \ll \Psi$ be an unprovable sequent. Then, $\sigma$ is strongly unprovable or at-unprovable or $\to$-unprovable.*

*Proof.* By induction on $\prec$. Let us assume that, for every $\sigma' \prec \sigma$, the lemma holds for $\sigma'$; we prove the lemma for $\sigma$ by a case analysis.

- Let $H$ be an $\mathrm{AF}^+$. Since the sequent $\sigma$ is unprovable then $\Theta; H \Longrightarrow \cdot; \Psi$ is unprovable. Hence by definition $\sigma$ is strongly unprovable.
- Let $H = a \to B$. If $a \notin \Theta$ then $\sigma$ is at-unprovable w.r.t. $a \to B$. Let $a \in \Theta$ and let $\sigma' = \Theta; B \ll \Psi$. Then $\sigma'$ is unprovable. Since $\sigma' \prec \sigma$, by IH $\sigma'$ is strongly unprovable or at-unprovable or $\to$-unprovable. If $\sigma'$ is strongly unprovable, by definition $\sigma$ is strongly unprovable. Let us assume that $\sigma'$ is at-unprovable w.r.t. $a' \to C$. Then $B = H_1 \to \cdots \to H_m \to a' \to C$ and $a' \notin \Theta$. This implies that $\sigma$ is at-unprovable w.r.t. $a' \to C$. Finally, let us assume that $\sigma'$ is $\to$-unprovable w.r.t. $(C \to D) \to E$. Then $B = H_1 \to \cdots \to H_m \to (C \to D) \to E$ and the sequent $\Theta; C, D \to E \Longrightarrow D; \cdot$ is unprovable. If follows that $\sigma$ is $\to$-unprovable w.r.t. $(C \to D) \to E$.

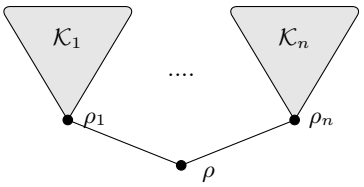



**Fig. 2.** The model Model(At, $\{\mathcal{K}_1, \ldots, \mathcal{K}_n\}$)

– Let $H = (B \to C) \to D$. If $\Theta; B, C \to D \Longrightarrow C; \cdot$ is unprovable, then by definition $\sigma$ is $\to$-unprovable w.r.t. $(B \to C) \to D$. Otherwise, let $\Theta; B, C \to D \Longrightarrow C; \cdot$ be provable. Then $\sigma' = \Theta; D \ll \Psi$ is unprovable. Since $\sigma' \prec \sigma$, by IH $\sigma'$ is strongly unprovable or at-unprovable or $\to$-unprovable. Reasoning as above, the lemma holds for $\sigma$. □

Let $\mathcal{S} = \{\mathcal{K}_1, \ldots \mathcal{K}_n\}$ be a (possibly empty) set of models $\mathcal{K}_i = \langle P_i, \leq_i, \rho_i, V_i \rangle$ $(1 \leq i \leq n)$, let At be a set of atoms such that, for every $1 \leq i \leq n$, At $\subseteq V_i(\rho_i)$; without loss of generality, we can assume that the sets $P_i$ are pairwise disjoint. By Model(At, $\mathcal{S}$) we denote the Kripke model $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ defined as follows:

1. If $\mathcal{S}$ is empty, then $\mathcal{K}$ is the Kripke model consisting of only the world $\rho$ and $V(\rho) =$ At.
2. Let $n \geq 1$. Then (see Fig. 2):
   - $\rho$ is new (namely, $\rho \notin \bigcup_{i \in \{1,\ldots,n\}} P_i$) and $P = \{\rho\} \cup \bigcup_{i \in \{1,\ldots,n\}} P_i$;
   - $\leq = \{ (\rho, \alpha) \mid \alpha \in P \} \cup \bigcup_{i \in \{1,\ldots,n\}} \leq_i$;
   - $V(\rho) =$ At and, for every $i \in \{1, \ldots, n\}$ and $\alpha \in P_i$, $V(\alpha) = V_i(\alpha)$.

It is easy to check that $\mathcal{K}$ is a well-defined Kripke model. In Point 2, for every $1 \leq i \leq n$, every $\alpha \in P_i$ and every formula $A$, it holds that $\mathcal{K}, \alpha \Vdash A$ iff $\mathcal{K}_i, \alpha \Vdash A$. A world $\beta$ of a model $\mathcal{K}$ is an immediate successor of $\alpha$ if $\alpha < \beta$ and, for every $\gamma$ such that $\alpha \leq \gamma \leq \beta$, either $\gamma = \alpha$ or $\gamma = \beta$.

**Lemma 5.** *Let $H = H_1 \to \cdots \to H_m \to A \to B$ $(m \geq 0)$, let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be a model such that $\mathcal{K}, \rho \nVdash A$ and, for every immediate successor $\alpha$ of $\rho$, it holds that $\mathcal{K}, \alpha \Vdash H$. Then $\mathcal{K}, \rho \Vdash H$.*

In the next lemma we show how to build a Kripke model of an unprovable sequent.

**Lemma 6.** *Let $\sigma = \Theta; \cdot \Longrightarrow \cdot; \Psi$ be an unprovable sequent such that, for every non-atomic $H \in \Theta$, the sequent $\Theta \setminus \{H\}; H \ll \Psi$ is at-unprovable or $\to$-unprovable. Let* At *be the set of atoms of $\Theta$ and let $\Theta_1$ be the set of non-atomic formulas $H$ of $\Theta$ such that the sequent $\Theta \setminus \{H\}; H \ll \Psi$ is not at-unprovable. Let $\mathcal{S}$ be a (possibly empty) set of models satisfying the following conditions:*

*(i) For every $H \in \Theta_1$, let $(A \to B) \to C$ such that $\Theta \setminus \{H\}; H \ll \Psi$ is $\to$-unprovable w.r.t. $(A \to B) \to C$; then $\mathcal{S}$ contains a model of the sequent $\Theta \setminus \{H\}; A, B \to C \Longrightarrow B; \cdot$.*

*(ii) For every* $A \to B \in \Psi$, $\mathcal{S}$ *contains a model of the sequent* $\Theta; A \Longrightarrow B; \cdot$.
*(iii) Every model of* $\mathcal{S}$ *is of type (i) or (ii).*

*Then,* $\mathrm{Model}(\mathrm{At}, \mathcal{S})$ *is a model of* $\sigma$.

*Proof.* Let us assume that the set of models $\mathcal{S}$ is empty. Then $\Theta_1$ is empty and $\Psi$ only contains atoms not belonging to At. By definition, $\mathcal{K} = \mathrm{Model}(\mathrm{At}, \mathcal{S})$ has only the world $\rho$. Since $V(\rho) = \mathrm{At}$, we immediately get $\mathcal{K}, \rho \Vdash a$, for every $a \in \mathrm{At}$, and $\mathcal{K}, \rho \nVdash a'$, for every $a' \in \Psi$. Let $H$ be a non-atomic formula of $\Theta$. Since $\Theta_1 = \emptyset$, the sequent $\Theta \setminus \{H\}; H \ll \Psi$ is at-unprovable. This means that $H = H_1 \to \cdots \to H_m \to a \to B$, where $a \notin \mathrm{At}$, hence $\mathcal{K}, \rho \Vdash H$. This proves that $\mathcal{K}, \rho \rhd \sigma$, thus $\mathcal{K}$ is a model of $\sigma$.

Let us assume that $\mathcal{S}$ contains the models $\mathcal{K}_1 = \langle P_1, \leq_1, \rho_1, V_1 \rangle$, $\ldots$, $\mathcal{K}_n = \langle P_n, \leq_n, \rho_n, V_n \rangle$ $(n \geq 1)$ and let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be the model $\mathrm{Model}(\mathrm{At}, \mathcal{S})$; we show that $\mathcal{K}$ is a model of $\sigma$.

If $a \in \mathrm{At}$, then $\mathcal{K}, \rho \Vdash a$ by definition of $V$.

Let $H$ be a non-atomic formula of $\Theta$. If $H \notin \Theta_1$, then the sequent $\Theta \setminus \{H\}; H \ll \Psi$ is at-unprovable, namely $H = H_1 \to \cdots \to H_m \to a \to B$, where $a \notin \mathrm{At}$. Firstly, we note that $\mathcal{K}_i, \rho_i \Vdash H$, for every $1 \leq i \leq n$; indeed, by (i)–(iii), $\mathcal{K}_i$ is a model of a sequent of the form $\Theta'; \Gamma' \Longrightarrow \Delta'; \cdot$ such that $H \in \Theta'$. It follows that $\mathcal{K}_i, \rho_i \Vdash H$, for every $1 \leq i \leq n$; hence $\mathcal{K}, \rho_i \Vdash H$. By definition of $V$, we have $\mathcal{K}, \rho \nVdash a$. By Lemma 5, we get $\mathcal{K}, \rho \Vdash H$.

Let $H \in \Theta_1$ and let $\Theta \setminus \{H\}; H \ll \Psi$ be $\to$-unprovable w.r.t. $(A \to B) \to C$. This mean that $H = H_1 \to \cdots \to H_m \to (A \to B) \to C$ and, by (i), $\mathcal{S}$ contains a model $\mathcal{K}_j$ of $\Theta \setminus \{H\}; A, B \to C \Longrightarrow B; \cdot$. This implies that:

(P1) $\mathcal{K}_j, \rho_j \Vdash A$;
(P2) $\mathcal{K}_j, \rho_j \Vdash B \to C$;
(P3) $\mathcal{K}_j, \rho_j \nVdash B$.

By (P1) and (P2) it follows that $\mathcal{K}_j, \rho_j \Vdash (A \to B) \to C$, which implies $\mathcal{K}_j, \rho_j \Vdash H$. Moreover, if $i \in \{1, \ldots, n\}$ and $i \neq j$, then by (i)– (iii) $\mathcal{K}_i$ is a model of a sequent $\Theta'; \Gamma' \Longrightarrow \Delta'; \cdot$ such that $H \in \Theta'$, hence $\mathcal{K}_i, \rho_i \Vdash H$. Thus, for every $1 \leq i \leq n$, it holds that $\mathcal{K}_i, \rho_i \Vdash H$, which implies $\mathcal{K}, \rho_i \Vdash H$. By (P1) and (P3), we have $\mathcal{K}, \rho_j \Vdash A$ and $\mathcal{K}, \rho_j \nVdash B$. Since $\rho < \rho_j$ in $\mathcal{K}$, we get $\mathcal{K}, \rho \nVdash A \to B$. By Lemma 5, we conclude $\mathcal{K}, \rho \Vdash H$.

Let $H \in \Psi$. If $H$ is an atom, then $H \notin \mathrm{At}$, otherwise $\sigma$ would be provable; hence $\mathcal{K}, \rho \nVdash H$. Let $H = A \to B$. By (ii), $\mathcal{S}$ contains a model $\mathcal{K}_j$ of $\Theta; A \Longrightarrow B; \cdot$. Thus, $\mathcal{K}_j, \rho_j \Vdash A$ and $\mathcal{K}_j, \rho_j \nVdash B$, which implies $\mathcal{K}, \rho \nVdash A \to B$. We conclude that $\mathcal{K}$ is a model of $\sigma$. □

We can now prove the completeness of **G4ipf**.

**Proposition 3 (Completeness).** *Let* $\sigma = \Theta; \Gamma \Longrightarrow \Delta; \Psi$. *If* $\sigma$ *is unprovable, then* $\sigma$ *is realizable.*

*Proof.* By induction on $\prec$. If $\Gamma, \Delta$ is not empty, the proposition easily follows by the induction hypothesis. For instance, let $\sigma = \Theta; \Gamma, A \vee B \Longrightarrow \Delta; \Psi$. By definition of the rule $\vee L$, one of the sequents $\sigma_A = \Theta; \Gamma, A \Longrightarrow \Delta; \Psi$ or $\sigma_B = \Theta; \Gamma, B \Longrightarrow \Delta; \Psi$ is unprovable. Since $\sigma_A \prec \sigma$ and $\sigma_B \prec \sigma$, by induction hypothesis there exists a model $\mathcal{K}$ of $\sigma_A$ or of $\sigma_B$. In either case $\mathcal{K}$ is a model of $\sigma$, hence $\sigma$ is realizable.

Let $\sigma = \Theta; \cdot \Longrightarrow \cdot; \Psi$. We distinguish two cases (C1) and (C2).

(C1) There is a non-atomic formula $H \in \Theta$ such that $\sigma' = \Theta \setminus \{H\}; H \ll \Psi$ is strongly unprovable.

By Lemma 3, $\sigma'$ is unprovable. Since $\sigma' \prec \sigma$, by induction hypothesis there exists a model $\mathcal{K}$ of $\sigma'$; since $\mathcal{K}$ is also a model of $\sigma$, we conclude that $\sigma$ is realizable.

(C2) For every non-atomic $H \in \Theta$, the sequent $\sigma' = \Theta \setminus \{H\}; H \ll \Psi$ is not strongly unprovable.

We build a model of $\sigma$ by applying Lemma 6. We point out that the hypothesis of Lemma 6 are satisfied. Indeed, for every non-atomic $H \in \Theta$, since $\sigma' = \Theta \setminus \{H\}; H \ll \Psi$ is not strongly unprovable, by Lemma 4 $\sigma'$ is at-unprovable or $\rightarrow$-unprovable. The (possibly empty) set of models $\mathcal{S}$ can be defined as follows:

(a) For every $H \in \Theta_1$, let us assume that $\Theta \setminus \{H\}; H \ll \Psi$ is $\rightarrow$-unprovable w.r.t. $(A \rightarrow B) \rightarrow C$. Then $H = H_1 \rightarrow \cdots \rightarrow H_m \rightarrow (A \rightarrow B) \rightarrow C$ and the sequent $\sigma_H = \Theta \setminus \{H\}; A, B \rightarrow C \Longrightarrow B; \cdot$ is unprovable. Since $\sigma_H \prec \sigma$, by induction hypothesis there exists a model of $\sigma_H$.
(b) For every $K = A \rightarrow B \in \Psi$, the sequent $\sigma_K = \Theta; A \Longrightarrow B; \cdot$ is unprovable (otherwise $\sigma$ would be provable). Since $\sigma_K \prec \sigma$, by induction hypothesis there exists a model of $\sigma_K$.

Thus, we can define $\mathcal{S}$ as the set of models $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ mentioned in (a) and in (b); note that, since $\mathrm{At} \subseteq \Theta$, we have $\mathrm{At} \subseteq V(\rho)$. By Lemma 6, $\mathrm{Model}(\mathrm{At}, \mathcal{S})$ is a model of $\sigma$, hence $\sigma$ is realizable. □

The above proof shows how to build a model of an unprovable sequent (see in particular points (a) and (b)). We remark that, in the model construction, only active sequents are relevant, while focused sequents are skipped. This justifies why standard model construction techniques are not directly applicable and a more involved machinery is needed.

By soundness and completeness of **G4ipf**, a sequent $\sigma$ is provable in **G4ipf** iff $\sigma$ is not realizable. By definition, $A \in$ **Int** iff the sequent $\cdot; \cdot \Longrightarrow A; \cdot$ is not realizable. We conclude that $A \in$ **Int** iff $A$ is provable in **G4ipf**.

## 4 Conclusions and future work

We have presented a focused version of the contraction-free calculus **G4ip** [21]. Essentially, every treatment of focusing [14] extends the (a)synchronous classification of connectives to *atoms*, assigning them a *bias* or *polarity*. Different

polarizations of atoms do not affect provability, but do influence significantly the *shape* of the derivation, allowing one to informally characterize forward and backward reasoning via respectively positive and negative bias assignments. Unfortunately, the contraction-free approach is essentially forward and negative bias do not work as expected. Here is why: standard presentations, where contraction on focus is allowed, use the following rules

$$\frac{}{\Theta; n \ll n, \Psi}\ \mathrm{Init}^{\mathrm{L}} \qquad \frac{\Theta; P \Longrightarrow \cdot; \Psi}{\Theta; P \ll \Psi}\ \mathrm{Blur}^{\mathrm{L}}$$

$$\frac{\Theta; \cdot \Longrightarrow \cdot; n \qquad \Theta; B \ll \Psi}{\Theta; n \to B \ll \Psi} \to \mathrm{at}- \qquad \frac{\Theta, p; B \ll \Psi}{\Theta, p; p \to B \ll \Psi} \to \mathrm{at}+$$

where $n$ is a negative atom, $p$ is a positive atom, $P$ an AF or a positive atom. These rules without contraction give rise to an incomplete calculus. For instance, let us consider the non-realizable sequent $\sigma = n \to p, (n \to p) \to n; \cdot \Longrightarrow \cdot; p$. The only rule applicable to $\sigma$ is $\mathrm{Focus}^{\mathrm{L}}$. If we select $n \to p$ we get:

$$\frac{\begin{array}{c}\vdots\\ (n \to p) \to n; \cdot \Longrightarrow \cdot; n \end{array} \qquad \begin{array}{c}\\ (n \to p) \to n; p \ll p\end{array}}{(n \to p) \to n; n \to p \ll p} \to \mathrm{at}-$$

But the left premise is unprovable. On the other hand, if we choose $(n \to p) \to n$ we get:

$$\frac{\begin{array}{c}\vdots\\ n \to p; n, p \to n \Longrightarrow p; \cdot \end{array} \qquad \begin{array}{c}\\ n \to p; n \ll p\end{array}}{n \to p; (n \to p) \to n \ll p} \to\!\to L$$

But the right premise is unprovable because there is no rule that can blur a negative atom from focus. To get a complete calculus we should allow $\mathrm{Blur}^{\mathrm{L}}$ on negative atoms, but in this case the calculus does not properly capture "backward chaining".

This paper is but a beginning of our investigation of focusing:

- It is commonly believed that every "reasonable" sequent calculus has a natural focused version. We aim to test this "universality" hypothesis further by investigating its applicability to a rather peculiar logic, Gödel-Dummett's, which is well-known to lead a double life as a super-intuitionistic (but not constructive) and as a quintessential fuzzy logic [17].
- We plan to investigate *counterexample* search in focused systems. The natural question is: considering that focused calculi restrict the shape of derivations, what kind of counter models do they yield, upon failure? How do they compare to calculi such as [2] or the calculus [11] designed to yield models of *minimal* depth?
- There seems to be a connection between contraction-free calculi and Gabbay's restart rule [12], a technique to make goal oriented provability with diminishing resources complete for intuitionistic provability. Focusing could be the key to understand this.

## References


1. J. Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.
2. A. Avellone, G. Fiorino, and U. Moscato. Optimization techniques for propositional intuitionistic logic and their implementation. *TCS*, 409(1):41–58, 2008.
3. A. Avron and B. Konikowska. Decomposition proof systems for Gödel-Dummett logics. *Studia Logica*, 69(2):197–219, 2001.
4. D. Baelde. Least and greatest fixed points in linear logic. *ACM Trans. Comput. Log.*, 13(1):2, 2012.
5. D. Baelde, D. Miller, and Z. Snow. Focused inductive theorem proving. In J. Giesl et al., editors, *IJCAR*, volume 6173 of *LNCS*, pp. 278–292. Springer, 2010.
6. A. Chagrov and M. Zakharyaschev. *Modal Logic*. Oxford University Press, 1997.
7. K. Chaudhuri. *The Focused Inverse Method for Linear Logic*. PhD thesis, Carnegie Mellon University, 2006.
8. K. Chaudhuri, F. Pfenning, and G. Price. A logical characterization of forward and backward chaining in the inverse method. *JAR*, 40(2-3):133–177, 2008.
9. R. Dyckhoff and S. Lengrand. LJQ: a strongly focused calculus for intuitionistic logic. In A. Beckmann et al., editors, *Computability in Europe 2006*, volume 3988, pages 173–185. Springer, 2006.
10. R. Dyckhoff and S. Negri. Admissibility of structural rules for contraction-free systems of intuitionistic logic. *J. Symb. Log.*, 65(4):1499–1518, 2000.
11. M. Ferrari, C. Fiorentini, and G. Fiorino. Contraction-Free Linear Depth Sequent Calculi for Intuitionistic Propositional Logic with the Subformula Property and Minimal Depth Counter-Models. *JAR*, pages 1–21, 2012.
12. D. Gabbay and N. Olivetti. *Goal-Directed Proof Theory*, volume 21 of *Applied Logic Series*. Kluwer Academic Publishers, August 2000.
13. A.S. Henriksen. A contraction-free focused sequent calculus for classical propositional logic. Leibnitz International Proc. in Informatics, Daghstul, April 2011.
14. C. Liang and D. Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theor. Comput. Sci.*, 410(46):4747–4768, 2009.
15. S. Maehara. Eine darstellung der intuitionistischen logik in der klassischen. *Nagoya Mathematical Journal*, pages 45–64, 1954.
16. S. McLaughlin and F. Pfenning. Imogen: Focusing the polarized inverse method for intuitionistic propositional logic. In I. Cervesato et al., editors, *LPAR*, volume 5330 of *LNCS*, pages 174–181. Springer, 2008.
17. G. Metcalfe, N. Olivetti, and D. Gabbay. *Proof Theory for Fuzzy Logics*. Springer Publishing Company, Incorporated, 1st edition, 2008.
18. D. Miller and E. Pimentel. A formal framework for specifying sequent calculus proof systems. *Theor. Comput. Sci.*, 474:98–116, 2013.
19. D. Miller and A. Saurin. From proofs to focused proofs: A modular proof of focalization in linear logic. In J. Duparc et al., editors, *CSL*, volume 4646 of *LNCS*, pages 405–419. Springer, 2007.
20. V. Nigam, E. Pimentel, and G. Reis. Specifying proof systems in linear logic with subexponentials. *Electr. Notes Theor. Comput. Sci.*, 269:109–123, 2011.
21. A.S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*, volume 43 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1996.
22. A. Waaler and L. Wallen. Tableaux for Intuitionistic Logics. In M. D'Agostino et al., editors, *Handbook of Tableaux Methods*, pages 255–296. Kluwer, 1999.

## Appendix

### Proof of Lemma 1

To prove that $\prec_s$ is a well-founded relation, we have to show that there is no infinite descending $\prec_s$-chain of the form

$$\cdots \prec_s \sigma_3 \prec_s \sigma_2 \prec_s \sigma_1$$

Note that all the sequents in the $\prec_s$-chain have the same kind. Thus, either all the sequents in the $\prec_s$-chain are focused or all are active.

Let $\sigma_1 = \Theta_1; A_1 \ll \Psi_1$ and $\sigma_2 = \Theta_2; A_2 \ll \Psi_2$ be two focused sequents such that $\sigma_1 \prec_s \sigma_2$. Then, $\Theta_1 = \Theta_2$, $\Psi_1 = \Psi_2$ and $\mathrm{wg}(A_1) < \mathrm{wg}(A_2)$, hence $\mathrm{wg}(\sigma_1) < \mathrm{wg}(\sigma_2)$. Since the weight of a sequent is a positive number, every descending $\prec_s$-chains containing focused sequents has finite length.

Let $\sigma_1 = \Theta_1; \Gamma_1 \Longrightarrow \Delta_1; \Psi_1$ and $\sigma_2 = \Theta_2; \Gamma_2 \Longrightarrow \Delta_2; \Psi_2$ be two active sequents such that $\sigma_1 \prec_s \sigma_2$. Then, one of the following conditions holds:

1. $\mathrm{wg}(\sigma_1) < \mathrm{wg}(\sigma_2)$;
2. $\mathrm{wg}(\sigma_1) = \mathrm{wg}(\sigma_2)$ and $\mathrm{wg}(\Gamma_1, \Delta_1) < \mathrm{wg}(\Gamma_2, \Delta_2)$.

Thus, every descending $\prec_s$-chains containing active sequents has finite length.

### Proof of Proposition 1

We have to prove that $\prec$ is a well-founded order relation. By definition, $\prec$ is transitive. We show that there exists no infinite descending $\prec$-chain; this also implies that $\prec$ is not reflexive. Let us assume, by absurd, that there exists an infinite $\prec$-chain $\mathcal{C}$ of sequents $\sigma_i$ ($i \geq 1$) such that $\sigma_{i+1} \prec \sigma_i$ for every $i \geq 1$. We have $\mathrm{wg}(\sigma_{i+1}) \leq \mathrm{wg}(\sigma_i)$ for every $i \geq 1$. Since, by Lemma 1, the relation $\prec_s$ is well-founded, $\mathcal{C}$ contains infinitely many occurrences of $\prec_d$. By Lemma 2, from $\mathcal{C}$ we can extract an infinite sequence of active sequents $\sigma'_i$ such that $\mathrm{wg}(\sigma'_{i+1}) < \mathrm{wg}(\sigma'_i)$ for every $i \geq 1$, a contradiction. We conclude that every descending $\prec$-chain has finite length, hence $\prec$ well-founded.

### Proof of Proposition 2

We have to prove that the rules of **G4ipf** are sound. All the cases except the one for $\rightarrow\rightarrow L$ and $\rightarrow$R rules are immediate.

Let $R$ be the rule $\rightarrow$R, let $\sigma = \Theta \gg A \rightarrow B; \Psi$ be the conclusion of $R$ and let $\mathcal{K} = \langle P, \leq, \rho, V \rangle$ be a Kripke model such that $\mathcal{K}, \rho \rhd \sigma$. Since $\mathcal{K}, \rho \nVdash A \rightarrow B$, there exists $\beta \in P$ such that $\mathcal{K}, \beta \Vdash A$ and $\mathcal{K}, \beta \nVdash B$. It follows that the submodel of $\mathcal{K}$ having root $\beta$ realizes the premise $\Theta; A \Longrightarrow B; \cdot$ of $R$.

Let $R$ be the rule $\rightarrow\rightarrow L$ , let $\sigma = \Theta; (A \rightarrow B) \rightarrow C \ll \Psi$ be the conclusion of $R$ and let us assume $\mathcal{K}, \rho \rhd \sigma$. If $\mathcal{K}, \rho \Vdash C$, we get $\mathcal{K}, \rho \rhd \Theta; C \ll \Psi$, hence the right-most premise of $R$ is realizable. Let us assume $\mathcal{K}, \rho \nVdash C$. Since $\mathcal{K}, \rho \Vdash (A \rightarrow B) \rightarrow C$, we have $\mathcal{K}, \rho \nVdash A \rightarrow B$. Then, there exists $\beta \in P$ such that $\mathcal{K}, \beta \Vdash A$ and $\mathcal{K}, \beta \nVdash B$. It follows that $\mathcal{K}, \beta \Vdash B \rightarrow C$, and this implies $\mathcal{K}, \beta \rhd \Theta; A, B \rightarrow C \Longrightarrow B; \cdot$; thus, the left-most premise of $R$ is realizable.

**Proof of Theorem 1 (Soundness of G4ipf)**

Let $\mathcal{D}$ be a closed derivation of $\sigma$ and let us assume that $\sigma$ is realizable. By Proposition 2, one of the initial sequents $\sigma$ of $\mathcal{D}$ is realizable. Since $\sigma$ is the conclusion of an axiom-rule, we get a contradiction.

# Verification of Imperative Programs by Transforming Constraint Logic Programs*

Emanuele De Angelis[1], Fabio Fioravanti[1],
Alberto Pettorossi[2], and Maurizio Proietti[3]

[1] DEC, University 'G. D'Annunzio', Pescara, Italy
{emanuele.deangelis,fioravanti}@unich.it
[2] DICII, University of Rome Tor Vergata, Rome, Italy
pettorossi@disp.uniroma2.it
[3] IASI-CNR, Rome, Italy maurizio.proietti@iasi.cnr.it

**Abstract.** We present a method for verifying partial correctness properties of imperative programs that manipulate integers and arrays by using techniques based on the transformation of constraint logic programs (CLP). We use CLP as a metalanguage for representing imperative programs, their executions, and their properties. First, we encode the correctness of an imperative program, say *prog*, as the negation of a predicate `incorrect` defined by a CLP program $T$. By construction, `incorrect` holds in the least model of $T$ if and only if the execution of *prog* from an initial configuration eventually halts in an error configuration. Then, we apply to program $T$ a sequence of transformations that preserve its least model semantics. These transformations are based on well-known transformation *rules*, such as *unfolding* and *folding*, guided by suitable transformation *strategies*, such as *specialization* and *generalization*. The objective of the transformations is to derive a new CLP program *TransfT* where the predicate `incorrect` is defined either by (i) the fact '`incorrect.`' (and in this case *prog* is *not* correct), or by (ii) the empty set of clauses (and in this case *prog* is correct). In the case where we derive a CLP program such that neither (i) nor (ii) holds, we iterate the transformation. Since the problem is undecidable, this process may not terminate. We show through examples that our method can be applied in a rather systematic way, and is amenable to automation by transferring to the field of program verification many techniques developed in the field of program transformation.

## 1 Introduction

In the last decade formal techniques have received a renewed attention as the basis of a methodology for increasing the reliability of software artifacts and reducing the cost of software production. In particular, great efforts have been made to devise automatic techniques such as *software model checking* [23], for verifying the correctness of programs with respect to their specifications.

* A preliminary version of this paper appears in [10].

In many software model checking techniques, the use of *constraints* has been very effective both for constructing models of programs and for reasoning about them [2, 8, 9, 12, 18, 20, 22, 33, 34]. Several kinds of constraints have been considered, such as equalities and inequalities over booleans, integers, reals, and finite or infinite trees. By using constraints we can represent in a symbolic, compact way the (possibly infinite) sets of values computed by programs and, in general, the sets of states which are reached during program executions. Then, by using powerful solvers specifically designed for the classes of constraints we have mentioned above, we can reason about program properties in an efficient way.

In this paper we consider a simple imperative programming language with integer and array variables and we use Constraint Logic Programming (CLP) [21] as a metalanguage for representing imperative programs, their executions, and the properties to be verified. We use constraints consisting of linear equalities and inequalities over integers. Note, however, that the method presented here is parametric with respect to the constraint domain which is used. By following an approach originally presented in [33], a given imperative program *prog* and its interpreter are first encoded as a CLP program. Then, the proofs of the properties of interest about the program *prog* are sought by analyzing that derived CLP program. In order to improve the efficiency of that analysis, it is advisable to first *compile-away* the CLP interpreter of the language in which *prog* is written. This is done by specializing the interpreter with respect to the given program *prog* using well-known *program specialization* techniques [24, 33].

In previous papers [9, 16] we have shown that program specialization can be used not only as a preprocessing step to improve the efficiency of program analysis, but also as a means of analysis on its own. In this paper, we extend that approach and we propose a verification method based on more general *unfold/fold transformation rules* for CLP programs [5, 13, 37].

Transformation-based verification techniques are very appealing because they are parametric with respect to both the programming languages in which programs are written, and the logics in which the properties of interest are specified. Moreover, since the output of a transformation-based verification method is a program which is *equivalent* to the given program with respect to the properties of interest, we can apply a *sequence* of transformations, thereby refining the analysis to the desired degree of precision (see, for instance, [9]).

The specific contributions of this paper are the following. We present a verification method based on a set of transformation rules which includes the rules for performing *conjunctive definition*, *conjunctive folding*, and *goal replacement*, besides the usual rules for *unfolding* and *constraint manipulation* which are used during program specialization. The rules for conjunctive definition and conjunctive folding allow us to introduce and transform new predicates defined in terms of *conjunctions* of old predicates, while program specialization can only deal with new predicates that correspond to specialized versions of exactly *one* old predicate. The goal replacement rule allows us to replace conjunctions of predicates and constraints by applying equivalences that hold in the least model of the CLP

program at hand, while program specialization can only replace conjunctions of constraints.

By using these more powerful definition and folding rules, we extend the specialization-based verification method in the following two directions: (i) we verify programs with respect to specifications given by sets of CLP clauses (for instance, recursively defined relations among program variables), whereas program specialization can only deal with specifications given by constraints, and (ii) we verify programs manipulating arrays and other data structures by applying equivalences between predicates that axiomatize suitable properties of those data structures (for instance, the ones deriving from the axiomatization of the theory of arrays [31]).

The paper is organized as follows. In Section 2 we present our transformation-based verification method. First, we introduce a simple imperative language and we describe how correctness properties of imperative programs can be translated into predicates defined by CLP programs. We also present a general strategy for applying the transformation rules to CLP programs, with the objective of verifying the properties of interest. Next, we present two examples of application of our verification method. In particular, in Section 3 we show how we deal with specifications given by recursive CLP clauses, and in Section 4 we show how we deal with programs which manipulate arrays. Finally, in Section 5 we discuss the related work which has been recently done in the area of automatic program verification.

## 2 The Transformation-Based Verification Method

We consider an imperative C-like programming language with integer and array variables, assignments (`=`), sequential compositions (`;`), conditionals (`if` and `if else`), while-loops (`while`), and jumps (`goto`). A program is a sequence of (labeled) commands, and in each program there is a unique `halt` command which, when executed, causes program termination.

The semantics of our language is defined by a *transition relation*, denoted $\Longrightarrow$, between *configurations*. Each configuration is a pair $\langle\!\langle c, \delta \rangle\!\rangle$ of a command $c$ and an *environment* $\delta$. An environment $\delta$ is a function that maps: (i) every integer variable identifier $x$ to its value $v$, and (ii) every integer array identifier $a$ to a *finite* function from the set $\{0, \ldots, dim(a)-1\}$, where $dim(a)$ is the dimension of the array $a$, to the set of the integer numbers. The definition of the relation $\Longrightarrow$ is similar to the 'small step' operational semantics given in [35], and is omitted.

Given an imperative program *prog*, we address the problem of verifying whether or not, starting from any *initial configuration* that satisfies the property $\varphi_{init}$, the execution of *prog* eventually leads to a *final configuration* that satisfies the property $\varphi_{error}$, also called an *error configuration*. This problem is formalized by defining an *incorrectness triple* of the form $\{\!\{\varphi_{init}\}\!\}$ *prog* $\{\!\{\varphi_{error}\}\!\}$, where $\varphi_{init}$ and $\varphi_{error}$ are encoded by CLP predicates defined by (possibly recursive) clauses. We say that a program *prog* is *incorrect* with respect to $\varphi_{init}$ and $\varphi_{error}$, whose free variables are assumed to be among $z_1, \ldots, z_r$, if there

exist environments $\delta_{init}$ and $\delta_h$ such that: (i) $\varphi_{init}(\delta_{init}(z_1), \ldots, \delta_{init}(z_r))$ holds, (ii) $\langle\!\langle \ell_0 : c_0,\ \delta_{init} \rangle\!\rangle \Longrightarrow^* \langle\!\langle \ell_h : \texttt{halt},\ \delta_h \rangle\!\rangle$, and (iii) $\varphi_{error}(\delta_h(z_1), \ldots, \delta_h(z_r))$ holds, where $\ell_0 : c_0$ is the first labeled command of *prog* and $\ell_h : \texttt{halt}$ is the unique `halt` command of *prog*. A program is said to be *correct* with respect to $\varphi_{init}$ and $\varphi_{error}$ iff it is not incorrect with respect to $\varphi_{init}$ and $\varphi_{error}$. Note that this notion of correctness is equivalent to the usual notion of *partial correctness* specified by the Hoare triple $\{\varphi_{init}\}\ prog\ \{\neg\varphi_{error}\}$.

Our verification method is based on the formalization of the notion of program incorrectness by using a predicate `incorrect` defined by a CLP program.

In this paper a CLP program is a finite set of clauses of the form `A :- c,B`, where `A` is an atom, `c` is a constraint (that is, a possibly empty conjunction of linear equalities and inequalities over the integers), and `B` is a goal (that is, a possibly empty conjunction of atoms). The conjunction `c,B` is called a *constrained goal*. A clause of the form: `A :- c` is called a *constrained fact*. We refer to [21] for other notions of CLP with which the reader might be not familiar.

We translate the problem of checking whether or not the program *prog* is incorrect with respect to the properties $\varphi_{init}$ and $\varphi_{error}$ into the problem of checking whether or not the predicate `incorrect` is a consequence of the CLP program $T$ defined by the following clauses:

```
incorrect :- initConf(X), reach(X).
reach(X) :- tr(X,X1), reach(X1).
reach(X) :- errorConf(X).
```

together with the clauses for the predicates `initConf(X)`, `errorConf(X)`, and `tr(X,X1)`. They are defined as follows: (i) `initConf(X)` encodes an initial configuration satisfying the property $\varphi_{init}$, (ii) `errorConf(X)` encodes an error configuration satisfying the property $\varphi_{error}$, and (iii) `tr(X,X1)` encodes the transition relation $\Longrightarrow$. (Note that in order to define `initConf(X)`, `errorConf(X)`, and `tr(X,X1)` and, in particular, to represent operations over the integer variables and the elements of arrays, we need constraints.) The predicate `reach(X)` holds if an error configuration `Y` such that `errorConf(Y)` holds, can be reached from the configuration `X`.

The imperative program *prog* is correct with respect to the properties $\varphi_{init}$ and $\varphi_{error}$ iff $\texttt{incorrect} \notin M(T)$, where $M(T)$ denotes the *least model* of program $T$ [21]. Due to the presence of integer variables and array variables, $M(T)$ is in general an infinite model, and both the bottom-up and top-down evaluation of the query `incorrect` may not terminate. In order to deal with this difficulty, we propose an approach to program verification which is symbolic and, by using program transformations, allows us to avoid the exhaustive exploration of the possibly infinite space of reachable configurations.

Our verification method consists in applying to program $T$ a sequence of program transformations that preserve the least model $M(T)$ [13, 15]. In particular, we apply the following *transformation rules*, collectively called *unfold/fold rules*: (i) (*conjunctive*) *definition*, (ii) *unfolding*, (iii) *goal replacement*, (iv) *clause removal*, and (v) (*conjunctive*) *folding*. Our verification method is made out of the following two steps.

*Step* (A): *Removal of the Interpreter.* Program $T$ is *specialized* with respect to the given *prog* (on which `tr` depends), `initConf`, and `errorConf`, thereby deriving a new program $T1$ such that: (i) $\texttt{incorrect} \in M(T)$ iff $\texttt{incorrect} \in M(T1)$, and (ii) `tr` does not occur explicitly in $T1$ (in this sense we say that the interpreter is removed or compiled-away).

*Step* (B): *Propagation of the Initial and Error Properties.* By applying a sequence of unfold/fold transformation rules, the CLP program $T1$ is transformed into a new CLP program $T2$ such that the program *prog* is correct with respect to the given initial and error properties iff $\texttt{incorrect} \notin M(T2)$.

The objective of Step (B) is to propagate the initial and the error properties so as to derive a program $T2$ where the predicate `incorrect` is defined by either (i) the fact '`incorrect.`' (in which case *prog* is incorrect), or (ii) the empty set of clauses (in which case *prog* is correct). In the case where neither (i) nor (ii) holds, that is, in program $T2$ the predicate `incorrect` is defined by a non-empty set of clauses not containing the fact '`incorrect.`', we cannot conclude anything about the correctness of *prog* and, similarly to what has been proposed in [9], we iterate Step (B) in the hope of deriving a program where either (i) or (ii) holds. Obviously, due to undecidability limitations, it may be the case that we never get a program where either (i) or (ii) holds.

Steps (A) and (B) are both instances of the *Transform* strategy outlined in Figure 1 below. These two instances are obtained by using two different ways of controlling the application of the transformation rules. In particular, in the instance of the *Transform* strategy that realizes Step (A) we never apply the goal replacement rule and the resulting strategy coincides with the fully automatic specialization strategy presented in [9].

In the *Transform* strategy we make use of the following rules, where $P$ is the input CLP program, and *Defs* is a set of clauses, called *definition clauses*, constructed as we indicate in that strategy.

*Definition Rule.* By this rule we introduce a clause of the form `newp(X) :- c,G`, where `newp` is a new predicate symbol, `X` is a tuple of variables occurring in `(c,G)`, `c` is a constraint, and `G` is a non-empty conjunction of atoms.

*Unfolding Rule.* Given a clause $C$ of the form `H :- c,L,A,R`, where `H` and `A` are atoms, `c` is a constraint, and `L` and `R` are (possibly empty) conjunctions of atoms, let us consider the set $\{\texttt{K}_i \texttt{ :- } \texttt{c}_i\texttt{,B}_i \mid i = 1, \ldots, m\}$ made out of the (renamed apart) clauses of $P$ such that, for $i = 1, \ldots, m$, `A` is unifiable with $\texttt{K}_i$ via the most general unifier $\vartheta_i$ and $(\texttt{c},\texttt{c}_i)\,\vartheta_i$ is satisfiable (thus, the unfolding rule performs some constraint solving operations). By unfolding $C$ w.r.t. $A$ using $P$, we derive the set $\{(\texttt{H :- c,c}_i\texttt{,L,B}_i\texttt{,R})\,\vartheta_i \mid i = 1, \ldots, m\}$ of clauses.

*Goal Replacement Rule.* If a constrained goal $\texttt{c}_1, \texttt{G}_1$ occurs in the body of a clause $C$, and $M(P) \models \forall\,(\texttt{c}_1, \texttt{G}_1 \leftrightarrow \texttt{c}_2, \texttt{G}_2)$, then we derive a new clause $D$ by replacing $\texttt{c}_1, \texttt{G}_1$ by $\texttt{c}_2, \texttt{G}_2$ in the body of $C$.

The equivalences which are needed for goal replacements are called *laws* and their validity in $M(P)$ can be proved once and for all, before applying the *Transform* strategy.

*Input*: Program $P$.
*Output*: Program *TransfP* such that `incorrect` $\in M(P)$ iff `incorrect` $\in M(\mathit{TransfP})$.

INITIALIZATION:
Let *InDefs* be the set of all clauses of $P$ whose head is the atom `incorrect`;
$\mathit{TransfP} := \emptyset$; $\mathit{Defs} := \mathit{InDefs}$;

*while* in *InDefs* there is a clause $C$ *do*
  UNFOLDING: Apply the unfolding rule at least once using $P$, and derive from $C$ a set $U(C)$ of clauses;
  GOAL REPLACEMENT: Apply a sequence of goal replacements, and derive from $U(C)$ a set $R(C)$ of clauses;
  CLAUSE REMOVAL: Remove from $R(C)$ all clauses whose body contains an unsatisfiable constraint;
  DEFINITION & FOLDING: Introduce a (possibly empty) set *NewDefs* of new predicate definitions and add them to *Defs* and to *InDefs*;
  Fold the clauses in $R(C)$ different from constrained facts by using the clauses in *Defs*, and derive a set $F(C)$ of clauses;
  $\mathit{InDefs} := \mathit{InDefs} - \{C\}$; $\mathit{TransfP} := \mathit{TransfP} \cup F(C)$;
*end-while*;

REMOVAL OF USELESS CLAUSES:
Remove from *TransfP* all clauses whose head predicate is useless.

**Fig. 1.** The *Transform* strategy.

*Folding Rule.* Given a clause $E$ of the form: `H :- e, L, Q, R` and a clause $D$ in *Defs* of the form `K :- d, D` such that: (i) for some substitution $\vartheta$, $\mathtt{Q} = \mathtt{D}\,\vartheta$, and (ii) $\forall\,(\mathtt{e} \rightarrow \mathtt{d}\,\vartheta)$ holds, then by folding $E$ using $D$ we derive `H :- e, L,` $\mathtt{K}\,\vartheta$`, R`.

*Removal of Useless Clauses.* The set of *useless predicates* in a given program $Q$ is the greatest set $U$ of predicates occurring in $Q$ such that `p` is in $U$ iff every clause with head predicate `p` is of the form $\mathtt{p(X)}$ `:-` $\mathtt{c}, \mathtt{G}_1, \mathtt{q(Y)}, \mathtt{G}_2$, for some `q` in $U$. A clause in a program $Q$ is *useless* if the predicate of its head is useless in $Q$.

The termination of the *Transform* strategy is guaranteed by suitable techniques for controlling the unfolding and the introduction of new predicates. We refer to [28] for a survey of techniques which ensure the finiteness of unfolding. The introduction of new predicates is controlled by applying *generalization operators* based on various notions, such as *widening*, *convex hull*, *most specific generalization*, and *well-quasi ordering*, which have been proposed for analyzing and transforming CLP programs (see, for instance, [8, 11, 17, 32]).

The correctness of the strategy with respect to the least model semantics directly follows from the fact that the application of the transformation rules complies with some suitable conditions that guarantee the preservation of that model [13].

**Theorem 1.** (Termination and Correctness of the *Transform* strategy) (i) *The Transform strategy terminates.* (ii) *Let program TransfP be the output of the*

*Transform strategy applied on the input program $P$. Then,* `incorrect` $\in M(P)$ *iff* `incorrect` $\in M(\mathit{TransfP})$.

## 3 Verification of Recursively Defined Properties

In this section we will show, through an example, that our verification method can be used when the initial properties and the error properties are specified by (possibly recursive) CLP clauses, rather than by constraints only (as done, for instance, in [9]). In order to deal with that kind of properties, during the DEFINITION & FOLDING phase of the *Transform* strategy, we allow ourselves to introduce new predicates which are defined by clauses of the form: `Newp :- c, G`, where `Newp` is an atom with a new predicate symbol, `c` is a constraint, and `G` is a conjunction of *one or more* atoms. This kind of predicate definitions allows us to perform program verifications that cannot be done by the technique presented in [9], where the goal `G` is assumed to be a single atom.

Let us consider the following program *GCD* that computes the greatest common divisor $z$ of two positive integers $m$ and $n$, denoted $gcd(m, n, z)$.

*GCD*: $\ell_0$: $x = m$;
$\ell_1$: $y = n$;
$\ell_2$: `while` $(x \neq y)$ { `if` $(x > y)$ $x = x - y$; `else` $y = y - x$; };
$\ell_3$: $z = x$;
$\ell_h$: `halt`

We also consider the incorrectness triple $\{\!\{\varphi_{init}(m, n)\}\!\}$ *GCD* $\{\!\{\varphi_{error}(m, n, z)\}\!\}$, where:
(i) $\varphi_{init}(m, n)$ is $m \geq 1 \wedge n \geq 1$, and (ii) $\varphi_{error}(m, n, z)$ is $\exists\, d\, (gcd(m, n, d) \wedge d \neq z)$. These properties $\varphi_{init}$ and $\varphi_{error}$ are defined by the following CLP clauses 1 and 2–5, respectively:

1. `phiInit(M,N) :- M≥1, N≥1.`
2. `phiError(M,N,Z) :- gcd(M,N,D), D≠Z.`
3. `gcd(X,Y,D) :- X>Y, X1=X−Y, gcd(X1,Y,D).`
4. `gcd(X,Y,D) :- X<Y, Y1=Y−X, gcd(X,Y1,D).`
5. `gcd(X,Y,D) :- X=Y, Y=D.`

The predicates `initConf` and `errorConf` specifying the initial and the error configurations, respectively, are defined by the following clauses:

6. `initConf(cf(cmd(0,asgn(int(x),int(m))),`
   `[[int(m),M],[int(n),N],[int(x),X],[int(y),Y],[int(z),Z]]))`
   `:- phiInit(M,N).`
7. `errorConf(cf(cmd(h,halt),`
   `[[int(m),M],[int(n),N],[int(x),X],[int(y),Y],[int(z),Z]])) :-`
   `phiError(M,N,Z).`

Thus, the CLP program encoding the given incorrectness triple consists of clauses 1–7 above, together with the clauses defining the predicates `incorrect`, `reach`, and `tr` given as indicated in Section 2.

Now we perform Step (A) of our verification method, which consists in the removal of the interpreter, and we derive the following CLP program:

8. incorrect :- M≥1, N≥1, X=M, Y=N, new1(M,N,X,Y,Z).
9. new1(M,N,X,Y,Z) :- X>Y, X1=X−Y, new1(M,N,X1,Y,Z).
10. new1(M,N,X,Y,Z) :- X<Y, Y1=Y−X, new1(M,N,X,Y1,Z).
11. new1(M,N,X,Y,Z) :- X=Y, Z=X, Z≠D, gcd(M,N,D).

By moving the constrained atom 'Z≠D, gcd(M,N,D)' from the body of clause 11 to the body of clause 8, we can rewrite clauses 8 and 11 as follows (this rewriting is correct because in clauses 9 and 10 the predicate new1 modifies neither the value of M nor the value of N):

8r. incorrect :- M≥1, N≥1, X=M, Y=N, Z≠D, gcd(M,N,D), new1(M,N,X,Y,Z).
11r. new1(M,N,X,Y,Z) :- X=Y, Z=X.

Note that we could avoid performing the above rewriting and obtain a similar program where the constraints characterizing the initial and the error properties occur in the same clause by starting our derivation from a more general definition of the reachability relation. However, an in-depth analysis of this variant of our verification method is beyond the scope of this paper.

Now we will perform Step (B) of the verification method by applying the *Transform* strategy to the derived program consisting of clauses $\{3, 4, 5, 8r, 9, 10, 11r\}$. Initially, we have that the sets *InDefs* and *Defs* of definition clauses are both equal to $\{8r\}$.

UNFOLDING. We start off by unfolding clause 8r w.r.t. the atom new1(M,N,X,Y,Z), and we get:

12. incorrect :- M≥1, N≥1, X=M, Y=N, X>Y, X1=X−Y, Z≠D, gcd(M,N,D), new1(M,N,X1,Y,Z).
13. incorrect :- M≥1, N≥1, X=M, Y=N, X<Y, Y1=Y−X, Z≠D, gcd(M,N,D), new1(M,N,X,Y1,Z).
14. incorrect :- M≥1, N≥1, X=M, Y=N, X=Y, Z=X, Z≠D, gcd(M,N,D).

By unfolding clauses 12, 13, and 14 w.r.t. the atom gcd(M,N,D), we derive:

15. incorrect :- M≥1, N≥1, M>N, X1=M−N, Z≠D, gcd(X1,N,D), new1(M,N,X1,N,Z).
16. incorrect :- M≥1, N≥1, M<N, Y1=N−M, Z≠D, gcd(M,Y1,D), new1(M,N,M,Y1,Z).

(The unfolding of clause 14 produces the empty set of clauses because the constraint 'X=M, Z=X, Z≠D, M=D' is unsatisfiable.) The GOAL REPLACEMENT and CLAUSE REMOVAL phases leave the set of clauses produced by the UNFOLDING phase unchanged, because no laws are available for the predicate gcd.

DEFINITIONS & FOLDING. In order to fold clauses 15 and 16, we perform a generalization step and we introduce a new predicate defined by the following clause:

17. new2(M,N,X,Y,Z,D) :- M≥1, N≥1, Z≠D, gcd(X,Y,D), new1(M,N,X,Y,Z).

The body of this clause 17 is the most specific generalization of the bodies of clause 8r (which is the only clause in *Defs*), and clauses 15 and 16 (which are the

clauses to be folded). Now, clauses 15 and 16 can be folded by using clause 17, thereby deriving:

18. `incorrect :- M≥1, N≥1, M>N, X1=M−N, Z≠D, new2(M,N,X1,N,Z,D).`
19. `incorrect :- M≥1, N≥1, M<N, Y1=N−M, Z≠D, new2(M,N,M,Y1,Z,D).`

Clause 17 defining the new predicate `new2` is added to *Defs* and *InDefs* and, since the latter set is not empty, we perform a new iteration of the while-loop body of the *Transform* strategy.

UNFOLDING. By unfolding clause 17 w.r.t. `new1(M,N,X,Y,Z)` and then unfolding the resulting clauses w.r.t. `gcd(X,Y,Z)`, we derive:

20. `new2(M,N,X,Y,Z,D) :- M≥1, N≥1, X>Y, X1=X−Y, Z≠D, gcd(X1,Y,D), new1(M,N,X1,Y,Z).`
21. `new2(M,N,X,Y,Z,D) :- M≥1, N≥1, X<Y, Y1=Y−X, Z≠D, gcd(X,Y1,D), new1(M,N,X,Y1,Z).`

DEFINITION & FOLDING. Clauses 20 and 21 can be folded by using clause 17, and we derive:

22. `new2(M,N,X,Y,Z,D) :- M≥1, N≥1, X>Y, X1=X−Y, Z≠D, new2(M,N,X1,Y,Z).`
23. `new2(M,N,X,Y,Z,D) :- M≥1, N≥1, X<Y, Y1=Y−X, Z≠D, new2(M,N,X,Y1,Z).`

No new predicate definition is introduced, and the *Transform* strategy exits the while-loop. The final program *TransfP* is the set $\{18, 19, 22, 23\}$ of clauses, which contains no constrained facts. Hence both predicates `incorrect` and `new2` are useless and all clauses of *TransfP* can be removed. Thus, the *Transform* strategy terminates with *TransfP*$=\emptyset$ and we conclude that the imperative program *GCD* is correct w.r.t. the given initial and error properties.

## 4 Verification of Array Programs

In this section we apply our verification method to the following program *ArrayMax* which computes the maximal element of an array:

*ArrayMax*: $\ell_0$: $i = 0$;
$\ell_1$: `while` $(i<n)$ { `if` $(a[i] > max)$ $max = a[i]$;
$i = i+1$; };
$\ell_h$: `halt`

We consider the following incorrectness triple:

$\{\!\{\varphi_{init}(i,n,a,max)\}\!\}$ *ArrayMax* $\{\!\{\varphi_{error}(n,a,max)\}\!\}$

where: (i) $\varphi_{init}(i,n,a,max)$ is $i \geq 0 \wedge n = dim(a) \wedge n \geq i+1 \wedge max = a[i]$, and (ii) $\varphi_{error}(n,a,max)$ is $\exists k\, (0 \leq k < n \wedge a[k] > max)$.

First, we construct a CLP program $T$ which encodes the above incorrectness triple, similarly to what has been done in Section 3. In particular, the properties $\varphi_{init}$ and $\varphi_{error}$ are defined by the following CLP clauses, respectively:

1. `phiInit(I,N,A,Max) :- I≥0, N≥I+1, read((A,N),I,Max).`
2. `phiError(N,A,Max) :- K≥0, N>K, Z>Max, read((A,N),K,Z).`

The clauses defining the predicates `initConf(X)` and `errorConf(X)` which specify the initial and the error configurations, respectively, are as follows:

3. initConf(cf(cmd(0, asgn(int(i), int(0))),
   [[int(i), I], [int(n), N], [array(a), (A, N)], [int(max), Max]])) :-
   phiInit(I, N, A, Max).
4. errorConf(cf(cmd(h, halt),
   [[int(i), I], [int(n), N], [array(a), (A, N)], [int(max), Max]])) :-
   phiError(N, A, Max).

Now we start off by applying Step (A) of our verification method which consists in the removal of the interpreter. From program $T$ we obtain the following program $T1$:

5. incorrect :- I=0, N≥1, read((A, N), I, Max), new1(I, N, A, Max).
6. new1(I, N, A, Max) :- I1=I+1, I<N, I≥0, M>Max, read((A, N), I, M),
   new1(I1, N, A, M).
7. new1(I, N, A, Max) :- I1=I+1, I<N, I≥0, M≤Max, read((A, N), I, M),
   new1(I1, N, A, Max).
8. new1(I, N, A, Max) :- I≥N, K≥0, N>K, Z>Max, read((A, N), K, Z).

As indicated in [9], in order to propagate the error property, we 'reverse' the derived program $T1$ and we get the following program $T1_{rev}$:

rev1. incorrect :- b(U), r2(U).
rev2. r2(V) :- trans(U, V), r2(U).
rev3. r2(U) :- a(U).

where the predicates a, b, and trans are defined as follows:

s4. a([new1, I, N, A, Max]) :- I=0, N≥1, read((A, N), I, Max)
s5. trans([new1, I, N, A, Max], [new1, I1, N, A, M]) :-
   I1=I+1, I<N, I≥0, M>Max, read((A, N), I, M).
s6. trans([new1, I, N, A, Max], [new1, I1, N, A, Max]) :-
   I1=I+1, I<N, I≥0, M≤Max, read((A, N), I, M).
s7. b([new1, I, N, A, Max]) :- I≥N, K≥0, K<N, Z>Max, read((A, N), K, Z).

This reversal transformation, which from program $T1$ derives program $T1_{rev}$, can easily be automatedand it is correct in the sense that incorrect $\in M(T1)$ iff incorrect $\in M(T1_{rev})$. This equivalence holds because: (i) in program $T1$ the predicate incorrect is defined in terms of the predicate new1 that encodes the reachability relation from an error configuration to an initial configuration, and (ii) in program $T1_{rev}$ the predicate incorrect is defined in terms of the predicate r2 that also encodes the reachability relation, but this time the encoding is, so to speak, 'in the reversed direction', that is, from an initial configuration to an error configuration.

Now let us apply Step (B) of our verification method starting from the program $T1_{rev}$.

UNFOLDING. First we unfold clause rev1 w.r.t. the atom b(U), and we get:

9. incorrect :- I≥N, K≥0, K<N, Z>Max, read((A, N), K, Z),
   r2([new1, I, N, A, Max]).

Neither GOAL REPLACEMENT nor CLAUSE REMOVAL is applied.

DEFINITION & FOLDING. In order to fold clause 9 we introduce the following clause:

10. $\mathtt{new2(I,N,A,Max,K,Z)}$ :- $\mathtt{I \geq N,\ K \geq 0, K < N, Z > Max,\ read((A,N),K,Z),}$
$\mathtt{r2([new1,I,N,A,Max])}$.

By folding clause 9 using clause 10, we get:

11. $\mathtt{incorrect}$ :- $\mathtt{I \geq N,\ K \geq 0, K < N,\ Z > Max,\ new2(I,N,A,Max,K,Z)}$.

Now we proceed by performing a second iteration of the body of the while-loop of the *Transform* strategy because *InDefs* is not empty (indeed, clause 10 belongs to *InDefs*).

UNFOLDING. After some unfoldings from clause 10 we get the following clauses:

12. $\mathtt{new2(I1,N,A,M,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I1,\ M > Max,\ Z > M,}$
$\mathtt{read((A,N),K,Z),\ read((A,N),I,M),\ r2([new1,I,N,A,Max])}$.

13. $\mathtt{new2(I1,N,A,Max,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I1,\ M \leq Max,\ Z > Max,}$
$\mathtt{read((A,N),K,Z),\ read((A,N),I,M),\ r2([new1,I,N,A,Max])}$.

GOAL REPLACEMENT. We use the following law which is a consequence of the fact that arrays are finite functions:

(L1) $\mathtt{read((A,N),K,Z),\ read((A,N),I,M)} \leftrightarrow$
$(\mathtt{K = I,\ Z = M,\ read((A,N),K,Z)}) \vee (\mathtt{K \neq I,\ read((A,N),K,Z),\ read((A,N),I,M)})$

Thus, (i) we replace the conjunction of atoms '$\mathtt{read((A,N),K,Z),\ read((A,N),I,M)}$' occurring in the body of clause 12 by the right hand side of law (L1), and then (ii) we split the derived clause with disjunctive body into the following two clauses, each of which corresponds to a disjunct of the right hand side of (L1). We get the following clauses:

12.1 $\mathtt{new2(I1,N,A,M,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I1,\ M > Max,\ Z > M,}$
$\mathtt{K = I,\ M = Z,\ read((A,N),K,Z),\ r2([new1,I,N,A,Max])}$.

12.2 $\mathtt{new2(I1,N,A,M,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I1,\ M > Max,\ Z > M,}$
$\mathtt{K \neq I,\ read((A,N),K,Z),\ read((A,N),I,M),\ r2([new1,I,N,A,Max])}$.

CLAUSE REMOVAL. The constraint '$\mathtt{Z > M, M = Z}$' in the body of clause 12.1 is unsatisfiable. Hence, this clause is removed from *TranfP*. By simplifying the constraints in clause 12.2 we get:

14. $\mathtt{new2(I1,N,A,M,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I,\ M > Max,\ Z > M,}$
$\mathtt{read((A,N),K,Z),\ read((A,N),I,M),\ r2([new1,I,N,A,Max])}$.

By applying similar goal replacements and clause removals, from clause 13 we get:

15. $\mathtt{new2(I1,N,A,Max,K,Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I,\ M \leq Max,\ Z > Max,}$
$\mathtt{read((A,N),K,Z),\ read((A,N),I,M),\ r2([new1,I,N,A,Max])}$.

DEFINITION & FOLD. In order to fold clause 14, we introduce the following definition:

16. $\mathtt{new3(I,N,A,Max,K,Z)}$ :- $\mathtt{K \geq 0, K < N, K < I, Z > Max,\ read((A,N),K,Z)),}$
$\mathtt{r2([new1,I,N,A,Max])}$.

Clause 16 is obtained from clauses 10 and 14 by applying a generalization operator called *WidenSum* [17], which is a variant of the classical widening operator [6]. Clause 16 can be used also for folding clause 15, and by folding clauses 14 and 15 using clause 16, we get:

17. $\mathtt{new2(I1, N, A, Max, K, Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I,\ M > Max,\ Z > M,}$
$\mathtt{read((A, N), I, M),\ new3(I, N, A, Max, K, Z)}$.
18. $\mathtt{new2(I1, N, A, M, K, Z)}$ :- $\mathtt{I1 = I+1,\ N = I1,\ K \geq 0,\ K < I,\ M \leq Max,\ Z > Max,}$
$\mathtt{read((A, N), I, M),\ new3(I, N, A, Max, K, Z)}$.

Now we perform the third iteration of the body of the while-loop of the strategy. After some unfolding, goal replacement, clause removal, and folding steps, from clause 16 we get:

19. $\mathtt{new3(I1, N, A, M, K, Z)}$ :- $\mathtt{I1 = I+1,\ K \geq 0,\ K < I,\ N \geq I1,\ M > Max,\ Z > M,}$
$\mathtt{read((A, N), I, M),\ new3(I, N, A, Max, K, Z)}$.
20. $\mathtt{new3(I1, N, A, Max, K, Z)}$ :- $\mathtt{I1 = I+1,\ K \geq 0,\ K < I,\ N \geq I1,\ M \leq Max,\ Z > Max,}$
$\mathtt{read((A, N), I, M),\ new3(I, N, A, Max, K, Z)}$.

Since we did not introduce any new definition, and no clause remains to be processed (indeed, the set *InDefs* of definitions is empty), the *Transform* strategy exits the while-loop and we get the program consisting of the set $\{11, 17, 18, 19, 20\}$ of clauses.

Since no clause in this set is a constrained fact, by the final phase of removing the useless clauses we get a final program consisting of the empty set of clauses. Thus, the program *ArrayMax* is correct with respect to the given $\varphi_{init}$ and $\varphi_{error}$ properties.

## 5 Related Work and Conclusions

The verification method presented in this paper is an extension of the one introduced in [9], where Constraint Logic Programming (CLP) and iterated specialization have been used to define a general verification framework that is parametric with respect to the programming language and the logic used for specifying the correctness properties. The main novelties of this paper are the following ones: (i) we have considered imperative programs acting on integer variables as well as array variables, and (ii) we have allowed a more expressive specification language, in which one can write properties about elements of arrays and, in general, elements of complex data structures.

In order to deal with this more general setting, we have defined the operational semantics of array manipulation, and we have also considered powerful transformation rules, such as conjunctive definition, conjunctive folding, and goal replacement. These transformation rules together with some strategies for guiding their application, have been implemented in the MAP transformation system [29], so that the proofs of program correctness have been performed in a semi-automatic way.

The idea of encoding imperative programs into logic programs for reasoning about the properties of those imperative programs is not novel. In particular,

for instance, this encoding has been recently used for reasoning about the type system of Featherweight Java programs in [1]. The use of constraint-based techniques for program verification is not novel either.Indeed, CLP programs have been successfully applied to perform model checking of both finite and infinite state systems [12, 14, 17] because through CLP programs one can express in a simple manner both (i) the symbolic executions of imperative programs and (ii) the invariants which hold during their executions. Moreover, there are powerful CLP-based tools, such as ARMC [34], TRACER [22], and HSF [20], that can be used for performing model checking of imperative programs. These tools are fully automatic, but they are applicable to classes of programs and properties that are much more limited than those considered in this paper. We have shown in [9] that, by focusing on verification tasks similar to those considered by ARMC, TRACER, and HSF, we can design a fully automatic, transformation-based verification technique whose effectiveness is competitive to the one of the above mentioned tools.

Our rule-based program transformation technique is also related to *conjunctive partial deduction* (CPD) [11], a technique for the specialization of logic programs with respect to conjunctions of atoms. There are, however, some substantial differences between CPD and the approach we have presented here. First, CPD is not able to specialize logic programs with constraints and, thus, it cannot be used to prove the correctness of the *GCD* program where the role of constraints is crucial. Indeed, using the ECCE conjunctive partial deduction system [27] for specializing the program consisting of clauses {3, 4, 5, 8r, 9, 10, 11r} with respect to the query `incorrect`, we obtain a residual program where the predicate `incorrect` is not useless. Thus, we cannot conclude that the atom `incorrect` does not belong to the least model of the program, and thus we cannot conclude that the program is correct. One more difference between CPD and our technique is that we may use goal replacement rules which allow us to evaluate terms over domain-specific theories. In particular, we can apply the goal replacement rules using well-developed theories for data structures like arrays, lists, heaps and sets (see [4, 30, 19, 3, 36, 39] for some formalizations of these theories).

An alternative, systemic approach to program transformation is supercompilation [38], which considers programs as machines. A supercompiler runs a program and, while it observes its behavior, produces an equivalent program without performing stepwise transformations of the original program.

The verification method we have presented in this paper is also related to several other methods for verifying properties of imperative programs acting on arrays. Those methods use techniques based on abstract interpretation, theorem proving and, in particular, Satisfiability Modulo Theory (see, for instance, [7, 25, 26]).

The application of the powerful transformation rules we have considered in this paper enables us to verify larger classes of properties, but the strategies to be applied for dealing with those classes are not all instances of the automated strategy introduced in [9].

In the future we intend to consider the issue of designing fully mechanizable strategies for guiding the application of our program transformation rules. In particular, we want to study the problem of devising suitable unfolding strategies and generalization operators, by adapting the techniques already developed for program transformation. We also envisage that the application of the laws used by the goal replacement rule can be automated by importing in our framework the techniques used in the fields of Theorem Proving and Term Rewriting. For some specific theories we could also apply the goal replacement rule by exploiting the results obtained by external theorem provers or Satisfiability Modulo Theory solvers.

We also plan to address the issue of proving correctness of programs acting on *dynamic data structures* such as lists or heaps, looking for a set of suitable goal replacement laws which axiomatize those structures.

## Acknowledgements

We would like to thank the anonymous referees for their helpful comments and constructive criticism.

## References

1. D. Ancona and G. Lagorio. Coinductive Type Systems for Object-Oriented Languages. In *Proceedings of the 23th European Conference on Object-Oriented Programming. ECOOP'09*, Lecture Notes in Computer Science 5653, pages 2–26. Springer, 2009.
2. D. Beyer, T. A. Henzinger, R. Majumdar, and A. Rybalchenko. Invariant synthesis for combined theories. In *Proceedings of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation. VMCAI'07*, Lecture Notes in Computer Science 4349, pages 378–394. Springer, 2007.
3. R. S. Bird. An introduction to the theory of lists. In *Proceedings of the NATO Advanced Study Institute on Logic of programming and calculi of discrete design*, pages 5–42. Springer-Verlag New York, Inc., 1987.
4. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation. VMCAI'06, Charleston, SC, USA*, Lecture Notes in Computer Science 3855. Springer, 2006.
5. R. M. Burstall and J. Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *Proceedings of the 4th ACM-SIGPLAN Symposium on Principles of Programming Languages, POPL'77*, pages 238–252. ACM Press, 1977.
7. P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. In *Proceedings of the 38th ACM Symposium on Principles of programming languages. POPL'11*, pages 105–118, 2011.

8. 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.
9. E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying Programs via Iterated Specialization. In *Proceedings of the ACM SIGPLAN 2013 Workshop on Partial Evaluation and Program Manipulation, PEPM'13*, pages 43–52, 2013.
10. E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Verification of Imperative Programs by Constraint Logic Program Transformation. In *Semantics, Abstract Interpretation, and Reasoning about Programs, SAIRP'13*, EPTCS 129, pages 186–210, 2013.
11. D. De Schreye, R. Glück, J. Jørgensen, M. Leuschel, B. Martens, and M. H. Sørensen. Conjunctive partial deduction: Foundations, control, algorithms, and experiments. *Journal of Logic Programming*, 41(2–3):231–277, 1999.
12. G. Delzanno and A. Podelski. Model checking in CLP. In R. Cleaveland, ed., *5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'99*, Lecture Notes in Computer Science 1579, pages 223–239. Springer-Verlag, 1999.
13. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.
14. F. Fioravanti, A. Pettorossi, and M. Proietti. Verifying CTL properties of infinite state systems by specializing constraint logic programs. In *Proceedings of the ACM SIGPLAN Workshop on Verification and Computational Logic VCL'01, Florence, Italy*, Technical Report DSSE-TR-2001-3, pages 85–96. University of Southampton, UK, 2001.
15. F. Fioravanti, A. Pettorossi, and M. Proietti. Transformation Rules for Locally Stratified Constraint Logic Programs. In K.-K. Lau and M. Bruynooghe, eds *Program Development in Computational Logic*, Lecture Notes in Computer Science 3049, pages 292–340. Springer, 2004.
16. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Improving reachability analysis of infinite state systems by specialization. In G. Delzanno and I. Potapov, eds., *Proceedings of the 5th International Workshop on Reachability Problems, RP'11, Genoa, Italy*, Lecture Notes in Computer Science 6945, pages 165–179. Springer, 2011.
17. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Generalization strategies for the verification of infinite state systems. *Theory and Practice of Logic Programming. Special Issue on the 25th Annual GULP Conference*, 13(2):175–199, 2013.
18. C. Flanagan. Automatic software model checking via constraint logic. *Sci. Comput. Program.*, 50(1–3):253–270, 2004.
19. S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Decision procedures for extensions of the theory of arrays. *Ann. Math. Artif. Intell.*, 50(3-4):231–254, 2007.
20. S. Grebenshchikov, A. Gupta, N. P. Lopes, C. Popeea, and A. Rybalchenko. HSF(C): A Software Verifier based on Horn Clauses. In *Proc. of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'12*, Lecture Notes in Computer Science 7214, pages 549–551. Springer, 2012.
21. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
22. J. Jaffar, V. Murali, J. A. Navas, and A. E. Santosa. TRACER: A symbolic execution tool for verification. In *CAV'12*, Lecture Notes in Computer Science 7358, pages 758–766. Springer, 2012.

23. R. Jhala and R. Majumdar. Software model checking. *ACM Computing Surveys*, 41(4):21:1–21:54, 2009.
24. N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall, 1993.
25. L. Kovács and A. Voronkov. Finding loop invariants for programs over arrays using a theorem prover. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering. FASE'09*, Lecture Notes in Computer Science 5503, pages 470–485. Springer, 2009.
26. D. Larraz, E. Rodríguez-Carbonell, and A. Rubio. SMT-based array invariant generation. In *14th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI'13, Rome, Italy*, Lecture Notes in Computer Science 7737, pages 169–188. Springer, 2013.
27. M. Leuschel. The ECCE partial deduction system and the DPPD library of benchmarks, Release 3, Nov. 2000. Available from `http://www.ecs.soton.ac.uk/~mal`.
28. 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.
29. MAP: The MAP transformation system. `http://www.iasi.cnr.it/~proietti/system.html`. Via a WEB interface: `http://www.map.uniroma2.it/mapweb`.
30. J. McCarthy. A basis for a mathematical theory of computation. In *Computer Programming and Formal Systems*, pages 33–70. North-Holland, 1963.
31. J. McCarthy. Towards a mathematical science of computation. In C. Popplewell, ed., *Information Processing. Proceedings of IFIP 1962*, pages 21–28, Amsterdam, 1963. North Holland.
32. J. C. Peralta and J. P. Gallagher. Convex hull abstractions in specialization of CLP programs. In M. Leuschel, ed., *Logic Based Program Synthesis and Tranformation, 12th International Workshop, LOPSTR'02, Madrid, Spain, Revised Selected Papers*, Lecture Notes in Computer Science 2664, pages 90–108, 2003.
33. J. C. Peralta, J. P. Gallagher, and H. Saglam. Analysis of Imperative Programs through Analysis of Constraint Logic Programs. In G. Levi, ed., *Static Analysis, 5th International Symposium, SAS'98, Pisa, Italy*, Lecture Notes in Computer Science 1503, pages 246–261. Springer, 1998.
34. A. Podelski and A. Rybalchenko. ARMC: The Logical Choice for Software Model Checking with Abstraction Refinement. In M. Hanus, ed., *Practical Aspects of Declarative Languages, PADL'07*, Lecture Notes in Computer Science 4354, pages 245–259. Springer, 2007.
35. C. J. Reynolds. *Theories of Programming Languages*. Cambridge University Press, 1998.
36. J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science LICS'02*, pages 55–74. IEEE Computer Society, 2002.
37. H. Tamaki and T. Sato. Unfold/fold transformation of logic programs. In S.-Å. Tärnlund, ed., *Proceedings of the Second International Conference on Logic Programming, ICLP'84*, pages 127–138, Uppsala, Sweden, 1984. Uppsala University.
38. V. F. Turchin. The concept of a supercompiler. *ACM TOPLAS*, 8(3):292–325, 1986.
39. M. Wirsing. Algebraic specification. In J. Van Leeuwen, ed., *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier, 1990.

# A semantics for Rational Closure: Preliminary Results

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

[1] DISIT - Università del Piemonte Orientale - Alessandria, Italy - laura@mfn.unipmn.it
[2] Dip. di Informatica - Univ. di Torino - Italy {gliozzi,pozzato}@di.unito.it
[3] Aix-Marseille Univ. - CNRS, LSIS UMR 7296 - France - {nicola.olivetti}@univ-amu.fr

**Abstract.** We provide a semantical reconstruction of rational closure. We first consider rational closure as defined by Lehman and Magidor for propositional logic, and we provide a semantical characterization based on minimal models mechanism on rational models. Then, we extend the whole formalism and semantics to Description Logics focusing our attention to the standard $\mathcal{ALC}$: we first naturally adapt to Description Logics Lehman and Magidor's propositional rational closure, starting from an extension of $\mathcal{ALC}$ with a typicality operator $\mathbf{T}$ that selects the most typical instances of a concept $C$ (hence $\mathbf{T}(C)$ stands for typical $C$s). Then, we provide for $\mathcal{ALC}$ plus $\mathbf{T}$ a semantical characterization similar to the one for propositional logic. Last, we extend the notion of rational closure to the ABox.

## 1 Introduction

In [18] Kraus, Lehmann and Magidor (henceforth KLM) proposed a set of natural properties of non-monotonic reasoning. Plausible inferences are represented by non-monotonic conditionals of the form $A \mathrel{|\!\sim} B$, to be read as "typically or normally $A$ entails $B$": for instance $monday \mathrel{|\!\sim} go\_work$ can be used to represent that "normally if it is Monday I go to work". Conditional entailment is non-monotonic since from $A \mathrel{|\!\sim} B$ one cannot derive $A \wedge C \mathrel{|\!\sim} B$, in our example from $monday \mathrel{|\!\sim} go\_work$ one cannot monotonically derive $monday \wedge ill \mathrel{|\!\sim} go\_work$ ("normally if it is Monday, even if I am ill I go to work"). KLM organized the core properties of non-monotonic reasoning into a hierarchy of systems, from the weakest to the strongest: cumulative logic **C**, loop-cumulative logic **CL**, preferential logic **P**. Preferential logic has been strengthened into rational logic **R** in [20]. In this work, we restrict our attention to the rational logic **R** on which rational closure is built.

KLM system **R** formalizes desired properties of non-monotonic inference but it is too weak to perform useful non-monotonic inferences. We have just seen that by the non-monotonicity of $\mathrel{|\!\sim}$, $A \mathrel{|\!\sim} B$ does not entail $A \wedge C \mathrel{|\!\sim} B$, and this is a wanted property of $\mathrel{|\!\sim}$. However, there are cases in which, in the absence of information to the contrary, we want to be able to tentatively infer that also $A \wedge C \mathrel{|\!\sim} B$, with the possibility of withdrawing the inference in case we discovered that it is inconsistent. For instance, we might want to infer that $A \wedge C \mathrel{|\!\sim} B$ when $C$ is irrelevant with respect to the property $B$: we might want to tentatively infer from $monday \mathrel{|\!\sim} go\_work$ that $monday \wedge shines \mathrel{|\!\sim} go\_work$ ("normally if it is Monday, even if the sun shines I go to work"), with the possibility of withdrawing the conclusion if we discovered that indeed the sun shining prevents from going to work. **R** cannot handle irrelevant information in conditionals, and the inferences just exemplified are not supported.

Partially motivated by this weakness, Lehmann and Magidor have proposed a true non-monotonic mechanism on the top of **R**. Rational closure on the one hand preserves

the properties of **R**, on the other hand it allows to perform some truthful non-monotonic inferences, like the one just mentioned ($monday \wedge shines \mathrel{|\!\sim} go\_work$). In [20] the authors give a syntactic procedure to calculate the set of conditionals entailed by the rational closure as well as a quite complex semantic construction. It is worth noticing that a strongly related construction has been proposed by Pearl [22] with his notion of 1-entailment, motivated by a probabilistic interpretation of conditionals.

The first problem we tackle in this work is that of giving a purely semantic characterization of the syntactic notion of rational closure. Our semantic characterization has as its main ingredient the modal semantics of logic **R**, over which we build a minimal models' mechanism, based on the minimization of the *rank* of worlds. Intuitively, we prefer the models that minimize the rank of domain elements: the lower the rank of a world, the more normal (or less exceptional) is the world and our minimization corresponds intuitively to the idea of minimizing less-plausible worlds (or maximizing most plausible ones). We show that a semantic reconstruction of rational closure can be given in terms of a specific case of a general semantic framework for non-monotonic reasoning.

In the second part of the paper we consider Description Logics (DLs for short). A large amount of discussion has recently been done in order to extend the basic formalism of DLs with non-monotonic reasoning features [1, 2, 4, 6, 7, 14, 19, 17, 3, 21]; the purpose of these extensions is that of allowing reasoning about prototypical properties of individuals or classes of individuals. In spite of the load of work in this direction, finding a solution to the problem of extending DLs for reasoning about prototypical properties seems far from being solved. The best known semantics for non-monotonic reasoning have been used to the purpose, from default logic [1], to circumscription [2], from Lifschitz's non-monotonic logic MKNF [6, 21] to KLM logics. Concerning KLM logics, in [10] a preferential extension of $\mathcal{ALC}$ is defined, based on the logic **P**, and in [14] a minimal model semantics for this logic is proposed; in [3], a defeasible description logic based on the logic **R** is introduced and, in [4], a notion of rational closure is defined for $\mathcal{ALC}$ through an algorithmic construction similar to the one introduced by Freund for the propositional calculus. Although [4] provides axiomatic properties of this notion of rational closure, it does not provide a semantics for it.
We here extend to $\mathcal{ALC}$ the definition of rational closure by Lehmann and Magidor [20] and define a minimal model semantics for rational closure in $\mathcal{ALC}$ by adapting the semantics introduced in the propositional case. We start from the extension of the description logic $\mathcal{ALC}$ with a typicality operator $\mathbf{T}$, first proposed in [10], that allows to directly express typical properties such as $\mathbf{T}(HeartPosition) \sqsubseteq Left$, $\mathbf{T}(Bird) \sqsubseteq Fly$, and $\mathbf{T}(Penguin) \sqsubseteq \neg Fly$, whose intuitive meaning is that normally, the heart is positioned in the left-hand side of the chest, that typical birds fly, whereas penguins do not. In this paper, the $\mathbf{T}$ operator is intended to enjoy the well-established properties of rational logic **R**. Even if $\mathbf{T}$ is a non-monotonic operator (so that for instance $\mathbf{T}(HeartPosition) \sqsubseteq Left$ does not entail that $\mathbf{T}(HeartPosition \sqcap SitusInversus) \sqsubseteq Left$) the logic itself is monotonic. Indeed, in this logic it is not possible to monotonically infer from $\mathbf{T}(Bird) \sqsubseteq Fly$, in the absence of information to the contrary, that also $\mathbf{T}(Bird \sqcap Black) \sqsubseteq Fly$. Nor it can non-monotonically be inferred from $Bird(tweety)$, in the absence of information to the contrary, that $\mathbf{T}(Bird)(tweety)$ and that $Fly(tweety)$. Non-monotonicity is achieved, from a semantic point of view, by defining, on the top of $\mathcal{ALC}$ with typicality, a minimal

model semantics which is similar to the one in [14], with the difference that the notion of minimality is based on the minimization of the ranks of the worlds, rather than on the minimization of specific formulas, as in [14]. This semantics provides a characterization to the rational closure construction for $\mathcal{ALC}$, which assigns a *rank* (a level of exceptionality) to every concept; this rank is used to evaluate defeasible inclusions of the form $\mathbf{T}(C) \sqsubseteq D$: the inclusion is supported by the rational closure whenever the rank of $C$ is strictly smaller than the one of $C \sqcap \neg D$.

Last, we tackle the problem of extending rational closure to ABox reasoning: in order to ascribe defeasible properties to individuals we maximize their typicality. This is done by minimizing their ranks (that is, their level of exceptionality). Because of the interaction between individuals (due to roles) it is not possible to separately assign a unique minimal rank to each individual and alternative minimal ranks must be considered. We end up with a kind of *skeptical* inference with respect to the ABox.

The rational closure construction that we propose has not just a theoretical interest and a simple minimal model semantics, we show that it is also *feasible*. Its complexity is ExpTime in the size of the knowledge base (and the query), the same complexity as the underlying logic $\mathcal{ALC}$. In this respect it is less complex than other approaches to non-monotonic reasoning in DLs [14, 2] and comparable with the approaches in [4, 21], and thus a good candidate to define effective non-monotonic extensions of DLs.

## 2 Propositional rational closure: a semantic characterization

### 2.1 KLM rational system R

The language of logic **R** consists just of conditional assertions $A \mathrel{|\!\sim} B$. Here we consider a richer language which also allows boolean combinations of assertions. Our language $\mathcal{L}$ is defined from a set of propositional variables $ATM$, the boolean connectives and the conditional operator $\mathrel{|\!\sim}$. We assume that the set $ATM$ is finite. We use $A, B, C, \ldots$ to denote propositional formulas (that do not contain conditional formulas), whereas $F, G, \ldots$ are used to denote all formulas (including conditionals). The formulas of $\mathcal{L}$ are defined as follows: if $A$ is a propositional formula, $A \in \mathcal{L}$; if $A$ and $B$ are propositional formulas, $A \mathrel{|\!\sim} B \in \mathcal{L}$; if $F$ is a boolean combination of formulas of $\mathcal{L}$, $F \in \mathcal{L}$. A knowledge base $K$ is any set of formulas: in this work we restrict our attention to finite knowledge bases.

Here is the axiomatization of logic **R** [11]. We use $\vdash_{PC}$ (resp. $\models_{PC}$) to denote provability (resp. validity) in the propositional calculus:

- All axioms and rules of propositional logic
- $A \mathrel{|\!\sim} A$ (REF)
- if $\vdash_{PC} A \leftrightarrow B$ then $(A \mathrel{|\!\sim} C) \rightarrow (B \mathrel{|\!\sim} C)$, (LLE)
- if $\vdash_{PC} A \rightarrow B$ then $(C \mathrel{|\!\sim} A) \rightarrow (C \mathrel{|\!\sim} B)$ (RW)
- $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \wedge B \mathrel{|\!\sim} C)$ (CM)
- $((A \mathrel{|\!\sim} B) \wedge (A \mathrel{|\!\sim} C)) \rightarrow (A \mathrel{|\!\sim} B \wedge C)$ (AND)
- $((A \mathrel{|\!\sim} C) \wedge (B \mathrel{|\!\sim} C)) \rightarrow (A \vee B \mathrel{|\!\sim} C)$ (OR)
- $((A \mathrel{|\!\sim} B) \wedge \neg(A \mathrel{|\!\sim} \neg C)) \rightarrow ((A \wedge C) \mathrel{|\!\sim} B)$ (RM)

The axiom (CM) is called cumulative monotony and it is characteristic of all KLM logics, axiom (RM) is called rational monotony and it characterizes the logic of rational entailment **R** (it is what distinguishes rational from the weaker preferential entailment). **R**

seems to capture the core properties of non-monotonic reasoning, as shown by Friedman and Halpern these properties are quite ubiquitous being characterized by different semantics (all of them being instances of so-called *plausibility structures* [8]).

The logic **R** enjoys a simple modal semantics, actually it turns out that it is the flat fragment (i.e. without nested conditionals) of the well-known conditional logic **VC**. The modal semantics is defined by considering a set of worlds $\mathcal{W}$ equipped by an accessibility (or preference) relation $<$. Intuitively the meaning of $x < y$ is that $x$ is more normal/less exceptional than $y$. We say that a conditional $A \mathrel{|\!\sim} B$ is true in a model if $B$ holds in all most normal worlds where $A$ is true, i.e. in all $<$-minimal worlds satisfying $A$.

**Definition 1.** *A* rational *model is a triple $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ where: • $\mathcal{W}$ is a non-empty set of worlds; • $<$ is an irreflexive, transitive relation on $\mathcal{W}$ satisfying modularity: for all $x, y, z$, if $x < y$ then either $x < z$ or $z < y$. $<$ further satisfies the Smoothness condition defined below; • $V$ is a function $V : \mathcal{W} \longmapsto 2^{ATM}$, which assigns to every world $w$ the set of atoms holding in that world. If $F$ is a boolean combination of formulas, its truth conditions ($\mathcal{M}, w \models F$) are defined as for propositional logic. Let $A$ be a propositional formula; we define $Min_<^{\mathcal{M}}(A) = \{w \in \mathcal{W} \mid \mathcal{M}, w \models A$ and $\forall w', w' < w$ implies $\mathcal{M}, w' \not\models A\}$. Hence $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if for all $w'$, if $w' \in Min_<^{\mathcal{M}}(A)$ then $\mathcal{M}, w' \models B$.*
*We define the* Smoothness condition*: if $\mathcal{M}, w \models A$, then $w \in Min_<^{\mathcal{M}}(A)$ or there is $w' \in Min_<^{\mathcal{M}}(A)$ s.t. $w' < w$. Validity and satisfiability of a formula are defined as usual. Given a set of formulas $K$ of $\mathcal{L}$ and a model $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$, we say that $\mathcal{M}$ is a model of $K$, written $\mathcal{M} \models K$, if for every $F \in K$ and every $w \in \mathcal{W}$, $\mathcal{M}, w \models F$. $K$ rationally entails a formula $F$ ($K \models F$) if $F$ is valid in all rational models of $K$.*

Since in this work we limit our attention to a language containing finitely many atoms, and to finite knowledge bases, we can restrict our attention to finite models, as the logic enjoys the finite model property (observe that in this case the smoothness condition is ensured trivially by the irreflexivity of the $<$). It is easy to see from Definition 1 that the truth condition of $A \mathrel{|\!\sim} B$ is "global" in a model $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$: given a world $w$, we have that $\mathcal{M}, w \models A \mathrel{|\!\sim} B$ if, for all $w'$, if $w' \in Min_<^{\mathcal{M}}(A)$ then $\mathcal{M}, w' \models B$. It immediately follows that $A \mathrel{|\!\sim} B$ holds in $w$ if and only if $A \mathrel{|\!\sim} B$ is valid in a model, i.e. it holds that $\mathcal{M}, w' \models A \mathrel{|\!\sim} B$, for all $w'$ in $\mathcal{W}$; for this reason we will often write $\mathcal{M} \models A \mathrel{|\!\sim} B$. Moreover, when the reference to the model $\mathcal{M}$ is unambiguous, we will simply write $Min_<(A)$ instead of $Min_<^{\mathcal{M}}(A)$.

Rational models can be equivalently defined by postulating the existence of a rank function $k : \mathcal{W} \to \mathbb{N}$, and then letting $x < y$ iff $k(x) < k(y)$. For this reason rational models are also called "ranked models".

**Definition 2 (Rank of a world).** *Given a model $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$, the rank $k_{\mathcal{M}}$ of a world $w \in \mathcal{W}$, written $k_{\mathcal{M}}(w)$, is the length of the longest chain $w_0 < \cdots < w$ from $w$ to a minimal $w_0$ (i.e. there is no $w'$ such that $w' < w_0$).*

**Definition 3 (Rank of a formula).** *The rank $k_{\mathcal{M}}(F)$ of a formula $F$ in a model $\mathcal{M}$ is $i = min\{k_{\mathcal{M}}(w) : \mathcal{M}, w \models F\}$. If there is no $w : \mathcal{M}, w \models F$, $F$ has no rank in $\mathcal{M}$.*

**Proposition 1.** *For any $\mathcal{M} = \langle \mathcal{W}, V, < \rangle$ and any $w \in \mathcal{W}$, we have $\mathcal{M} \models A \mathrel{|\!\sim} B$ iff $k_{\mathcal{M}}(A \wedge B) < k_{\mathcal{M}}(A \wedge \neg B)$ or $A$ has no rank in $\mathcal{M}$.*

### 2.2 Lehmann and Magidor's definition of rational closure

As already mentioned, although the operator $\mid\sim$ is *non-monotonic*, the notion of logical entailment just defined is itself *monotonic*. In order to strengthen **R** and to obtain non-monotonic entailment, Lehmann and Magidor in [20] propose the well-known mechanism of rational closure. Since in rational closure no boolean combinations of conditionals are allowed, in the following, the knowledge base $K$ is just a finite set of positive conditional assertions of the form $A \mid\sim B$.

**Definition 4 (Exceptionality of propositional formulas and conditional formulas).** *Let $K$ be a knowledge base (i.e. a finite set of positive conditional assertions) and $A$ a propositional formula. $A$ is said to be* exceptional *for $K$ if and only if $K \models \top \mid\sim \neg A$. A conditional formula $A \mid\sim B$ is exceptional for $K$ if its antecedent $A$ is exceptional for $K$. The set of conditional formulas which are exceptional for $K$ will be denoted as $E(K)$.*

It is possible to define a non increasing sequence of subsets of $K$, $C_0 \supseteq C_1, \ldots$ by letting $C_0 = K$ and, for $i > 0$, $C_i = E(C_{i-1})$. Observe that, being $K$ finite, there is a $n \geq 0$ such that for all $m > n, C_m = C_n$ or $C_m = \emptyset$.

**Definition 5 (Rank of a formula).** *Let $K$ be a knowledge base and let $A$ be a propositional formula. $A$ has* rank $i$ *(for $K$) if and only if $i$ is the least natural number for which $A$ is not exceptional for $C_i$. If $A$ is exceptional for all $C_i$ then $A$ has no rank.*

Definition 5 above allows to define the rational closure of a knowledge base $K$.

**Definition 6 (Rational closure $\bar{\mathbf{K}}$ of $K$).** *Let $K$ be a conditional knowledge base. The rational closure $\bar{K}$ of $K$ is the set of all $A \mid\sim B$ such that either (1) the rank of $A$ is strictly less than the rank of $A \wedge \neg B$ (this includes the case $A$ has a rank and $A \wedge \neg B$ has none), or (2) $A$ has no rank.*

This mechanism, which is now well-established, allows to overcome some weaknesses of **R** . First of all it is closed under rational monotonicity (RM): if $(A \mid\sim B) \in \bar{\mathrm{K}}$ and $(A \mid\sim \neg C) \notin \bar{\mathrm{K}}$ then $(A \wedge C) \mid\sim B \in \bar{\mathrm{K}}$. Furthermore, rational closure supports some of the wanted inferences that **R** does not support. For instance rational closure allows to deal with irrelevance: from $monday \mid\sim go\_work$, it does support the non-monotonic conclusion that $monday \wedge shines \mid\sim go\_work$.

### 2.3 A semantical characterization of rational closure

We provide a semantical reconstruction of rational closure in terms of a minimal models' mechanism, thus providing an instantiation of the following general recipe for non-monotonic reasoning:
(i) fix an underlying modal semantics for conditionals (here we concentrate on **R** but another possible choice could have been the weaker **P** as in [12]),
(ii) obtain non-monotonic inference by restricting semantic consequence to a class of *minimal* models. These minimal models should be chosen on the basis of semantic considerations, independent from the *language* and from the *set of conditionals* (knowledge base) whose non-monotonic consequences we want to determine.
In the next proposition we will use $\mathcal{M}_i$ defined as follows. Let $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ be any rational model of $K$. Let $\mathcal{M}_0 = \mathcal{M}$ and, for all $i$, let $\mathcal{M}_i = \langle \mathcal{W}_i, <_i, V_i \rangle$ be the

rational model obtained from $\mathcal{M}$ by removing all the worlds $w$ with $k_{\mathcal{M}}(w) < i$, i.e., $\mathcal{W}_i = \{w \in \mathcal{W} : \ k_{\mathcal{M}}(w) \geq i\}$.

**Proposition 2.** *Let $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ be any rational model of $K$. For any propositional formula $A$, if rank($A$) $\geq i$, then 1) $k_{\mathcal{M}}(A) \geq i$, and 2) if $A \mathrel{|\!\sim} B$ is entailed by $C_i$, then $\mathcal{M}_i$ satisfies $A \mathrel{|\!\sim} B$.*

The semantics we propose is a *fixed interpretations minimal semantics*, for short $FIMS$. In some respects our approach is similar in spirit to minimal models approaches to non-monotonic reasoning, such as circumscription[4].

**Definition 7** **($FIMS$).** *Given $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ and $\mathcal{M}' = \langle \mathcal{W}', <', V' \rangle$ we say that $\mathcal{M}$ is preferred to $\mathcal{M}'$ with respect to the fixed interpretations minimal semantics, and we write $\mathcal{M} <_{FIMS} \mathcal{M}'$, if $\mathcal{W} = \mathcal{W}'$, $V = V'$, and for all $x$, $k_{\mathcal{M}}(x) \leq k_{\mathcal{M}'}(x)$ whereas there exists $x' : k_{\mathcal{M}}(x') < k_{\mathcal{M}'}(x')$. We say that $\mathcal{M}$ is minimal w.r.t. $<_{FIMS}$ in case there is no $\mathcal{M}'$ such that $\mathcal{M}' <_{FIMS} \mathcal{M}$. We say that $K$ minimally entails a formula $F$ w.r.t. FIMS, and we write $K \models_{FIMS} F$, if $F$ is valid in all models of $K$ which are minimal w.r.t. $<_{FIMS}$.*

Can we capture rational closure within the semantics of Definition 7 above? We are soon forced to recognize that this is not the case. For instance, consider the following:

*Example 1.* Let $K = \{penguin \mathrel{|\!\sim} bird, penguin \mathrel{|\!\sim} \neg fly, bird \mathrel{|\!\sim} fly\}$. We derive that $K \not\models_{FIMS} penguin \wedge black \mathrel{|\!\sim} \neg fly$. Indeed in $FIMS$ there can be a model $\mathcal{M}$ in which $\mathcal{W} = \{x, y, z\}$, $V(x) = \{penguin, bird, fly, black\}$, $V(y) = \{penguin, bird\}$, $V(z) = \{bird, fly\}$, and $z < y < x$. $\mathcal{M}$ is a model of $K$, and it is minimal with respect to $FIMS$ (indeed once fixed $V(x), V(y), V(z)$ as above, it is not possible to lower the rank of $x$ nor of $y$ nor of $z$ unless we falsify $K$). Furthermore, in $\mathcal{M}$, $x$ is a typical world in which "it flies" and "it is black" hold (since there is no other world satisfying the same propositions which is preferred to it). Therefore, $K \not\models_{FIMS} penguin \wedge black \mathrel{|\!\sim} \neg fly$.

We have that $\{penguin \mathrel{|\!\sim} bird, penguin \mathrel{|\!\sim} \neg fly, bird \mathrel{|\!\sim} fly\} \not\models_{FIMS} penguin \wedge black \mathrel{|\!\sim} \neg fly$. On the contrary, it can be verified that $penguin \wedge black \mathrel{|\!\sim} \neg fly$ is in the rational closure of $\{penguin \mathrel{|\!\sim} bird, penguin \mathrel{|\!\sim} \neg fly, bird \mathrel{|\!\sim} fly\}$. Therefore, $FIMS$ as it is does not allow us to define a semantics corresponding to rational closure. Things change if we consider $FIMS$ applied to models that contain *all possible valuations compatible (see Definition 8 below) with a given knowledge base $K$*. We call these models *canonical models*.

*Example 2.* Consider Example 1 above. If we restrict our attention to models that also contain a $w$ with $V(w) = \{penguin, bird, black\}$ which satisfies "it is a penguin", "it is black" and "it does not fly" in which $w$ is a typical world satisfying "it is a penguin", we are able to conclude that typically it holds that if it is a penguin and it is black then it does not fly, as in rational closure. Indeed, in all minimal models of $K$ that also contain $w$ with $V(w) = \{penguin, bird, black\}$, it holds that $penguin \wedge black \mathrel{|\!\sim} \neg fly$.

[4] As for circumscription, there are mainly two ways of comparing models with the same domain: by keeping the valuation function fixed (only comparing $\mathcal{M}$ and $\mathcal{M}'$ if $V$ and $V'$ in the two models respectively coincide); or by also comparing $\mathcal{M}$ and $\mathcal{M}'$ in case $V \neq V'$. In this work we consider the latter alternative.

We are led to the conjecture that $FIMS$ restricted to canonical models could be the right semantics for rational closure. Canonical models are defined w.r.t. the language $\mathcal{L}$. A truth assignment $v : ATM \longrightarrow \{true, false\}$ is *compatible* with $K$, if there is no formula $A \in \mathcal{L}$ such that $v(A) = true$ and $K \models A \mathrel{|\!\sim} \bot$.

**Definition 8.** *A model $\mathcal{M} = \langle \mathcal{W}, <, V \rangle$ satisfying a knowledge base $K$ is said to be* canonical *if it contains (at least) a world associated to each truth assignment compatible with $K$, that is to say: if $v$ is compatible with $K$, then there exists a world $w$ in $\mathcal{W}$, such that for all propositional formulas $B$ $\mathcal{M}, w \models B$ iff $v(B) = true$.*

It can be shown that for any knowledge base a minimal canonical model exists: this is any canonical model in which every possible world $w$ has the rank associated to the conjunction of all atoms and negated atoms in $\mathcal{L}$ that it satisfies. We can also prove that the canonical models that are minimal with respect to $FIMS$ are an adequate semantic counterpart of rational closure.

**Theorem 1.** *Let $K$ be a knowledge base and $\mathcal{M}$ be a canonical model of $K$ minimal w.r.t. $<_{FIMS}$. We show that, for all conditionals $A \mathrel{|\!\sim} B$, $\mathcal{M} \models A \mathrel{|\!\sim} B$ if and only if $A \mathrel{|\!\sim} B \in \overline{K}$, where $\overline{K}$ is the rational closure of $K$.*

## 3 Rational closure in Description Logics

As mentioned, the interest towards non-monotonic reasoning in DLs has grown in the last years. In this section, we extend to $\mathcal{ALC}$ the notion of rational closure proposed by Lehmann and Magidor [20], recalled in Section 2.2, and we define a semantic characterization of this notion of rational closure by introducing a minimal model semantics for $\mathcal{ALC}$ with defeasible inclusions. This semantics is a direct generalization of the minimal (canonical) model semantics introduced in Section 2.3

To express defeasible inclusions, $\mathcal{ALC}$ is extended with a typicality operator $\mathbf{T}$, following the approach in [10, 14]. Differently from [14], here we consider special kinds of preferential models, namely, rational models, to define the semantics of the $\mathbf{T}$ operator, and we use a different notion of preference between models, namely, the preference relation $<_{FIMS}$, introduced in Section 2.3. Given the typicality operator, the defeasible assertion $\mathbf{T}(C) \sqsubseteq D$ (all the typical C's are D's) plays the role of the conditional assertion $C \mathrel{|\!\sim} D$ in $\mathbf{R}$.

### 3.1 The logic $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$

Similarly to rational closure which is a non-monotonic mechanism built over $\mathbf{R}$, our application of rational closure to DLs is done in two steps. First, similarly to what done in [10], we extend the standard $\mathcal{ALC}$ by a typicality operator $\mathbf{T}$ that allows to single out the typical instances of a concept $\mathbf{T}$. Since we are dealing here with rational closure (that builds over $\mathbf{R}$), we attribute to $\mathbf{T}$ properties related to $\mathbf{R}$. The resulting logic is called $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$. As a second step, we build over $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ a rational closure mechanism.

Our starting point is therefore the extension of logic $\mathcal{ALC}$ with a typicality operator $\mathbf{T}$. The intuitive idea is to extend the standard $\mathcal{ALC}$ allowing concepts of the form $\mathbf{T}(C)$ whose intuitive meaning is that $\mathbf{T}(C)$ selects the *typical* instances of a concept $C$. 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 typical such instances ($\mathbf{T}(C) \sqsubseteq D$).

**Definition 9.** *We consider an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individual constants $\mathcal{O}$. Given $A \in \mathcal{C}$ and $R \in \mathcal{R}$, we define $C_R := A \mid \top \mid \bot \mid \neg C_R \mid C_R \sqcap C_R \mid C_R \sqcup C_R \mid \forall R.C_R \mid \exists R.C_R$, and $C_L := C_R \mid \mathbf{T}(C_R)$. A KB is a pair (TBox, ABox). TBox contains a finite set of concept inclusions $C_L \sqsubseteq C_R$. ABox contains assertions of the form $C_L(a)$ and $R(a,b)$, where $a, b \in \mathcal{O}$.*

The $\mathbf{T}$ operator satisfies a set of postulates that are essentially a reformulation of rational logic $\mathbf{R}$: in this respect, the $\mathbf{T}$-assertion $\mathbf{T}(C) \sqsubseteq D$ is equivalent to the conditional assertion $C \mathrel{|\!\sim} D$ in $\mathbf{R}$.

A first semantic characterization of $\mathbf{T}$ can be given by means of a set of postulates that are essentially a restatement of axioms and rules of non-monotonic entailment in rational logic $\mathbf{R}$. Given a domain $\Delta$ and a valuation function $I$ one can define the function $f_{\mathbf{T}}(S)$ that selects the *typical* instances of $S$, and in case $S = C^I$ for a concept $C$, it selects the typical instances of $C$. In this semantics, $(\mathbf{T}(C))^I = f_{\mathbf{T}}(C^I)$, and $f_{\mathbf{T}}$ has the following intuitive properties for all subsets $S$ of $\Delta$:

$(f_{\mathbf{T}} - 1)$ $f_{\mathbf{T}}(S) \subseteq S$ $\qquad$ $(f_{\mathbf{T}} - 2)$ if $S \neq \emptyset$, then also $f_{\mathbf{T}}(S) \neq \emptyset$

$(f_{\mathbf{T}} - 3)$ if $f_{\mathbf{T}}(S) \subseteq R$, then $f_{\mathbf{T}}(S) = f_{\mathbf{T}}(S \cap R)$ $\qquad$ $(f_{\mathbf{T}} - 4)$ $f_{\mathbf{T}}(\bigcup S_i) \subseteq \bigcup f_{\mathbf{T}}(S_i)$

$(f_{\mathbf{T}} - 5)$ $\bigcap f_{\mathbf{T}}(S_i) \subseteq f_{\mathbf{T}}(\bigcup S_i)$ $\qquad$ $(f_{\mathbf{T}} - \mathbf{R})$ if $f_{\mathbf{T}}(S) \cap R \neq \emptyset$, then $f_{\mathbf{T}}(S \cap R) \subseteq f_{\mathbf{T}}(S)$

$(f_{\mathbf{T}} - 1)$ enforces that typical elements of $S$ belong to $S$. $(f_{\mathbf{T}} - 2)$ enforces that if there are elements in $S$, then there are also *typical* such elements. $(f_{\mathbf{T}} - 3)$ expresses a weak form of monotonicity, namely *cautious monotonicity*. The next properties constraint the behavior of $f_{\mathbf{T}}$ wrt $\cap$ and $\cup$ in such a way that they do not entail monotonicity. Last, $(f_{\mathbf{T}} - \mathbf{R})$ corresponds to rational monotonicity, and forces again a form of monotonicity: if there is a typical $S$ having the property $R$, then all typical $S$ and $R$s inherit the properties of typical $S$s.

The semantics of $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ can be equivalently formulated in terms of rational models: models of $\mathcal{ALC}$ are equipped by a *preference relation* $<$ on the domain, whose intuitive meaning is to compare the "typicality" of domain elements, that is to say $x < y$ 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 (s.t. there is no other member of $C$ more typical than $x$). This semantics with one single preference relation $<$ is the one that, as we will show, corresponds to rational closure[5].

**Definition 10 (Semantics of $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$).** *A model $\mathcal{M}$ of $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ is any structure $\langle \Delta, <, I \rangle$ where: $\Delta$ is the domain; $<$ is an irreflexive, transitive and modular relation over $\Delta$ ($<$ is modular if, for all $x, y, z \in \Delta$, if $x < y$ then either $x < z$ or $z < y$); $I$ 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 $\mathcal{ALC}$, $C^I$ is defined in the usual way. For the $\mathbf{T}$ operator, we have $(\mathbf{T}(C))^I = Min_<(C^I)$, where $Min_<(S) = \{u : u \in S$ and $\nexists z \in S$*

[5] One may think of considering a sharper semantics with several preference relations. We aim to explore this possibility in future works, for the moment, we just notice that (i) the definition of such a semantics is not straightforward (what does differentiate one preference relation from another? What are the dependencies between the different preference relations? Has the typicality operator to be made parametric?) (ii) it cannot be expected that the resulting semantics, being stronger than the one just proposed, can correspond to rational closure below.

*s.t.* $z < u\}$. *Furthermore,* $<$ *satisfies the* Smoothness Condition*, i.e., for all concepts* $C$*,* $C^I$ *is* smooth*. For* $S \subseteq \Delta$*, we say that* $S$ *is* smooth *iff for all* $x \in S$*, either* $x \in Min_<(S)$ *or* $\exists y \in Min_<(S)$ *such that* $y < x$*,*

**Theorem 2.** *[Theorem* 1 *in [9]] A KB=(TBox,ABox) is satisfiable in a model described in Definition 10 iff it is satisfiable in a model* $\langle \Delta, I, f_{\mathbf{T}} \rangle$ *where* $f_{\mathbf{T}}$ *satisfies* $(f_{\mathbf{T}} - 1) - (f_{\mathbf{T}} - 5)$ *and* $(f_{\mathbf{T}} - \mathbf{R})$*, and* $(\mathbf{T}(C))^I = f_{\mathbf{T}}(C^I)$*.*

In the following, we will refer to the definition of the semantics given in Definition 10.

**Definition 11 (Model satisfying a Knowledge Base).** *Given a model* $\mathcal{M}$*,* $I$ *is extended to assign a distinct element* $a^I$ *of* $\Delta$ *to each individual constant* $a$ *of* $\mathcal{O}$ *(i.e. we assume the* unique name assumption*).*
*We say that: a model* $\mathcal{M}$ *satisfies an inclusion* $C \sqsubseteq D$ *if it holds* $C^I \subseteq D^I$*;* $\mathcal{M}$ *satisfies an assertion* $C(a)$ *if* $a^I \in C^I$*; and* $\mathcal{M}$ *satisfies an assertion* $R(a, b)$ *if* $(a^I, b^I) \in R^I$*.*
*We say that:* $\mathcal{M}$ satisfies a knowledge base $K$=(TBox,ABox)*, if it satisfies both its TBox and its ABox, where:* $\mathcal{M}$ *satisfies TBox if* $\mathcal{M}$ *satisfies all inclusions in TBox and* $\mathcal{M}$ *satisfies ABox if* $\mathcal{M}$ *satisfies all assertions in ABox.*

From now on, in this section, we restrict our attention to $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ and to finite models. Given a knowledge base $K$ and an inclusion $C_L \sqsubseteq C_R$, we say that the inclusion is derivable from $K$ (we write $K \models_{\mathcal{ALC}^{\mathbf{R}}\mathbf{T}} C_L \sqsubseteq C_R$) if $C_L^I \subseteq C_R^I$ holds in all models $\mathcal{M} = \langle \Delta, <, I \rangle$ satisfying $K$.

**Definition 12 (Rank of a domain element).** *The rank* $k_{\mathcal{M}}$ *of a domain element* $x$ *in a model* $\mathcal{M}$ *is the length of the longest chain* $x_0 < \cdots < x$ *from* $x$ *to a minimal* $x_0$ *(s.t. for no* $x'$*,* $x' < x_0$*).*

Finite $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ models can be equivalently defined by postulating the existence of a function $k : \Delta \to \mathbb{N}$, and then letting $x < y$ iff $k(x) < k(y)$.

**Definition 13 (Rank of a concept).** *Given a model* $\mathcal{M} = \langle \Delta, <, I \rangle$*, the* rank $k_{\mathcal{M}}(C_R)$ of a concept $C_R$ in the model $\mathcal{M}$ *is* $i = min\{k_{\mathcal{M}}(x) : x \in C_R^I\}$*. If* $C_R^I = \emptyset$*, then* $C_R$ *has no rank and we write* $k_{\mathcal{M}}(C_R) = \infty$*.*

**Proposition 3.** *For any* $\mathcal{M} = \langle \Delta, <, I \rangle$*, we have that* $\mathcal{M}$ *satisfies* $\mathbf{T}(C) \sqsubseteq D$ *iff* $k_{\mathcal{M}}(C \sqcap D) < k_{\mathcal{M}}(C \sqcap \neg D)$*.*

As already mentioned, although the typicality operator $\mathbf{T}$ itself is non-monotonic (i.e. $\mathbf{T}(C) \sqsubseteq D$ does not imply $\mathbf{T}(C \sqcap E) \sqsubseteq D$), the logics $\mathcal{ALC}+\mathbf{T}$ and $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ are monotonic: what is inferred from $K$ can still be inferred from any $K'$ with $K \subseteq K'$. This is a clear limitation in DLs. As a consequence of non-monotonicity in $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ one cannot deal with irrelevance for instance. So one cannot derive from $K = \{Penguin \sqsubseteq Bird, \mathbf{T}(Bird) \sqsubseteq Fly, \mathbf{T}(Penguin) \sqsubseteq \neg Fly\}$ that $K \models_{min} \mathbf{T}(Penguin \sqcap Black) \sqsubseteq \neg Fly$, even if the property of being black is irrelevant with respect to flying. In the same way if we added to $K$ the information that jim is a bird ($Bird(jim)$), in $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ one cannot non-monotonically derive that it is a typical bird and therefore flies ( $\mathbf{T}(Bird)(jim)$ and $Fly(jim)$ ). We investigate the possibility of overcoming this weakness by extending to $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$ the notion of rational closure. We first consider the rational closure of the TBox alone. Next we will consider rational closure that also takes into account the ABox.

### 3.2 Rational Closure of the TBox in $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$

Let us first define the notion of *query*. Intuitively, a query is either an inclusion relation or an assertion of the ABox; we want to check whether it is entailed from a given KB.

**Definition 14 (Query).** *A* query $F$ *is either an assertion* $C_L(a)$ *or an inclusion relation* $C_L \sqsubseteq C_R$. *Given a model* $\mathcal{M} = \langle \Delta, <, I \rangle$, *a query* $F$ *holds in* $\mathcal{M}$ *if* $\mathcal{M}$ *satisfies* $F$.

**Definition 15.** *Let* $T_B$ *be a TBox and* $C$ *a concept.* $C$ *is said to be* exceptional *for* $T_B$ *iff* $T_B \models_{\mathcal{ALC}^{\mathbf{R}}\mathbf{T}} \mathbf{T}(\top) \sqsubseteq \neg C$. *A* $\mathbf{T}$*-inclusion* $\mathbf{T}(C) \sqsubseteq D$ *is exceptional for* $T_B$ *if* $C$ *is exceptional for* $T_B$. *The set of* $\mathbf{T}$*-inclusions of* $T_B$ *which are exceptional in* $T_B$ *will be denoted as* $\mathcal{E}(T_B)$.

Given a DL knowledge base $K$=(TBox,ABox), it is possible to define a sequence of non-increasing subsets of TBox $E_0 \supseteq E_1, \ldots$ by letting $E_0 =$ TBox and, for $i > 0$, $E_i = \mathcal{E}(E_{i-1}) \cup \{C \sqsubseteq D \in \text{TBox s.t. } \mathbf{T} \text{ does not occurr in } C\}$. Observe that, being $K$ finite, there is an $n \geq 0$ such that for all $m > n, E_m = E_n$ or $E_m = \emptyset$. Observe also that the definition of the $E_i$'s is the same as the definition of the $C_i$'s in Lehmann and Magidor's definition of rational closure in Section 2.2, except for the fact that here, at each step, we also add all the strict inclusions.

**Definition 16.** *A concept* $C$ *has* rank $i$ *(denoted by* $rank(C) = i$*) for* $K$*=(TBox,ABox), iff* $i$ *is the least natural number for which* $C$ *is not exceptional for* $E_i$. *If* $C$ *is exceptional for all* $E_i$ *then* $rank(C) = \infty$, *and we say that* $C$ *has no rank.*

As for propositional logic, the notion of rank of a formula allows to define the rational closure of the TBox of a knowledge base $K$.

**Definition 17 (Rational closure of TBox).** *Let* $K$*=(TBox,ABox) be a DL knowledge base. We define,* $\overline{TBox}$*, the* rational closure of TBox*, as*

$$\overline{TBox} = \{\mathbf{T}(C) \sqsubseteq D \mid \textit{either } rank(C) < rank(C \sqcap \neg D) \textit{ or } rank(C) = \infty\} \cup \{C \sqsubseteq D \mid K \models_{\mathcal{ALC}} C \sqsubseteq D\}$$

It can be easily seen that the rational closure of TBox is a non-monotonic strengthening of $\mathcal{ALC}^{\mathbf{R}}\mathbf{T}$. For instance it allows to deal with irrelevance. If TBox = $\{Penguin \sqsubseteq Bird, \mathbf{T}(Bird) \sqsubseteq Fly, \mathbf{T}(Penguin) \sqsubseteq \neg Fly\}$, then it can be verified that $\mathbf{T}(Bird \sqcap Black) \sqsubseteq Fly \in \overline{TBox}$. This is a non-monotonic inference that does no longer follow if we knew that indeed black birds are non typical birds that do not fly: in this case from TBox'= TBox $\cup \{\mathbf{T}(Bird \sqcap Black) \sqsubseteq \neg Fly\}$ (in this case $\mathbf{T}(Bird \sqcap Black) \sqsubseteq Fly \notin \overline{TBox'}$). Similarly, as for the propositional case, rational closure is closed under rational monotonicity: from $\mathbf{T}(Bird) \sqsubseteq Fly \in \overline{TBox}$ and $\mathbf{T}(Bird) \sqsubseteq \neg LivesEurope \notin \overline{TBox}$ it follows that $\mathbf{T}(Bird \sqcap LivesEurope) \sqsubseteq Fly \in \overline{TBox}$.

As for the propositional case, in order to semantically characterize the rational closure, we first restrict our attention to minimal rational models that minimize *the rank of domain elements*. Informally, given two models of $K$, one in which a given domain element $x$ has rank 2 (because for instance $z < y < x$) , and another in which it has rank 1 (because only $y < x$), we would prefer the latter, as in this model the element $x$ is "more normal" than in the former.

From now on, we restrict our attention to *canonical minimal models*. First, we define a set of concepts $\mathcal{S}$ closed under negation and subconcepts. We assume that all the concepts in $K$ and in the query $F$ belong to $\mathcal{S}$. In order to define canonical models, we consider all the sets of concepts $\{C_1, C_2, \ldots, C_n\} \subseteq \mathcal{S}$ that are *consistent with* $K$, i.e., s.t. $K \not\models_{\mathcal{ALC}} C_1 \sqcap C_2 \sqcap \cdots \sqcap C_n \sqsubseteq \bot$.

**Definition 18 (Canonical model w.r.t. $\mathcal{S}$).** *Given $K$=(TBox,ABox) and a query $F$, a model $\mathcal{M} = \langle \Delta, <, I \rangle$ satisfying $K$ is* canonical w.r.t. $\mathcal{S}$ *if it contains at least a domain element $x \in \Delta$ s.t. $x \in (C_1 \sqcap C_2 \sqcap \cdots \sqcap C_n)^I$, for each set of concepts $\{C_1, C_2, \ldots, C_n\} \subseteq \mathcal{S}$ that are consistent with $K$.*

**Definition 19 (Minimal canonical models (w.r.t. $\mathcal{S}$)).** *Consider two models $\mathcal{M} = \langle \Delta, <, I \rangle$ and $\mathcal{M}' = \langle \Delta', <', I' \rangle$, canonical w.r.t. $\mathcal{S}$. We say that $\mathcal{M}$ is preferred to $\mathcal{M}'$ ($\mathcal{M} < \mathcal{M}'$) if $\Delta = \Delta'$, and for all $x \in \Delta$, $k_{\mathcal{M}}(x) \leq k_{\mathcal{M}'}(x)$ whereas there exists $y \in \Delta$ such that $k_{\mathcal{M}}(y) < k_{\mathcal{M}'}(y)$. Given a knowledge base $K$, we say that $\mathcal{M}$* is a minimal canonical model of $K$ *if it is a canonical model satisfying $K$ and there is no canonical model $\mathcal{M}'$ satisfying $K$ such that $\mathcal{M}' < \mathcal{M}$.*

The following results hold (more details and proofs can be found in [15, 16]):

**Theorem 3.** *For any $K$ there exists a minimal canonical model w.r.t. TBox.*

**Theorem 4.** *Let $K$=(TBox,ABox) be a knowledge base and $C \sqsubseteq D$ a query. We have that $C \sqsubseteq D \in \overline{TBox}$ if and only if $C \sqsubseteq D$ holds in all minimal canonical models of $K$ with respect to $\mathcal{S}$.*

**Theorem 5 (Complexity of rational closure over the TBox).** *Given a knowledge base $K$ =(TBox,ABox), the problem of deciding whether $\mathbf{T}(C) \sqsubseteq D \in \overline{TBox}$ is in* EXPTIME.

### 3.3 Rational Closure Over the ABox

In this section we extend the notion of rational closure defined in the previous section in order to take into account the individual constants in the ABox. We address this question by first considering the semantic aspect, in order to treat individuals explicitly mentioned in the ABox in a uniform way with respect to the other domain elements: as for all the domain elements we would like to attribute to each individual constant named in the ABox the lowest possible rank. So we further refine Definition 19 of minimal canonical models with respect to TBox by taking into account the interpretation of individual constants of the ABox: given two minimal canonical models $\mathcal{M}$ and $\mathcal{M}'$, we prefer $\mathcal{M}$ to $\mathcal{M}'$ if there is an individual constant $b$ occurring in ABox such that $k_{\mathcal{M}}(b^I) < k_{\mathcal{M}}(b^{I'})$ (whereas $k_{\mathcal{M}}(a^I) \leq k_{\mathcal{M}}(a^{I'})$ for all other individual constants occurring in ABox).

**Definition 20 (Minimal canonical model of $K$ minimally satisfying ABox).** *Given $K$=(TBox,ABox), let $\mathcal{M} = \langle \Delta, <, I \rangle$ and $\mathcal{M}' = \langle \Delta', <', I' \rangle$ be two canonical models of $K$ which are minimal w.r.t. Definition 19. We say that $\mathcal{M}$ is preferred to $\mathcal{M}'$ with respect to ABox ($\mathcal{M} <_{ABox} \mathcal{M}'$) if for all individual constants $a$ occurring in ABox, $k_{\mathcal{M}}(a^I) \leq k_{\mathcal{M}}(a^{I'})$ and there is at least one individual constant $b$ occurring in ABox such that $k_{\mathcal{M}}(b^I) < k_{\mathcal{M}}(b^{I'})$.*

**Theorem 6.** *For any $K = (TBox, ABox)$ there exists a minimal canonical model of $K$ minimally satisfying ABox.*

In order to see the power of the above semantic notion, consider the standard birds and penguins example.

*Example 3.* Suppose we have a knowledge base $K$ where TBox = $\{\mathbf{T}(Bird) \sqsubseteq Fly$, $\mathbf{T}(Penguin) \sqsubseteq \neg Fly, Penguin \sqsubseteq Bird\}$, and ABox = $\{Penguin(pio), Bird(tweety)\}$. Knowing that tweety is a bird and pio is a penguin, we would like to be able to assume, in the absence of other information, that tweety is a typical bird, whereas pio is a typical penguin, and therefore tweety flies whereas pio does not. Consider any minimal canonical model $\mathcal{M}$ of $K$. Being canonical, $\mathcal{M}$ will contain, among other elements:

- $x \in (Bird)^I$, $x \in (Fly)^I$, $x \in (\neg Penguin)^I$, $k_{\mathcal{M}}(x) = 0$;
- $y \in (Bird)^I$, $y \in (\neg Fly)^I$, $y \in (\neg Penguin)^I$, $k_{\mathcal{M}}(y) = 1$;
- $z \in (Penguin)^I$, $z \in (Bird)^I$, $z \in (\neg Fly)^I$ , $k_{\mathcal{M}}(z) = 1$;
- $w \in (Penguin)^I$, $w \in (Bird)^I$, $w \in (Fly)^I$ , $k_{\mathcal{M}}(w) = 2$;

Notice that in the definition of minimal canonical model there is no constraint on the interpretation of the ABox constants tweety and pio. As far as Definition 19 is concerned for instance tweety can be mapped onto $x$ ($(tweety)^I = x$) or onto $y$ ($(tweety)^I = y$): the minimality of $\mathcal{M}$ with respect to Definition 19 is not affected by this choice. However in the first case it would hold that tweety is a typical bird, in the second tweety is not a typical bird. We want to prefer the first case, and this is what derives from Definition 20: if in $\mathcal{M}$ $tweety^I = x$ whereas in $\mathcal{M}_1$ (which for the rest is identical to $\mathcal{M}$) it holds that $tweety^I = y$, then $\mathcal{M}$ is preferred to $\mathcal{M}_1$. The same for $pio$. As a result in all models of $K$ minimal with respect to both TBox and ABox (Definition 20), it holds what we wanted: that tweety is a typical bird ($T(Bird)(tweety)$), and therefore it flies, whereas pio is a typical penguin ($T(Penguin)(pio)$), and therefore it does not fly.

We conclude this section by providing an algorithmic construction for the rational closure of ABox, whose idea is that of considering all the possible minimal consistent assignments of ranks to the individuals explicitly named in the ABox. Each assignment adds some properties to named individuals which can be used to infer new conclusions. We adopt a skeptical view of considering only those conclusions which hold for all assignments. The equivalence with the semantics shows that the minimal entailment captures a skeptical approach when reasoning about the ABox.

More formally, in order to calculate the rational closure of ABox ($\overline{ABox}$) for all individual constants of the ABox we find out what is the lowest possible rank they can have in minimal canonical models w.r.t. Definition 19, with the idea that an individual constant $a_i$ can have a given rank ($k_j(a_i)$) just in case it is compatible with all the inclusions of the TBox whose antecedent $A$'s rank is $\geq k_j(a_i)$ (the inclusions whose antecedent $A$'s rank is $< k_j(a_i)$ do not matter. The minimal possible rank assignment $k_j$ for all $a_i$ is computed in the algorithm below: $\mu_i^j$ computes all the concepts that $a_i$ would need to satisfy in case it had the rank attributed by $k_j$ ($k_j(a_i)$). The algorithm verifies whether $\mu_i^j$ is compatible with ($\overline{TBox}$, ABox) and whether it is minimal. Notice that in this phase all constants are considered simultaneously (indeed the possible ranks of different individual constants depend on each other). For this reason $\mu^j$ takes into

account the ranks attributed to all individual constants, being the union of all $\mu_i^j$ for all $a_i$, and the consistency of this union with ($\overline{TBox}$, ABox) is verified (instead of the consistency of all separate $\mu_i^j$). Once computed the minimal rank assignments these are used to define $\overline{ABox}$) as the set of all assertions derivable in $\mathcal{ALC}$ from ABox $\cup \mu^j$ for all minimal consistent rank assignments $k_j$.

**Definition 21 ($\overline{ABox}$: rational closure of ABox).** *Let $a_1, \ldots, a_m$ be the individuals explicitly named in the ABox. Let $k_1, k_2, \ldots, k_h$ be all the possible rank assignments (ranging from 1 to $n$) to the individuals occurring in ABox.*

- *Given a rank assignment $k_j$ we define:*
  - *for each $a_i$: $\mu_i^j = \{(\neg C \sqcup D)(a_i)$ s.t. $C, D \in \mathcal{S}$, $\mathbf{T}(C) \sqsubseteq D$ in $\overline{TBox}$, and $k_j(a_i) \leq rank(C)\} \cup \{(\neg C \sqcup D)(a_i)$ s.t. $C \sqsubseteq D$ in TBox $\}$;*
  - *let $\mu^j = \mu_1^j \cup \cdots \cup \mu_m^j$ for all $\mu_1^j \ldots \mu_m^j$ just calculated for all $a_1, \ldots, a_m$ in the ABox*
- *$k_j$ is* minimal and consistent *with ($\overline{TBox}$, ABox) if:*
  - *ABox $\cup \mu^j$ is consistent in $\mathcal{ALC}$;*
  - *there is no $k_i$ consistent wih ($\overline{TBox}$, ABox) s.t. for all $a_i$, $k_i(a_i) \leq k_j(a_i)$ and for some $b$, $k_i(b) < k_j(b)$.*
- *The rational closure of ABox ($\overline{ABox}$) is the set of all assertions derivable in $\mathcal{ALC}$ from ABox $\cup \mu^j$ for all minimal consistent rank assignments $k_j$, i.e:*

$$\overline{ABox} = \bigcap_{k_j \textit{minimal consistent}} \{C(a) : \textit{ABox} \cup \mu^j \models_{\mathcal{ALC}} C(a)\}$$

The following theorems hold (again, see [15, 16] for details and proofs):

**Theorem 7 (Soundness and Completeness of $\overline{ABox}$).** *Given $K$=(TBox, ABox), for all individual constant $a$ in ABox, we have that $C(a) \in \overline{ABox}$ if and only if $C(a)$ holds in all minimal canonical models of $K$ minimally satisfying ABox.*

**Theorem 8 (Complexity of rational closure over the ABox).** *Given a knowledge base $K$ =(TBox,ABox), an individual constant $a$ and a concept $C$, the problem of deciding whether $C(a) \in \overline{ABox}$ is* EXPTIME*-complete.*

## 4 Related work

In [14] non-monotonic extensions of DLs based on the **T** operator have been proposed. In these extensions, the semantics of **T** is based on preferential logic **P**. Non-monotonic inference is obtained by restricting entailment to *minimal models*, where minimal models are those that minimize the truth of formulas of a special kind. In this work, we have presented an alternative approach. First, the semantics underlying the **T** operator is **R** . Moreover and more importantly, we have adopted a minimal model semantics, where, as a difference with [14], the notion of minimal model is completely independent from the language and is determined only by the relational structure of models.

Casini and Straccia [4] study the application of rational closure to DLs. They extend to $\mathcal{ALC}$ the algorithmic construction proposed by Freund for capturing the rational closure in the propositional calculus. While in the propositional calculus this construction is proved to be equivalent with the notion of rational closure in [20], the equivalence

is not known to hold for the case of $\mathcal{ALC}$. While Casini and Straccia prove axiomatic properties of their notion of rational closure, here we focus on an extension of Lehmann and Magidor definition of rational closure for $\mathcal{ALC}$ and we define a semantics for it. [4] also keeps the ABox into account, and defines closure operations over individuals. It introduces a consequence relation $\Vdash$ among a knowledge base $K$ and assertions, under the requirement that the TBox is unfoldable and the ABox is closed under completion rules, such as, for instance, that if $a : \exists R.C \in$ ABox, then both $aRb$ and $b : C$ (for some individual constant $b$) must belong to the ABox too. Under such restrictions they are able to define a procedure to compute the rational closure of the ABox assuming that the individuals explicitly named are linearly ordered, and different orders determine different sets of consequences. The authors show that, for each order $s$, the consequence relation $\Vdash_s$ is rational and can be computed in PSPACE. In a subsequent work [5], the authors introduce an approach based on the combination of rational closure and *Defeasible Inheritance Networks* (INs).

## 5 Conclusions

In the first part of the paper we have provided a semantic reconstruction of the well known rational closure, in detail a minimal model semantics based on the idea that preferred rational models are those ones in which the height of the worlds is minimized. Adding suitable possibility assumptions to a knowledge base, such a minimal model semantics corresponds to rational closure.

The correspondence between the proposed minimal model semantics and rational closure suggests the possibility of defining variants of rational closure by varying the ingredients underlying our approach, namely: (i) the properties of the preference relation $<$: for instance just preorder, or multi-linear or weakly-connected; (ii) the comparison relation on models: based for instance on the rank of the worlds or on the inclusion between the relations $<$, or on negated boxed formulas satisfied by a world, as in the logic $\mathbf{P}_{min}$ [12]. The systems obtained by various combinations of these ingredients are largely unexplored and may give rise to useful non-monotonic logics.

In the second part of the paper we have defined a rational closure construction for the Description Logic $\mathcal{ALC}$ extended with a typicality operator and provided a minimal model semantics for it, based on the idea of minimizing the rank of objects in the domain, that is their level of "untypicality". This semantics corresponds to a natural extension to DLs of Lehmann and Magidor's notion of rational closure. We have also extended the notion of rational closure to the ABox, by providing an algorithm for computing it that is sound and complete with respect to the minimal model semantics. Last, we have shown an EXPTIME upper bound for the algorithm.

In future work, concerning Description Logics, we will consider further ingredients in the recipe for non-monotonic DLs. First, we aim to study stronger versions of rational closure that allow to overcome the weaknesses of the basic one, for instance the fact that we cannot reason separately on the inheritance of different properties. Furthermore, non-monotonic extensions of *low complexity* DLs based on the **T** operator have been recently provided [13]. In future works, we aim to study the application of the proposed semantics to DLs of the $\mathcal{EL}$ and DL-Lite families, in order to define a rational closure for low complexity DLs.

## References


1. F. Baader and B. Hollunder. Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *J. Autom. Reasoning*, 15(1):41–68, 1995.
2. Piero A. Bonatti, Carsten Lutz, and Frank Wolter. The Complexity of Circumscription in DLs. *Journal of Artificial Intelligence Research (JAIR)*, 35:717–773, 2009.
3. Katarina Britz, Johannes Heidema, and Thomas Meyer. Semantic preferential subsumption. In G. Brewka and J. Lang, editors, *KR 2008*, pages 476–484, 2008. AAAI Press.
4. G. Casini and U. Straccia. Rational Closure for Defeasible Description Logics. In T. Janhunen and I. Niemelä, editors, *Proc. of JELIA 2010*, LNAI 6341, pages 77–90, 2010. Springer.
5. Giovanni Casini and Umberto Straccia. Defeasible Inheritance-Based Description Logics. In *Proc of IJCAI 2011*, pages 813–818, 2011. Morgan Kaufmann.
6. F. M. Donini, D. Nardi, and R. Rosati. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic (ToCL)*, 3(2):177–225, 2002.
7. T. Eiter, T. Lukasiewicz, R. Schindlauer, and H. Tompits. Combining Answer Set Programming with Description Logics for the Semantic Web. In *KR 2004*, pages 141–151, 2004.
8. N. Friedman and J. Y. Halpern. Plausibility measures and default reasoning. *Journal of the ACM*, 48(4):648–685, 2001.
9. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Preferential vs Rational Description Logics: which one for Reasoning About Typicality? *ECAI 2010*, pp. 1073 - 1074, IOS Press.
10. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. ALC+T: a preferential extension of Description Logics. *Fundamenta Informaticae*, 96:1–32, 2009.
11. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Analytic Tableaux Calculi for KLM Logics of Nonmonotonic Reasoning. *ACM Trans. on Comput. Logics (TOCL)*, 10(3), 2009.
12. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. A nonmonotonic extension of KLM preferential logic P. In *LPAR 2010*, LNCS 6397, pages 317–332, 2010. Springer-Verlag.
13. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. Reasoning about typicality in low complexity DLs: the logics $\mathcal{EL}^{\perp}\mathbf{T}_{min}$ and *DL-Lite*$_c\mathbf{T}_{min}$. In *Proc. of IJCAI 2011*, pages 894–899, 2011. Morgan Kaufmann.
14. L. Giordano, V. Gliozzi, N. Olivetti, and G. L. Pozzato. A NonMonotonic Description Logic for Reasoning About Typicality. *Artificial Intelligence*, pages 165–202, 2012.
15. Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Preliminary result on the definition of a minimal model semantics for Rational Closure . Technical report, Dipartimento di Informatica, Università degli Studi di Torino, 2013.
16. Laura Giordano, Valentina Gliozzi, Nicola Olivetti, and Gian Luca Pozzato. Minimal model semantics and rational closure in description logics. In *Informal Proc. of DL2013)*, CEUR 1014, pages 168–180, 2013.
17. P. Ke and U. Sattler. Next Steps for Description Logics of Minimal Knowledge and Negation as Failure. In *Proc. of DL2008*, CEUR 353, 2008. CEUR-WS.org.
18. S. Kraus, D. Lehmann, and M. Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1-2):167–207, 1990.
19. Adila Alfa Krisnadhi, Kunal Sengupta, and Pascal Hitzler. Local closed world semantics: Keep it simple, stupid! In *Proc. of DL2011*, CEUR 745, 2011.
20. Daniel Lehmann and Menachem Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55(1):1–60, 1992.
21. Boris Motik and Riccardo Rosati. Reconciling Description Logics and rules. *Journal of the ACM*, 57(5), 2010.
22. J. Pearl. System Z: A natural ordering of defaults with tractable applications to nonmonotonic reasoning. In *TARK*, pages 121–135,1990. Morgan Kaufmann.

# Reasoning by Analogy Using Past Experiences

F. Leuzzi[1] and S. Ferilli[1,2]

[1] Dipartimento di Informatica – Università di Bari
{fabio.leuzzi, stefano.ferilli}@uniba.it
[2] Centro Interdipartimentale per la Logica e sue Applicazioni – Università di Bari

**Abstract.** Reasoning by analogy is essential to provide new conclusions helpful to solve a problem. Here we present the definition of a new operator aimed at reasoning by analogy. The proposed reasoner relies on the *Roles Mapping Engine*. It finds analogous roles encoded in descriptions that use domain-specific terminology, overcoming syntactical constraints that limit the relations to have the same name. We employ also a structural similarity function to face cases affected by ambiguity. We evaluate our approach using examples proposed in other works, producing comparable results.

## 1 Introduction

As pointed out by [6], reasoning by analogy is essential in order to produce new conclusions helpful to solve a problem. In particular, some perspectives make this type of reasoning a primary issue. First, in the study of learning, analogies are important in the transfer of knowledge and inferences across different concepts, situations, or domains. Second, analogies are often used in problem solving and reasoning. Third, analogies can serve as mental models to understand new domains. Fourth, analogy is important in creativity. Studies in history science show that analogy was a frequent mode of thought for such great scientists as Faraday, Maxwell, and Kepler. Fifth, analogy is used in communication and persuasion. Sixth, analogy and its cousin, similarity, underlie many other cognitive process. [6] defines the analogies as partial similarities between different situations that support further inferences. Specifically, analogy is defined as a kind of similarity in which the same system of relations holds across different objects. Analogies thus capture parallels across different situations.

This proposal consists in the definition of a new operator aimed at reasoning by analogy. The reasoner relies on the *Roles Mapping Engine* (RME), that finds common roles across descriptions. The long term objective is to embed in such definition a strategy for the cooperation with other reasoning operators.

The remainder of this work is organized as follow: in Section 2 related works are presented with related criticisms, in Section 3 some considerations about the analogy process are reported, then we present the proposed mapping procedure in details, presenting an evaluation in the successive section, finally we conclude with some considerations and future works.

## 2 Related Work

Plenty of works studied an operator aimed to perform reasoning by analogy. The most popular line of thought regards the research of identities between predicates across domain, using as representation formalism propositional or first order logic. In particular, in [13] the author aims to compose goal and sub-goal using analogous experiences. The authors claim that retrieving one (or many) analogies consists in finding *similar* past cases, then imposing to find the same predicate in both the experiences. This proposal contrasts with the canonical definition of analogy, in which only a correlation between roles is expected.

Similar assumption can be found in [10], in which the author proposes a strategy of knowledge projection between domains. This procedure is based on a definition of analogy presented in [9], that relies on the assumption that some given terms are analogous across domains if they are tied by the same predicate (as stated also in [8, 11]). In [5, 3, 7], the central statement is not so different: the author claims that analogy is characterized by the mapping of relations (having same predicate name) between objects, rather than attributes of objects, from base to target. The particular contribution that is slightly different from the other works is that this work isolates four primary classes of matching: literal similarity (in which both predicates and attributes are mapped), analogy (in which mainly predicates are mapped), abstraction (having an analogy like behaviour, but aimed to use the mapping for different goals) and anomaly (that is a wrong trying of analogical mapping). The current class depends on the domain in which an analogy is sought.

Again the same assumption is kept into account in [12], in which the author trains a classifier with a set of word pairs having as label the name of their relationship. The output of its algorithm is a classification, where the learned class describes a given relationship. All the pairs that are part of this class are claimed as analogues. We point out some limitations. In first place, the evaluation of a single relationship for each time is equivalent to consider each relationship in a complex context as independent from the others. In second place, this work proposes a supervised approach that require a concept description with a fixed list of features, such requirement is not always available in real cases.

A different approach has been proposed in [1], in which analogies are carried out using Evolutionary Computation. The authors represent the knowledge in semantic graphs and generate the dataset using common sense knowledge bases. At this point potential analogies are generated through evolution (i.e. using pairs of descriptions as parents probabilistically selected from population with reselection allowed) and evaluated through the Structure Mapping Engine (SME, [7]) that provides a fitness measure score. Despite the novelty of this approach, the methodology relies on the mapping of equal relation only.

In [2], the authors use the SME [7] in order to recognize similar sketches. In particular, they propose a software system in which an expert can draw a sketch that is stored as ground truth. In such a way a student can draw a sketch in turn, that is compared to the stored one with the aim to catch analogical aspects, checking its correctness. Unfortunately, this is a typical task of pattern

recognition, that fits with similarity evaluations. In fact, this work proposes an algorithm that exploits a numerical technique to revise the SME mistakes, instead of considering that analogy involves semantic aspects of reasoning that are far from pattern recognition in sketches.

In a general view, the attempt is to produce analogies searching identities between predicates across domains, we must underline that, despite the reliability of these works (shown by the experimental results), the research of identical predicates alone could be not enough to generate useful analogies.

## 3 The analogical reasoner

In this Section we will present our approach. For the sake of clarity, the description used as prior knowledge is referred as *base* domain, whereas the description of the current problem is referred as *target* domain.

We introduce some assumptions that limit our scopes: (1) we assume that all the descriptions are encoded using the same abstraction level, (2) this work does not keep into account the formalization of the goal, suggesting then all plausible analogies.

**Retrieving oldest useful knowledge** We want to face the retrieval of potentially analogues experiences. A good starting point can be an evaluation of common knowledge, that provides hints about potential points of contact between descriptions. Furthermore, the evaluation of the subnet of common knowledge relations allows to include the direct dependences between common statements. The addition of relational dimension allows the soundness check of the results.

In [11], the authors propose to face this step using a structural similarity evaluation, because they hope that this choice would increase the possibility of retrieval surprising and creative source domains. They describe the experiences in a $n$-dimensional structure space, where each dimension represents a particular topological quality of that domain. In such a way they can project the experiences in a vector space. Unfortunately, using such representation formalism some information are lost (e.g. connections among concepts).

For this reason, we decided to evaluate the similarity using the structural measure proposed in [4] and denoted as *fs*, because it performs a multi-level evaluation combining the similarities at terms, literals and clauses level. We apply the *fs* measure between a given description and each other one in the background knowledge. Unfortunately, such measure presents a drawback. Suppose given two short similar descriptions. In such a case the *fs* score will indicate similarity. Despite its score, the knowledge that can be effectively used for inferences could be poor. For this reason, such score is smoothed by a *multiplicative factor* denoted as *mf*. Given a set of clauses $S$, two clauses $C' \in S$ and $C'' \in S$, *mf* is computed as:

$$mf(C', C'') = \frac{|C'| + |C''|}{2 * L} = \frac{\mu(|C'|, |C''|)}{L}$$

**Algorithm 1** Roles Mapping Engine.

**Input:** A pair of Horn clauses $< C', C'' >$.
**Output:** Two sets: term mappings $\theta_t$ and predicate mappings $\theta_p$.

$\theta_p = \emptyset$
$H =< h', h'' >|$ $h'$ is the head of $C'$, $h''$ is the head of $C''$
$\theta_t \leftarrow \theta_t \cup \{$term mappings in $H\}$
$RA =$ literals having the same predicate in $C'$ and $C''$
$\theta_t \leftarrow \theta_t \cup \{$term mappings in $RA\}$
$\theta_p \leftarrow \theta_p \cup \{$predicate mappings in $RA\}$
**repeat**
  **repeat**
    $\theta_p^{init} \leftarrow \theta_p$, $\theta_t^{init} \leftarrow \theta_t$
    **repeat**
      $\theta_p^{prev} \leftarrow \theta_p$, $\theta_t^{prev} \leftarrow \theta_t$
      $RA =$ literals having the same predicate, and some terms in $\theta_t$
      $\theta_t \leftarrow \theta_t \cup \{$term mappings in $RA\}$
      $\theta_p \leftarrow \theta_p \cup \{$predicate mappings in $RA\}$
    **until** $\theta_p \neq \theta_p^{prev} \vee \theta_t \neq \theta_t^{prev}$
    **repeat**
      $\theta_p^{prev} \leftarrow \theta_p$, $\theta_t^{prev} \leftarrow \theta_t$
      $RA =$ literals having different predicates, and all terms in $\theta_t$
      $RRA = RA$ ranked by reliability score
      $\theta_p \leftarrow \theta_p \cup \{$ predicate mappings in $RRA\}$
      $RA =$ literals having different predicates, and some terms in $\theta_t$
      $RRA = RA$ ranked by reliability score
      $\theta_t \leftarrow \theta_t \cup \{$term mappings in $RRA\}$
    **until** $\theta_p \neq \theta_p^{prev} \vee \theta_t \neq \theta_t^{prev}$
  **until** $\theta_p \neq \theta_p^{init} \vee \theta_t \neq \theta_t^{init}$
  $\theta_p^{score} \leftarrow \theta_p$, $\theta_t^{score} \leftarrow \theta_t$
  $S = (l', l'')$ | $l' \in C'$, $l'' \in C''$, $S$ is $\arg\max_{(l_1,l_2)\in C'\times C''} rs(l_1, l_2)$, $l'$ and $l''$ are not fully mapped
  $\theta_t \leftarrow \theta_t \cup \{$term mappings in $S\}$
  $\theta_p \leftarrow \theta_p \cup \{$predicate mappings in $S\}$
**until** $\theta_p \neq \theta_p^{score} \vee \theta_t \neq \theta_t^{score}$

where: $RA = \{< \{L' \mid L' \subseteq C'\}, \{L'' \mid L'' \subseteq C''\} >, ...\}$ is a set of pairs of sets in which new mappings are sought; $RRA$ is the list of sets in $RA$ ranked by reliability score.

where $L = \arg\max_{C\in S} |C|$. Such a trick could avoid obvious or useless analogies between too short or too unbalanced descriptions.

**Roles Mapping Engine** Reasoning by analogy cannot be reduced to looking for equal predicates across domains, because the derived inference could be useless or trivial. In this section our mapping strategy is presented through the RME.

Each concept is represented in Horn clause logic. Telling our strategy in a nutshell, starting points are sought, then the mapping is expanded in breadth. Both steps are executed keeping consistency requirements.

We will discuss the RME approach using the Algorithm 1. In particular, such algorithm presents iterations on three levels: the external level has an iteration enclosing the whole life cycle of the analogical mapping; the middle level has an

iteration that checks whether one of the inner researches for mappings of identical and non-identical predicates produce novel mappings or not; the internal level has an iteration which aims to search for mappings of identical predicates and another one aimed to research mappings of non-identical predicates.

Such phases will be described in a formal notation and they will be equipped with a running example. Such example refers to the analogy between a proverb stating that "when the fox cannot reach the grapes, he says they are not ripe." and a life context in which "a man cannot have a girl, he spreads a bad opinion about her". We propose such example in order to clarify each step of our proposal.

*Example 1 (Proverb and life context).*

*proverb(fox, grape)* :- wants(fox, grape), cannot_take(fox, grape, fox_does_not_reach_grape), is(fox, crafty), cause(fox_does_not_reach_grape, bad_opinion), says(fox, grape_is_not_ripe), is(grape, not_ripe, grape_is_not_ripe), have(john, bad_opinion).

*situation(john, carla)* :- loves(john, carla), cannot_have(john, carla, john_cannot_have_carla), says(john, carla_is_bad), is(carla, bad, carla_is_bad), uses(jealous, craftiness).

In order to understand the mapping procedure, we need to define what are starting points and how to map terms and predicates as well.

**Definition 1 (Starting point).** *Given two clauses $C'$ and $C''$, a starting point $S$ is a binding $S = [e'/e'']$ where $(e' \in predicates(C') \wedge e'' \in predicates(C'')) \vee (e' \in terms(C') \wedge e'' \in terms(C''))$.*

**Definition 2 (Term mapping).** *Given two clauses $C'$ and $C''$, two sets of literals $L' \subseteq C'$ and $L'' \subseteq C''$, a pair of terms $t'$ and $t''$, a consistent term association $\theta \subseteq terms(C') \times terms(C'')$; then $t'/t''$ is a* term mapping *if*

- *$\{t'/t''\} \cup \theta$ is a consistent term association;*
- *either $\forall l' \in L'$ s.t. $t'$ is a term of $l'$, $\forall l'' \in L''$ s.t. $t''$ is a term of $l''$, both $t'$ and $t''$ have position $p$; or $\exists l', l'' \in L' \times L''$ s.t. $t'$ is a term of $l'$, $t''$ is a term of $l''$ and both $t'$ and $t''$ have name $n$ and position $p$.*

*A term association $\theta$ is said to be* consistent *if it is a bijection,* inconsistent *otherwise.*

**Definition 3 (Predicate mapping).** *Given two clauses $C'$ and $C''$, two sets of literals $L' \subseteq C'$ and $L'' \subseteq C''$ where $\forall l' \in L'$, $l' = p'(t'_1, ..., t'_a)$ and $\forall l'' \in L''$, $l'' = p''(t''_1, ..., t''_a)$, a consistent predicate association $\theta_p \subseteq predicates(C') \times predicates(C'')$, a consistent term association $\theta_t \subseteq terms(C') \times terms(C'')$; then $p'/p''$ is a* predicate mapping *if*

- *$\{p'/p''\} \cup \theta_p$ is a consistent predicate association;*
- *either $p' = p''$; or a matching association defined as $\theta_{l'/l''} = \{t'_1/t''_1, ..., t'_a/t''_a\}$ s.t. $\forall i = 1, ..., a$, $t'_i/t''_i$ is a term mapping, holds (i.e. $\theta_{l'/l''} \subseteq \theta_t$).*

*A predicate association $\theta_p$ is said to be* consistent *if it is a bijection,* inconsistent *otherwise.*

**Definition 4 (Compatibility under term and predicate mapping).** *Two term mappings $\theta'$ and $\theta''$ are* t-compatible *iff $\theta' \cup \theta''$ is consistent. Two literals, or two sequences of literals, are* p-compatible *iff all their predicate mappings are defined and consistent.*

Suppose given two clauses $C' = p'(t'_1, ..., t'_n)$ :- $l'_1, ..., l'_k$ and $C'' = p''(t''_1, ..., t''_m)$ :- $l''_1, ..., l''_h$ that the Algorithm 1 takes in input. Initially, we have an empty global term mapping $\theta_t$ and an empty global predicate mapping $\theta_p$.

The heads mapping could be a good starting point. Since an analogy is sought between the descriptions of the concepts represented by the heads, this choice reflects the intrinsic strength of the heads relationship. More formally, if $n = m$, then $\theta'_t = \{t'_1/t''_1, ..., t'_{n=m}/t''_{n=m}\}$ is a term mapping. Since $\theta_t$ is empty, $\theta_t$ and $\theta'_t$ are t-compatible, then $\theta_t \leftarrow \theta_t \cup \theta'_t$.

Although the entities in the descriptions will play specific roles (likely using a domain specific terminology), it is plausible that part of explanations are encoded using common sense, providing potential points of contact. Then a research of equal predicates across descriptions makes sense. The underlying idea is that two domains in the same knowledge share the representation formalism (implying the common sense as intersection between any pair of descriptions).

The first iteration of the internal level in the Algorithm 1 can be formalized as follow. Using Definitions 2, 3 and 4, we carry on our mappings searching for shared knowledge. Given two subsets $L' \subseteq C'$ and $L'' \subseteq C''$ s.t. $\forall l' \in L'$, $l' = p(t'_1, ..., t'_a)$ and $\forall l'' \in L''$, $l'' = p(t''_1, ..., t''_a)$, then $\theta'_p = \{p/p\}$ is a predicate mapping. If $\theta_p$ and $\theta'_p$ are p-compatible, then $\theta_p \leftarrow \theta_p \cup \theta'_p$.

*Example 2 (Proverb and life context).* Firstly, the heads are mapped, since they have the same arity, obtaining <*grape, carla*>, <*fox, john*>. At this point the literals having a predicate used by both clauses are isolated, in order to search for deterministic alignment. Then we have {*says(fox, grape_is_not_ripe)*} and {*says(john, carla_is_bad)*}, from which <*says/2, says/2*> becomes a mapped predicate and <*grape_is_not_ripe, carla_is_bad*> becomes a mapped term. Going on, the sets {*is(grape, not_ripe, grape_is_not_ripe)*} and {*is(carla, bad, carla_is_bad)*} are isolated, in which some terms are mapped, and from which we can add to the mapped predicates <*is/3, is/3*> and to the mapped terms <*not_ripe, bad*>.

In Algorithm 1, the second iteration of the internal level tries to map non-identical predicates expanding in breadth the previous mappings to those concepts that are syntactically different but that play an analogous role. The research of predicates and terms association goes on until new mappings are done. More formally, suppose given two subsets of literals $L' \subseteq C'$ and $L'' \subseteq C''$ for which $\exists t'/t'' \in \theta_t$ s.t. $\forall l' \in L'$, $t'$ is a term of $l'$ and $\forall l'' \in L''$, $t''$ is a term of $l''$. If exists a term mapping $\theta'_t$ s.t. $\forall t'/t'' \in \theta'_t$, $t'$ is a term of $l' \in L'$ in position $w$, $t''$ is a term of $l'' \in L''$ in position $w$, $\theta_t$ and $\theta'_t$ are t-compatible, then $\theta_t \leftarrow \theta_t \cup \theta'_t$. Moreover, if exists a predicate mapping $\theta'_p$ s.t. $\forall p'/p'' \in \theta'_p$, $\forall l' \in L'$, $l' = p'(t'_1, ..., t'_a)$, $\forall l'' \in L''$, $l'' = p''(t''_1, ..., t''_a)$, $\forall i = 1, ..., a$ then $\exists t'_i/t''_i \in \theta_t$, $\theta_p$ and $\theta'_p$ are p-compatible, then $\theta_p \leftarrow \theta_p \cup \theta'_p$.

**Table 1.** Mapping between proverb and life context.

| Outcome | Base clause | Target clause |
|---|---|---|
| mapped predicates | says/2 | says/2 |
| | is/3 | is/3 |
| | wants/2 | loves/2 |
| | cannot_take/3 | cannot_have/3 |
| mapped terms | fox | john |
| | grape | carla |
| | grape_is_not_ripe | carla_is_bad |
| | not_ripe | bad |
| | fox_does_not_reach_grape | john_cannot_have_carla |
| | crafty | craftiness |

*Example 3 (Proverb and life context).* In the Example 2 a set of mapped predicates and a set of mapped terms have been obtained. Since {<*grape, carla*>, <*fox, john*>} $\subset \theta_t$ (see Algorithm 1), the expansion in breadth produces the pair of sets {*wants(fox, grape)*} and {*loves(john, carla)*}, from which <*wants/2, loves/2*> is added to $\theta_p$. Consequently, {*cannot_take(fox, grape, fox_does_not_reach_grape)*} and {*cannot_have(john, carla, john_cannot_have_carla)*} is found, allowing to map <*cannot_take/3, cannot_have/3*> that can be added to $\theta_p$ and <*fox_does_not_reach_grape, john_cannot_have_carla*> that can be added to $\theta_t$.

The middle level iteration in Algorithm 1 ensures that the internal iterations are repeated until at least one of them extends the mappings. Then, the middle level performs all deterministic mappings based on previous ones.

Unfortunately, it could be the case in which common knowledge does not exist, or cannot be used as starting point, or deterministic mappings are not available. A strategy relying on the structure analysis becomes a primary issue, in order to suggest starting points that do not share the representation. Then a structural similarity is exploited in order to obtain a reliability score between literals (denoted as $rs$). In such a way we design a pair of literals as new starting point, if it is the most similar, it is not already mapped and its literals have the same arity. This is the reason for which the external level exists. It attempts to restart the mappings expansion using such evaluation to overcome the absence of deterministic mappings. This attempt can be seen as the last opportunity to make novel deterministic mappings after the end of the algorithm.

More formally, we obtain a pair $l'/l'' \in C' \times C''$ s.t. $l' = p'(t'_1, ..., t'_a)$, $l'' = p''(t''_1, ..., t''_a)$, $rs(l', l'')$ is the maximum score, $l'/l''$ has not been fully mapped through term and predicate mappings. Then $\theta'_t = \{t'_1/t''_1, ..., t'_a/t''_a\}$ and $\theta'_p = \{p'/p''\}$. If $\theta_t \cup \theta'_t$ is t-compatible, then $\theta_t \leftarrow \theta_t \cup \theta'_t$. If $\theta_p$ and $\theta'_p$ are p-compatible, then $\theta_p \leftarrow \theta_p \cup \theta'_p$.

*Example 4 (Proverb and life context).* At this point, the mapping expanded in breadth pursued in the Example 3 (see Table 1) cannot be carried on because there are not novel deterministic bindings. The similarity function suggests {*is(fox, crafty)*} and {*uses(jealous, craftiness)*}. Here, only <*crafty, craftiness*>

is a deterministic mapping. Since *fox* is already in $\theta_t$ as <*fox, john*>, *jealous* cannot be mapped, impeding the association of the respective predicates.

**Ranking of hypotheses** A known problem is the ranking of the hypotheses to learn. Such problem affects reasoning by analogy, since each new mapping relies on previous ones, and so on. In this proposal, the hypotheses ranking problem has been faced using relative frequencies.

Given a clause $C$ and a literal $l \in C$, the score of $l$ is obtained averaging the relative frequencies for each term $t$ in $l$, then:

$$rf_\mu(l) = \frac{\sum_{i=1}^{a} o_i}{n * a}$$

where $o$ is the number of literals in $C$ in which the $t$ appears, $n$ is the total number of literals in $C$, $a$ is the arity of $l$. The idea behind the $rf_\mu(l)$ scores is to represent the centrality of $l$ in its clause. Then, given two literals $l'$ and $l''$, the reliability score $rs(l', l'')$ for their hypothesis of mapping is computed as:

$$rs(l', l'') = rf_\mu(l') * rf_\mu(l'')$$

Mappings hypotheses are ranked using a descendant $rs(l', l'')$ score. A ranking so defined means that each new mapping maximizes the coverage of literals in both clauses.

**Making inference** During the mapping phase, the attention has been focused on the recognition of analogous roles cross-domains (both for objects and relations). As highlighted in [3], one-to-one alignment has a primary importance, since part of the structural consistency is verified if the bijection holds. For this reason, the inference is carried out starting from a one-to-one alignment of the mapped literals (i.e. both for the predicate and its arguments), and ending with the projection of all the residual knowledge.

Giving a more practical view of the procedure, after the filtering out of the one-to-one correspondences, the procedure seeks base knowledge that could be novel in the target domain. Consistently with the assumption that common knowledge is fundamental for analogical inference, it is possible that the lacking mappings are part of common knowledge, then it is projected using a Skolem function. An inference having Skolem functions is a hypothesis having an unreliability degree directly proportional to the number of Skolem functions.

*Example 5 (Proverb and life context).* The expected explanation of the phenomena sounds like: "John has a bad opinion about Carla because he cannot have the love of Carla, then he uses his craftiness in order to do not appear rejected from the girl". Let us examine the inference hypotheses from the proverb to the life context:

1. *skolem_is(john, craftiness)*
2. *skolem_cause(john_cannot_have_carla, skolem_bad_opinion)*
3. *skolem_has(john, skolem_bad_opinion)*

These hypotheses fully satisfy the expected interpretation.

**Table 2.** Literal mappings between proverb and life context.

| Proverb | Life context |
|---|---|
| proverb(fox, grape) | situation(john, carla) |
| says(fox, grape_is_not_ripe) | says(john, carla_is_bad) |
| is(grape, not_ripe, grape_is_not_ripe) | is(carla, bad, carla_is_bad) |
| wants(fox, grape) | loves(john, carla) |
| cannot_take(fox, grape, fox_does_not_reach_grape) | cannot_have(john, carla, john_cannot_have_carla) |

**Meta-Pattern formalization** Using the RME we can carry out an analogical mapping between each pair of clauses representing experiences, contexts or concepts. From such a mapping we can outline a pattern.

Let us to give a formal view of a pattern. Given two clauses $C' = h' \text{ :- } l'_1, ..., l'_n$ and $C'' = h'' \text{ :- } l''_1, ..., l_m$, a set $\theta_t$ containing the mapped terms and a set $\theta_p$ containing the mapped predicates, we can outline a generalized pattern $C = h \text{ :- } l_1, ..., l_k$ with $k \leq \min(n, m)$, s.t. $\forall l \in C$, $\exists l'/l''$ s.t. $l' \in C'$, $l'' \in C''$, $l' = p'(t'_1, ..., t'_a)$, $l'' = p''(t''_1, ..., t''_a)$, $p'/p'' \in \theta_p$ and $t'_i/t''_i \in \theta_t$ $(\forall i = 1, ..., a)$. Moreover, $l = p(t_1, ..., t_a)$ where: if $p' = p''$, then $p = p' = p''$, otherwise $p = p*$, where $p*$ is a new predicate; if $t'_i = t''_i$, then $t_i = t'_i = t''_i$, otherwise $t_i = t*_i$, where $t*_i$ is a new term.

Since each analogy has a reason to exist with respect to a specific perspective, we cannot expect that each pattern will become general. Conversely, very often analogies remain specific. Then each refinement of a pattern needs to be consider a new pattern, because we have not any reason to consider too specific or useless the older pattern. In any case, we can say that trying an analogical mapping between a pattern and a third description (or another pattern), if the pattern(s) is not fully mapped, a novel and more general *meta-pattern* arises.

The reason behind such a choice is that if the pattern is used to make a novel analogy, many domains can support a mapping or suggest the argument of the relative Skolem function. To note that for each use or refinement of a pattern, the novel domain contributes to support each survived mapping in the pattern, giving further confirmation that the mappings in which the name of the original predicate or term has been preserved are effectively common sense expressions.

The story of each predicate/term in the pattern can be recognized, since the origin of each predicate/term in the pattern is stored using the formalism:

*db(head, type, pattern_name, original_name)*

where 'head' stands for the head of the original clause, 'type' indicates if we are storing a predicate or a term, 'pattern_name' represents the name reported in the pattern and the 'original_name' reports the name in the original clause.

*Example 6 (Proverb and life context).* The mappings presented in Table 1 bring to the literals alignment in Table 2, from which we obtain the following pattern.

*pattern(proverb(fox, grape), situation(john, carla))* :-
wants_OR_loves(fox_OR_john, grape_OR_carla),

cannot_take_OR_cannot_have(fox_OR_john, grape_OR_carla,
fox_does_not_reach_grape_OR_john_cannot_have_carla),
says(fox_OR_john, grape_is_not_ripe_OR_carla_is_bad),
is(grape_OR_carla, not_ripe_OR_bad, grape_is_not_ripe_OR_carla_is_bad).

**What is an analogy?** In order to provide a formal definition of analogy, we need to formalize what is the role that an object plays in a description. Keeping in mind that an object is an abstract entity that can be materialized differently in each description, let us to define a role.

**Definition 5 (Role).** *Given a clause $C$, and a literal $l = p(t_1, ..., t_a)$ s.t. $l \in C$, then the* role *of a term $t_i (1 \leq i \leq a)$ is a set $R_{t_i} = \{l_1, ..., l_k\}$ s.t. $\forall l' \in R_{t_i}$, $l' \in C$ and $t_i$ appears in $l'$. Such a set refers to the role that the term $t_i$ plays with respect to the other terms involved in the relations in which it is involved.*

In any situation a task can be recognized. Any task requests a focus, i.e. the set of objects and relations necessary and sufficient to carry out the current task. Sometimes objects or relations lack, then an analogy could represent a way to hypothesize them from previous experiences. We need to retrieve the experience having an alignable focus, in order to derive hypotheses. In the light of such premises, we define the analogical perspective.

**Definition 6 (Analogical perspective).** *Given two descriptions $< D', D'' >$ representing respectively base and target domain, $K'$ and $K''$ denote the aligned knowledge among $D'$ and $D''$, $T'$ and $T''$ denote the aligned knowledge among $D'$ and $D''$ that solves the current task, an* analogical perspective *holds if either*

$$T' \subseteq K' \subseteq D' \wedge T'' \subseteq K'' \subseteq D''$$

*or*

$$T' \subseteq K' \subseteq D' \wedge T'' \subseteq (K'' \cup I) \subseteq D''$$

*Where $I$ is the inference obtained from the base domain.*

**Definition 7 (Analogy).** *Given two relational descriptions $< D', D'' >$ representable as sets of roles $< R_{D'}, R_{D''} >$ and a perspective $P$, we say that $D'$ and $D''$ are analogous if there exist two subsets $S_{D'} \subseteq R_{D'}$ and $S_{D''} \subseteq R_{D''}$ s.t. a bijective function $f : S_{D'} \rightarrow S_{D''}$ holds, and $f$ satisfies $P$. $f$ satisfies $P$ if for each role $r \in R_{D'} \vee R_{D''}$ necessary to explain $P$, $f$ holds (i.e., $r \in S_{D'} \vee S_{D''}$).*

*Remark 1 (Analogy).* Given a description, each object plays a specific role with respect to each other. Common roles across descriptions are essential to the analogy, relations analysis is fundamental for roles identification, common objects can be a clue for relations analysis.

## 4 Evaluation

We present a qualitative evaluation, in such a way we can further clarify our proposal. In [8] the analogical reasoning is explored from the cognitive psychology perspective. The authors want to study the analogical process of reasoning in humans. Then they give to a group of humans two stories: the former talks about a general that wants to capture a fortress, whereas the latter talks about a doctor that wants to defeat a tumor. These stories are respectively the base and the target domain. The human subjects completed the latter story in the light of the former one (i.e. trying to recognize the knowledge that solves the problem in the target story). Without any suggestion, only the 57% of the subjects provided a complete solution to the analogy, whereas our software system implementing the RME provides directly the correct analogical solution.

**RME assessment** Let us give details about the stories and the relative expected solution. The base story follows.

> A fortress was located in the center of the country. Many roads radiated out from the fortress. A general wanted to capture the fortress with his army. The general wanted to prevent mines on the roads from destroying his army and neighbouring villages. As a result the entire army could not attack the fortress along one road. However, the entire army was needed to capture the fortress. So an attack by one small group would not succeed. The general therefore divided his army into several small groups. He positioned the small groups at the heads of different roads. The small groups simultaneously converged on the fortress. In this way the army captured the fortress.

The base story is translated in a Horn clause having as head *conquer(fortress)*. Each item in the following list encodes a sentence in the story.

1. *located(fortress,center), partof(center,country),*
2. *radiated(oneroad,fortress), radiated(roads,fortress), partof(oneroad,roads),*
3. *capture(general,fortress), use(general,army),*
4. *prevent(general,mines), located(mines,oneroad), located(mines,roads), destroy(mines,army), destroy(mines,villages),*
5. *couldnotuse(army,oneroad),*
6. *capture(army,fortress),*
7. *couldnotuse(subgroup,oneroad),*
8. *splittable(army,subgroups), partof(subgroup,subgroups), partof(subgroups,army), destroy(mines,subgroup), notenough(subgroup),*
9. *distribute(subgroups,roads),*
10. *converge(subgroups,fortress),*
11. *capture(subgroups,fortress).*

The target story follows.

> A tumor was located in the interior of a patient's body. A doctor wanted to destroy the tumor with rays. The doctor wanted to prevent the rays form destroying healthy tissue. As a result the high-intensity rays could not be applied to the tumor along one path. However, high-intensity rays were needed to destroy the tumor. So applying one low-intensity ray would not succeed.

**Table 3.** Military and medical mapping outcomes.

| Outcome | Base clause | Target clause |
|---|---|---|
| mapped predicates | destroy/2 | aredestroyed/2 |
| | capture/2 | defeat/2 |
| | partof/2 | partof/2 |
| | couldnotuse/2 | couldnotuse/2 |
| | splittable/2 | splittable/2 |
| | use/2 | use/2 |
| | radiated/2 | radiated/2 |
| | prevent/2 | prevent/2 |
| | located/2 | located/2 |
| | notenough/1 | notenough/1 |

| Outcome | Base clause | Target clause |
|---|---|---|
| mapped arguments | country | body |
| | center | interior |
| | roads | slits |
| | subgroups | subrays |
| | oneroad | oneslit |
| | army | rays |
| | mines | healthytissue |
| | general | doc |
| | subgroup | ray |
| | fortress | tumor |

The target story becomes a Horn clause having the head *heal(tumor)*. The last item encodes implicit knowledge. In particular such item says that the cancer can be reached from one or many directions. It encodes also that a slit is one of many slits, that the rays can be splitted and that healthy tissue can be damaged and/or destroyed from rays or sub-rays.

1. *located(tumor,interior), partof(interior,body),*
2. *defeat(doc,tumor), use(doc,rays),*
3. *prevent(doc,healthytissue), located(healthytissue,oneslit), located(healthytissue,slits), aredestroyed(healthytissue,rays),*
4. *couldnotuse(rays,oneslit),*
5. *defeat(rays,tumor),*
6. *couldnotuse(ray,oneslit),*
7. (additional) *radiated(oneslit,tumor), radiated(slits,tumor), partof(oneslit,slits), splittable(rays,ray), partof(ray,subrays), partof(subrays,rays), aredestroyed(healthytissue,ray), aredestroyed(healthytissue,subrays), notenough(ray).*

The expected result must contain the lacking knowledge of the target story, that is: "The doctor therefore divided the rays into several low-intensity rays. He positioned the low-intensity rays at multiple locations around the patient's body. The low intensity rays simultaneously converged on the tumor. In this way the rays destroyed the tumor."

Given the mapping in Table 3, the inference hypotheses from base to target domain are:

1. *defeat(subrays,tumor)*
2. *splittable(rays,subrays)*
3. *skolem_distribute(subrays,slits)*
4. *skolem_converge(subrays,tumor)*
5. *aredestroyed(healthytissue,skolem_villages)*

The hypothesis 1 can be identified as the goal of the problem, it is made of fully mapped components, making it a conclusive inference. The hypotheses 2, 3 and 4 represent the procedure useful to reach the goal, their predicates are

**Table 4.** Pattern and Pharmaceutical mapping outcomes.

| Outcome | Pattern | Target clause |
|---|---|---|
| mapped predicates | capture_OR_defeat/2 | purchase/2 |
| | partof/2 | partof/2 |
| | couldnotuse/2 | couldnotuse/2 |
| | located/2 | located/2 |
| | use/2 | use/2 |
| | prevent/2 | mustface/2 |
| mapped terms | country_OR_body | pharmacy |
| | subgroups_OR_subrays | many_partial_amounts |
| | subgroup_OR_ray | partial_amount |
| | center_OR_interior | warehouse |
| | oneroad_OR_oneslit | one_money_source |
| | army_OR_rays | medicine_total_amount |
| | mines_OR_healthytissue | not_enough_money |
| | general_OR_doc | patient |
| | fortress_OR_tumor | medicine |

not mapped, then *skolem* has been added (i.e. a Skolem function), in order to emphasize that the relation could be common sense knowledge, making it projectable without further elaborations. In any case, this type of inference needs to be considered contingent instead of conclusive. The hypothesis 5 does not make sense, then it is a case of fake inference. It is straightforward to highlight that all these statements was absent in the target domain, and have been completely obtained using the base domain. Finally, it is easy to note the consistency with the expected analogical reasoning.

In order to evaluate the similarity of the relational structure, the similarity between the clauses has been evaluated using the *fs* measure [4] before and after the use of the RME. The *fs* ranges between ]0,1[. The original clauses score is 0.65, whereas after alignments, the score became 0.85. The alignment allowed the recognition of the 20% of the structure. Such portion of the clauses appeared unrelated before the RME.

**Patterns assessment** The proposed approach has been evaluated using a qualitative experiment that carries on the running example of analogical reasoning between military and medical domains, that produced a pattern as described in Section 3. We chosen a third domain telling about the purchase of a medicine for which an offertory is needed. We refer to this story as *Pharmaceutical*.

> A medicine is located in the warehouse of a pharmacy. A patient needs to purchase the medicine. The patient must face the problem that his money is not enough. As a result the patient cannot purchase the medicine paying the total price. However, the total amount is needed to purchase the medicine. So applying a minor amount cannot succeed.

The clause encoding such concepts has the head *get(medicine)*.

**Table 5.** Suggestions from original domains.

| Pharmaceutical | Mapping | Source |
|---|---|---|
| mustface/2 | prevent/2 | Military<br>Medical |
| use/2 | use/2 | Military<br>Medical |
| located/2 | located/2 | Military<br>Medical |
| couldnotuse/2 | couldnotuse/2 | Military<br>Medical |
| partof/2 | partof/2 | Military<br>Medical |
| purchase/2 | capture/2<br>defeat/2 | Military<br>Medical |
| medicine | fortress<br>tumor | Military<br>Medical |

| Pharmaceutical | Mapping | Source |
|---|---|---|
| patient | general<br>doc | Military<br>Medical |
| not_enough_money | mines<br>healthytissue | Military<br>Medical |
| medicine_total_amount | army<br>rays | Military<br>Medical |
| one_money_source | oneroad<br>oneslit | Military<br>Medical |
| warehouse | center<br>interior | Military<br>Medical |
| partial_amount | subgroup<br>ray | Military<br>Medical |
| many_partial_amounts | subgroups<br>subrays | Military<br>Medical |
| pharmacy | country<br>body | Military<br>Medical |

1. *located(medicine, warehouse), located(warehouse, pharmacy),*
2. *purchase(patient, medicine), use(patient, medicinetotalamount),*
3. *mustface(patient, notenoughmoney),*
4. *couldnotuse(medicinetotalamount, onemoneysource),*
5. *couldnotuse(partialamount, onemoneysource),*
6. *purchase(medicinetotalamount, medicine),*
7. (additional) *splittable(medicinetotalamount, partialamount), partof(partialamount, manypartialamounts), partof(manypartialamounts, medicinetotalamount).*

In the light of the mappings in Table 4, the RME suggests analogous domains using the stored original mappings, reported in Table 5. For the predicates *use/2*, *located/2*, *couldnotuse/2* and *partof/2* there is no novelty. Instead, the predicates *mustface/2* and *purchase/2* are more interesting because the suggestion indicates that *mustface/2* is the "difficulty" that the protagonist must solve, whereas *purchase/2* stands for the main action on which Pharmaceutical story is built.

The term mapping suggestions have not shared knowledge, then each term is traced to different terms in base domains. For instance *patient* is traced to the main actors in the other domains (*general* and *doc*), such as *medicine* that is the target object in the story is traced to *fortress* and *tumor*, and so on.

As for the analogy between Military and Medical domains, also here the *fs* measure has been exploited to evaluate the gain in the alignment of the portion of the structure for which the analogy holds. The original clauses score is 0.4, whereas the score after the RME is 0.74. Then the 34% of the structure that appeared not related in the original clauses, has been aligned in order to bring out the analogy.

The evaluation of the inference step is important in turn. Since the projection of knowledge is not empty, we can conclude that the analogy with Pharmaceutical domain is well represented by another pattern. The consequence is that a novel pattern has been produced.

## 5 Conclusions

In this work we propose a strategy aimed to recognize analogies and to build meta-patterns for further reasoning, generalizing the analogical schemas. Our proposal differs from the existing literature since it allows to learn patterns representing both the intuition to know any potential solution to the problem, and a computational trick that allows to reuse analogies computed in the past. Moreover, the RME captures non-syntactic alignments without meta-descriptions.

Future improvements will regard relations with opposite sense (perhaps using a common sense knowledge). Another interesting direction could be the use of a probabilistic approach to mappings of non identical predicates. We plan also to face the factual validity using the abductive procedure, in order to check if the inferred knowledge (mapped or projected) is consistent with the constraints of the world. Last but not least, we will equip the solution with an abstraction operator, in order to shift the representation if needed.

## References

[1] Atilim Günes Baydin, Ramon López de Mántaras, and Santiago Ontañón. Automated generation of cross-domain analogies via evolutionary computation. *CoRR*, abs/1204.2335, 2012.

[2] Maria de los Angeles Chang and Kenneth D. Forbus. Using quantitative information to improve analogical matching between sketches. In *IAAI*, 2012.

[3] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63, 1989.

[4] S. Ferilli, M. Biba, N. Di Mauro, T.M. Basile, and F. Esposito. Plugging taxonomic similarity in first-order logic horn clauses comparison. In *Emergent Perspectives in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 131–140. Springer, 2009.

[5] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[6] Dedre Gentner. Analogy. *A companion to cognitive science*, pages 107–113, 1998.

[7] Dedre Gentner and Arthur B. Markman. Structure mapping in analogy and similarity. *American psychologist*, 52:45–56, 1997.

[8] Mary L. Gick and Keith J. Holyoak. Analogical problem solving. *Cognitive Psychology*, 12(3):306–355, 1980.

[9] Makoto Haraguchi. Towards a mathematical theory of analogy. *Bulletin of informatics and cybernetics*, 21(3):29–56, 1985.

[10] Makoto Haraguchi. Analogical reasoning using transformations of rules. In *Proceedings of the 4th conference on Logic programming '85*, pages 56–65, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[11] Diarmuid P. O'Donoghue and Mark T. Keane. A creative analogy machine: Results and challenges. In *Proceedings of the International Conference on Computational Creativity 2012*, 2012.

[12] Peter D. Turney. A uniform approach to analogies, synonyms, antonyms, and associations. *CoRR*, abs/0809.0124, 2008.

[13] Manuela M. Veloso and Jaime G. Carbonell. Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. In *Machine Learning*, pages 249–278. Kluwer Academic Publishers, Boston, 1993.

# Probabilistic Abductive Logic Programming using Possible Worlds

F. Rotella[1] and S. Ferilli[1,2]

[1] Dipartimento di Informatica – Università di Bari
{fulvio.rotella, stefano.ferilli}@uniba.it
[2] Centro Interdipartimentale per la Logica e sue Applicazioni – Università di Bari

**Abstract** Reasoning in very complex contexts often requires purely deductive reasoning to be supported by a variety of techniques that can cope with incomplete data. Abductive inference allows to guess information that has not been explicitly observed. Since there are many explanations for such guesses, there is the need for assigning a probability to each one. This work exploits logical abduction to produce multiple explanations consistent with a given background knowledge and defines a strategy to prioritize them using their chance of being true. Another novelty is the introduction of probabilistic integrity constraints rather than hard ones. Then we propose a strategy that learns model and parameters from data and exploits our Probabilistic Abductive Proof Procedure to classify never-seen instances. This approach has been tested on some standard datasets showing that it improves accuracy in presence of corruptions and missing data.

## 1 Introduction

In the field of *Artificial Intelligence* (AI) so far two approaches have been attempted: numerical/statistical on one hand, and relational on the other. Statistical methods are not able to fully seize the complex network of relationships, often hidden, between events, objects or combinations thereof. In order to apply AI techniques to learn/reason in the real world, one might be more interested on producing and handling complex representations of data than flat ones. This first challenge has been faced by exploiting First-Order Logic (FOL) for representing the world. This setting is useful when data are certain and complete. However this is far for being always true, there is the need for handling incompleteness and noise in data. Uncertainty in relational data makes things even more complex. In particular it can the affect the features, the type or more generally the identity of an object and the relationships in which it is involved. When putting together logical and statistical learning, the former provides the representation language and reasoning strategies, and the latter enforces robustness. This gave rise to a new research area known as *Probabilistic Inductive Logic Programming* [14] (PILP) or *Statistical Relational Learning* [6] (SRL). Although clearly relevant in complex domains such as Social or Biological data, it inherits well-known

problems from *Probabilistic Graphical Models* [11] (PGM) (parameter and model learning, inference).

Furthermore, reasoning in contexts characterized by a high degree of complexity often requires purely deductive reasoning to be supported by a variety of techniques that cope with the lack or incompleteness of the observations. Abductive inference can tackle incompleteness in the data by allowing to guess information that has not been explicitly observed [7]. For instance, if one is behind a corner and a ball comes out of it, a good explanation might be that someone has kicked it. However there are many other plausible explanations for the ball's movement, and thus that inference provides no certainty that someone really stroke the ball. While humans are able to discriminate which explanations are consistent with their previous knowledge and which ones have to be discarded, embedding in machines this capability is not easy.

This work faces two issues: the generation of multiple (and minimal) explanations consistent with a given background knowledge, and the definition of a strategy to prioritize different explanations using their chance of being true. It is organized as follows: the next section describes some related works; Section 3 introduces the Abductive Logic Programming framework; our Probabilistic Abductive Logic Proof procedure is presented in Section 4; then in Section 5 we propose a strategy to exploit our approach in classification tasks; finally there is an empirical evaluation on three standard datasets followed by some considerations and future works.

## 2 Related Work

Abductive reasoning is typically used to face uncertainty and incompleteness. In the literature there are two main approaches: uncertainty has been faced by bayesian probabilistic graphical models [13] and incompleteness by means of the classical approach based on pure logic [7]. Nowadays many works combined logical abduction and statistical inference, which allows to rank all possible explanations and choose the best one.

One of the earliest approaches is [12] where a program contains non-probabilistic definite clauses and probabilistic disjoint declarations $\{h_1 : p_1, ..., h_n : p_n\}$ where an abducible atom $h_i$ is considered true with probability $p_i$ and $i \in \{1, ..., N\}$. That work focuses on the representation language and proposes a simple language for integrating logic and probability. It does not integrate any form of logic-based abductive proof procedure with statistical learning, and considers hard assumptions for the nature of constraints (i.e. only disjoint declarations) in order to keep simple the general procedure. This framework does not assign probabilities to constraints but only to ground literals.

PRISM [16] is a system based on logic programming with multivalued random variables. It provides no support for integrity constraints but includes a variety of top-level predicates which can generate abductive explanations. Since a probability distribution over abducibles is introduced, the system chooses the

best explanation using a generalized Viterbi algorithm. Another central feature of PRISM is its capability to learn probabilities from training data.

Two approaches have merged directed and undirected graphical models with logic. The former [15] exploits Bayesian Logic Programs [14] (BLPs) as a representation language for abductive reasoning and uses the Expectation Maximization algorithm to learn the parameters associated to the model. The latter [9], exploiting Markov Logic Networks (MLN), carries out abduction by adding reverse implications for every rule in the knowledge base (since MLNs provide only deductive inference). However, the addition of these rules increases the size and complexity of the model, resulting computationally expensive. It's worth noting that like MLNs, most SRL formalisms use deduction for logical inference, and hence they cannot be used effectively for abductive reasoning.

An approach for probabilistic abductive Logic programming with Constraint Handling Rules has been proposed in [3]. It differs from other approaches to probabilistic logic programming by having both interaction with external constraint solvers and integrity constraints. Moreover exploits probabilities to optimize the search for explanations using Dijkstra's shortest path algorithm. Hence, the approach explores always the most probable direction, so that the investigation of less probable alternatives is suppressed or postponed. Although the knowledge representation formalism is similar to our one, there are several differences in the approaches. The first difference regards the support for negation. Their framework do not handle negation, and so they simulate it by introducing new predicate symbols (eg. haspower(X) $\rightarrow$ hasnopower(X)). The other difference involves the definition of the constraints that, due to the lack of negation, might be tricky. Conversely our ones allow a flexible and intuitive representation without such restrictions. The last difference is the lack of a strategy to learn the probabilities.

In [1] abduction is conducted with Stochastic Logic Programs [10] (SLP) by considering a number of *possible worlds*. Abductive reasoning is carried out by reversing the deductive flow of proof and collecting the probabilities associated to each clause. Although this approach is probabilistically consistent with the SLP language, abduction through reversion of deduction is quite hazardous because abductive reasoning by means of deduction without constraints may lead to wrong conclusions.

## 3 Abductive Logic Programming framework

Our proposal is based on Abductive Logic Programming [7] (ALP), a high-level knowledge representation framework that allows to solve problems declaratively based on abductive reasoning. It extends Logic Programming by allowing some predicates, called abducible predicates, to be incompletely defined. Problem solving is performed by deriving hypotheses on these abducible predicates (abductive hypotheses) as solutions of the problems to be solved. These problems can be either observations that need to be explained (as in classical abduction) or goals

to be achieved (as in standard logic programming). An abductive logic program is made up of a triple $< P, A, IC >$, where:

- $P$ is a standard logic program;
- $A$ (Abducibles) is a set of predicate names;
- $IC$ (Integrity Constraints or domain-specific properties) is a set of first order formulae that must be satisfied by the abductive hypotheses.

These three components are used to define *abductive explanations.*

**Definition 1 (Abductive explanation).** *Given an abductive theory* $T =< P, A, IC >$ *and a formula* $G$*, an* abductive explanation $\Delta$ *for* $G$ *is a set of ground atoms of predicates in* $A$ *s.t.* $P \cup \Delta \models G$ *(*$\Delta$ *explains* $G$*) and* $P \cup \Delta \models IC$ *(*$\Delta$ *is consistent). When it exists,* $T$ abductively entails $G$*, in symbols* $T \models_A G$*.*

Suppose a clause $C$: $h(t_1, .., t_n)$ :- $l_1, ..l_{n'}$, $h$ is the unique literal of the head, $n$ is its arity, $l_i$ with $i = \{1, ..., n'\}$ are the literals in the body and $\overline{l_i}$ stands for the negative literal $\neg l_i$. For instance, Example 1 defines a logic program $P$ for the concept $printable(X)$ that describes the features that a document must own in order to be printed by a particular printer.

*Example 1 (Example theory for paper domain).*
$c_1 : printable(X) \leftarrow a4(X), \overline{text(X)}$
$c_2 : printable(X) \leftarrow a4(X), table(X), black\_white(X)$
$c_3 : printable(X) \leftarrow a4(X), text(X), black\_white(X)$
$A = \{image,\ text,\ black\_white,\ printable,\ table,\ a4,\ a5,\ a3\}$

In this framework, a proof procedure for abductive logic programs has been presented in [8]. It interleaves phases of *abductive* and *consistency derivations*: an *abductive derivation* is the standard Logic Programming derivation extended in order to consider abducibles. When an abducible literal $\delta$ has to be proved, it is added to the current set of assumptions (if not already included). Since the addition of $\delta$ must not violate any integrity constraint, a *consistency derivation* starts to check that all integrity constraints containing $\delta$ fail. In the *consistency derivation* an *abductive derivation* is used to solve each goal. This might cause an extension of the current set of assumptions.

More specifically an abductive derivation from $(G_1\ \Delta_1)$ to $(G_n\ \Delta_n)$ in $< P, A, IC >$ of a literal from a goal, is a sequence

$$(G_1\ \Delta_1), (G_2\ \Delta_2), ..., (G_n\ \Delta_n)$$

such that each $G_i$ has the form $\leftarrow L_1, ..., L_k$ and $(G_{i+1}\Delta_{i+1})$ is obtained according to one of the following rules:

1. If $L_j$ is not abducible, then $G_{i+1} = C$ and $\Delta_{i+1} = \Delta_i$ where $C$ is the resolvent of some clause in $P$ with $G_i$ on the selected literal $L_j$;
2. If $L_j$ is abducible and $L_j \in \Delta_i$, then $G_{i+1} = \leftarrow L_1, ..., L_{j-1}, L_{j+1}, ..., L_k$ and $\Delta_{i+1} = \Delta_i$;
3. If $L_j$ is a ground abducible, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$ and there exists a *consistency derivation* from $(\{L_j\}\ \Delta_i \cup \{L_j\})$ to$(\{\}\ \Delta')$ then $G_{i+1} = \leftarrow L_1, ..., L_{j-1}, L_{j+1}, ..., L_k$ and $\Delta_{i+1} = \Delta'$

In the first two steps the logical resolution is performed exploiting: (1) the rules of $P$ and (2) the abductive assumptions already made. In the last step before adding a new assumption to the current set of assumption a consistency checking is performed.

A consistency derivation for an abducible $\alpha$ from $(\alpha, \Delta_1)$ to $(F_n\ \Delta_n)$ in $< P, A, IC >$ is a sequence

$$(\alpha\ \Delta_1), (F_1\ \Delta_1), ..., (F_n\ \Delta_n)$$

where:

1. $F_1$ is the union of all goals of the form $\leftarrow L_1, ..., L_n$ obtained by resolving the abducible $\alpha$ with the constraints in $IC$ with no such goal being empty;
2. for each $i > 1$ let $F_i$ have the form $\{\leftarrow L_1, ..., L_k\} \cup F_i'$, then for some $j = 1, ..., k$ and each $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:
   (a) If $L_j$ is not abducible, then $F_{i+1} = C' \cup F_i'$ where $C'$ is the set of all resolvents of clauses in $P$ with $\leftarrow L_1, ..., L_k$ on literal $L_j$ and the empty goal $[] \notin C'$, and $\Delta_{i+1} = \Delta$
   (b) If $L_j$ is abducible, $L_j \in \Delta_i$ and $k > 1$, then $F_{i+1} = \{\leftarrow L_1, ..., L_k\} \cup F_i'$ and $\Delta_{i+1} = \Delta_i$
   (c) If $L_j$ is abducible, $\overline{L_j} \in \Delta_i$ then $F_{i+1} = F_i'$ and $\Delta_{i+1} = \Delta_i$
   (d) If $L_j$ is a ground abducible, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$ and there exists an *abductive derivation* from $(\leftarrow \overline{L_j}\ \Delta_i)$ to $([]\ \Delta')$ then $F_{i+1} = F_i'$ and $\Delta_{i+1} = \Delta'$;
   (e) If $L_j$ is equal to $\overline{A}$ with $A$ a ground atom and there exists an *abductive derivation* from $(\leftarrow \overline{A}\ \Delta_i)$ to $([]\ \Delta')$ then $F_{i+1} = F_i'$ and $\Delta_{i+1} = \Delta'$.

In the first case 2a the current branch is split into the number of resolvents of $\leftarrow L_1, ..., L_k$ with the clauses in $P$ on $L_j$. If we get the empty clause the whole check fails. In the second case if $L_j$ belongs to $\Delta_i$, it is discarded and if it is alone, the derivation fails. In case 2c the current branch is consistent with the assumptions in $\Delta_i$ and so it is dropped from the checking. In the last two cases 2d and 2e the current branch can be dropped if respectively $\leftarrow \overline{L_j}$ and $A$ are abductively probable.

The procedure returns the minimal abductive explanation set if any, otherwise it fails. It is worth noting that according to the implementation in [8] the $\Delta$s in the abductive explanations must be ground. Thus if a non ground abducible is encountered it is first unified with a clause in the background knowledge, with an example or with a previously abduced literal. If this is not possible, it is grounded with a skolem constant.

## 4 Probabilistic Abductive Logic Programming

This work extends the technique shown in Section 3 in order to smooth the classical rigid approach with a statistical one.

In order to motivate our approach we can assume that to abduce a fact, there is the need for checking if there are constraints that prevent such an abduction.

The constraints can be either universally valid laws (such as temporal or physical ones), or domain-specific restrictions. Constraints verification can involve other hypotheses that have to be abduced, and others that can be deductively proved. We can be sure that if a hypothesis is deductively verified, it can be surely assumed true. Conversely, if a hypothesis involves other abductions, there is the need of evaluating all possible situations before assuming the best one. In this view each abductive explanation can be seen as a possible world, since each time one assumes something he conjectures the existence of that situation in a specific world. Some abductions might be very unlikely in some worlds, but most likely in other ones. The likelihood of an abduction can be assessed considering what we have seen in the real world and what we should expect to see.

Moreover, this work handles a new kind of constraints since typically the constraints are only the nand of the conditions and are not probabilistic. They allow a more suitable and understandable combination of situations. The first kind is the classical *nand* denial where at least one condition must be false, the type *or* represent a set of conditions where at least one must be true, and the type *xor* requires that only one condition must be true. Moreover, due to noise and uncertainty in the real world, we have smoothed each constraint with a probability that reflects the degree of the personal belief in the likelihood of the whole constraint.

In Example 2 it can be seen that each probabilistic constraint is a triple $\langle Pr, S, T\rangle$ where $Pr$ is a probability and expresses its reliability, or our confidence on it, $T$ is the *type* of constraint and represents the kind of denial, and $S$ is the set of literals of the constraint. For example $ic_1$ states that a document can be only of one of the three size formats (a3, a4 or a5) and that our personal belief in the likelihood of this constraint is 0.8, $ic_2$ states that a document can be composed either of tables, images or text, and so on.

*Example 2 (Typed Probabilistic Constraints).*
$ic_1 = \langle 0.8, [a3(X), a4(X), a5(X)], xor\rangle$
$ic_2 = \langle 0.9, [table(X), text(X), image(X)], or\rangle$
$ic_3 = \langle 0.3, [text(X), color(X)], nand\rangle$
$ic_4 = \langle 0.3, [table(X), color(X)], nand\rangle$
$ic_5 = \langle 0.6, [black\_white(X), color(X)], xor\rangle$

Our probabilistic approach to logical abductive reasoning can be described from two perspectives: the logical proof procedure and the computation of probabilities.

### 4.1 ALP perspective

The logical proof procedure consists of an extended version of the classical one [7, 8]. While the classical procedure resolved the goal by looking for one minimal abductive explanation set, our approach is to generate different (minimal) abductive explanations and then evaluate them all. This goal can be achieved by changing each assumption that may constrain the subsequent abductive explanations. For example, supposing that a document is black and white, all subsequent

assumptions must be consistent with this. But if we change this assumption, assuming that in another *world* that document is not black and white, we might obtain another set of hypoteses consistent with this last assumption. Each time the procedure assumes something two *possible worlds* can be considered: one where the assumption holds and another where it does not. The overall view is analogous to the exploration of a tree in which each interior node corresponds to a decision (an assumption), as for example changing the truth value of a literal, an edge represents the consequences of that decision and a leaf is the conclusion of the abductive reasoning in a particular *possibile consistent world*. In order to generate such *possible worlds* in which a literal holds and others in which it does not, the classical procedure has been extended by means of introducing two rules for the derivation procedures. The 4th rule of the *abductive derivation* will be:

4. If $\overline{L_j}$ is a ground abducible, $\overline{L_j} \notin \Delta_i$ and $L_j \notin \Delta_i$ and there exists a *consistency derivation* from $(\{\overline{L_j}\}\ \Delta_i \cup \{\overline{L_j}\})$ to $(\{\}\ \Delta')$ then $G_{i+1} = \leftarrow L_1, ..., L_{j-1}, L_{j+1}, ..., L_k$ and $\Delta_{i+1} = \Delta'$

A corresponding rule 2d* for the *consistency derivation* is introduced between 2d and 2e, and states:

2d* If $\overline{L_j}$ is a ground abducible, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$ and there exists an *abductive derivation* from $(\leftarrow L_j\ \Delta_i)$ to $([]\ \Delta')$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta'$;

It can be noted that the rules 3 and 4 of the *abductive derivation* are the choice points on which the procedure can backtrack in order to assume both values (positive and negative) of a literal. In fact in a *possible world* $L_j$ is added to the set of hypoteses by means of 3rd rule, and in the other *possible world* $\overline{L_j}$ holds by means of 4th rule. The analogous choice points in the *consistency derivation* are respectively the rules 2d* and 2d.

The choice of the predicate definition, that is the 1st rule of the *abductive derivation*, is another choice point where the procedure can backtrack to explore different explanations and consequently other *possible worlds*. In fact choosing different definitions, other literals will be taken into account and thus other constraints must be satisfied, and so on. It is worth noting that if some assumptions do not preserve the consistency, the procedure discards them along with the corresponding *possible worlds*. Section 4.2 will present a probabilistic strategy to choose the most likely explanation among all *possible worlds*.

*Example 3 (Observation $o_1$, Query and Possible Explanations).*

| $a4(o_1)$ | ?- $printable(o_1)$ |
|---|---|
| $\Delta_1 = \{\overline{text(o_1)}, table(o_1)\}$ | $Ic_1 = \{ic_2, ic_3, ic_4\}$ |
| $\Delta_2 = \{text(o_1), \overline{table(o_1)}, image(o_1)\}$ | $Ic_2 = \{ic_2, ic_3, ic_4\}$ |
| $\Delta_3 = \{table(o_1), black\_white(o_1)\}$ | $Ic_3 = \{ic_2, ic_4, ic_5\}$ |
| $\Delta_4 = \{text(o_1), black\_white(o_1)\}$ | $Ic_4 = \{ic_2, ic_3, ic_5\}$ |

In order to motivate our point of view, let's consider Example 3 belonging to the "paper" domain presented in Section 3. In order to abductively cover $o_1$ by means of the concept definition $printable(X)$ (i.e. query ?- $printable(o_1)$), there is the need for abducing other literals since the concept definitions $c_i$ with $i = \{1, ..., 3\}$ need some facts that are not in the knowledge base (KB) (i.e. only

$a4(o_1)$ holds). This is the classical situation in which deductive reasoning fails and there is the need of abductive one.

The procedure executes an *abductive derivation* applying the 1st rule and obtaining one of the resolvent in $P$, for instance, the first clause $c_1 : printable(X) \leftarrow a4(X), \overline{text(X)}$. In this case $a4(o_1)$ holds, and so we can exclude it from the abductive procedure continuing with the next literal $\overline{text(o_1)}$. This literal does not hold, and so there is the need of abducing it. This abduction involves the verification of $ic_2$ and $ic_3$ by the *consistency derivation*. Since $ic_2$ is a *or* constraint and thus at least one literal must be true, there are two possible ways to satisfy it considering that $text(o_1)$ must not hold (by current hypothesis). Thus there are two *possible worlds*, one in which $table(o_1)$ holds, and the other in which $table(o_1)$ does not.

In the former *world* rule 2d* fires and $table(o_1)$ is abduced by performing a *consistency derivation* on the constraints $ic_2$ and $ic_4$. The former constraint is already verified by means of the previous abductions (rule 2b). Since $ic_4$ is a *nand* constraint and thus at least one literal must be false, it is satisfied because *color* is not abducible (see abducible predicates $A$ in Example 1) and does not hold (if a literal is not abducible, its value depends on background knowledge, i.e. rule 2a). Thus coming back to the initial abduction $\overline{text(o_1)}$, $ic_2$ is satisfied abducing $table(o_1)$ and $ic_3$ is already satisfied (since it is a *nand* constraint, it is falsified by abducing $\overline{text(o_1)}$). The first abductive explanation $\Delta_1$ for the goal $printable(o_1)$ is so formed by the abductions of $\overline{text(o_1)}$ and $table(o_1)$ (see Example 3).

Similarly to the classical procedure that in backtracking returns all minimal abductive explanations, our procedure comes back to each choice point changing the truth values of the literals in order to explore different possible explanations (and thus other possible worlds). The first choice point, as mentioned above, regards the abduction of $table(o_1)$. While in the former *possible world* $table(o_1)$ holds, now the procedure abduces $\overline{table(o_1)}$ (rule 2d) thus exploring the other *possible worlds*. In this *world* to abduce $\overline{table(o_1)}$ the constraints $ic_2$ and $ic_4$ are taken into account. The former constraint is verified by abducing $\overline{image(o_1)}$ (i.e. *or* constraint) and the latter is satisfied by the initial abduction $\overline{text(o_1)}$ as in explanation $\Delta_1$. Then $\overline{text(o_1)}$ can be abduced also in this *world* since $ic_2$ has been just verified and $ic_3$ as before. So $\Delta_2$ is the second explanation obtained by changing the assumption on literal $table(o_1)$ and thus exploiting another *possible world*.

The explanations $\Delta_1$ and $\Delta_2$ are the only two *possible worlds* given the first clause $c_1$ because other literal configurations are inconsistent. Hence, the procedure comes back to last backtracking point that were the choice of the predicate definition. This time the 1st rule of the *abductive derivation* can be applied to obtain the second predicate definition $c_2 : printable(X) \leftarrow a4(X), table(X), black\_white(X)$. So there is the need of abducing $table(o_1)$ and $black\_white(o_1)$. The former can be abduced because $ic_2$ is satisfied (i.e. *or* constraint), and $ic_4$ is verified by means of $color(o_1)$. The latter literal can be abduced because $ic_5$ is a *xor* constraint and thus at most one literal must be true. In explanation $\Delta_3$ no other

literals must be abduced by the *consistency derivation* and so no other worlds must be explored since the abduced literals $table(o_1)$ and $black\_white(o_1)$ are constrained by the predicate definition.

Once again the procedure comes back to last backtracking point and chooses the predicate definition $c_3 : printable(X) \leftarrow a4(X), text(X), black\_white(X)$. The *abductive derivation*, similarly to the previous run, returns only one possible explanation $\Delta_4$ as it can been seen in Example 3.

It easy to note that rules (2d) and (2d*) are candidate entry points for backtracking in the *consistency derivation* , and rules 1,3 and 4 are the analogues in the *abductive derivation*. In this way all possible (minimal) explanations are obtained along with all *possible consistent worlds*.

### 4.2 Probabilistic perspective

After all different explanations have been found by the above abductive proof procedure, the issue of selecting the best one arises. The simple approach of choosing the minimal explanation is reliable when there is only one minimal proof or when there are no ways to assess the reliability of the explanations. However, Example 3 shows that there might be different explanations for the same observation and so we need to assess the probability of each explanation in order to choose the best one. To face this issue, our approach regards each set of abductions as a *possible world* and so a chance of being true can be assigned to it. The abduction probability of each ground literal through *the possible worlds* can be obtained considering two aspects: the chance of being true in the real world and all sets of assumptions made during the *consistency derivation* in each *possible world*.

Let's introduce some notation: $\Delta = \{P_1 : (\Delta_1, Ic_1), ..., P_T : (\Delta_T, Ic_T)\}$ is the set of the $T$ consistent *possible worlds* that can be assumed for proving a goal $G$ (i.e. the observation to be proved). Each $(\Delta_i, Ic_i)$ is a pair of sets: $\Delta_i = \{\delta_1, ..., \delta_J\}$ contains the ground literals $\delta_j$ with $j \in \{1, ..., J\}$ abduced in a single abductive proof, and $Ic_i = \{ic_1, ..., ic_K\}$ is the set of the constraints $ic_k$ with $k = \{1, ..., K\}$ involved in the explanation $\Delta_i$. Both $\Delta_i$ and $Ic_i$ may be empty. Moreover, we have used the following symbols in our equations: $n(\delta_j)$ is the number of true grounding of the predicate used in literal $\delta_j$, $n(cons)$ is total number of constants encountered in the world, $a(\delta_j)$ is the arity of literal $\delta_j$ and $P(ic_k)$ is the probability of the $kth$-constraint. Thus the chance of being true of a ground literal $\delta_j$ can be defined as:

$$P(\delta_j) = \frac{n(\delta_j)}{\frac{n(cons)!}{(n(cons)-a(\delta_j))!}} \tag{1}$$

Then the unnormalized probability of the abductive explanation can be assessed by Equation 2 and the probability of the abductive explanation normalized over

all $T$ consistent worlds can be computed as in Equation 3:

$$P'_{(\Delta_i, Ic_i)} = \prod_{j=1}^{J} P(\delta_j) * \prod_{k=1}^{K} P(ic_k) \quad (2) \qquad P_{(\Delta_i, Ic_i)} = \frac{P'_{(\Delta_i, Ic_i)}}{\sum_{t=1}^{T} P'_{(\Delta_t, Ic_t)}} \quad (3)$$

Equation 1 expresses the ratio between true and possible groundings of literal $\delta_j$. The intuition behind this equation can be expressed with a simple example: given a feature $f(\cdot)$ and an item *obj* that does not have such a feature, if we want to assess the probability that *obj* owns $f(\cdot)$ (i.e. $P(f(obj))$), we should consider how often we found items that hold $f(\cdot)$ over all items that might own it in real world.

It is worth noting that we are considering the Object Identity [17] assumption and so within a clause, terms (even variables) denoted with different symbols must be distinct (i.e. they must refer to different objects). Thus only literal groundings without constants repetitions are allowed in Equation 1. It is important to underline that such a bias does not limit the expressive power of the language, since for any clause/program it is possible to find an equivalent version under Object Identity. The first part of the formula 2 encloses the probability that all abduced literals are true in that particular *world*. The second part expresses the reliability of the constraints involved in the $i$-th abductive explanation. Although our approach is focused on the computation of the *most probable explanation* and hence there is no need of normalizing the probabilities of the explanations over all possible worlds (i.e some worlds are ruled out due to the presence of integrity constraints), it follows that Equation 3 is presented for completeness. The probability of $\overline{\delta_j}$ is equal to $1 - P(\delta_j)$.

*Example 4 (Probability assessment of the Abductive Explanations).*
$A = \{0.2{:}image, 0.4{:}text, 0.1{:}black\_white, 0.6{:}printable, 0.1{:}table, 0.9{:}a4, 0.1{:}a5, 0.1{:}a3\}$
$P'(\Delta_1, Ic_1) = 0.00486 \quad P'(\Delta_2, Ic_2) = 0.00875$
$P'(\Delta_3, Ic_3) = 0.00162 \quad P'(\Delta_4, Ic_4) = 0.00648$

For the sake of clarity, each abducible literal of the current example has been labeled with its probability using formula (1) as shown in Example 4. For instance, the probability of the first explanation using (2) can be computed as $P'_{(\Delta_1, Ic_1)} = P(\overline{text(o_1)}) * P(table(o_1)) * P(ic_2) * P(ic_3) * P(ic_4)$ and thus $P'_{(\Delta_1, Ic_1)} = 0.6 * 0.1 * 0.9 * 0.3 * 0.3 = 0.00486$. Then, Example 4 shows the probability of all explanations computed using equation (2).

Finally we can state the maximum probability among all abductive explanations. It corresponds to the maximum between all $T$ possible consistent worlds (in this example $T$ is equal to 4) s.t. $P'(printable(o_1)) = \max_{1 \leq i \leq T} P'_i(\Delta_i, Ic_i)$, that is $\Delta_2$. It is worth noting that this behaviour claims the need of the logical abductive reasoning to be supported by a probabilistic assessment of all abductive explanations rather than relying on the minimal one.

## 5 Improving Classification Exploiting Probabilistic Abductive Reasoning

Now the above proof procedure can be exploited to classify never-seen instances. In particular we first learns from data the model (i.e. the Abductive Logic Program $< P, A, IC >$) and the parameters (i.e. literals probabilities), and then our Probabilistic Abductive Logic proof procedure can be exploited to classify new instances. The strategy, presented in Algorithm 1, can be split into two

**Algorithm 1** Probabilistic Classification Algorithm

**Require:** $A$ is the set of abducibles, a couple $<Train_i, Test_i>$
**Ensure:** $Pred_i$, the set of examples labelled with most likely class.

```
1: T_i ← learn_background_theory(Train_i)
2: IC_i ← learn_integrity_constraints(Train_i)
3: ProbLit_i ← compute_literals_probabilities(Test_i)
4: Pred_i = ∅
5: for each example e in Test_i do
6:    R = ∅
7:    for each class c in T_i do
8:       <P(c,e), Δ_p> ← probabilistic_abductive_proof(ProbLit_i, c, e)
9:       <P(¬c,e), Δ_n> ← probabilistic_abductive_proof(ProbLit_i, ¬c, e)
10:      if P(c,e) > P(¬c,e) then
11:         R ← R ∪ <P(c,e), Δ_p>
12:      else if P(c,e) < P(¬c,e) then
13:         R ← R ∪ <P(¬c,e), Δ_n>
14:      else
15:         discard e, inconsistency
16:   <P(c*,e), Δ*> ← most_likely_class_in(R)
17:   Pred_i ← Pred_i ∪ <P(c*,e), Δ*>
```

parts: the former prepares the data (model and parameters), and the latter performs the classification. In the former part, given a train set $Train_i$ and a set of abducible literals $A$ (possibly empty) our approach learns the corresponding Theory $T_i$ (line 1) by exploiting INTHELEX [4], a well-known ILP system, and then obtains the integrity constraints (line 2) with the procedure described in [5]. Such procedure returns a set of *nand* constraints of different sizes and descriptor type domain. This last information can be useful to define a new kind of constraint called *type_domain* that can be dealt as an *xor* constraint. In fact if the descriptor type domain for the color property is {blue, red, yellow, black, green}, and the object X is part of an observation, it will be impossible to abduce two different color descriptors from the above set applied to X. However since our procedure handles natively a probability value associated to the constraints, and this procedure does not return those values, the constraints should be manually labelled or they will be automatically considered true with probability of 1.0. In any case, the set of abducible $A$ can be left empty, because our system considers

abducible all predicates without definitions in $T_i$. The last step of the first part (line 3) computes the Equation 1 for each literal in $Train_i$ before starting the abductive procedure since those values depend only on $Train_i$.

Hence, the second part of the strategy starts at line 5. As can be seen at lines 8 and 9, our algorithm tries to abductively cover the example considering both as positive and as negative for the class $c$. In fact, when an example is considered negative, our procedure discovers all *possibile worlds* in which it cannot be abductively proved as instance of concept $c$. Specularly if the example is positive, it discovers all the *possibile worlds* in which must be abduced something to prove it. Those two executions return an explanation probability that can be compared each other in order to choose the best class. Then the algorithm selects the best classification between all concept probabilities as can be seen at line 16.

It is worth noting that this strategy cannot be performed by a pure abductive logic proof procedure since in such context we do not need a logical response (true/false) but we want a probabilistic value.

## 6 Experimental Evaluation

The evaluation is aimed at assessing the quality of the results obtained by the probabilistic classification when it faces incomplete and noisy data. All experiments were performed on datasets obtained from UCI [2] machine learning repository.

A 10-fold split of each dataset has been performed in order to obtain a set of 10 couples $<Train, Test>$. Then each test-set has been replaced by a set of corrupted versions, in which we removed at random a K% of each example description, with K varying from 10% to 70% with step 10. This procedure has been repeated 5 times for each corrupted test-set in order to randomize the example corruption. In this way 35 test-sets for each fold have been obtained (7 levels of corruption by 5 runs for level). In order to compare the outcomes with a complete test-set we exploited deductive reasoning on the original test-set (i.e corruption level 0%). Moreover we exploited only deductive reasoning to the same corrupted test-set in order to show the improvement of our approach. The maximum length of constraints has been set to 4 for all datasets. Since we do not have any previous knowledge on the datasets we assumed true all obtained constraints with a probability of 1.0 as described in the Section 5.

Then the performances of the system can be evaluated with the aim of understanding how the approach is sensible to the progressive lack of knowledge across the 10 folds. The following synthetic descriptions of the datasets refer to values averaged on the folds.

**Breast-Cancer** contains 201 instances of the benign class and 85 instances of the malignant class and each instance is described by 9 literals. There is the presence of less than 10 instances with missing values. The learned theory consists of 30 clauses where each one has an average of 6 literals in the body. The learned integrity constraints are 1784 (55% are constraints of length 4, 35%

of length 3 and 10% of length 2), and 9 type domain constraints (one for each literal in the example description language).

**Congressional Voting Records** contains 435 instances (267 democrats, 168 republicans) classified as democrats or republicans according to their votes. Each instance is described by 16 literals. The obtained theory consists of 35 clauses where each one has an average of 4.5 literals in the body. The learned integrity constraints are 4173 (16% are constraints of length 4, 37% of length 3 and 47% of length 2), and 16 type domain constraints (as before).

**Tic-Tac-Toe** dataset contains 958 end game board configurations of tic-tac-toe (about 65.3% are positive), where x is assumed to have played first. The target concept win(x) represents one of the 8 possible ways to create a three-in-a-row. Thus, each instance is described by 8 literals. The learned theory consists of 18 clauses where each has an average of 4 literals in the body. The learned integrity constraints are 1863 (99% are constraints of length 4, 1% of length 3), and 16 type domain constraints (as before).

### Results and Discussion

Figure 1 shows the average accuracy obtained on each dataset with respect to

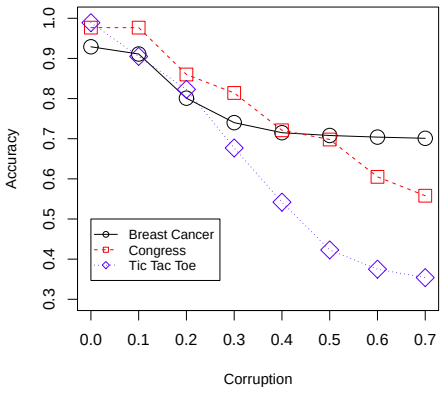



**Figure 1:** Average Accuracy Curves

| Dataset | Corr. | Abductive Reas. | | | Deductive Reas. | | |
|---|---|---|---|---|---|---|---|
| | | *Prec.* | *Rec.* | $F_1$ | *Prec.* | *Rec.* | $F_1$ |
| Breast | 0% | 0.891 | 0.870 | 0.881 | 0.891 | 0.870 | 0.881 |
| | 10% | 0.865 | 0.835 | 0.850 | 0.634 | 0.454 | 0.227 |
| | 20% | 0.853 | 0.411 | 0.556 | 0.571 | 0.118 | 0.195 |
| | 30% | 0.800 | 0.188 | 0.584 | 0.500 | 0.029 | 0.056 |
| | 40% | 1.000 | 0.059 | 0.111 | — | — | — |
| | 50% | 1.000 | 0.035 | 0.068 | — | — | — |
| | 60% | 1.000 | 0.023 | 0.046 | — | — | — |
| | 70% | 1.000 | 0.012 | 0.023 | — | — | — |
| Congress | 0% | 1.000 | 0.961 | 0.980 | 1.000 | 0.961 | 0.980 |
| | 10% | 1.000 | 0.961 | 0.981 | 0.971 | 0.793 | 0.873 |
| | 20% | 1.000 | 0.769 | 0.869 | 0.971 | 0.761 | 0.853 |
| | 30% | 1.000 | 0.680 | 0.809 | 0.982 | 0.714 | 0.827 |
| | 40% | 1.000 | 0.538 | 0.700 | 0.979 | 0.623 | 0.761 |
| | 50% | 1.000 | 0.500 | 0.667 | 1.000 | 0.425 | 0.596 |
| | 60% | 1.000 | 0.346 | 0.514 | 1.000 | 0.333 | 0.500 |
| | 70% | 1.000 | 0.269 | 0.424 | 1.000 | 0.264 | 0.418 |
| TikTakToe | 0% | 1.000 | 0.983 | 0.992 | 1.000 | 0.983 | 0.992 |
| | 10% | 1.000 | 0.833 | 0.909 | 0.842 | 0.743 | 0.789 |
| | 20% | 1.000 | 0.730 | 0.844 | 0.808 | 0.531 | 0.641 |
| | 30% | 1.000 | 0.508 | 0.673 | 0.796 | 0.387 | 0.521 |
| | 40% | 1.000 | 0.302 | 0.463 | 0.829 | 0.261 | 0.397 |
| | 50% | 1.000 | 0.127 | 0.225 | 0.697 | 0.103 | 0.180 |
| | 60% | 1.000 | 0.048 | 0.090 | 0.777 | 0.031 | 0.060 |
| | 70% | 1.000 | 0.016 | 0.031 | 1.000 | 0.004 | 0.009 |

**Table 1:** Results of the experiments

the corruption levels. The probabilistic classification performed on the first two datasets allows to assess the strength and robustness of the approach. In fact their accuracy curves go down less fast than the third's one and even when the

70% of each example description has been removed, the system is able to classify never-seen instances with a mean accuracy of 0.7 and 0.6 respectively.

In order to understand the non-linear trend of the accuracy on the third dataset we can analyze Table 1. It shows the classification performances averaged within the same fold (i.e. 5 random corruptions) and between different folds for each level of corruption. We can find a sharp decay of recall for the third dataset in correspondence of the descending accuracy curve. This happens for two reasons: each example is less described than the ones of the other datasets (8 literals for description) and its background theory consists of an average of 4 literals for clause. Those two aspects make the approach more prone to abduce few literals (often a single one) to make positive examples never covered by the class definitions.

Comparing these results with the deductive approach, it can be noted that in the first dataset after the 30% of corruption, no positive example has been classified correctly. This deficiency has never affected our approach in all experiments. The performances of the deductive approach on last two datasets degrade faster compared to the abductive one.

In general if the examples are less described (as in Tic-Tac-Toe and Breast-Cancer), the number of misclassifications increases due to missing information. It follows that corruption levels greater than 50% rise up the chance of removing important object features, and so such experiments have been performed only for putting stress on the proposed approach.

## 7 Conclusions

Reasoning in very complex contexts often requires pure deductive reasoning to be supported by a variety of techniques that can cope with incomplete and uncertain data. Abductive inference allows to guess information that has not been explicitly observed. Since there are many explanations for such guesses, there is the need for assigning a probability to them in order to choose the best one.

In this paper we propose a strategy to rank all possible minimal explanations according to their chance of being true. We have faced two issues: the generation of multiple explanations consistent with the background knowledge, and the definition of a strategy to prioritize among different ones using their chance of being true according to the notion of *possible worlds*. Another novelty has been the introduction of probabilistic integrity constraints rather than hard ones as in the classical Abductive Logic Programming framework.

The proposal has been described from two perspectives: the abductive proof procedure and the computation of the probabilities. The former, extending the classical one, allows to generate many different explanations for each abduction, while the latter provides a probabilistic assessment of each explanation in order to choose the most likely one. Moreover we introduce a strategy to exploit our Probabilistic Abductive Proof procedure in classification tasks. The approach has been evaluated on three standard datasets, showing that it is able to cor-

rectly classify unseen instances in the presence of noisy and missing information. Further studies will be focused on learning the probabilistic constraints, and on the use of a richer probabilistic model for the literal probability distribution. Then we aim to exploit the Probabilistic Abductive Proof Procedure in other tasks such as natural language understanding and plan recognition.

## References

[1] Andreas Arvanitis, Stephen H. Muggleton, Jianzhong Chen, and Hiroaki Watanabe. Abduction with stochastic logic programs based on a possible worlds semantics. In *In Short Paper Proc. of 16th ILP*, 2006.

[2] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[3] Henning Christiansen. Implementing probabilistic abductive logic programming with Constraint Handling Rules. pages 85–118. 2008.

[4] Floriana Esposito, Giovanni Semeraro, Nicola Fanizzi, and Stefano Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 38:133–156, 2000.

[5] Stefano Ferilli, Teresa M. A. Basile, Nicola Di Mauro, and Floriana Esposito. Automatic induction of abduction and abstraction theories from observations. In *Proc. of the 15th ILP*, ILP'05, pages 103–120, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] Lise Carol Getoor. *Learning statistical models from relational data*. PhD thesis, Stanford, CA, USA, 2002. AAI3038093.

[7] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming, 1993.

[8] Antonis C. Kakas and Fabrizio Riguzzi. Abductive concept learning. *New Generation Comput.*, 18(3):243–294, 2000.

[9] Rohit J. Kate and Raymond J. Mooney. Probabilistic abduction using markov logic networks. In *Proceedings of the IJCAI-09 Workshop on Plan, Activity, and Intent Recognition (PAIR-09)*, Pasadena, CA, July 2009.

[10] S. Muggleton. Stochastic Logic Programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logi c Programming*. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[11] Judea Pearl. Graphical models for probabilistic and causal reasoning. In *The Computer Science and Engineering Handbook*, pages 697–714. 1997.

[12] David Poole. Probabilistic horn abduction and bayesian networks. *Artif. Intell.*, 64(1):81–129, 1993.

[13] David Poole. Learning, Bayesian probability, graphical models, and abduction. In *Abduction and Induction: Essays on their Relation and Integration, Chapter 10*, pages 153–168. Kluwer, 1998.

[14] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *ALT*, pages 19–36, 2004.

[15] Sindhu V. Raghavan. Bayesian abductive logic programs: A probabilistic logic for abductive reasoning. In Toby Walsh, editor, *IJCAI*, pages 2840–2841, 2011.

[16] Taisuke Sato. Em learning for symbolic-statistical models in statistical abduction. In *Progress in Discovery Science, Final Report of the Japanese Discovery Science Project*, pages 189–200, London, UK, UK, 2002. Springer-Verlag.

[17] Giovanni Semeraro, Floriana Esposito, Donato Malerba, Nicola Fanizzi, and Stefano Ferilli. A logic framework for the incremental inductive synthesis of datalog theories. In Norbert E. Fuchs, editor, *LOPSTR*, volume 1463 of *Lecture Notes in Computer Science*, pages 300–321. Springer, 1997.

# Explicit Constructive Logic ECL: a New Representation of Construction and Selection of Logical Information by an Epistemic Agent

Paolo Gentilini[1] and Maurizio Martelli[2]

[1] DIMA, University of Genoa , Via Dodecaneso 35, 16146 Genova, and IMATI-CNR, via de Marini 6, 16149 Genova, Italy
[2] DIBRIS, University of Genoa, Via Dodecaneso 35, 16146 Genova, Italy

**Abstract.** One of the seminal goals of *Explicit Constructive Logic* (*ECL*) is to provide a constructive formulation of full higher order logic (*Classical Type Theory* $\mathbf{LK}_\omega$) that can be seen as a foundation for knowledge representation. Moreover, the development of this work has produced the basis of a new approach to constructivism in Logic.*ECL* is introduced as a sub-system $\mathbf{Z}_\omega$ of $\mathbf{LK}_\omega$. Also the first order case **Z1** and the propositional case **ZP** of *ECL* are examined. A comparison between *ECL*'s constructivism and the corresponding features of *Intuitionistic Logic*, and *Constructive Paraconsistent Logic* is proposed.

**Keywords:** Constructivism in Logic, Higher Order Logic, Intuitionistic and Constructive Paraconsistent Logic

## 1 Introduction

Full higher order logic can be an extremely powerful tool for knowledge representation if some of its features could be simplified and controlled. A constructive formulation of it is the goal of Explicit Constructive Logic (ECL) and will be presented in this paper. We will start from the sequent version $\mathbf{LK}_{\boldsymbol{\omega}}$ of Classical Type Theory as presented in [7] where the Church formalism is used and logical connectives are expressed as typed formulas.For the extensive definitions of the syntax of typed language and the basic notions of proof-theory (sequents, rules, proof-trees and so on) see [7] Sections 2 and 3. To realize the foundational principles of the *ECL* inference we will define the systems $\mathbf{Z}_\omega$ and $\mathbf{Z}^*_\omega$,(included in $\mathbf{LK}_{\boldsymbol{\omega}}$), that, even maintaining a very high expressive power, show strong constructivity properties and could admit a new organization of proofs (through a *Normal Form Theorem*). $\mathbf{Z}_\omega$ could be also seen as a generalization, at the theoretical level, of the features of Higher Order Uniform Logic, introduced in [8] to express Higher Order Logic Programming. Indeed, in both cases the very specific behaviour of proofs is that the principal formula in the conclusion of a logical rule can be deduced only if some constraints on the introductions of the auxiliary formulas in the rule premise(s) are respected. We will also examine the first order case **Z1** and the propositional case **ZP** of *ECL*. The real novelty

of any proposed new logical framework must arise clearly at the propositional level, and this is the case for Intuitionistic Logic and Paraconsistent Logic. The Church formalism is maintained also for the first order case and the propositional case, since it is very convenient for a deep analysis of logical connectives. A parallel and a comparison between $ECL$ and Intuitionistic Logic and Paraconsistent Logic *as different kinds of constructive logics* will be often proposed in this paper. The constructivity of Intuitionistic Logic [10] doesn't need explanations. As to the constructivity of Paraconsistent Logic we consider only a well delimited area of paraconsistency, given by the Logics of Formal Inconsistency (**LFI**) and the included $C$-system family, introduced in [2]. The formal notion of *constructive paraconsistent logic* is introduced in [6].

## 2 Explicit Logical Constructivism: Syntactic Environment and Epistemological Basis

We will now give a short synthesis of the epistemological basis of *Explicit Constructive Logic*: the style will be heuristic and intuitive.

We will use the logical connectives à la Church [3]:

$$\left\{\neg_{o\to o},\ \wedge_{o\to(o\to o)},\ \vee_{o\to(o\to o)},\supset_{o\to(o\to o)},\ \forall_{(\alpha\to o)\to o}\ ,\ \exists_{(\alpha\to o)\to o}\ ,\ \bot_o\ ,\ \top_o\right\}$$

where the type subscript is in general omitted in writing formulas. We call *judgments* of the epistemic subject the *compound logical formulas*: we want to convey the idea that logical information is provided by compound propositions and is introduced by the subject starting from elementary data, expressed by atomic formulas. The elementary data reflect the elementary facts that take place inside a fixed empirical world that is assumed as reference for constructing knowledge. The difference in the logical and epistemic role between atomic formulas and compound formulas is so relevant, that we introduce for it specific meta-symbols: thus, latin capital letters $A, B, C, ...$ will indicate arbitrary formulas, whereas latin capital letters with the $+$ superscript $A^+, B^+, C^+, ...$ will indicate arbitrary *atomic* formulas. We recall that logical connectives are not atomic formulas. A formula is *atomic* if the outermost symbol is not a logical connective. A formula is an *atom* if it has not proper Church sub-term. In particular, the $o$-typed logical connectives, i.e. the logical constants $\bot_o$, $\top_o$ are *atoms* but not atomic formulas. The first expresses the judgement which is always acceptable (for which no criticism is possible) so that the sequent $\vdash \top$ is always provable; the latter expresses the judgement which is always refutable (for which no corroboration is possible) so that the sequent $\bot \vdash$ is always provable. (see also [7], Sec. 2 and 3). A further technical remark is that in this paper we will always use the sequent version of the considered logical systems. Other foundational assumptions are the following. In the demonstration (argumentation) produced by the epistemic subject, *judgments are not received from the external world*: they must be *explicitly constructed* alongside the demonstration itself. Only *elementary data* are necessarily provided by the external world. Therefore, logical

information has to be always reconstructed by the epistemic subject and never merely acquired; differently, elementary data are merely acquired. Simmetrically, judgements, alongside a demonstration, *cannot be eliminated without an explicit logical motivation:* this must have the form of another explicit judgment of the espistemic subject, that is by applying a *logical* rule. The eliminating rules, in a context where the cut rule in strictly bounded and anyway eliminable (as it happens in the main logical calculi) are the so called *Comprehension rules* (*Comp rules*) in [7], i.e $\forall - L$ and $\exists - R$. An exhaustive discussion of the elimination power of *Comp Rules* is given in [7] Section 3. Thus, the full higher order system $\mathbf{Z}_\omega$ for $ECL$ Logic that we are going to define, is a subsystem of $\mathbf{LK}_{\boldsymbol{\omega}}$ where:

- only atomic formulas occur in the logical axioms;

- weakening rules are admitted only to introduce atomic formulas, i.e. *weakening cannot introduce logical information*;

- cut rule is admitted only with atomic cut formulas, i.e. *cut cannot eliminate logical information*;

- each logical rule is always thought as occurring in some proof P, and have constraints on its auxiliary formulas that take into account their introduction in the whole proof-segment above the premise(s) of the rule occurrence in P. That is, they are *global* and not *local* constraints. The underlying idea is that the construction of a judgment introducing new logical information must be based only on previously produced logical information which has been itself acceptably constructed.

We note that the restrictions on weakening and cut rules immediately follow from the epistemic assumptions mentioned above. The constraints on the logical rules will be presented and discussed in detail in the next Sections. Moreover, the constructivity properties we assign to $\mathbf{Z}_\omega$ are also justified through the comparison with those logics that in the literature are already considered as constructive (Intuitionistic Logic, Paraconsistent Logic, Uniform Logic).

The constructivism of Intuitionistic Logic **LJ** is well known. We recall that, from a merely technical point of view, it is characterized by the refutation of the *excluded middle* (or *tertium non datur*) principle, syntactically expressed by the schema $A \vee \neg A$ and , collaterally, also by the refutation of the *left double negation principle*, expressed by the schema $\neg\neg A \supset A$. The standard sequent version **LJ** is characterized by the following condition: *each sequent in a proof has the empty set or a singleton as succedent.* However, this is not strictly necessary: in the sequent version of Maehara [9] p. 52, such condition is replaced by local contraints on three logical rules, among which, centrally, the negation rule *on the right* $\neg - R$. Even if intuitionism has many peculiar constructive features, that cannot be reduced to the mere refutation of *tertium non datur*, it is a fact that **LJ** *plus* $\vdash A \vee \neg A \equiv$ classical **LK**.

The constructivism of Paraconsistent Logic has been only recently defined in a formal way, and the set of paraconsistent logics to which the notion can be applied must be clearly delimited. We consider here the system **CI** examined in [6]. Paraconsistent Logic arises form the refutation of the classical (syntactic) principle *ex contraditione quodlibet* that can be expressed by the **LK**-provable

sequent schema $A \wedge \neg A \vdash$ which is classically equivalent to the *non contradiction principle* $\vdash \neg(A \wedge \neg A)$. **CI** and the various systems in the $C$-system family do not prove contradictions, but can support axioms of the form $B \wedge \neg B$ without trivializing, i.e. without proving the empty sequent " $\vdash$ ". Even if the refutation of *ex contradictione quodlibet* could seem today a position which is naturally constructive, in [6] a formal definition of *constructive paraconsistent logic* is given, also employing the introduction of antisimmetry connections between $C$-system paraconsistency and intuitionistic logic.

The Uniform Logic introduced in [8], that we indicate here with $\mathbf{LU}_{\boldsymbol{\omega}}$, is a sub-system of Higher Order Intuitionistic Logic $\mathbf{LJ}_{\boldsymbol{\omega}}$ where a particular constraint is added for the application of logical rules inside a proof-tree. Essentially, if a logical rule occurrence $\mathcal{R}$ *has the auxiliary formula(s) in the premise succedent(s), then it (they) must be the principal formula(s) of the logical rule occurrence(s) immediately above* $\mathcal{R}$ *in the branch.* Uniform Logic specifies the intuitionism constructivity in the direction of computation: indeed, it expresses *abstract logic programming languages.* Moreover, as to the main discussion of this paper, it must be remarked that the rule-constraint mentioned above is a first example of *global*, i.e. referred to the context of the rule-occurrence in the proof, and not *local* constraint.

As detailed later, $\mathbf{Z}_{\omega}$ shares relevant specific features with Intuitionistic Logic, since it **does not** prove both $\aleph_0$ instances of *excluded middle principle* $A \vee \neg A$ and $\aleph_0$ instances of *left double negation principle* $\neg\neg A \supset A$. It could be called also a pseudo-intuitionistic system (such notion is formally introduced in [6]).

$\mathbf{Z}_{\omega}$ shares relevant specific features with Paraconsistent Logic, since it **does not** prove $\aleph_0$ instances of *non contradiction principle* $\neg(A \wedge \neg A)$ and can be extended by $\aleph_0$ contradictions without trivializing. It could be called also a paraconsistent system.

$\mathbf{Z}_{\omega}$ shares some properties with Uniform Logic. In fact it **does not** hold, (in general, for $\mathbf{Z}_{\omega}$ proof trees), the possibility of any permutation of the order of propositional rules in a proof branch without changing the end-sequent, even in those cases where classical logic would admit such a permutation.

We also point out that $ECL$ has a strong expression power. Indeed, besides the $\mathbf{Z}_{\omega}$-proof capabilities allowed by the higher order, the following properties also hold for **Z1** and **ZP**:

$\mathbf{Z}_{\omega}$ proves $\aleph_0$ arbitratrily complex instances of *excluded middle principle* $A \vee \neg A$ that $\mathbf{LJ}_{\boldsymbol{\omega}}$ does not prove;

$\mathbf{Z}_{\omega}$ proves $\aleph_0$ arbitratrily complex instances of *non contradiction principle* $\neg(A \wedge \neg A)$ that $\mathbf{CI}_{\omega}$ does not prove.

## 3 The Systems $\mathbf{Z}_{\boldsymbol{\omega}}$, Z1, ZP for Explicit Constructive Logic (ECL)

In the sequel we briefly call *bottom* and *top* the formulas $\bot$ and $\top$, recalling that they *are not* atomic formulas. Moreover, if $\mathbf{Z}_{\omega}$ proves the sequent $\vdash B$ we say that

$B$ is a *theorem* of $\mathbf{Z}_\omega$, if $\mathbf{Z}_\omega$ proves the sequent $E \vdash$ we say that $E$ is a *refuted* by $\mathbf{Z}_\omega$. The language of $\mathbf{Z}_\omega$ is that of $\mathbf{LK_\omega}$ (see [7] Sec. 2) *with the exclusion of all the equality symbol* $=_{o\to(o\to o)}$ that would express the *equality relation* between $o$-typed formulas, since such relation is not considered by $ECL$. For the notions concerning general proof theory, the analysis of proofs as well as the notions of *ancestor*, *descendant*, *auxiliary formula*, *principal formula* and so on, we refer to [7] Section 2 and 6.

The sequent system $\mathbf{Z}_\omega$ is the following:

(In a sequent $\Omega$, $\Delta$, $\Gamma$, $\Pi$, $\Theta$, ... will be used as meta-expressions for finite and possibly empty sets of $o$-typed formulas, $A, B, C, D$, ...for arbitrary isolated formulas in a sequent, $A^+, B^+, C^+, D^+$, ... for arbitrary *atomic* isolated formulas. The writing $\Omega, \Delta$ denotes $\Omega \cup \Delta$ )

**Axioms**

*Logical axioms* $A^+ \vdash A^+$ with the following constraint:

*if the atomic $A^+$ is a $\beta$-redex, possible repeated application of the $\lambda$-rule does not produce any $\beta$-contractum $F$ which is a descendant of $A^+$ and is a non-atomic formula.*

*Top axiom* $\vdash \top$

*Bottom axiom* $\bot \vdash$

**Rules**

**Strong Logical Rules***:*

*Propositional rules:*

$$\frac{A, B, \Gamma \vdash \Delta}{A \wedge B, \Gamma \vdash \Delta} \wedge -L \qquad \frac{\Gamma \vdash \Delta, A \qquad \Lambda \vdash X, B}{\Gamma, \Lambda \vdash \Delta, X, A \wedge B} \wedge -R$$

$$\frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B} \vee -R \qquad \frac{A, \Gamma \vdash \Delta \qquad B, \Lambda \vdash X}{A \vee B, \Gamma, \Lambda \vdash \Delta, X} \vee -L$$

$$\frac{A, \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \supset B} \supset -R \qquad \frac{\Gamma \vdash \Delta, A \qquad B, \Lambda \vdash X}{A \supset B, \Gamma, \Lambda \vdash \Delta, X} \supset -L$$

$$\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \neg - L \qquad \frac{A, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg A} \neg - R$$

It can be noted that the forms of $\wedge - L$ and $\vee - R$ are not the standard one. This is a specific requirement of the Explicit Constructive Logic that will be discussed in the next Sections.

*Quantifier rules*:

$$\frac{[t_\alpha/x_\alpha]\, A, \Gamma \vdash \Delta}{\forall x_\alpha A, \Gamma \vdash \Delta} \forall - L \qquad \frac{\Gamma \vdash \Delta, [b_\alpha/x_\alpha]\, A}{\Gamma \vdash \Delta, \forall x_\alpha A} \forall - R$$

$$\frac{[b_\alpha/x_\alpha]\, A, \Gamma \vdash \Delta}{\exists x_\alpha A, \Gamma \vdash \Delta} \exists - L \qquad \frac{\Gamma \vdash \Delta, [t_\alpha/x_\alpha]\, A}{\Gamma \vdash \Delta, \exists x_\alpha A} \exists - R$$

where: in $\forall - L$, $\exists - R$, $t_\alpha$ is an arbitrary term and in the corresponding $\forall x_\alpha A$, $\exists x_\alpha A$, $t_\alpha$ may still occur, that is $t_\alpha$may be not fully quantified in $\forall x_\alpha A$, $\exists x_\alpha A$; on the other hand, in $\forall - R$, $\exists - L$, the free variable $b_\alpha$ occurring in $[b_\alpha/x_\alpha]\, A$ is uniformly replaced in $\forall x_\alpha A$, $\exists x_\alpha A$ by the bound variable $x_\alpha$ having the same index, and $b_\alpha$ does not occur in $\Gamma$, $\Delta$. $b_\alpha$ is the *proper variable* or *eigenvariable* of the rule.

*$\lambda$ -rule*: $$\frac{\Gamma\prime \vdash \Delta\prime}{\Gamma \vdash \Delta} \lambda$$

where the sets $\Gamma$ and $\Gamma\prime$ and the sets $\Delta$ and $\Delta\prime$ differ only in that zero or 1 formula in them is replaced by some formula to which it is $\beta$-reducible. Note that the rule is defined so that the $\beta$-reduction may work either upwards or downwards. We observe that differently from the more usual version (see e.g. [8], [4]) it is imposed here that $\lambda$-rule works on 1 auxiliary formula only, and not simultaneously on any arbitrary set of auxiliary formulas. This option is more coherent with the $ECL$ perspective and with the inclusion of the rule among the strong logical rules, where the control of all the origins of the auxiliary formulas of the rule -occurrence in the above standing proof-segment is required. Finally, the inclusion of $\lambda$-rule among strong logical rules is due to the possibility that a rule occurrence $\mathcal{R}$ in a proof P of $\mathbf{Z}_\omega$ may $\beta-$reduce a $\beta$-redex to a non-atomic formula $D$ arbitrarily complex, with a main logical connective (the outermost symbol of $D$) that can be seen as introduced by $\mathcal{R}$.

**Strong logical rules must fulfil the following constraints**:

*- If $\mathcal{R}$ is a 1 premise strong logical rule then each occurrence of $\mathcal{R}$ in a $\mathbf{Z}_\omega$-proof* P *is such that at least 1 auxiliary formula has at least 1 uppermost ancestor introduced by an axiom.*

*- If $\mathcal{R}$ is a 2 premise strong logical rule, having i.e. two auxiliary formulas, then each occurrence of $\mathcal{R}$ in a $\mathbf{Z}_\omega$-proof* P *is such that each auxiliary formula has at least 1 uppermost ancestor introduced by an axiom.*

*- If $\mathcal{R}$ is a $\lambda$-rule then each $\mathcal{R}$- occurrence in a proof* P *in $\mathbf{Z}_\omega$ is such that its auxiliary formula has at least 1 uppermost ancestor introduced by an axiom.*

**Weak Logical Rules**

*bottom rule* $\dfrac{\bot \ \vdash}{\bot \ \vdash A}$

where $A$ is an arbitrary formula without sub-formulas of the form $\forall x_o(x_o)$, $\exists x_o(x_o)$.

*top rule*: $\dfrac{\vdash \ \top}{B \ \vdash \ \top}$

where $B$ is san arbitrary formula without sub-formulas of the form $\forall x_o(x_o)$, $\exists x_o(x_o)$.

**Structural Rules**

*Wakening rules*: $\dfrac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta,\ A^+}W1-R \qquad \dfrac{\Gamma \vdash \Delta}{A^+, \Gamma \vdash \Delta}W1-L$

$\dfrac{\vdash}{\vdash \ F}W2-R \qquad \dfrac{\vdash}{F \vdash}W2-L$

*Cut Rules* $\dfrac{\Gamma \vdash \Delta, B^+ \qquad B^+, \Gamma \vdash \Delta}{\Gamma \vdash \Delta}Cut1 \qquad \dfrac{\vdash F \qquad F \vdash}{\vdash}Cut2$

**Structural rules must fulfil the following constraints**:

*- Each $W1$ -principal formula is atomic, and in any $W1$-rule at least 1 set of the contex is non-empty.*

*- Each $W2$- principal formula may be arbitrary.*

*- $Cut1$-formula $B^+$ is atomic, such that if it is a $\beta$-redex, possible repeated application of the $\lambda$-rule does not produce any $\beta$-contractum $G$ which is a descendant of $B^+$ and is a non-atomic formula. Moreover, at leaast 1 context set is non-empty.*

*- $Cut2-$formula $F$ may be arbitrary.*

**3.1 The First Order System Z1 of ECL**

**The language of Z1 is defined as follows:**

The *well formed expressions* of **Z1** are Church-terms with the following constraints:

- variables are only of type $i$;

- $\lambda$-abstractions are only over variables of type $i$ and on formulas of type $o$, i.e. have only the form $\lambda x_i A_o$ with type $i \to o$;

- quantifiers occur only with type $(i \to o) \to o$ , i.e. with the forms $\exists_{(i\to o)\to o}$ $\forall_{(i\to o)\to o}$ ;

- if $\tau$ is a type occurrence in any **Z1**-espression, no occurrences of the type $o$ in $\tau$ precede any occurrence of the type $i$ in $\tau$;

- non-logical constants of **Z1** are only of a type $\tau$ such that:

either in $\tau$ the type $o$ does not occur, or the type $o$ has at most one occurrence as *tail* of $\tau$; $\tau$ has a *condensed writing* of the form: $i \to i \to i \to ... \to u$, where $u$ is a primitive type.

**Deduction apparatus of Z1**:

It is identical to $\mathbf{Z}_\omega$ deduction apparatus, with the constraint that rules are restricted to sequents of **Z1**-formulas, so that only $i$-typed terms can be quantified. The $\lambda$-rule could be useful but is not strictly necessary for the expressivity of the system. Thus, we denote **Z1**$\lambda$ the version including $\lambda$-rule, **Z1** the $\lambda$-rule free version.

**3.2 The Propositional Calculus ZP of ECL**

The language of **ZP** is obtained from **Z1** with the following restrictions:

-quantifiers, variables, $\lambda$-abstraction expressions, do not occur in the language;

- the only non logical constants are $o$-typed atoms, also called *propositional letters*.

The deduction apparatus is obtained from that of **Z1** by deleting quantifier rules.

**3.3 Immediate Properties and Definitions of $\mathbf{Z}_\omega$, Z1, ZP**

In this Section we will focus mainly on the most powerful and expressive system, which is $\mathbf{Z}_\omega$; however, many definitions and properties naturally extend to **Z1** and **ZP**.

*Remark 1.* Weak logical rules are also imposed by the necessity to give a suitable proof power to $ECL$ systems. For example, in $\mathbf{LK_\omega}$, the axiom $\bot \vdash$ makes superfluous a rule of the form

$$\frac{\bot \ \vdash}{\bot, U \ \vdash \ V}$$

$U, V$ arbitrary sets, due to constraint free weakening rules of $\mathbf{LK_\omega}$. In $ECL$ systems, such approach would not be conceivable.

**Definition 1.** Let P be a proof tree in $\mathbf{Z}_\omega$. Then we say that *the occurrence of the formula* **A** *in P is strongly introduced* if it is integral descendant of an axiom formula or it is integral descendant of the principal formula of a strong logical rule. We say that *the occurrence of the formula* **A** *in P is weakly introduced* if

it is integral descendant of the principal formula of a weak logical rule or of a weakening formula.

For the definition of *integral descendant of a formula occurrence in a proof* see [7] Def. 6.5.ii p. 750. Intuitively the *integral descendant* $B$ of the formula occurence $F$ in a proof branch is such that $B$ and $F$ are occurrences of the same formula, i.e. $B \equiv F$, connected by a proof-path where $F$ (or $B$) is *never* an auxiliary formula of any rule.

**Definition 2.** Among the strong logical rules we call *major (strong) logical rules* those where all auxiliary formulas must be strongly introduced, *minor (strong) logical rules* those where at least 1 auxiliary formula may be weakly introduced.

**Corollary 1.** $\{\vee - L, \wedge - R, \supset - L, \neg - R, \neg - L, \forall - L, \forall - R, \exists - L, \exists - R, \lambda - rule\}$ *is the set of major logical rules in* $\mathbf{Z}_\omega$, $\{\vee - R, \wedge - L, \supset - R\}$ *is the set of minor logical rules in* $\mathbf{Z}_\omega$.

**Definition 3.** Let P be a proof tree in $\mathbf{Z}_\omega$. Then we say that *a sequent $S$ occurring in $P$ is strongly proven in $P$* if each formula of S is strongly introduced in P. We say that *$S$ is weakly proven in $P$* otherwise.

Caveat: the same $S$ may be strongly proven in a proof P and, simultaneously, weakly proven in a different proof Q. In $ECL$ logic the *proof-context* of a sequent or of a formula has a substantial role, and this is coherent with the fact that the constraints on the application of a logical rule in $ECL$ are always *global* and not *local*. We also mention these two evident facts: $\mathbf{Z}_\omega$ is consistent, since it is a $\mathbf{LK_\omega}$ subsystem; moreover, if P is a $\mathbf{Z}_\omega$-proof, it cannot have an end-sequent where all formulas are weakly introduced.

## 4 General Epistemological and Logical Justifications for Axioms and Rules of $\mathbf{Z_\omega}$, Z1, ZP, and Further Properties of Connectives and Rules in ECL

### 4.1 The Formulas Bottom and Top

$o$-typed logical constants $\bot$ and $\top$ standardly occur in the Church presentation of Type Theories ([7], [8], [4]). In the Explicit Constructive Logic $ECL$ $\top$ *expresses the judgment that the subject thinks as always acceptable*, beyond any possible confutation, and $\bot$ *expresses the judgment that the subject thinks as always refutable*, beyond any possible corroboration. That's why, *in general*, we can state: if $B$ is a $\mathbf{Z}_\omega$-theorem different from $\top$, then it *is not* provable in $\mathbf{Z}_\omega$ the sentence $(\top \supset B) \wedge (B \supset \top)$ (or $B \longleftrightarrow \top$), and if $E$ is a $\mathbf{Z}_\omega$-refuted different from $\bot$, then it *is not* provable in $\mathbf{Z}_\omega$ the sentence $(\bot \supset E) \wedge (E \supset \bot)$ (or $E \longleftrightarrow \bot$). In particular, as to the conjunctions that would give the mentioned $\mathbf{Z}_\omega$-logical equivalences, it is not provable, *in general*, in the first case the conjunct $\top \supset B$ and in the second case the conjunct $E \supset \bot$ . Remarkable exceptions may exist. For example $\top \wedge \top$ is a theorem and $\top \supset \top \wedge \top$ is provable. However,

for each atomic $A^+$, $A^+ \vee \neg A^+$ is a theorem but $\top \supset A^+ \vee \neg A^+$ is not provable. For example, for each atomic $B^+$, $\bot \wedge B^+$ is a refuted and $\bot \wedge B^+ \supset \bot$ is provable. However, for each atomic $B^+$, $B^+ \wedge \neg B^+$ is a refuted but $B^+ \wedge \neg B^+ \supset \bot$ is not provable. These examples suggest that explicit constructivity includes *a criticism to classical implication*. Moreover, the usual systems of classical, intuitionistic and paraconsistent logics lack such fine separation capability: all of them make *top particles* equivalent to *theorems*, and *bottom particles* equivalent to *refuted sentences*.

**4.2 The Weak Logical Rules**

Let's comment on and justify the *weak logical rules*, i.e. the *top rule* and the *bottom rule*. They are *logical* since they *realize through an information transformation process the presumed logical content of the logical connectives top and bottom*. Note that, in the foundational perspective of $ECL$, without such rules, the presence in the system of the mentioned $o$-typed logical connectives would be not motivated and they should be excluded from the language. On the other side, they are the *only rules* through which *not explicitly constructed logical information can be introduced in the argumentative discourse*. Indeed, they represent a very constrained and regulated way to partially have that information introduction power of the *standard weakening rule*. This is also why they are called *weak*, and their principal formulas are qualified as *weakly introduced*. This causes inferential limitations. If $\bot\vdash C$ and $D \vdash \top$ are the conclusions of any *bottom rule* and *top rule* respectively, we cannot apply to them a $\supset -L$ rule and infer $C \supset D, \bot \vdash \top$ , since both the auxiliary formulas are weakly introduced. As a matter of fact $\bot \vdash \bot$ and $\top \vdash \top$ **are not** logical axioms, they are conclusions of weak logical rules, so that one of the two cedents ([1] p. 10) is always weakly introduced. Nevertheless, the contribution of weak logical rules to $ECL$ inference is substantial; *otherwise the information sources of $ECL$ proofs would be too poor*. In addition, their weakly introduced principal formula can anyway contribute to infer strongly introduced formulas: from $D \vdash \top$, we can infer $\vdash D \supset \top$ that is a strongly introduced formula, as the principal formula of a $\supset -R$ must be. We shall prove in Section 6 that weak logical rules must only occur as the *initial rule* in a branch: in addition, *their principal formula has not auxiliary formula, so that it has no predecessor*. This justifies the requirement that formulas $\forall x_o(x_o)$, $\exists y_o(y_o)$ do not occur as sub-formulas of the principal formula of any weak logical rule. In a $\mathbf{Z}_\omega-$proof, $\forall x_o(x_o)$, $\exists y_o(y_o)$ mark elimination judgments about previously produced logical information. Then, their occurrence is *meaningless* in an initial (uppermost) formula of the proof tree.

**4.3 The Exclusion of Equational Logic $\mathbf{EQ}_\omega$ from $\mathbf{Z}_\omega$**

In $\mathbf{LK}_\omega$ and $\mathbf{LJ}_\omega$, Equational Higher Order Logic $\mathbf{EQ}_\omega$ can be fully included in the system and works together with the logical part. For the $\mathbf{EQ}_\omega$-axioms see [7], Sec. 2.4. What must be clearly emphasized is that in the higher order context and inside the Church formalism $\mathbf{EQ}_\omega$ becomes extremely powerful and mixes itself with the logical connectives' deduction action. This is clear if we consider that each theorem $B$ of $\mathbf{LK}_\omega$ and $\mathbf{LJ}_\omega$ can be provably constricted to the atomic formula $B =_o \top$, and that the equality predicate on type $o$ can

be provably identified with the logical equivalence between propositions, i.e., in the $ECL$ perspective, between *judgments*. From the standpoint of $ECL$, aiming to obtain a very fine characterization of logical connectives through a constructive approach, this is not admissible. *Equational Logic is explicitly excluded from* $ECL$. More generally, we point out that, in the context of explicit constructivism, is also *inadmissible* the confusion between the *equality relation* $=_o$ and the *logical implication* $\supset$ or *double-implication* $\longleftrightarrow$. We think that the equality relation can be only defined *a priori* in a *platonic universe*. In a knowledge representation setting, we are not able to imagine an epistemic subject that, inside an *empirical world* and through an *effective process* can establish that two *objects* are *equal*, with the same meaning owned by the statement "these two Euclidean triangles are equal" affirmed inside a platonic universe. On the contrary, if we consider the epistemic subject that formulates judgments on the world, the implication or double-implication relation must be established by a construction which increases *the complexity of the judgment through the logical rules*, starting from *elementary data*. Observe that in $\mathbf{Z}_\omega$ *theorems that are atomic formulas do not exist* (with the minor exception of possible $\beta$-redexes, which are a bit artificial form both for possible judgments and for possible data), coherently with the principle that *elementary data cannot be judgments.* Differently, Equational Logic $\mathbf{EQ}_\omega$ produces a multitude of atomic theorems, most of them having a substantial and non-artificial information content, such as, for example, the assertion $B =_o F \wedge \neg F$, establishing that the arbitrarily complex formula $B$ is *logically* equivalent to a contradiction.

**4.4 The Strong Logical Rules**

The originality of the proposed logic is mainly expressed in these rules. *In fact, the constraints involving these rules are not local, i.e. they do not operate on the occurrence of the specific rule* $\mathcal{R}$ *in a proof, but are conditions concerning the whole proof P in which the rule occurs*: to apply $\mathcal{R}$ it is necessary, in general, to examine all the introductions of the uppermost ancestors of the auxiliary formulas of $\mathcal{R}$ in the proof segment of P standing above the $\mathcal{R}$ -premise(s). We deem that relevant innovations in Logic could be obtained by changing the praxis of imposing only local constraints on a single rule. This lightly changes the usual notion of performing a proof, and could produce innovative results and situations, perhaps more than the introduction of new connectives and new rules. By recalling the Definitions of Section 3, the distinction between *strongly introduced formula in a proof* P and *weakly introduced formula in a proof* P should result natural. *Axioms* are the *choices of the epistemic subject*, on which it *decides to found* its reasoning. Weakening and weak logical rules are auxiliary tools, useful to introduce information. On the other hand, a proof without at least one axiom occurrence cannot exist, while infinite proofs may exist without weakening or weak rule occurrences. It must be emphasized that *the* ***minor*** *strong logical rules* $\vee - R$ , $\wedge - L$ *are differently presented w.r.t. the usual standard presentation, that is both the auxiliary formulas must occur as isolated in the premise.* This reflects two crucial requirements. First, if this would not be the case, the introduction of one of the maximal disjunct (conjunct) of the principal formula

would be *an arbitrary hidden weakening.* Furthermore, since we need to constraint all the uppermost ancestors of *both* the auxiliary formulas in the whole proof-segment above, *both* the formulas must explicitly occur in the premise.

**4.5 The Structures of Weakening and Cut in $\mathbf{Z}_\omega$**

The constraints on the weakening rule, i.e. the imposition on principal formulas to be *atomic*, should be quite clear. It is at the basis of Explicit Constructive Logic: *only elementary data can be used without having been constructed.* A first non trivial fact follows immediately, thus clarifying the differences from $\mathbf{LK_\omega}$ and $\mathbf{LJ_\omega}$: if $X \vdash Y$ is $\mathbf{Z}_\omega$-provable, its over-sequents $U \vdash V$ with $X \subset U$ and $Y \subset V$, are, in general, not $\mathbf{Z}_\omega$-provable. A motivation of the presence of *two rules* $W1$ and $W2$ is due: while W1 is immediately understandable, less obvious is the necessity of $W2$, i.e. of *arbitrary* weakenings on the *empty* sequent. The reason arises from the fact that $\mathbf{Z}_\omega$ is supposed to be possibly extended to various (countably many) axiomatized theories $\mathbf{T_j}$'s. Some of them are expected to be absolutely inconsistent, and we usually identify this situation with the provability of the empty sequent. But this does not work in the $ECL$-framework, since from the empty sequent, $\mathbf{Z}_\omega$-rules *minus* $W2$, in general, cannot derive *all* formulas of the language as theorems. Therefore, $W2$ is added. The motivations of $Cut2$ are also linked to the ones of $W2$: if a $\mathbf{Z}_\omega$-based theory $\mathbf{T}$ has any *non atomic theorem* and any *non atomic refuted* that are identical, i.e. $\mathbf{T}$ proves both $B \vdash$ and $\vdash B$, $\mathbf{Z}_\omega$-rules *minus* $Cut2$, in general, cannot derive the empty sequent. Thus, $Cut2$ is added. We consider now the substantial restrictions that are imposed to the cut rule. They reflect the requirements already considered about the introduction and the elimination of logical information by a reasoning epistemic subject: *logical information is not eliminated by a material deletion, but only by a logical transformation.* The selection of logical information is a thinking act, requires elaboration, *then* it must involve logical rules. Thus, cut at most deletes atomic formulas, i.e. elementary data. On the other hand, the peculiar structure of the cut rule *cannot become a clandestine arbitrary weakening,* and this is obtained by imposing the same context for the two premises.

## 5 Possible New Effective Applicability of ECL-based Higher Order Logic

It is well known that, in full Higher Order Logic, the unbounded quantification on $o$-typed formulas, or on formulas of arbitrary types where $o$-typed formulas occur as Church-subterms, makes it extremely difficult to establish any useful link between the formulas occurring in the root of a *cut-free* proof-tree P and the formulas occurring in the overstanding sequents alongside the branches of P.

To find how to exploit the information and proof power of higher order quantification, without chaotic or collapsing phenomena, is a main goal of our constructive view of higher order logic. The envisaged road is the following:

**to suitably and lightly normalize** *higher order quantification by a set of constraints allowing a* **Normal Form Theorem** *for the proofs of an adequately expressive sub-system of* $\mathbf{LK_\omega}$.

A Normal Form Theorem (NFT) for a system **V** (see [1] or for example [5] where a NFT for Arithmetic is proposed) is characterized by:

*i*) an effective description of the transformation of a **V**-proof into a **V**-proof tree with the same root, partitioned in blocks such that each block includes only homogeneous rules or axioms;

*ii*) a set of effective procedures such that given a root sequent $L$, the following reasonable estimates about the features of a possible proof Q of $L$ in **V** can be produced:

an estimate of the possible **V**-rule instance set occurring in Q

an estimate of the possible **V**-axiom instance set occurring in Q

an estimate of the length and the width of Q

an estimate of the formula (or term) set occurring in Q.

It is quite clear that an efficient Normal Form Theorem for an adequate and non redundant $\mathbf{LK}_{\boldsymbol{\omega}}$ sub-system can give a new basis for higher order automated deduction and knowledge representation. We believe that:

*the very strong and peculiar ECL constructivity of the system* $\mathbf{Z}_\omega$ *allows us to state those* ***normative constraints*** *on its quantification power that can generate an adequately expressive subsystem* $\mathbf{Z}^*_\omega$ *that admits a* **Normal Form Theorem.** We present now a first hint of the system $\mathbf{Z}^*_\omega$ that should exemplify how further constructive conditions imposed on the quantification judgements of the epistemic subject may lead to a Normal Form Theorem. As a basic feature of $\mathbf{Z}^*_\omega$ we introduce the notion of $\mathbf{Z}_\omega-$*proof with the witness property.* We previously recall that the language includes, for each type $\gamma$, $\aleph_0$ free variables (atoms) $\left\{b^j_\gamma\right\}$, univocally individuated by their index $j \in \mathbf{N}$, and $\aleph_0$ bound variables (atoms) $\left\{y^i_\gamma\right\}$, univocally individuated by their index $i \in \mathbf{N}$ (see [7] Section 2.2). In the following definition, the most relevant point is b):

**Definition 4.** A $\mathbf{Z}_\omega-$*proof* P *has the witness property* if the following conditions hold:

a) Each time $o$-typed formulas/terms $B^j_o$, $j = 1, ..., m$, $m \geq 1$ occur in P as the auxiliary formula $E$ of a *Comp rule,* or are included in it as sub-terms, then the bound variable $z_o$ in the corresponding principal formula $\mathcal{Q}z_o(z_o)$, $\mathcal{Q} \in \{\forall, \exists\}$ has as index the *gödel number* ([7] Section 2.5) of their sequence.

b) Any auxiliary formula of a *Comp rule* in P may have $o$-typed sub-terms only if the quantification is over the type $o$.

c) In P isolated formulas of the form $a_o$, $b_o$, i.e. *free variables* of type $o$ which occur as isolated formulas in a sequent, *cannot* be auxiliary formulas of quantifier rules.

**Proposition 1.** *The property "to be a proof with the witness property of the system* $\mathbf{Z}_\omega$*" is a recursive relation, it can be expressed by a recursive predicate inside Primitive Recursive Arithmetic* **PRA** *and is a decidable property.*

*Proof.* The Definition above describes exactly effective conditions to get a $\mathbf{Z}_\omega - proof$ with the witness property.

Only as one of the possible example of the effective control on proofs allowed by the witness property we mention these results:

**Proposition 2.** *a) Let $P$ be a proof in $\mathbf{Z}_\omega$ with the witness property. If in $P$ a formula of the form $\mathcal{Q}z_o(z_o)$, $\mathcal{Q} \in \{\forall, \exists\}$ is introduced, then in the root at least one formula of the form $\mathcal{H}z_o(z_o)$, $\mathcal{H} \in \{\forall, \exists\}$ occurs.*

*b) Let $P$ be a proof in $\mathbf{Z}_\omega$ with the witness property. Let us suppose that quantified formulas over the type $o$ do not occur in the P-root. Then each atom of type $o$ occurring in $P$ occurs also in the root of $P$.*

**Definition 5.** The *weakly normalized system* $\mathbf{Z}^*_\omega$ is so defined: *a)* Axioms, propositional rules, structural rules of $\mathbf{Z}^*_\omega$ are the same as that of $\mathbf{Z}_\omega$ and with the same constraints; *b)* Quantifier rules of $\mathbf{Z}^*_\omega$ have the same constraints as the ones of $\mathbf{Z}_\omega$, and moreover are applied in a proof-tree in a way such that the resulting proof has the witness property, i.e. it fulfils all the conditions stated in the previous Definition of the witness property. *c)* The $\lambda$-rule is omitted from the deduction aparatus.

Therefore, in $\mathbf{Z}^*_\omega$ all proofs have the witness property. Thus, even if the higher order quantification power is not dramatically bounded at all, some interesting links between the formulas occurring in the root and the rule instances and formulas occurring in the above proof-segments are at disposal. We will see such links at work in particular in Section 7.

## 6 Elementary Proof-theory and Expressivity of $\mathbf{Z}_\omega$

**Proposition 3.** $\mathbf{Z}_\omega$ *admits cut-elimination.*

*Proof.* The proof is straightforward. Indeed, $Cut1$-formulas can be only atomic. Then they can be introduced in any $\mathbf{Z}_\omega-$proof only by weakenings or by logical axioms. This makes the proof-reductions to get cut-elimination very easy. As to $Cut2$ the set of $Cut2$-occurrences in the $\mathbf{Z}_\omega-$proofs is empty, due to the consistency of $\mathbf{Z}_\omega$.

In the sequel we assume to work only with cut-free $\mathbf{Z}_{\boldsymbol{\omega}}-$proofs. Moreover, by coherence with $ECL$ setting, we will consider only the equality free versions of type theories $\mathbf{LK}_{\boldsymbol{\omega}}$ $\mathbf{LJ}_{\boldsymbol{\omega}}$, that so have the full cut elimination property. The next results could seem to have obvious proofs. This would be a misunderstanding, since in $ECL$ the form of the logical rules is essentially standard, but their application conditions are not standard at all. For example, if $F$ *is not* atomic, then $F \vdash F$ *is not* a $\mathbf{Z}_{\boldsymbol{\omega}}-$logical axioms, and, in general, *nobody can easily assert or deny* that it must be a $\mathbf{Z}_{\boldsymbol{\omega}}-$theorem.

**Proposition 4.** $\mathbf{Z}_\omega$ *proves* $\aleph_0$ *instances both of the excluded middle principle* $B \vee \neg B$ *and of the left double negation principle* $\neg\neg B \supset B$ *that Intuitionistic Higher Order Logic* $\mathbf{LJ}_{\boldsymbol{\omega}}$ *does not prove.*

*Proof.* Let $G^+$, $H^+$ different non logical constants of type $o$. It is easy to see that $\mathbf{Z}_\omega$ proves the sequent $G^+ \wedge H^+ \vdash G^+ \wedge H^+$ with all the formulas *strongly introduced* in the proof. Then we produce in $\mathbf{Z}_\omega$ the following proof segment:

$$\dfrac{\dfrac{G^+ \wedge H^+ \vdash G^+ \wedge H^+}{\vdash G^+ \wedge H^+, \neg(G^+ \wedge H^+)}\ \neg - R}{\vdash (G^+ \wedge H^+) \vee \neg(G^+ \wedge H^+)}\ \vee - R$$

where all the rules have *strongly introduced* auxiliary formulas. Differently, by applying the cut-elimination property of $\mathbf{LJ_\omega}$ it is evident that $\mathbf{LJ_\omega}$ cannot prove the same end-sequent without breaking the local constraints of each $\mathbf{LJ_\omega}-$rule, imposing at most one formula in each succedent. Analogous considerations hold for the sequent $\vdash \neg\neg H^+ \supset H^+$. Moreover, the thesis can be easily extends to arbitrarily complex instances of the mentioned principles.

**Proposition 5.** $\mathbf{Z}_\omega$ *proves* $\aleph_0$ *instances of the non contradiction principle* $\neg(A \wedge \neg A)$ *that equality free paraconsistent type theory* $\mathbf{CI}_\omega$*, extending the system* $\mathbf{CI}$*, cannot prove.*

*Proof.* Let $B^+$ be an atomic formula. It is wellknown that $\mathbf{CI}_\omega$ does not prove $\vdash \neg(B^+ \wedge \neg B^+)$ (for the details on $\mathbf{CI}$ see [6]). The following proof can be produced in $\mathbf{Z}_\omega$ :

$$\dfrac{\dfrac{\dfrac{B^+ \vdash B^+}{B^+, \neg B^+ \vdash}\ \neg - L}{B^+ \wedge \neg B^+ \vdash}\ \wedge - R}{\vdash \neg(B^+ \wedge \neg B^+)} \neg - R$$

where all the auxiliary formulas of the rules are *strongly introduced.*

In the following, we simply state some lemmas without proofs:

**Lemma 1.** *The weak logical rules top rule and bottom rule can be only initial rules in a proof branch, i.e. they always occur as the uppermost rules of the branch.*

**Lemma 2.** *Let* $Q$ *be a proof in* $\mathbf{Z}_\omega$ *with root* $X \vdash Y, B$ *where* $B$ *is the integral descendant of the principal formulas of a set* $W \equiv \left\{ \dfrac{\bot\vdash}{\bot\vdash B}\ \mathcal{R}_j \right\}$ *of bottom rules in* $Q$*. Then we can replace each element of* $W$ *in* $Q$ *with the axiom* $\bot\vdash$ *obtaining a proof* $P$ *of* $X \vdash Y$ *in* $\mathbf{Z}_\omega$*.*

**Lemma 3.** *Analogous to the last Lemma, by replacing "bottom rule" with "top rule" and the formula bottom* $\bot$ *with the formula top* $\top$*.*

## 7 What ECL does not want to prove: $\mathbf{Z}^*_\omega$, Z1, ZP as paraconsistent and pseudo-intuitionistic systems

A very remarkable property of *ECL* is that *without imposing any local constraint* on negation rules of its systems, it nevertheless shows simultaneously a relevant and interesting *intuitionistic* and *paraconsistent* behaviour of its proofs. We will use now the *weakly normalized system* $\mathbf{Z}^*_\omega$ (Definition 5) that is a convenient setting of our epistemic consideration. We have the following results[3]:

[3] in the sequel the superscript $(.)^+$ in atomic formulas will be omitted

**Theorem 6.** *Consider the following instance of non contradiction principle expressed by the sequent S:* $\vdash \neg[(\bot \wedge(B \wedge C)) \wedge \neg(\bot \wedge(B \wedge C))]$ *where B and C are different o-typed non logical constants. Then S is not* $\mathbf{Z}^*_\omega-$*provable. Since S belongs also to propositional and first order languages the same holds for* **Z1** *and* **ZP**.

*Proof.* Suppose ad absurdum that $S$ is the root of a proof Q of $\mathbf{Z}^*_\omega$. If the root formula $B$ is also the integral descendant of the principal formula of a set of of bottom rule occcurences in Q, by Lemma 2 we delete such rules and get a proof Q' where the root formula $B$ is never introduced by a bottom rule occurrence: indeed, we exclude that the root of Q' could result the empty sequent, by the absolute consistency of $\mathbf{Z}^*_\omega$. Therefore, by properties of $\mathbf{Z}^*_\omega$, the end rule of Q' must be a $\neg - L$ rule having the sequent $M \equiv (\bot \wedge(B \wedge C)) \wedge \neg(\bot \wedge(B \wedge C)) \vdash$ as premise, and let $H$ be its root formula. In $M$ top formulas do not occur: then, by Proposition 2 (item $b$)), in Q' neither top formulas nor top rules can occur. Thus, neither $H$ nor $H$ sub-formulas can be integral descendant of principal formulas of top rule occurrences in Q'. $H$ must be so the conclusion of a $\wedge - L$ rule, with premise $K \equiv (\bot \wedge(B \wedge C)), \neg(\bot \wedge(B \wedge C)) \vdash$ . By analogous reasons $K$ is the conclusion of a $\neg - R$ rule with premise $N \equiv (\bot \wedge(B \wedge C)) \vdash (\bot \wedge(B \wedge C))$. Let $D$ be the succedent of $N$. We have to examine the possibility that $D$ has been introduced by bottom rules in Q'. We have two possible cases. The first one is that $D$ is exclusively the integral descendant of principal formulas of bottom rule occurrences in Q'. By Lemma 2 we delete them, and get $G \equiv \bot \wedge(B \wedge C) \vdash$ as the root of a $\mathbf{Z}^*_\omega$-proof W. Since top rules do not exist in W, the premise of $G$ in W must be $\bot, B \wedge C \vdash$, that necessarily has $\bot, B, C \vdash$ as premise: this is absurd, since being *both* $B$ and $C$ obviously weakly introduced, they cannot be auxiliary formulas of a $\wedge - L$ rule. The second case is that $D$ is not only the integral descendant of principal formulas of bottom rule occurrences so that, having deleted these by Lemma 2, we obtain a proof V of the sequent $L \equiv \bot \wedge(B \wedge C) \vdash \bot \wedge(B \wedge C)$ where no cedent is the integral descendant of principal formulas of weak logical rule occurrences. $L$ must be the conclusion of a strong logical rule. Indeed, suppose that the end rule of V is any $\wedge - R$ rule. Then its premises, that are *both necessarily* $\mathbf{Z}^*_\omega$-provable, are $J1 \equiv \bot \wedge(B \wedge C) \vdash \bot$ and $J2 \equiv \bot \wedge(B \wedge C) \vdash (B \wedge C)$: unfortunately, it is evident that the succedent of $J1$ cannot be strongly introduced, so that the $\wedge - R$ constraint would not be respected. We must so assume that the end rule of V is any $\wedge - L$ rule, with premise $J3 \equiv \bot, B \wedge C \vdash \bot \wedge(B \wedge C)$. $J3$ can be the conclusion either of a $\wedge - L$ or of a $\wedge - R$. In the first case the possible premise is $J4 \equiv \bot, B, C \vdash \bot \wedge(B \wedge C)$, in the second case the two possible premises, *both necessarily* $\mathbf{Z}^*_\omega$-provable, are $J5 \equiv \bot, B \wedge C \vdash \bot$ and $J6 \equiv \bot, B \wedge C \vdash B \wedge C$. However, the sucedent of $J5$ can be only weakly introduced and the same problems observed for $J1$ stop the examined possibility. Thus, we have to examine the possible provability of $J4$. The possibility that $J4$ is the conclusion of any $\wedge - R$ rule gives the same problems already noted for $J1$ and $J5$. We have then to suppose that the succedent of $J4$ has been introduced in V only by a set of bottom rule occurrences, and that the $\bot$ in the $J4$-antecedent is the integral descendant of the premise formulas

of such bottom rules. By Lemma 2, we delete such bottom rule occurrences and get a proof Z of the sequent $J7 \equiv \bot, B, C \vdash$ where by construction $B, C$ are the *only* integral ancestors of the auxiliary formulas of the $\wedge - L$ with conclusion $J3 \equiv \bot, B \wedge C \vdash \bot \wedge (B \wedge C)$. But this is absurd, since *both* $B$ and $C$ in $J7$ would be only introduced by weakenings in V, i.e. both are weakly introduced, and the $\wedge - L$ constraints would not be respected[4].

**Theorem 7.** *Consider the following instance of excluded middle principle expressed by the sequent* $M: \vdash [\top \vee \exists x_i B] \vee \neg[\top \vee \exists x_i B]$ *with* $B$ *o-typed non logical constant. Then* $M$ *is not* $\mathbf{Z}^*_\omega-$*provable. Since* $M$ *belongs also to the first order language the same holds for* **Z1**.

## 8 Conclusions

The introduction of the ECL logic and the first results regarding its relations with intuitionistic and paraconsistent logics are the main topics of the paper. We stressed the higher order setting since we believe that it is a relevant issue to find ways to make HOL a foundational setting for knowledge representation (constructive and with controlled use of instantiations). The future work will be devoted to further analyze the characteristics of ECL. As some considerations already present in the paper suggest the constructivity of ECL is not related to the "not rules" but to some peculiarities of the implication and of the possible proofs that can be accepted. These properties should be relevant for the use of ECL as the foundation logic of both Logic Programming and Theorem Proving.

## References

1. S.R. Buss (Ed.), *Handbook of Proof Theory*, Elsevier, Amsterdam, 1998.
2. W. A. Carnielli, M. E. Coniglio, J. Marcos, 'Logics of formal inconsistency' in *Handbook of Philosophical Logic*, D. Gabbay, F. Guenthner (Eds.), 2nd ed., volume 14, Kluwer, Dordrecht, 2005.
3. A. Church, 'A Formulation of Simple Theory of Types', Journal of Symbolic Logic, 5, 1940, 56-68.
4. M. De Marco, J. Lipton, 'Completeness and Cut-elimination in the Intuitionistic Theory of Types', Journal of Logic and Computation, 15, 2005, 821-854.
5. P. Forcheri, P. Gentilini , M.T. Molfino, 'Informational Logic in knowledge representation and automated deduction', *AI COMMUNICATIONS*, vol. 12,1999, 185-208.
6. P. Gentilini, 'Proof Theory and Mathematical Meaning of Paraconsistent C-Systems', Journal of Applied Logic, vol. 9, 3, 2011, 171-202.
7. P. Gentilini, M. Martelli, 'Abstract Deduction and Inferential Models for Type Theory', Information and Computation, vol. 208, Issue 7, July 2010, 737-77.
8. D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov, 'Uniform Proofs as a Foundation for Logic Programming'. Annals of Pure and Applied Logic 51, 1991, 125-157.
9. G. Takeuti, *Proof Theory*, North-Holland, Amsterdam, 1987.
10. A.S. Troelstra, D. van Dalen, *Constructivism in Mathematics,* Vol.1, Elsevier, 1988.

[4] It is not difficult to imagine that infinitely many sequents having a form similar to $S$ are not $\mathbf{Z}^*_\omega-$provable.

# CUD@ASP: Experimenting with GPUs in ASP solving*

Flavio Vella[1], Alessandro Dal Palù[2], Agostino Dovier[3],
Andrea Formisano[1], and Enrico Pontelli[4]

[1] Dip. di Matematica e Informatica, Univ. di Perugia
[2] Dip. di Matematica, Univ. di Parma
[3] Dip. di Matematica e Informatica, Univ. di Udine
[4] Dept. of Computer Science, NMSU

**Abstract.** This paper illustrates the design and implementation of a prototype ASP solver that is capable of exploiting the parallelism offered by general purpose graphical processing units (GPGPUs). The solver is based on a basic conflict-driven search algorithm. The core of the solving process develops on the CPU, while most of the activities, such as literal selection, unit propagation, and conflict-analysis, are delegated to the GPU. Moreover, a deep non-deterministic search, involving a very large number of threads, is also delegated to the GPU. The initial results confirm the feasibility of the approach and the potential offered by GPUs in the context of ASP computations.

## 1 Introduction

Answer Set Programming (ASP) [22, 20] has gained momentum in the logic programming and artificial intelligence communities as a paradigm of choice for a variety of applications. In comparison to other non-monotonic logics and knowledge representation frameworks, ASP is syntactically simpler and, at the same time, very expressive. The mathematical foundations of ASP have been extensively studied; in addition, there exist a large number of building block results about specifying and programming using ASP. ASP has offered novel and highly declarative solutions in several application areas, including intelligent agents, planning, software verification, complex systems diagnosis, semantic web services composition and monitoring, and phylogenetic inference.

An important push towards the popularity of ASP has come from the development of very efficient ASP solvers, such as CLASP and DLV. In particular, systems like CLASP and its variants have been shown to be competitive with the state of the art in several domains, including competitive performance in SAT solving competitions. In spite of the efforts in developing fast execution models for ASP, execution of large programs remains a challenging task, limiting the scope of applicability of ASP in certain domains (e.g., planning). In this work, we offer parallelism as a viable approach to enhance performance of ASP inference engines. In particular, we are interested in devising techniques that can take advantage of recent architectural developments in the field of *General Purpose Graphical Processing Units (GPGPUs)*. Modern GPUs are multi-core platforms, offering massive levels of parallelism; vendors like NVIDIA have started

* Research partially supported by GNCS-13 project.

supporting the use of GPUs for applications different from graphical operations, providing dedicated APIs and development environments. Languages and language extensions like *OpenCL* [16] and *CUDA* [29] support the development of general purpose applications on GPUs, beyond the limitations of graphical APIs. To the best of our knowledge, the use of GPUs for ASP computations has not been explored and, as demonstrated in this paper, it opens an interesting set of possibilities and issues to be resolved.

The work proposed in this paper builds on two existing lines of research. The exploitation of parallelism from ASP computations has been explored in several research works, starting with seminal papers by Pontelli et al. and Finkel et al. [25, 9], and later continued in several other projects (e.g., [26, 12, 24]). Most of the existing proposals have primarily focused on parallelization of the search process underlying the construction of answer sets, by distributing parts of the search tree among different processors/cores; furthermore, the literature focused on parallelization on traditional multi-core or Beowulf architectures. These approaches are not applicable in the context of GPGPUs—the models of parallelization used on GPGPUs are deeply different (e.g., GPGPUs are designed to operate with large number of threads, operating in a synchronous way; GPGPUs have significantly more complex memory organizations, that have great impact on parallel performance) and existing parallel ASP models are not scalable on GPGPUs. Furthermore, our focus on this work is not primarily on search parallelism, but on parallelization of the various operations associated to unit propagation and management of nogoods.

The second line of research that supports the effort proposed in this paper is the recent developments in the area of GPGPUs for SAT solving and constraint programming. The work in [6] illustrates how to parallelize the search process employed by the DPLL procedure in solving a SAT problem on GPGPUs; the outcomes demonstrate the potential benefit of delegating to GPGPUs the tails of the branches of the search tree—an idea that we have also applied in the work presented in this paper. Several other proposals have appeared in the literature suggesting the use of GPGPUs to parallelize parts of the SAT solving process—e.g., the computation of variable heuristics [18]. The work presented in [4] provides a preliminary investigation of parallelization of constraint solving (applied to the specific domain of protein structure prediction) on GPGPUs. The work we performed in [4] provided inspiration for the ideas used in this paper to parallelize unit propagation and other procedures.

The main contribution of the research presented in this paper is the analysis of a state of the art algorithm for answer set computation (i.e., the algorithm underlying CLASP) to identify potential sources of parallelism that are suitable to the peculiar parallel architecture provided by CUDA.

## 2 Background

### 2.1 Answer Set Programming

**Syntax.** In this section we will briefly review the foundations of ASP, starting with its syntax. Let us consider a language composed of a set of propositional symbols (atoms) $\mathcal{P}$. An ASP rule has the form

$$p_0 \leftarrow p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n \tag{1}$$

where $p_i \in \mathcal{P}$.[5] Given a rule $r$ of type (1), $p_0$ is referred to as the *head* of the rule ($head(r)$), while the set of atoms $\{p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n\}$ is referred to as the body of the rule ($body(r)$). In particular, $body^+(r) = \{p_1, \ldots, p_m\}$ and $body^-(r) = \{p_{m+1}, \ldots, p_n\}$. We identify particular types of rules: a *constraint* is a rule of the form

$$\leftarrow p_1, \ldots, p_m, \mathit{not}\ p_{m+1}, \ldots, \mathit{not}\ p_n \tag{2}$$

while a *fact* is a rule of the form $p_0 \leftarrow$. A program $\Pi$ is a collection of ASP rules. We will use the following notation: $atom(\Pi)$ denotes the set of all atoms present in $\Pi$, while $body_\Pi(p)$ denotes the set $\{body(r) \mid r \in \Pi,\ head(r) = p\}$.

Let $\Pi$ be a program; its *positive dependence graph* $\mathcal{D}^+_\Pi = (V, E)$ is a directed graph satisfying the following properties:

- The set of nodes $V = atom(\Pi)$;
- $E = \{(p, q) \mid r \in \Pi, head(r) = p, q \in body^+(r)\}$.

In particular, we are interested in recognizing cycles in $\mathcal{D}^+_\Pi$; the number of non-self loops in $\mathcal{D}^+_\Pi$ is denoted by $loop(\Pi)$. A program $\Pi$ is *tight* (*non-tight*) if $loop(\Pi) = 0$ ($loop(\Pi) > 0$). A *strongly connected component (scc)* of $\mathcal{D}^+_\Pi$ is a maximal subgraph of $X$ of $\mathcal{D}^+_\Pi$ such that there exists a path between each pair of nodes in $X$.

**Semantics.** The semantics of ASP programs is provided in terms of *answer sets*. Intuitively, an answer set is a minimal model of the program which supports each atom in the model—i.e., for each atom there is a rule in the program that has such atom in the head and whose body is satisfied by the model. Formally, a set of atoms $M$ is an answer set of a program $\Pi$ if $M$ is the minimal model of the *reduct program* $\Pi^M$, where the reduct is obtained from $\Pi$ as follows:

- remove from $\Pi$ all rules $r$ such that $M \cap body^-(r) \neq \emptyset$;
- remove all negated atoms from the remaining rules.

$\Pi^M$ is a *definite program*, i.e., a set of rules that does not contain any occurrence of *not*. Definite programs are characterized by the fact that they admit a unique minimal model. Each answer set of a program $\Pi$ is, in particular, a minimal model of $\Pi$.

*Example 1.* The following program $\Pi$ has two answer sets: $\{a, c\}$ e $\{a, d\}$.

$$\Pi = \left\{ \begin{array}{lll} a \leftarrow & c \leftarrow a, \mathit{not}\ d & e \leftarrow b \\ b \leftarrow \neg a & d \leftarrow \mathit{not}\ c, \mathit{not}\ e & e \leftarrow e \end{array} \right\}$$

**Answer Set Computation.** In the rest of this section, we provide a brief overview of techniques used in the computation of the answer sets of a program; the material presented is predominantly drawn from the implementation techniques used in CLASP [11, 10].

Several ASP solvers rely directly or indirectly on techniques drawn from the domain of SAT solving, properly extended to include procedures to determine minimality and stability of the models (these two procedures can be quickly performed in time linear in the number of occurrences of atoms in the program, namely $|\Pi|$)). Several ASP

[5] A rule that includes first-order atoms with variables is simply seen as a syntactic sugar for all its ground instances.

solvers (e.g., CMODELS [13]) rely on a translation of $\Pi$ into a SAT problem and on the use of SAT solvers to determine putative answer sets. Other systems (e.g., CLASP) implement native ASP solvers, that combine search techniques with backjumping along with techniques drawn from the field of constraint programming [27].

The CLASP system relies on a search in the space of all truth value assignments to the atoms in $\Pi$, organized as a binary tree. The successful construction of a branch in the tree corresponds to the identification of an answer set of the program. If a, possibly partial, assignment fails to satisfy the rules in the program, then backjumping procedures are used to backtrack to the node in the tree that caused the failure. The design of the tree construction and the backjumping procedure in CLASP is implemented in such a way to guarantee that if a branch is successfully constructed, then the outcome is indeed an answer set of the program. CLASP's search is also guided by special assignments of truth values to subsets of atoms that are known not to be extendable into an answer set—these are referred to as *nogoods* [7, 27]. Assignments and nogoods are sets of assigned atoms—i.e., entities of the form $Tp$ ($Fp$) denoting that $p$ has been assigned `true` (`false`). For assignments it is also required that for each atom $p$ at most one between $Tp$ and $Fp$ is contained. Given an assignment $A$, we denote with $A^T = \{p \,|\, Tp \in A\}$ and $A^F = \{p \,|\, Fp \in A\}$. A is total if it assigns a truth value to every atom, otherwise it is partial. Given a (possibly partial) assignment $A$ and a nogood $\delta$, we say that $\delta$ is *violated* if $\delta \subseteq A$. In turn, a partial assignment $A$ is a *solution* for a set of nogoods $\Delta$ if no $\delta \in \Delta$ is violated by $A$.

The concept of nogood can be also used during deterministic propagation phases (a.k.a. *unit propagation*) to determine additional assignments. Given a nogood $\delta$ and a partial assignment $A$ such that $\delta \setminus A = \{Fp\}$ ($\delta \setminus A = \{Tp\}$), then we can infer the need to add $Tp$ ($Fp$) to $A$ in order to avoid violation of $\delta$. In the context of ASP computation, we distinguish two types of nogoods: *completion nogoods* [8], which are derived from Clark's completion of a logic program (we will denote with $\Delta_{\Pi_{cc}}$ the set of completion nogoods for the program $\Pi$), and *loop nogoods* [17], which are derived from the loop formula of $\Pi$ (denoted by $\Lambda_\Pi$). Before proceeding with the formal definitions of these two classes of nogoods, let us review the two fundamental results associated to them (see [10]). Let $\Pi$ be a program and $A$ an assignment:

- If $\Pi$ is a tight program then: $atom(\Pi) \cap A^T$ is an answer set of $\Pi$ iff $A$ satisfies all the nogoods in $\Delta_{\Pi_{cc}}$.
- If $\Pi$ is a non-tight program, then: $atom(\Pi) \cap A^T$ is an answer set of $\Pi$ iff $A$ satisfies all the nogoods in $\Delta_{\Pi_{cc}} \cup \Lambda_\Pi$.

Let us now proceed in the formal definitions of nogoods. Let us start by recalling the notion of Clark completion of $\Pi$ ($\Pi$):

$$\Pi_{cc} = \left\{\beta_r \leftrightarrow \bigwedge_{a \in body^+(r)} a \wedge \bigwedge_{b \in body^-(r)} \neg b \,|\, r \in \Pi\right\} \cup \left\{p \leftrightarrow \bigvee_{r \in body_\Pi(p)} \beta_r \,|\, p \in atom(\Pi)\right\} \quad (3)$$

Where $\beta_r$ is a new variable, introduced for each rule $r \in \Pi$, logically equivalent to the body of $r$. Assignments need to deal with $\beta_r$ variables, as well. The *completion nogoods* reflect the structure of the implications present in the definition of $\Pi_{cc}$. In particular:

- the implication present in the original rule $p \leftarrow body(r)$ implies the nogood $\{F\beta_r\} \cup \{Ta \,|\, a \in body^+(r)\} \cup \{Fb \,|\, b \in body^-(r)\}$.
- the implication in each rule also implies that the body should be false if any of its element is falsified, leading to the set of nogoods of the form: $\{T\beta_r, Fa\}$ for each $a \in body^+(r)$ and $\{T\beta_r, Tb\}$ for each $b \in body^-(r)$.
- the closure of an atom definition (as disjunction of the rule bodies supporting it) leads to a nogood expressing that the atom is true if any of its rule is true: $\{Fp, T\beta_r\}$ for each $r \in body_\Pi(p)$.
- similarly, the atom cannot be true if all its rules have a false body. This yields the nogood $\{Tp\} \cup \{F\beta_r \,|\, r \in body_\Pi(p)\}$.

$\Delta_{\Pi_{cc}}$ is the set of all the nogoods defined as above.

The *loop nogoods* derive instead from the need to capture loop formulae, thus avoiding cyclic support of truth. Let us provide some preliminary definitions. Given a set of atoms $U$, we define the *external bodies* of $U$ (denoted by $EB_\Pi(U)$) as the set $\{\beta_r \,|\, r \in \Pi, body^+(r) \cap U = \emptyset\}$. Furthermore, let us define $U$ to be an unfounded set with respect to an assignment $A$ if, for each rule $r \in \Pi$, we have *(i)* $head(r) \notin U$, or *(ii)* $body(r)$ is falsified by $A$, or *(iii)* $body^+(r) \cap U \neq \emptyset$. The loop nogoods capture the fact that, for each unfounded set $U$, its elements have to be false. This is encoded by the following nogoods: for each set of atoms $U$ and for each $p \in U$, we create the nogood $\{Tp\} \cup \{F\beta_r \mid \beta_r \in EB_\Pi(U)\}$. We denote with $\Lambda_\Pi$ the set of all loop nogoods, and with $\Delta_\Pi$ the whole set of nogoods: $\Delta_\Pi = \Delta_{\Pi_{cc}} \cup \Lambda_\Pi$.

## 2.2 CUDA

Our proposal focuses on exploring the use GPGPU parallelism in ASP solving. GPGPU is a general term indicating the use of the multicores available within modern graphical processing units (GPUs) for general purpose parallel computing. NVIDIA is one of the pioneering manufacturers in promoting GPGPU computing, especially thanks to its *Computing Unified Device Architecture (CUDA)* [29]. The underlying conceptual model of parallelism supported by CUDA is *Single-Instruction Multiple-Thread (SIMT),* a variant of the SIMD model, where, in general, the same instruction is executed by different threads that run on identical cores, while data and operands may differ from thread to thread. CUDA's architectural model is represented in Figure 1.

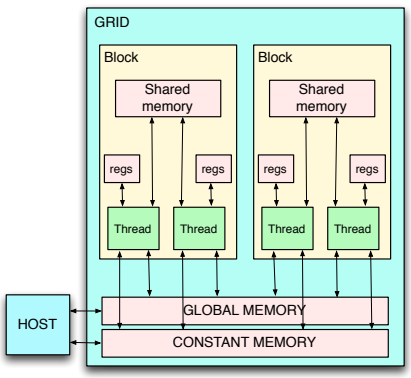



**Fig. 1:** CUDA Logical Architecture

Different NVIDIA GPUs are distinguished by the number of cores, their organization, and the amount of memory available. The GPU is composed of a series of *Streaming MultiProcessors (SMs);* the number of SMs depends on the specific characteristics of each family of GPU—e.g., the Fermi architecture provides 16 SMs. In turn, each SM contains a collection of computing cores; the number of cores per SM may range from

8 (in the older G80 platforms) to 32 (e.g., in the Fermi platforms). Each GPU provides access to both on-chip memory (used for thread registers and shared memory—defined later) and on-chip memory (used for L2 cache, global memory and constant memory). Notice that the architecture of the GPU also determines both the *GPU Clock* and the *Memory Clock* rates. A logical view of computations is introduced by CUDA, in order to define abstract parallel work and to schedule it among different hardware configurations (see Figure 1). A typical CUDA program is a C/C++ program that includes parts meant for execution on the CPU (referred to as the *host*) and parts meant for parallel execution on the GPU (referred as the *device*). A parallel computation is described by a collection of *kernels*—each kernel is a function to be executed by several threads.

The host program contains all instructions to initialize the data in GPUs, to define the threads number and to manage the kernel. Instead, a kernel is a set of instruction performed in GPUs across a set of concurrent threads. The programmer or compiler organizes these threads in thread *blocks* and *grids* of thread blocks. A grid is an array of thread blocks that execute the same kernel, read data input from global memory, write results to global memory. Each thread within a thread block executes an instance of the kernel, and has a thread ID within its thread block. When a CUDA program on the host CPU invokes a kernel grid, the blocks of the grid are enumerated and distributed to multiprocessors with available execution capacity; the kernel is executed in N blocks, each consisting of M threads. The threads in the same block can share data, using shared high-throughput on-chip memory; on the other hand, the threads belonging to different blocks can only share data through global memory. Thus, the block size allows the programmer to define the granularity of threads cooperation. Figure 1 shows the CUDA threads hierarchy [23].

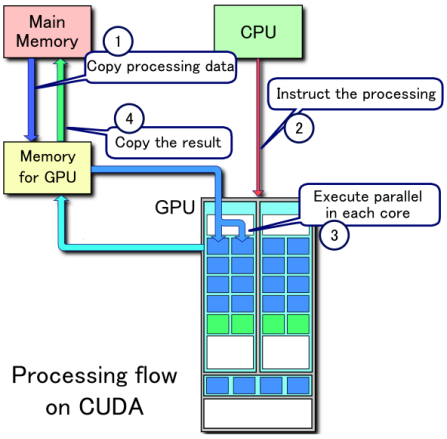



**Fig. 2:** Generic workflow in CUDA

CUDA provides an API to interact with GPU and C for CUDA, an extension of C language to define kernels. Referring to Figure 2, a typical CUDA application can be summarized as follow:

*Memory data allocation and transfer:* The data before being processed by kernels, must be allocated and transferred to Global memory. The CUDA API supports this operations through the functions `cudaMalloc()` and `cudaMemcpy()`. The call `cudaMalloc()` allows the programmer to allocate the space needed to store the data while the call `cudaMemcpy()` transfers the data from the memory of the host to the space previously allocated in Global Memory, or *vice versa*. The transfer rate is dependent on the bus bandwidth where the Graphics Card is physically connected.

*Kernels definition:* Kernels are defined as standard C functions; the annotation used to communicate to the CUDA compiler that a function should be treated as kernel has the

form: `__global__ void kernelName (Formal Arguments)` where `__global__` is the qualifier that shows to the compiler that the next statement is a kernel code.
*Kernels execution:* A kernel can be launched from the host program using a new:
`kernelName <<< GridDim, ThreadsPerBlock >>> (Actual Arguments)`
execution configuration syntax where `kernelName` is the specified name in kernel function prototype, `GridDim` is the number of blocks of the grid and `ThreadsPerBlock` specifies the number of threads in each block. Finally, the `Actual Arguments` are typically pointer variables, referring to the previously allocated data in Global Memory.
*Data retrieval:* After the execution of the kernel, the host needs to retrieve the data—representing results of the kernel. This is performed with another transfer operation from Global Memory to Host Memory, using the function `cudaMemcpy()`.

## 3 Design of an conflict-based CUDA ASP Solver

In this section, we will present the `CUD@ASP` procedure. This procedure is based on the `CDNL-ASP` procedure adopted in the CLASP system [11, 10]. The procedure assumes that the input is a ground ASP program. The novelty of `CUD@ASP` is the off-loading of several time consuming operations to the GPU—with particular focus on conflict analysis, exploration of the set of possible assignments and execution of the phases of unit-propagation. The rest of this section is organized as follows: we will start with an overview of the serial structure of the `CUD@ASP` procedure (Subsection 3.1). In the successive subsections, we will illustrate the parallel versions of the key procedures used in `CUD@ASP`: literal selection (Subsection 3.2), nogoods analysis (Subsection 3.3), unit propagation (Subsection 3.4), conflict analysis (Subsection 3.5), and analysis of stability (Subsection 3.7). In addition, we illustrate a method to use the GPU to handle the search process in the tail part of the search tree (Subsection 3.6).

### 3.1 The General `CUD@ASP` Procedure

The overall `CUD@ASP` procedure is summarized in Algorithm 3.2. The procedures that appear underlined in the algorithm are those that are delegated to the GPU for parallel execution. The algorithm makes use of the following notation. The input (ground) program is denoted by $\Pi$; $\Pi_{cc}$ denotes the completion of $\Pi$ (eq. 3). The overall set of nogoods is denoted by $\Delta_\Pi$, composed of the completion nogoods and the loop nogoods. For each program atom $p$, the notation $\overline{p}$ represents the atom with a truth value assigned; $\neg\overline{p}$ denotes, instead, the complement truth value with respect to $\overline{p}$.

Lines 1–5 of Algorithm 3.2 represent the initialization phase of the ASP computation. In particular, the `Parsing` procedure (Line 5) is in charge of computing the completion of $\Pi$ and extracting the nogoods. The set $A$ will keep track of the atoms that have already been assigned a truth value. It is initialized to the empty set in Line 1 and updated by the `Selection` procedure at Line 22. Two variables ($current\_dl$ and $k$) are introduced to support the rest of the computation. In particular, the variable $current\_dl$ represents the *decision level*; this variable acts as a counter that keeps track of the number of "choices" that have been made in the computation of an answer set. Line 6 invokes the procedure `StronglyConnectedComponent`, which

**Algorithm 3.2** CUD@ASP

**Input** $\Pi$ ground ASP program
**Output** An answer set, or null

```
 1: A := ∅                                                    ▷ Atoms assignment
 2: Δ_Π := ∅                                                  ▷ Nogoods
 3: current_dl := 0                                           ▷ Current Decision Level
 4: k := 32                                                   ▷ Threshold for Exhaustive Procedure
 5: (Δ_Π, k, Π_cc) := Parsing(Π)                              ▷ Initialize Δ_Π as Δ_{Π_cc}
 6: scc := StronglyConnectedComponent(Π)
 7: loop
 8:     Violation := NoGoodCheck(A, Δ_Π)
 9:     if (Violation is true) ∧ (current_dl = 0) then return no answer set
10:     end if
11:     if Violation is true then
12:         (current_dl, δ) = ConflictAnalysis(Δ_Π, A)
13:         Δ_Π = Δ_Π ∪ {δ}
14:         A := A \ {p̄ ∈ A | current_dl < dl(p̄)}
15:     else
16:         if ∃ δ ∈ Δ_Π such that δ \ A = {p̄} and p̄ ∉ A then
17:             A := UnitPropagation(A, Δ_Π)
18:         end if
19:     end if
20:     if There are atoms not assigned then
21:         if Number of atoms to assign > k then
22:             p̄ := Selection(Π_cc, A)
23:             current_dl := current_dl + 1
24:             dl(p̄) := current_dl
25:             A := A ∪ {p̄}
26:         else          ▷ At most k unassigned atoms: Non-deterministic GPU computation
27:             if There is a successful thread for Exhaustive(A) then
28:                 for each successful thread returning A := Exhaustive(A) do
29:                     if StableTest(A, Π_cc) is true then return A^T ∩ atom(Π)
30:                     end if
31:                 end for
32:             end if
33:         end if
34:     else return A^T ∩ atom(Π)
35:     end if
36: end loop
```

determines the positive dependence graph and its strongly connected components; in absence of loops, the program $\Pi$ is tight, thus not requiring the use of loop nogoods ($\Lambda_\Pi$). We have implemented the classical Tarjan's algorithm, running in $O(n+e)$, on CPU (where $n$ and $e$ are the numbers of nodes and edges, respectively). The loop in Lines 7–36 represents the core of the computation. It alternates the process of testing consistency and propagating assignments (through the nogoods), and of guessing a possible assignment to atoms that are still undefined. Each cycle starts with a call to the

procedure `NoGoodCheck` (Line 8)—which, given a partial assignment $A$, validates whether all the nogoods in $\Delta_\Pi$ are still satisfied. If a violation is detected, then the procedure `ConflictAnalysis` is used to determine the decision level causing the nogood violation, backtrack to such point in the search tree, and generate an additional nogood to prune that branch of the search space (Lines 11–14). If $\overline{p}$ is the assignment at the decision level determined by `ConflictAnalysis`, then the nogood will prompt the unit propagation process to explore the branch starting with the truth assignment $\neg\overline{p}$ (thus ensuring completeness of the computation [10]).

If the `ConflictAnalysis` procedure does not detect nogood violations, then the procedure might be in one of the following situations:

- If there is a nogood that is completely covered by $A$ except for one element $\overline{p}$, then the `UnitPropagation` procedure is called to determine assignments that are implied by the nogoods (starting with the assignment $\neg\overline{p}$) (Lines 16–17). Note that this procedure does not modify the decision level. In the case of non-tight programs, the `UnitPropagation` procedure will also execute a subroutine in charge of validating the loop nogoods.
- If there are atoms left to assign (Line 20), then additional selections will need to be performed. We distinguish two possibilities. If the number of unassigned atoms is larger than a threshold $k$, then one of them, say $\overline{p}$, is selected and the current decision level is recorded (by setting the value of the variable $dl(\overline{p})$—see Line 24). The `Selection` procedure is in charge for selecting a literal. The assignment is extended accordingly and the current decision level is increased (Lines 23–25). If the number of unassigned atoms is small, then a specialized parallel procedure (`Exhaustive`) systematically explores all the possible missing assignments. For each possible assignment of the remaining atoms, the procedure `StableTest` validates that all nogoods are satisfied and that the overall assignment $A$ is stable (necessary test in the case of non-tight programs). This is described in Lines 27–32.

### 3.2 `Selection` Procedure

The purpose of this procedure is to determine an unassigned atom in the program and a truth value for it. A number of heuristic strategies have been studied to determine atom and assignment, often derived from analogous strategies developed in the context of SAT solving or constraint solving [27, 2]. As soon as an atom has been selected, it is necessary to assign a truth value to it. A traditional strategy [10] consists of assigning at the beginning the value `true` to bodies of rules, while atoms are initially assigned `false`—aiming at maximizing the number of resulting implications.

There is no an optimal strategy for all problems, of course. In the current implementation, we provide three selection strategies: the most frequently occurring literal strategy which selects the atom that appears in the largest number of nogoods (that aims at determining violations as soon as possible or to lead to early propagations through the nogoods), the leftmost-first strategy (which selects the first unassigned atom found), and the *Jeroslow-Wang* strategy (also based on the frequency of occurrence of an atom, but placing a greater value on smaller nogoods). All the three strategies are implemented by allowing kernels on the GPU to concurrently compute the rank of each atom; these rankings are re-evaluated at each backjump.

**Algorithm 3.3** `NoGoodCheck` ▷ Kernel executed by thread $i$

**Input** $A$, $\Delta_\Pi = \{\delta_1, \ldots, \delta_m\}$ ▷ An assignment $A$ and a set of nogoods $\Delta_\Pi$
**Output** True or False

```
 1: if i ≤ m then
 2:     state := 0
 3:     covered := 0
 4:     Atom to propagate := NULL
 5:     for all p̄ ∈ δ_i do
 6:         if ¬p̄ ∈ A then state := 1
 7:         else if p̄ ∈ A then covered := covered + 1
 8:         else Atom to propagate := p̄
 9:         end if
10:     end for
11:     if covered = |δ_i| then return Violation := True
12:     else if covered = |δ_i| − 1 and state = 0 then
13:         Make Atom to propagate global
14:     end if
15:     return Violation := False
16: end if
```

### 3.3 `NoGoodCheck` Procedure

The `NoGoodCheck` procedure (see Algorithm 3.3) is primarily used to verify whether the current partial assignment $A$ violates any of the nogoods in a given set $\Delta_\Pi$. The procedure plays also the additional rôle of identifying opportunities for unit propagation—i.e., recognizing nogoods $\delta$ such that $\delta \setminus A = \{\overline{p}\}$ and $\neg\overline{p} \notin A$. In this case, the element $p$ will be the target of a successive unit propagation phase.

The pseudocode in Algorithm 3.3 describes a CUDA kernel (i.e., running on GPU) implementing the `NoGoodCheck`. Each thread handles one of the nogoods in $\Delta_\Pi$ and performs a linear scan of its assigned atoms (Lines 5–10). The local flag `state` keeps track of whether the nogood is satisfied by the assignment (`state` equal to 1). The counter `covered` keeps track of how many elements of $\delta_i$ have already been found in $A$. The condition of `state` equal to zero and the `covered` counter equal to the size of the nogood implies that the nogood is violated by $A$. The first thread to detect a violation will communicate it to the host by setting a variable (`Violation`—Line 11) in global memory (used in Lines 9 and 11 of the general `CUD@ASP` procedure).

Lines 12–13 implement the second functionality of the `NoGoodCheck` procedure—by identifying and making global the single element of the nogood that is not covered by the $A$ assignment. Note that the identification of the element `Atom to Propagate` can be conveniently performed in `NoGoodCheck` since the procedure is already performing the scanning of the nogood to check its validity.

### 3.4 `UnitPropagation` Procedure

The `UnitPropagation` procedure is performed only if the `NoGoodCheck` has detected no violations and has exposed at least one atom for propagation (as in Lines

12–13 of Algorithm 3.3). `UnitPropagation` is implemented as a CUDA kernel—which allows us to distribute the different nogoods among threads, each in charge of extending the partial assignment $A$ with one additional assignment. The procedure is iterated until a fixpoint is reached. The extension of $A$ is an immediate consequence of the work done in `NoGoodCheck`: if the check of a nogood $\delta_i$ identifies $\overline{p}$ as the only element in $\delta_i$ not covered by $A$ (i.e., $\{\overline{p}, \neg\overline{p}\} \cap A = \emptyset$), then $A$ is extended as $A := A \cup \{\neg\overline{p}\}$.

If the program $\Pi$ is non-tight, then the `UnitPropagation` procedure includes an additional phase aimed at performing the computation of the unfounded sets determined by the partial assignment $A$ and the corresponding loop nogoods $\Lambda_\Pi$. This process is implemented by the procedure `UnfoundedSetCheck` and follows the general structure of the analogous procedure used in the implementation of CLASP [10]. This procedure performs an analysis of the strongly connected components of the positive dependence graph $\mathcal{D}^+_\Pi$ (already computed at the beginning of the computation of `CUD@ASP`—Line 6). For each $p \in atoms(\Pi)$, $scc(p)$ denotes the set of atoms that belong to the same strongly connected component as $p$. An atom $p$ is said to be cyclic if there exists a rule $r \in \Pi$ such that: $head(r) \in scc(p)$ and $body^+(r) \cap scc(p) \neq \emptyset$, otherwise $p$ is *acyclic*. Cyclic atoms are the core of the search for unfounded sets—since they are the only ones that can appear in the unfounded loops. Cyclic atoms along with the knowledge of elements assigned by $A$ allow the computation of unfounded sets, as discussed in [17, 10]. In the current implementation `UnfoundedSetCheck` runs on the host. Some parts are inherently parallelizable (e.g., the computation of the external-support, or a splitting to different threads of the analysis of each scc component)—their execution on the device is work in progress.

### 3.5 `ConflictAnalysis` Procedure

The `ConflictAnalysis` procedure is used to resolve a conflict detected by the `NoGoodCheck` by identifying a level $dl$ and assignment $\overline{p}$ the computation should backtrack to, in order to remove the nogood violation. This process allows classical backjumping in the search tree generated by the Algorithm 3.2 [28, 27]. In addition to this, the procedure produces a new nogood to be added to the nogoods set, in order to prevent the same assignments in future. This procedure is implemented by a sequence of kernels, and it is executed after some nogood violations have been detected by `NoGoodCheck`. This procedure works as follows:

- Each thread is assigned to a unique nogood ($\delta$).
- The thread determines the last two assigned literals in $\delta$, say $\ell_M(\delta)$ and $\ell_m(\delta)$. The two (not necessarily distinct) decision levels of these assignments are stored in $dl_M(\delta) = dl(\ell_M(\delta))$ and $dl_m(\delta) = dl(\ell_m(\delta))$, respectively.
- The thread verifies whether $\delta$ is violated.
- Then, the violated nogood $\overline{\delta}$ with lowest value of $dl_M$ is determined.

At this point, a nogood learning procedure is activated. A kernel function (again, one thread for each existing nogood) determines each nogood $\varepsilon$, such that: *(a)* $\neg\ell_M(\overline{\delta}) \in \varepsilon$ and *(b)* $\varepsilon \setminus \{\neg\ell_M(\overline{\delta})\} \subseteq A$. Heuristic functions (see, e.g., [1]) can be applied to select one of these $\varepsilon$. Currently, the smallest one is selected in order to generate small new

nogoods—as future work, we will consider all the set of these nogoods. The next step performs a sequence of steps, by repeatedly setting $\overline{\delta} := (\varepsilon \backslash \{\neg \ell_M(\overline{\delta})\}) \cup (\overline{\delta} \backslash \{\ell_M(\overline{\delta})\})$ and coherently updating the values of $dl_M(\overline{\delta})$ and $dl_m(\overline{\delta})$, until $dl_M(\overline{\delta}) \neq dl_m(\overline{\delta})$. This procedure ends with the identification of a unique implication point (UIP [21]) that determines the lower decision level/literal among those causing the detected conflicts. We use such value for backjumping (Line 14 of Algorithm 3.2). The last nogood obtained in this manner is also added to the set of nogoods.

### 3.6 `Exhaustive` Procedure

GPU are typically employed for data parallelism. However, as shown in [6], when the size of the problem is manageable, it is possible to use them for massive search parallelism. We have developed the `Exhaustive` procedure for this task. It is called when at most $k$ atoms remains undecided—where $k$ is a parameter that can be set by the user (by default, $k = 32$). The nogood set is simplified using the current assignment (this is done in parallel by a kernel that assigns each nogood to a thread). This simplified sets will be then processed by a second kernel with $2^k$ threads, that non-deterministically explores all of the possible assignments. Each thread verifies that the assignments do not violate the nogoods set. If this happens, in case of a tight program, we have found an answer set. Otherwise the `StableTest` procedure (Sect. 3.7) is launched (Lines 27–28 of Algorithm 3.2). The efficiency of this procedure is obtained by a careful use of low-level data-structures. For example, the Boolean assignment of 32 atoms is stored in a single integer variable. Similarly, the nogood representation is stored using bit-strings, and violation control is managed by low-level bit operations.

### 3.7 `StableTest` Procedure

In order to verify whether an assignment found by the `Exhaustive` procedure is a stable model, we have implemented a GPU kernel that behaves as follows:

- It computes the reduct of the program: each thread takes care of an individual rule; as result, some threads may become inactive due to rule elimination, threads dealing with rules with all negative literals not in the model simply ignore them, while all other threads are idle.
- A computation of the minimum fixpoint is performed. Each thread handles one rule (internally modified by the first step above) and, if the body is satisfied, updates the sets of derived atoms. Once a rule is triggered, it becomes inactive, speeding-up the consecutive computations.
- When a fixpoint is reached, the computed and the guessed models are compared.

## 4 Concluding discussion

We have reported on our working project of developing an ASP solver running (partially) on GPGPUs. We implemented a working prototypical solver. The first results in experimenting with different GPU architectures are encouraging. Table 1 shows an

excerpt of the results obtained on some instances (taken from the Second ASP Competition). The differences between the performance obtained by exploiting different GPUs are evident and indicates the strong potential for enhanced performance and the scalability of the approach.

Table 2 reports on the performance of different serial ASP solvers, on the same collection of instances. Far from being a deep and fair comparison of these solvers against the GPU-based prototype, these results show that even at this stage of its development, the parallel prototype can compete, in some cases, with the existing and highly optimized serial solvers. Notice that the GPU-based prototype does not benefit from a number of refined heuristics and search/decision strategies exploited, for instance, by the state of the art solver CLASP.

It should be noticed that, in order to profitably exploit in full the computational power of the GPUs, one has to carefully tune its parallel application w.r.t. the characteristics of the specific device at hand. The architectural features and characteristics of the specific GPU family has to be carefully taken into account. Moreover, even considering a given GPU, different options can be adopted both in partitioning tasks among threads/warps and in allocating/transferring data on the device's memory. Clearly, such choices sensibly affect the performance of the whole application. This can be better explained by considering Table 3. It shows the performance obtained by three versions of the GPU-based solver, differing in the way the device's global memory is used. Apart from the default allocation mentioned in Sect. 2.2, CUDA provides two other basic kind of memory allocation. A first possibility uses *page-locking* to speed up address resolution. *Mapped* allocation allows one to map a portion of host memory into the device global memory. In this way the data transfer between host and device is implicitly ensured by the system and explicit memory transfers (by means of the function `cudaMemcpy()`) can be avoided. The first column of Table 3 shows the performance of a version of the prototype that allocates all data by using *mapped* memory. The behavior of a faster version of the solver which exploits *page-locking* to deal with the main data structures (essentially those representing the set of nogoods), is shown in the second column. Clearly, this approach requires additional programming effort (in optimizing and keeping track of memory transfers). Even better performance has been achieved by a third version of the solver that adopts *page-locking* to allocate all data structures, only on the device. This solution may appear, in some sense, unappealing, because it imposes to implement on the device also some intrinsically-serial functionalities. Even if these functions cannot fully exploit the parallelism of the cores, considerable advantage is achieved by avoiding most of the memory transfer between host and device.

In this work we made initial steps towards the creation of a GPU-based ASP-solver; however, further effort is needed to improve the solver. In particular, some procedures need to be optimized in order to take greater advantage from the high data-/task-parallelism offered by GPGPUs and the different types of available memories. Moreover, some parts of the solver currently running on the host, should be replaced by suitable parallel counterparts (examples are the computation of the strongly connected components of the dependence graph and the computation of the unfounded sets). We plan to develop the stability test that avoids analyzing the whole program and the implementation of the `NoGoodCheck` that makes use of watched literals.

| Instance | GT520 | GT640 | GTX580 |
|---|---|---|---|
| channelRoute_3 | 5.44 | 1.73 | 0.37 |
| knights_11_11 | 0.70 | 0.23 | 0.06 |
| knights_13_13 | 1.70 | 0.51 | 0.12 |
| knights_15_15 | 1.71 | 0.51 | 0.12 |
| knights_17_17 | 2.40 | 0.69 | 0.16 |
| knights_20_20 | 8.57 | 2.34 | 0.46 |
| labyrinth.0.5 | 0.08 | 0.08 | 0.05 |
| schur_4_41 | 0.24 | 0.16 | 0.07 |
| schur_4_42 | 0.31 | 0.20 | 0.07 |

**Table 1.** Results obtained with three different Nvidia GeForce GPUs: GT520 (48 cores, capability 2.0, GPU clock 1.62 GHz, memory clock rate 0.50 GHz, global memory 1GB), GT640 (384 cores, capability 3.0, GPU clock 0.90 GHz, memory clock rate 0.89 GHz, global memory 2GB), GTX580 (512 cores, capability 2.0, GPU clock 1.50 GHz, memory clock rate 2.00 GHz, global memory 1.5GB). The timing is in seconds.

## References


[1] C. Anger, M. Gebser, and T. Schaub. Approaching the Core of Unfounded Sets. Proceedings of the International Workshop on Nonmonotonic Reasoning. 2006.

[2] A. Biere. *Handbook of Satisfiability*, IOS Press, 2009.

[3] H. Blair and A. Walker. *Towards a theory of declarative knowledge*. IBM Watson Research Center, 1986.

[4] F. Campeotto, A. Dovier, and E. Pontelli. Protein Structure Prediction on GPU: an experimental report. *Proc. of RCRA*, Rome, June 2013.

[5] K. Clark. Negation as Failure. *Logic and Databases*, Morgan Kaufmann, 1978.

[6] A. Dal Palù, A. Dovier, A. Formisano, E. Pontelli. Exploiting Unexploited Computing Resources for Computational Logics. *Proc. of CILC*, CEUR, vol 857, 2012.

[7] R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.

[8] F. Fages. Consistency of Clark's Completion and Existence of Stable Models. *Journal of Methods of Logic in Computer Science*, 1(1):51–60, 1994.

[9] R. Finkel et al. Computing Stable Models in Parallel. *Answer Set Programming*, AAAI Spring Symposium, 2001.

[10] M. Gebser, B. Kaufmann, and T. Schaub. Conflict-driven Answer Set Solving: From Theory to Practice. *Artificial Intelligence* 187: 52–89, 2012.

[11] M. Gebser et al. *Answer Set Solving in Practice.* Morgan & Claypool, 2012.

[12] M. Gebser et al. Multi-Threaded ASP Solving with CLASP. *TPLP*, 12(4-5), 2012.

[13] E. Giunchiglia et al. Answer Set Programming Based on Propositional Satisfiability. *Journal of Automated Reasoning*, 36(4):345–377, 2006.

[14] J. Herbrand. *Recherches sur la théorie de la démonstration*. Doctoral Dissertation, Univ. of Paris, 1930.

[15] R. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. *Annals of Math and AI*, 1:167-187, 1990.

[16] Khronos Group Inc. OpenCL Reference Pages. http://www.khronos.org, 2011.

[17] F. Lin and Y. Zhao. Assat: Computing Answer Sets of a Logic Program by SAT Solvers. *Artificial Intelligence*, 157(1):115–137, 2004.

[18] P. Manolios and Y. Zhang. Implementing Survey Propagation on Graphics Processing Units. *SAT*, Springer Verlag, 2006.

| Instance | SMODELS | CMODELS | CLASP-None | CLASP | GTX580 |
|---|---|---|---|---|---|
| channelRoute_3 | 2.08 | 1.42 | 69.27 | 0.24 | 0.37 |
| knights_11_11 | 0.34 | 0.11 | 0.03 | 0.03 | 0.06 |
| knights_13_13 | 1.12 | 0.21 | 0.06 | 0.06 | 0.12 |
| knights_15_15 | 1.12 | 0.24 | 0.05 | 0.07 | 0.12 |
| knights_17_17 | 0.91 | 1.99 | 0.05 | 0.06 | 0.16 |
| knights_20_20 | 9.61 | 3.85 | 0.22 | 0.20 | 0.46 |
| labyrinth.0.5 | 0.02 | 0.01 | 0.01 | 0.01 | 0.05 |
| schur_4_41 | 0.05 | 0.70 | 0.02 | 0.02 | 0.07 |
| schur_4_42 | 0.07 | 0.60 | 0.02 | 0.05 | 0.07 |

**Table 2.** Results obtained with different solvers. All experiments where run on the same machine (host: QuadCore Intel i7 CPU, 2.93GHz, 4GB RAM; device GTX580). Serial solvers: SMODELS v. 2.34; CMODELS v. 3.85 exploiting minisat; CLASP v. 2.1.0. The column 'CLASP-None' shows results obtained with CLASP by inhibiting its decision heuristics. This makes its selection strategy analogous to the one used in our implementation. The timing is in seconds.

| Instance | All data mapped | $\Delta_\Pi$ page-locked | All data page-locked |
|---|---|---|---|
| knights_11_11 | 0.87 | 0.16 | 0.06 |
| knights_13_13 | 2.50 | 0.34 | 0.12 |
| knights_15_15 | 2.49 | 0.35 | 0.12 |
| knights_17_17 | 3.60 | 0.50 | 0.16 |
| knights_20_20 | 14.14 | 1.60 | 0.46 |
| labyrinth.0.5 | 0.82 | 0.03 | 0.05 |
| schur_4_41 | 19.71 | 0.09 | 0.07 |
| schur_4_42 | 24.75 | 0.12 | 0.07 |

**Table 3.** Results obtained with GTX580 with different use of memory resources.


[19] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM (JACM)*, 38(3):587–618, 1991.

[20] W. Marek and M. Truszczyński. Stable Models as an Alternative Programming Paradigm. *The Logic Programming Paradigm*, Springer Verlag, 1999.

[21] J. Marques-Silva and K. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48:506-521, 1999.

[22] I. Niemela. Logic Programming with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Math and AI*, 25:241-273, 1999.

[23] J. Nickolls and W.J. Dally. The GPU Computing Era. In *IEEE Micro*, 30(2):56-59, 2010.

[24] S. Perri, F. Ricca, and M. Sirianni. Parallel Instantiation of ASP Programs: Techniques and Experiments. *TPLP*, 13(2), 2013.

[25] E. Pontelli and O. El-Khatib. Exploiting Vertical Parallelism from Answer Set Programs. *Answer Set Programming*, AAAI Spring Symposium, 2001.

[26] E. Pontelli, H. Le and T. Son. An Investigation in Parallel Execution of ASP on Distribute Memory Platforms. *Computer Languages, Systems & Structures*, 36(2):158-202, 2010.

[27] F. Rossi, P. Van Beek, and T. Walsh. *Handbook of Constraint Programming*, Elsevier, 2006.

[28] S. Russell et al. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[29] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison Wesley, 2011.

[30] A. Van Gelder et al. The Well-founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):619–649, 1991.

# Dealing with Incompleteness and Vagueness in Inductive Logic Programming

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

[1] Dipartimento di Informatica, Università degli Studi di Bari "Aldo Moro", Italy
[2] ISTI - CNR, Pisa, Italy

**Abstract.** Incompleteness and vagueness are inherent properties of knowledge in several real world domains and are particularly pervading in those domains where entities could be better described in natural language. In order to deal with incomplete and vague structured knowledge, several fuzzy extensions of Description Logics (DLs) have been proposed in the literature. In this paper, we address the issues raised by incomplete and vague knowledge in Inductive Logic Programming (ILP). We present a novel ILP method for inducing fuzzy DL inclusion axioms from crisp DL knowledge bases and discuss the results obtained in comparison with related works.

## 1 Introduction

Incompleteness and vagueness are inherent properties of knowledge in several real world domains and are particularly pervading in those domains where entities could be better described in natural language. The issues raised by incomplete and vague knowledge have been traditionally addressed in the field of Knowledge Representation (KR).

*Incomplete knowledge.* The *Open World Assumption* (OWA) is used in KR to codify the informal notion that in general no single agent or observer has complete knowledge. The OWA limits the kinds of inference and deductions an agent can make to those that follow from statements that are known to the agent to be true. In contrast, the *Closed World Assumption* (CWA) allows an agent to infer, from its lack of knowledge of a statement being true, anything that follows from that statement being false. Heuristically, the OWA applies when we represent knowledge within a system as we discover it, and where we cannot guarantee that we have discovered or will discover complete information. In the OWA, statements about knowledge that are not included in or inferred from the knowledge explicitly recorded in the system may be considered unknown, rather than wrong or false. Description Logics (DLs) are KR formalisms compliant with the OWA, thus turning out to be particularly suitable for representing *incomplete* knowledge [1].

*Vague knowledge.* It is well known that "classical" DLs are not appropriate to deal with *vague* knowledge [20]. We recall for the inexpert reader that there has been a long-lasting misunderstanding in the literature of artificial intelligence and

uncertainty modelling, regarding the role of probability/possibility theory and vague/fuzzy theory. A clarifying paper is [5]. Specifically, under *uncertainty theory* fall all those approaches in which statements are true or false to some *probability* or *possibility* (for example, "it will rain tomorrow"). That is, a statement is true or false in any world/interpretation, but we are "uncertain" about which world to consider as the right one, and thus we speak about, *e.g.*, a probability distribution or a possibility distribution over the worlds. On the other hand, under *fuzzy theory* fall all those approaches in which statements (for example, "the car is long") are true to some *degree*, which is taken from a truth space (usually $[0, 1]$). That is, an interpretation maps a statement to a truth degree, since we are unable to establish whether a statement is entirely true or false due to the involvement of vague concepts, such as "long car" (the degree to which the sentence is true depends on the length of the car). Here, we shall focus on fuzzy logic only.

*Learning in fuzzy DLs.* Although a relatively important amount of work has been carried out in the last years concerning the use of fuzzy DLs as ontology languages [20] and the use of DLs as representation formalisms in Inductive Logic Programming (ILP) [13], the problem of automatically managing the evolution of fuzzy ontologies by applying ILP algorithms still remains relatively unaddressed. Konstantopoulos and Charalambidis [9] propose an ad-hoc translation of fuzzy Łukasiewicz $\mathcal{ALC}$ DL constructs into LP in order to apply a conventional ILP method for rule learning. However, the method is not sound as it has been recently shown that the mapping from fuzzy DLs to LP is incomplete [17] and entailment in Łukasiewicz $\mathcal{ALC}$ is undecidable [4]. Iglesias and Lehmann [7] propose an extension of DL-Learner [10] with some of the most up-to-date fuzzy ontology tools, e.g. the *fuzzyDL* reasoner [2]. Notably, the resulting system can learn fuzzy OWL DL [3] equivalence axioms from FuzzyOWL 2 ontologies. [4] However, it has been tested only on a toy problem with crisp training examples and does not build automatically fuzzy concrete domains. Lisi and Straccia [14] present *Soft*FOIL, a logic-based method for learning fuzzy $\mathcal{EL}$ inclusion axioms from fuzzy DL-Lite ontologies (also, *Soft*FOIL has not been implemented and tested).

*Contribution of this paper.* In this paper, we describe a novel method, named FOIL-$\mathcal{DL}$, for learning fuzzy $\mathcal{EL}(\mathbf{D})$ inclusion axioms from any crisp DL knowledge base. [5] Similarly to *Soft*FOIL, it adapts the popular rule induction method FOIL [18]. However, FOIL-$\mathcal{DL}$ differs from *Soft*FOIL mainly by the fact that the latter learns fuzzy $\mathcal{EL}$ inclusion axioms from fuzzy DL-Lite ontologies, while the former learns fuzzy $\mathcal{EL}(\mathbf{D})$ inclusion axioms from any crisp DL ontology.

*Structure of the paper.* The paper is structured as follows. For the sake of self-containment, Section 2 introduces some basic definitions we rely on. Section 3 describes the learning problem and the solution strategy of FOIL-$\mathcal{DL}$. Section 4 illustrates some results obtained in a comparative study between FOIL-$\mathcal{DL}$ and

[3] `http://www.w3.org/TR/2009/REC-owl2-overview-20091027/`
[4] `http://www.straccia.info/software/FuzzyOWL`
[5] $\mathcal{DL}$ stands for any DL.

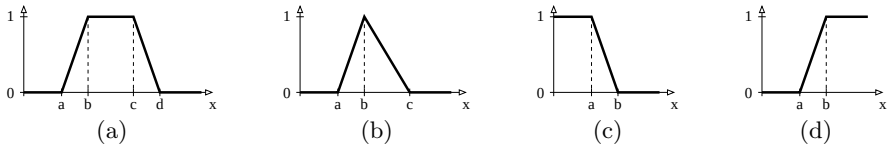



**Fig. 1.** (a) Trapezoidal function $trz(a, b, c, d)$, (b) triangular function $tri(a, b, c)$, (c) left-shoulder function $ls(a, b)$, and (d) right-shoulder function $rs(a, b)$.

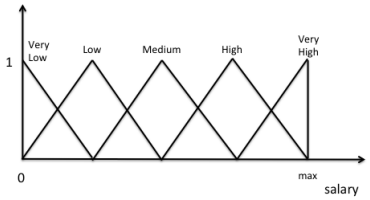
DL-Learner on the popular ILP problem of Michalski's trains. Section 5 concludes the paper by discussing limits of the current work, related work and possible directions of future work.

## 2 Preliminaries

*Mathematical Fuzzy Logic.* Fuzzy Logic is the logic of fuzzy sets. A *fuzzy set* $A$ over a countable crisp set $X$ is a function $A\colon X \to [0, 1]$. Let $A$ and $B$ be two fuzzy sets. The standard fuzzy set operations conform to $(A \cap B)(x) = \min(A(x), B(x))$, $(A \cup B)(x) = \max(A(x), B(x))$ and $\bar{A}(x) = 1 - A(x)$, while the *inclusion degree* between $A$ and $B$ is defined typically as

$$deg(A, B) = \frac{\sum_{x \in X}(A \cap B)(x)}{\sum_{x \in X} A(x)} \ . \tag{1}$$

The trapezoidal (Fig. 1 (a)), the triangular (Fig. 1 (b)), the left-shoulder function, Fig. 1 (c)), and the right-shoulder function, Fig. 1 (d)) are frequently used to specify *membership functions* of fuzzy sets. Although fuzzy sets have a greater expressive power than classical crisp sets, their usefulness depend critically on the capability to construct appropriate membership functions for various given concepts in different contexts. The problem of constructing meaningful membership functions is a difficult one (see, *e.g.*, [8, Chapter 10]). However, one easy and typically satisfactory method to define the membership functions is to uniformly partition the range of values into 5 or 7 fuzzy sets using either trapezoidal functions, or triangular functions. The latter is the more used one, as it has less parameters and is also the approach we adopt. For instance, the figure below illustrates salary values (bounded by a minimum and maximum value), partitioned uniformly into 5 fuzzy sets.




In *Mathematical Fuzzy Logic* [6], 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 *e.g.* $[0, 1]$. This degree is called *degree of truth*

**Table 1.** Syntax and semantics of constructs for the $\mathcal{ALC}$ DL.

| | | |
|---|---|---|
| bottom (resp. top) concept | $\bot$ (resp. $\top$) | $\emptyset$ (resp. $\Delta^{\mathcal{I}}$) |
| atomic concept | $A$ | $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ |
| role | $R$ | $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ |
| individual | $a$ | $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ |
| concept negation | $\neg C$ | $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$ |
| concept intersection | $C_1 \sqcap C_2$ | $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ |
| concept union | $C_1 \sqcup C_2$ | $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$ |
| value restriction | $\forall R.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \forall y\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$ |
| existential restriction | $\exists R.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |
| general concept inclusion | $C_1 \sqsubseteq C_2$ | $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$ |
| concept assertion | $a : C$ | $a^{\mathcal{I}} \in C^{\mathcal{I}}$ |
| role assertion | $(a,b) : R$ | $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ |

of the logical statement $\phi$ in the interpretation $\mathcal{I}$. For us, *fuzzy statements* have the form $\langle \phi, \alpha \rangle$, where $\alpha \in (0,1]$ and $\phi$ is a statement, encoding that the degree of truth of $\phi$ is *greater or equal* $\alpha$.

A *fuzzy interpretation* $\mathcal{I}$ maps each atomic statement $p_i$ into $[0,1]$ and is then extended inductively to all statements: $\mathcal{I}(\phi \wedge \psi) = \mathcal{I}(\phi) \otimes \mathcal{I}(\psi)$, $\mathcal{I}(\phi \vee \psi) = \mathcal{I}(\phi) \oplus \mathcal{I}(\psi)$, $\mathcal{I}(\phi \rightarrow \psi) = \mathcal{I}(\phi) \Rightarrow \mathcal{I}(\psi)$, $\mathcal{I}(\neg\phi) = \ominus\mathcal{I}(\phi)$, $\mathcal{I}(\exists x.\phi(x)) = \sup_{y \in \Delta^{\mathcal{I}}} \mathcal{I}(\phi(y))$, $\mathcal{I}(\forall x.\phi(x)) = \inf_{y \in \Delta^{\mathcal{I}}} \mathcal{I}(\phi(y))$, 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 [6]. Any other continuous t-norm can be obtained from them. The combination functions in Gödel logic are defined as follows:

$$a \otimes b = \min(a,b), a \oplus b = \max(a,b), a \Rightarrow b = \begin{cases} 1 & \text{if } a \leq b \\ b & \text{otherwise} \end{cases}, \ominus a = \begin{cases} 1 & \text{if } a = 0 \\ 0 & \text{otherwise} \end{cases}. \quad (2)$$

The notions of satisfiability and logical consequence are defined in the standard way, where a fuzzy interpretation $\mathcal{I}$ *satisfies* a fuzzy statement $\langle \phi, \alpha \rangle$ or $\mathcal{I}$ is a *model* of $\langle \phi, \alpha \rangle$, denoted as $\mathcal{I} \models \langle \phi, \alpha \rangle$, iff $\mathcal{I}(\phi) \geq \alpha$.

*Fuzzy Description Logics.* Description Logics (DLs) are a family of decidable First Order Logic (FOL) fragments that allow for the specification of structured knowledge in terms of classes (*concepts*), instances (*individuals*), and binary relations between instances (*roles*) [1]. Complex concepts (denoted with $C$) can be defined from atomic concepts ($A$) and roles ($R$) by means of the constructors available for the DL in hand. The set of constructors for the $\mathcal{ALC}$ DL is reported in Table 1. A DL *Knowledge Base* (KB) $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a pair where $\mathcal{T}$ is the so-called *Terminological Box* (TBox) and $\mathcal{A}$ is the so-called *Assertional Box* (ABox). The TBox is a finite set of *General Concept Inclusion* (GCI) axioms which represent

is-a relations between concepts, whereas the ABox is a finite set of *assertions* (or *facts*) that represent instance-of relations between individuals (resp. couples of individuals) and concepts (resp. roles). Thus, when a DL-based ontology language is adopted, an ontology is nothing else than a TBox, and a populated ontology corresponds to a whole KB (*i.e.*, encompassing also an ABox).

The semantics of DLs can be defined directly with set-theoretic formalizations (as shown in Table 1 for the case of $\mathcal{ALC}$) or through a mapping to FOL (as shown in [3]). An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for a DL KB consists of a domain $\Delta^{\mathcal{I}}$ and a mapping function $\cdot^{\mathcal{I}}$. For instance, $\mathcal{I}$ maps a concept $C$ into a set of individuals $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, *i.e.* $\mathcal{I}$ maps $C$ into a function $C^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow \{0, 1\}$ (either an individual belongs to the extension of $C$ or does not belong to it). Under the *Unique Names Assumption* (UNA) [19], individuals are mapped to elements of $\Delta^{\mathcal{I}}$ such that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$. However UNA does not hold by default in DLs. An interpretation $\mathcal{I}$ is a *model* of a KB $\mathcal{K}$ iff it satisfies all axioms and assertions in $\mathcal{T}$ and $\mathcal{A}$ . In DLs a KB represents many different interpretations, *i.e.* all its models. This is coherent with the OWA that holds in FOL semantics. A DL KB is *satisfiable* if it has at least one model.

The main reasoning task for a DL KB $\mathcal{K}$ is the *consistency check* which tries to prove the satisfiability of $\mathcal{K}$. Another well known reasoning service in DLs is *instance check*, i.e., the check of whether an ABox assertion is a logical implication of a DL KB. A more sophisticated version of instance check, called *instance retrieval*, retrieves, for a DL KB $\mathcal{K}$, all (ABox) individuals that are instances of the given (possibly complex) concept expression $C$, i.e., all those individuals $a$ such that $\mathcal{K}$ entails that $a$ is an instance of $C$.

Concerning fuzzy DLs, several fuzzy extensions of DLs have been proposed (see the survey in [15]). We recap here the fuzzy variant of the DL $\mathcal{ALC}(\mathbf{D})$ [21].

A *fuzzy concrete domain* or *fuzzy datatype theory* $\mathbf{D} = \langle \Delta^{\mathbf{D}}, \cdot^{\mathbf{D}} \rangle$ consists of a datatype domain $\Delta^{\mathbf{D}}$ and a mapping $\cdot^{\mathbf{D}}$ that assigns to each data value an element of $\Delta^{\mathbf{D}}$, and to every $n$-ary datatype predicate $d$ an $n$-ary fuzzy relation over $\Delta^{\mathbf{D}}$. We will restrict to unary datatypes as usual in fuzzy DLs. Therefore, $\cdot^{\mathbf{D}}$ maps indeed each datatype predicate into a function from $\Delta^{\mathbf{D}}$ to $[0, 1]$. Typical examples of datatype predicates $\mathbf{d}$ are the well known membership functions

$$\mathbf{d} := ls(a, b) \mid rs(a, b) \mid tri(a, b, c) \mid trz(a, b, c, d) \mid \geq_v \mid \leq_v \mid =_v ,$$

where *e.g.* $ls(a, b)$ is the left-shoulder membership function and $\geq_v$ corresponds to the crisp set of data values that are greater or equal than the value $v$.

In $\mathcal{ALC}(\mathbf{D})$, each role is either an *object property* (denoted with $R$) or a *datatype property* (denoted with $T$). Complex concepts are built according to the following syntactic rules:

$$C \rightarrow \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid C_1 \rightarrow C_2 \mid \exists R.C \mid \forall R.C \mid \exists T.\mathbf{d} \mid \forall T.\mathbf{d} \ . \quad (3)$$

Axioms in a fuzzy $\mathcal{ALC}(\mathbf{D})$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ are graded, *e.g.* a GCI is of the form $\langle C_1 \sqsubseteq C_2, \alpha \rangle$ (*i.e.* $C_1$ is a sub-concept of $C_2$ to degree at least $\alpha$). We may omit the truth degree $\alpha$ of an axiom; in this case $\alpha = 1$ is assumed.

Concerning the semantics, let us fix a fuzzy logic. In fuzzy DLs, $\mathcal{I}$ maps $C$ into a function $C^{\mathcal{I}} : \Delta^{\mathcal{I}} \rightarrow [0, 1]$ and, thus, an individual belongs to the extension of $C$ to some degree in $[0, 1]$, *i.e.* $C^{\mathcal{I}}$ is a fuzzy set. Specifically, a *fuzzy*

*interpretation* is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a nonempty (crisp) set $\Delta^{\mathcal{I}}$ (the *domain*) and of a *fuzzy interpretation function* $\cdot^{\mathcal{I}}$ that assigns: *(i)* to each atomic concept $A$ a function $A^{\mathcal{I}}\colon \Delta^{\mathcal{I}} \to [0,1]$; *(ii)* to each object property $R$ a function $R^{\mathcal{I}}\colon \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \to [0,1]$; *(iii)* to each data type property $T$ a function $T^{\mathcal{I}}\colon \Delta^{\mathcal{I}} \times \Delta^{\mathbf{D}} \to [0,1]$; *(iv)* to each individual $a$ an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$; and *(v)* to each concrete value $v$ an element $v^{\mathcal{I}} \in \Delta^{\mathbf{D}}$.

Now, $\cdot^{\mathcal{I}}$ is extended to concepts as specified below (where $x \in \Delta^{\mathcal{I}}$):

$\bot^{\mathcal{I}}(x) = 0, \quad \top^{\mathcal{I}}(x) = 1,$
$(C \sqcap D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \otimes D^{\mathcal{I}}(x), \quad (C \sqcup D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \oplus D^{\mathcal{I}}(x),$
$(\neg C)^{\mathcal{I}}(x) = \ominus C^{\mathcal{I}}(x), \quad (C \rightarrow D)^{\mathcal{I}}(x) = C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x),$
$(\forall R.C)^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x,y) \Rightarrow C^{\mathcal{I}}(y)\}, \quad (\exists R.C)^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathcal{I}}} \{R^{\mathcal{I}}(x,y) \otimes C^{\mathcal{I}}(y)\},$
$(\forall T.\mathbf{d})^{\mathcal{I}}(x) = \inf_{y \in \Delta^{\mathbf{D}}} \{T^{\mathcal{I}}(x,y) \Rightarrow \mathbf{d}^{\mathbf{D}}(y)\}, \quad (\exists T.\mathbf{d})^{\mathcal{I}}(x) = \sup_{y \in \Delta^{\mathbf{D}}} \{T^{\mathcal{I}}(x,y) \otimes \mathbf{d}^{\mathbf{D}}(y)\}\ .$

Hence, for every concept $C$ we get a function $C^{\mathcal{I}} : \Delta^{\mathcal{I}} \to [0,1]$.

The *satisfiability of axioms* is then defined by the following conditions: *(i)* $\mathcal{I}$ satisfies an axiom $\langle a{:}C, \alpha\rangle$ if $C^{\mathcal{I}}(a^{\mathcal{I}}) \geq \alpha$; *(ii)* $\mathcal{I}$ satisfies an axiom $\langle (a,b){:}R, \alpha\rangle$ if $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) \geq \alpha$; *(iii)* $\mathcal{I}$ satisfies an axiom $\langle C \sqsubseteq D, \alpha\rangle$ if $(C \sqsubseteq D)^{\mathcal{I}} \geq \alpha$ where $(C \sqsubseteq D)^{\mathcal{I}} = \inf_{x \in \Delta^{\mathcal{I}}} \{C^{\mathcal{I}}(x) \Rightarrow D^{\mathcal{I}}(x)\}$. $\mathcal{I}$ is a model of $\mathcal{K}$ iff $\mathcal{I}$ satisfies each axiom in $\mathcal{K}$. We say that $\mathcal{K}$ *entails* an axiom $\langle \tau, \alpha\rangle$, denoted $\mathcal{K} \models \langle \tau, \alpha\rangle$, if any model of $\mathcal{K}$ satisfies $\langle \tau, \alpha\rangle$. The *best entailment degree* of $\tau$ w.r.t. $\mathcal{K}$, denoted $bed(\mathcal{K}, \tau)$, is defined as

$$bed(\mathcal{K}, \tau) = \sup\{\alpha \mid \mathcal{K} \models \langle \tau, \alpha\rangle\}\ . \tag{4}$$

## 3 Learning fuzzy $\mathcal{EL}(\mathbf{D})$ axioms with Foil-$\mathcal{DL}$

### 3.1 The problem statement

The problem considered in this paper concerns the automated induction of fuzzy $\mathcal{EL}(\mathbf{D})$ [6] GCI axioms providing a sufficient condition for a given atomic concept $H$. It can be cast as a rule learning problem, provided that positive and negative examples of $H$ are available. This problem can be formalized as follows.

Given:

- a consistent crisp $\mathcal{DL}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}\rangle$ (the *background theory*);
- an atomic concept $H$ (the *target concept*);
- a set $\mathcal{E} = \mathcal{E}^+ \cup \mathcal{E}^-$ of crisp concept assertions labelled as either positive or negative examples for $H$ (the *training set*);
- a set $\mathcal{L}_{\mathcal{H}}$ of fuzzy $\mathcal{EL}(\mathbf{D})$ GCI axioms (the *language of hypotheses*)

the goal is to find a set $\mathcal{H} \subset \mathcal{L}_{\mathcal{H}}$ (a *hypothesis*) such that: $\forall e \in \mathcal{E}^+, \mathcal{K} \cup \mathcal{H} \models e$ (completeness), and $\forall e \in \mathcal{E}^-, \mathcal{K} \cup \mathcal{H} \not\models e$ (consistency).

Here we assume that $\mathcal{K} \cap \mathcal{E} = \emptyset$. Also, the language $\mathcal{L}_{\mathcal{H}}$ is given implicitly by means of syntactic restrictions over a given alphabet. In particular, the alphabet underlying $\mathcal{L}_{\mathcal{H}}$ is a subset of the alphabet for the language $\mathcal{L}_{\mathcal{K}}$ of the background

[6] $\mathcal{EL}(\mathbf{D})$ is a fragment of $\mathcal{ALC}(\mathbf{D})$.

theory. However, $\mathcal{L}_\mathcal{H}$ differs from $\mathcal{L}_\mathcal{K}$ as for the form of axioms. Please note that we do not make any specific assumption about the DL the background theory refers to. 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 that $\mathcal{K} \cup \mathcal{H} \not\models \bot$, which means that $\mathcal{K} \cup \mathcal{H}$ is a consistent theory, *i.e.* has a model. An axiom $\phi \in \mathcal{L}_\mathcal{H}$ *covers* an example $e \in \mathcal{E}$ iff $\mathcal{K} \cup \{\phi\} \models e$.

**The training examples.** Given the target concept $H$, the training set $\mathcal{E}$ consists of concept assertions of the form $a{:}H$, where $a$ is an individual occurring in $\mathcal{K}$. Note that both $\mathcal{K}$ and $\mathcal{E}$ is crisp. Also, $\mathcal{E}$ is split into $\mathcal{E}^+$ and $\mathcal{E}^-$. Note that, under OWA, $\mathcal{E}^-$ consists of all those individuals which can be proved to be instance of $\neg H$. On the other hand, under CWA, $\mathcal{E}^-$ is the collection of individuals, which cannot be proved to be instance of $H$.

**The language of hypotheses.** Given the target concept $H$, the hypotheses to be induced are fuzzy GCIs of the form

$$B \sqsubseteq H \ , \tag{5}$$

where the left-hand side is defined according to the following $\mathcal{EL}(\mathbf{D})$ syntax

$$B \longrightarrow \top \mid A \mid \exists R.B \mid \exists T.\mathbf{d} \mid B_1 \sqcap B_2 \ . \tag{6}$$

The language $\mathcal{L}_\mathcal{H}$ generated by this syntax is potentially infinite due, *e.g.*, to the nesting of existential restrictions yielding to complex concept expressions such as $\exists R_1.(\exists R_2 \ldots.(\exists R_n.(C))\ldots)$. $\mathcal{L}_\mathcal{H}$ is made finite by imposing further restrictions on the generation process such as the maximal number of conjuncts and the depth of existential nesting allowed in the left-hand side. Also, note that the learnable GCIs do not have an explicit truth degree. However, as we shall see later on, once we have learned a fuzzy GCI of the form (5), we attach to it a confidence degree that is obtained by means of the $cf$ function (see Eq. (8)). Finally, note that the syntactic restrictions of Eq. (6) w.r.t. Eq. (3) allow for a straightforward translation of the inducible axioms into rules of the kind "if $x$ is a $C_1$ and ... and $x$ is a $C_n$ then $x$ is an $H$", which corresponds to the usual pattern in fuzzy rule induction (in our case, $B \sqsubseteq H$ is seen as a rule "if $B$ then $H$") .

### 3.2 The solution strategy

The solution proposed for the learning problem defined in Section 3.1 is inspired by FOIL. FOIL is a popular ILP algorithm for learning sets of rules which performs a greedy search in order to maximise a gain function [18].

In FOIL-$\mathcal{DL}$, the learning strategy of FOIL (*i.e.*, the so-called *sequential covering* approach) is kept. The function LEARN-SETS-OF-AXIOMS (reported in Figure 2) carries on inducing axioms until all positive examples are covered. When an axiom is induced (step 3.), the positive examples covered by the axiom (step 5.) are removed from $\mathcal{E}$ (step 6.). In order to induce an axiom, the function

**function** LEARN-SETS-OF-AXIOMS($\mathcal{K}$, $H$, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{L}_\mathcal{H}$): $\mathcal{H}$
**begin**
1. $\mathcal{H} := \emptyset$;
2. **while** $\mathcal{E}^+ \neq \emptyset$ **do**
3. $\quad \phi :=$ LEARN-ONE-AXIOM($\mathcal{K}$, $H$, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{L}_\mathcal{H}$);
4. $\quad \mathcal{H} := \mathcal{H} \cup \{\phi\}$;
5. $\quad \mathcal{E}^+_\phi := \{e \in \mathcal{E}^+ | \mathcal{K} \cup \phi \models e\}$;
6. $\quad \mathcal{E}^+ := \mathcal{E}^+ \setminus \mathcal{E}^+_\phi$;
7. **endwhile**
8. **return** $\mathcal{H}$
**end**

**Fig. 2.** FOIL-$\mathcal{DL}$: Learning a set of GCI axioms.

LEARN-ONE-AXIOM (reported in Figure 3) starts with the most general axiom (*i.e.* $\top \sqsubseteq H$) and specializes it by applying the refinement rules implemented in the function REFINE (step 7.). The iterated specialization of the axiom continues until the axiom does not cover any negative example and its *confidence degree* is greater than a fixed threshold ($\theta$). The confidence degree of axioms being generated with REFINE allows for evaluating the *information gain* obtained on each refinement step by calling the function GAIN (step 9.).

Due to the peculiarities of the language of hypotheses in FOIL-$\mathcal{DL}$, necessary changes are made to FOIL as concerns the functions REFINE and GAIN. Details about these novel features are provided in the next two subsections.

**The refinement operator.** The function REFINE implements a *specialization* operator with the following refinement rules:

$Add_A$**:** adds an atomic concept $A$
$Add_{\exists R.\top}$**:** adds a complex concept $\exists R.\top$ by existential role restriction
$Add_{\exists T.\mathbf{d}}$**:** adds a complex concept $\exists T.\mathbf{d}$ by existential role restriction
$Subst_A$**:** replaces an atomic concept $A$ with another atomic concept $A'$ s.t. $A' \sqsubseteq A$

At each refinement step (*i.e.* at each call of REFINE), the rules are applied first to the left-hand side of the axiom being specialized and then recursively to the range of all the conjuncts defined with existential role restriction. For example, let us consider that $H$ is the target concept, $A$, $A'$, $B$, $R$, $R'$, $T$ are concepts and properties occurring in $\mathcal{K}$, and $A' \sqsubseteq A$ holds in $\mathcal{K}$. Under these assumptions, the axiom $\exists R.B \sqsubseteq H$ is specialized into the following axioms:

- $A \sqcap \exists R.B \sqsubseteq H$, $B \sqcap \exists R.B \sqsubseteq H$, $A' \sqcap \exists R.B \sqsubseteq H$;
- $\exists R'.\top \sqcap \exists R.B \sqsubseteq H$, $\exists T.\mathbf{d} \sqcap \exists R.B \sqsubseteq H$;
- $\exists R.(B \sqcap A) \sqsubseteq H$, $\exists R.(B \sqcap A') \sqsubseteq H$;
- $\exists R.(B \sqcap \exists R.\top) \sqsubseteq H$, $\exists R.(B \sqcap \exists R'.\top) \sqsubseteq H$, $\exists R.(B \sqcap \exists T.\mathbf{d}) \sqsubseteq H$.

The application of the refinement rules is not blind. It takes the background theory into account in order to avoid the generation of redundant or useless

**function** LEARN-ONE-AXIOM($\mathcal{K}$, $H$, $\mathcal{E}^+$, $\mathcal{E}^-$, $\mathcal{L}_\mathcal{H}$): $\phi$
**begin**
1. $B := \top$;
2. $\phi := B \sqsubseteq H$;
3. $\mathcal{E}^-_\phi := \mathcal{E}^-$;
4. **while** $cf(\phi) < \theta$ **or** $\mathcal{E}^-_\phi \neq \emptyset$ **do**
5. $B_{best} := B$;
6. $maxgain := 0$;
7. $\Phi := \text{REFINE}(\phi, \mathcal{L}_\mathcal{H})$
8. **foreach** $\phi' \in \Phi$ **do**
9. $gain := \text{GAIN}(\phi', \phi)$;
10. **if** $gain \geq maxgain$ **then**
11. $maxgain := gain$;
12. $B_{best} := B'$;
13. **endif**
14. **endforeach**
15. $\phi := B_{best} \sqsubseteq H$;
16. $\mathcal{E}^-_\phi := \{e \in \mathcal{E}^- | \mathcal{K} \cup \phi \models e\}$;
17. **endwhile**
18. **return** $\phi$
**end**

**Fig. 3.** FOIL-$\mathcal{DL}$: Learning one GCI axiom.

hypotheses. For example, if the concept $B'$ is the range of $R'$ in $\mathcal{K}$, the function REFINE adds the conjunct $\exists R'.B'$ instead of $\exists R'.\top$. One such "informed" refinement operator is able to perform "cautious" big steps in the search space.

Note that a specialization operator reduces the number of examples covered by a GCI. More precisely, the aim of a refinement step is to reduce the number of covered negative examples, while still keeping some covered positive examples. Since learned GCIs cover only positive examples, $\mathcal{K}$ will remain consistent after the addition of a learned GCI.

**The heuristic.** The function GAIN implements an information-theoretic criterion for selecting the best candidate at each refinement step according to the following formula:

$$\text{GAIN}(\phi', \phi) = p * (log_2(cf(\phi')) - log_2(cf(\phi))) \ , \tag{7}$$

where $p$ is the number of positive examples covered by the axiom $\phi$ that are still covered by $\phi'$. Thus, the gain is positive iff $\phi'$ is more informative in the sense of Shannon's information theory, *i.e.* iff the confidence degree ($cf$) increases. If there are some refinements, which increase the confidence degree, the function GAIN tends to favour those that offer the best compromise between the confidence degree and the number of examples covered. Here, $cf$ for an axiom $\phi$ of the form (5) is computed as a sort of fuzzy set inclusion degree (see Eq. (1)) between the fuzzy set represented by concept $B$ and the (crisp) set represented by concept $H$. More formally:

$$cf(\phi) = cf(B \sqsubseteq H) = \frac{\sum_{a \in \mathsf{Ind}^+_H(\mathcal{A})} bed(\mathcal{K}, a{:}B)}{|\mathsf{Ind}_H(\mathcal{A})|} \quad (8)$$

where $\mathsf{Ind}^+_H(\mathcal{A})$ (resp., $\mathsf{Ind}_H(\mathcal{A})$ ) is the set of individuals occurring in $\mathcal{A}$ and involved in $\mathcal{E}^+_\phi$ (resp., $\mathcal{E}^+_\phi \cup \mathcal{E}^-_\phi$) such that $bed(\mathcal{K}, a{:}B) > 0$. We remind the reader that $bed(\mathcal{K}, a{:}B)$ denotes the best entailment degree of the concept assertion $a{:}B$ w.r.t. $\mathcal{K}$ as defined in Eq. (4). Note that for individuals $a \in \mathsf{Ind}^+_H(\mathcal{A})$, $\mathcal{K} \models a{:}H$ holds and, thus, $bed(\mathcal{K}, a{:}B \sqcap H) = bed(\mathcal{K}, a{:}B)$. Also, note that, even if $\mathcal{K}$ is crisp, the possible occurrence of fuzzy concrete domains in expressions of the form $\exists T.\mathbf{d}$ in $B$ may imply that both $bed(\mathcal{K}, B \sqsubseteq H) \notin \{0, 1\}$ and $bed(\mathcal{K}, a{:}B) \notin \{0, 1\}$.

### 3.3 The implementation

A variant of FOIL-$\mathcal{DL}$ has been implemented in the *fuzzyDL-Learner* [7] system and provided with two GUIs: One is a stand-alone Java application, the other is a tab widget plug-in for the ontology editor Protégé [8] (release 4.2).

Several implementation choices have been made. Notably, fuzzy GCIs in $\mathcal{L}_\mathcal{H}$ are interpreted under Gödel semantics (see Eq. (2)). However, since $\mathcal{K}$ and $\mathcal{E}$ are represented in crisp DLs, we have used a classical DL reasoner, together with a specialised code, to compute the confidence degree of fuzzy GCIs. Therefore, the system relies on the services of DL reasoners to solve all the deductive inference problems necessary to FOIL-$\mathcal{DL}$ to work, namely instance retrieval, instance check and subclasses retrieval. In particular, the sets $\mathsf{Ind}^+_H(\mathcal{A})$ and $\mathsf{Ind}_H(\mathcal{A})$ are computed by posing instance retrieval problems to the DL reasoner. Conversely, $bed(\mathcal{K}, a{:}\exists T.\mathbf{d})$ can be computed from the derived $T$-fillers $v$ of $a$, and applying the fuzzy membership function of $\mathbf{d}$ to $v$. The examples covered by a GCI, and, thus, the entailment tests in LEARN-SETS-OF-AXIOMS and LEARN-ONE-AXIOM, have been determined in a similar way.

The implementation of FOIL-$\mathcal{DL}$ features several optimizations w.r.t. the solution strategy presented in Section 3.2. Notably, the search in the hypothesis space can be optimized by enabling a backtracking mode. This option allows to overcome one of the main limits of FOIL, *i.e.* the sequential covering strategy. Because it performs a greedy search, formulating a sequence of rules without backtracking, FOIL does not guarantee to find the smallest or best set of rules that explain the training examples. Also, learning rules one by one could lead to less and less interesting rules. To reduce the risk of a suboptimal choice at any search step, the greedy search can be replaced in FOIL-$\mathcal{DL}$ by a *beam search* which maintains a list of $k$ best candidates at each step instead of a single best candidate. Additionally, to guarantee termination, we provide two parameters to limit the search space: namely, the maximal number of conjuncts and the maximal depth of existential nesting allowed in a fuzzy GCI. In fact, the computation may end without covering all positive examples.

[7] `http://straccia.info/software/FuzzyDL-Learner`
[8] `http://protege.stanford.edu/`

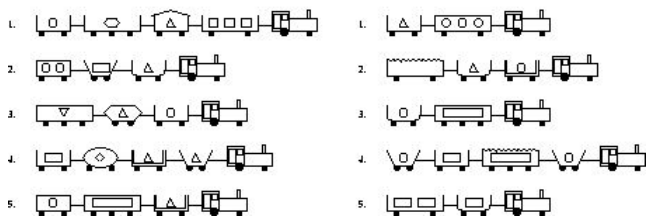


**Fig. 4.** Michalski's example of eastbound trains (left) and westbound trains (right) (illustration taken from [16]).

## 4 Comparing Foil-$\mathcal{DL}$ and DL-Learner

In this section we report the results of a comparison between Foil-$\mathcal{DL}$ and DL-Learner on a very popular learning task in ILP proposed 20 years ago by Ryszard Michalski [16] and illustrated in Figure 4. Here, 10 trains are described, out of which 5 are eastbound and 5 are westbound. The aim of the learning problem is to find the discriminating features between these two classes.

For the purpose of this comparative study, we have considered two slightly different versions, *trains2* and *trains3*, of an ontology encoding the original Trains data set. [9] The former has been adapted from the version distributed with DL-Learner in order to be compatible with Foil-$\mathcal{DL}$. Notably, the target classes `EastTrain` and `WestTrain` have become part of the terminology and several class assertion axioms have been added for representing positive and negative examples. The metrics for *trains2* are reported in Table 2. The ontology does not encompass any data property. Therefore, no fuzzy concept can be generated when learning GCIs from *trains2* with Foil-$\mathcal{DL}$. However, the ontology can be slightly modified in order to test the fuzzy concept generation feature of Foil-$\mathcal{DL}$. Note that in *trains2* cars can be classified according to the classes `LongCar` and `ShortCar`. Instead of one such crisp classification, we may want a fuzzy classification of cars. This is made possible by removing `LongCar` and `ShortCar` (together with the related class assertion axioms) from *trains2* and introducing the data property `hasLenght` with domain `Car` and range `double` (together with several data property assertions). The resulting ontology, called *trains3*, presents the metrics reported in Table 2.

DL-Learner [10] features several algorithms. Among them, the closest to Foil-$\mathcal{DL}$ is ELTL since it implements a refinement operator for concept learning in $\mathcal{EL}$ [12]. Conversely, CELOE learns class expressions in the more expressive OWL DL [11]. Both work only under OWA and deal only with crisp DLs.

### 4.1 Results on the ontology *trains2*

*Trial with* Foil-$\mathcal{DL}$. The settings for this experiment allow for the generation of hypotheses with up to 5 conjuncts and 2 levels of existential nestings. Under

[9] `http://archive.ics.uci.edu/ml/datasets/Trains`

[10] `http://dl-learner.org/Projects/DLLearner`

**Table 2.** Ontology metrics for *trains2.owl* and *trains3.owl* according to Protégé.

| | # logical axioms | # classes | # object prop. | # data prop. | # individuals | DL expressivity |
|---|---|---|---|---|---|---|
| *trains2* | 345 | 32 | 5 | 0 | 50 | $\mathcal{ALCO}$ |
| *trains3* | 343 | 30 | 5 | 1 | 50 | $\mathcal{ALCO}(\mathbf{D})$ |

these restrictions, the GCI axioms learned by FOIL-$\mathcal{DL}$ for the target concept `EastTrain` are:

```
Confidence Axiom
1,000      3CarTrain and hasCar some (2LoadCar) subclass of EastTrain
1,000      3CarTrain and hasCar some (3WheelsCar) subclass of EastTrain
1,000      hasCar some (ElipseShapeCar) subclass of EastTrain
1,000      hasCar some (HexagonLoadCar) subclass of EastTrain
```

whereas the following GCI axioms are returned by FOIL-$\mathcal{DL}$ for `WestTrain`:

```
Confidence Axiom
1,000      2CarTrain subclass of WestTrain
1,000      hasCar some (JaggedCar) subclass of WestTrain
```

The algorithm returns the same GCIs under both OWA and CWA. Note that an important difference between learning in DLs and standard ILP is that the former works under OWA whereas the latter under CWA. In order to complete the Trains example we would have to introduce definitions and/or assertions to model the closed world. However, the CWA holds naturally in this example, because we have complete knowledge of the world, and thus the knowledge completion was not necessary. This explains the behaviour of FOIL-$\mathcal{DL}$ which correctly induces the same hypotheses in spite of the opposite semantic assumptions.

*Trial with ELTL.* For the target concept `EastTrain`, the class expression learned by ELTL is the following :

```
EXISTS hasCar.(ClosedCar AND ShortCar) (accuracy: 1.0)
```

whereas the following finding has been returned for the target concept `WestTrain`:

```
EXISTS hasCar.LongCar (accuracy: 0.8)
```

The latter is not fully satisfactory as for the example coverage.

*Trial with CELOE.* For the target concept `EastTrain`, CELOE learns several class expressions of which the most accurate is:

```
hasCar some (ClosedCar and ShortCar) (accuracy: 1.0)
```

whereas, for the target concept `WestTrain`, the most accurate among the ones found is the following:

```
hasCar only (LongCar or OpenCar) (accuracy: 1.0)
```

Note that the former coincide with the corresponding result obtained with ELTL while the latter is a more accurate variant of the corresponding class expression returned by ELTL. The increase in example coverage is due to the augmented expressive power of the DL supported in CELOE.

```
- hasLenght_low: hasLenght, triangular(23.0,32.0,41.0)
- hasLenght_fair: hasLenght, triangular(32.0,41.0,50.0)
- hasLenght_high: hasLenght, triangular(41.0,50.0,59.0)
- hasLenght_veryhigh: hasLenght, rightShoulder(50.0,59.0)
- hasLenght_verylow: hasLenght, leftShoulder(23.0,32.0)
```

**Fig. 5.** Fuzzy concepts derived by FOIL-$\mathcal{DL}$ from the data property `hasLenght`.

### 4.2 Results on the ontology *trains3*

*Trial with* FOIL-$\mathcal{DL}$. The outcomes for the target concepts `EastTrain` and `WestTrain` remain unchanged when FOIL-$\mathcal{DL}$ is run on *trains3* with the same configuration of the first trial. Yet, fuzzy concepts are automatically generated by FOIL-$\mathcal{DL}$ from the data property `hasLenght` (see Figure 5). However, from the viewpoint of discriminant power, these concepts are weaker than the other crisp concepts occurring in the ontology. In order to make the fuzzy concepts emerge during the generation of hypotheses, we have appropriately biased the language of hypotheses. In particular, by enabling only the use of object and data properties in $\mathcal{L}_{\mathcal{H}}$, FOIL-$\mathcal{DL}$ returns the following axiom for `EastTrain`:

```
Confidence Axiom
1,000      hasCar some (hasLenght_fair) and hasCar some (hasLenght_veryhigh)
                and hasCar some (hasLenght_verylow) subclass of EastTrain
```

Conversely, for `WestTrain`, a lighter bias is sufficient to make fuzzy concepts appear in the learned axioms. In particular, by disabling the class `2CarTrain` in $\mathcal{L}_{\mathcal{H}}$, FOIL-$\mathcal{DL}$ returns the following axioms:

```
Confidence Axiom
1,000      hasCar some (2WheelsCar and 3LoadCar) and hasCar some (3LoadCar and CircleLoadCar)
           subclass of WestTrain
1,000      hasCar some (0LoadCar) subclass of WestTrain
1,000      hasCar some (JaggedCar) subclass of WestTrain
1,000      hasCar some (2LoadCar and hasLenght_high) subclass of WestTrain
1,000      hasCar some (ClosedCar and hasLenght_fair) subclass of WestTrain
```

*Trial with ELTL.* For the target class `EastTrain`, ELTL returns a class expression which leaves some positive example uncovered (incomplete hypothesis):

```
(EXISTS hasCar.TriangleLoadCar AND EXISTS hasCar.ClosedCar) (accuracy: 0.9)
```

whereas, for the target concept `WestTrain`, it returns an overly general hypothesis which covers also negative examples (inconsistent hypothesis):

```
TOP (accuracy: 0.5)
```

This bad performance of ELTL on *trains3* is due to the low expressivity of $\mathcal{EL}$ and to the fact that the classes `LongCar` and `ShortCar`, which appeared to be discriminant in the first trial, do not occur in *trains3* and thus can not be used anymore for building hypotheses.

*Trial with CELOE.* The most accurate class expression found by CELOE for the target concept `EastTrain` is:

```
((not 2CarTrain) and hasCar some ClosedCar) (accuracy: 1.0)
```

However, interestingly, CELOE learns also the following class expressions containing classes obtained by numerical restriction from the data property `hasLenght`:

```
hasCar some (ClosedCar and hasLenght <= 48.5) (accuracy: 1.0)
hasCar some (ClosedCar and hasLenght <= 40.5) (accuracy: 1.0)
hasCar some (ClosedCar and hasLenght <= 31.5) (accuracy: 1.0)
```

These "interval classes" are just a step back from the fuzzification which, conversely, FOIL-$\mathcal{DL}$ is able to do. It is acknowledged that using fuzzy sets in place of "intervall classes" improves the readability of the induced knowledge about the data. As for the target concept `WestTrain`, the most accurate class expression among the ones found by CELOE is:

```
(2CarTrain or hasCar some JaggedCar) (accuracy: 1.0)
```

Once again, the augmented expressivity increases the effectiveness of DL-Learner.

## 5 Conclusions and future work

We have described a novel method, named FOIL-$\mathcal{DL}$, which addresses the problem of learning fuzzy $\mathcal{EL}(\mathbf{D})$ GCI axioms from crisp $\mathcal{DL}$ assertions. The method extends FOIL in a twofold direction: from crisp to fuzzy and from rules to GCIs. Notably, vagueness is captured by the definition of confidence degree reported in (8) and incompleteness is dealt with the OWA. Also, thanks to the variable-free syntax of DLs, the learnable GCIs are highly understandable by humans and translate easily into natural language sentences. In particular, FOIL-$\mathcal{DL}$ present the learned axioms according to the user-friendly presentation style of the Manchester OWL syntax [11] (the same used in Protégé).

We would like to stress the fact that FOIL-$\mathcal{DL}$ provides a different solution from *Soft*FOIL [14] as for the KR framework, the refinement operator and the heuristic. Also, unlike *Soft*FOIL, FOIL-$\mathcal{DL}$ has been implemented and tested. The experimental results are quite promising and encourage the application of FOIL-$\mathcal{DL}$ to more challenging real-world problems. Notably, in spite of the low expressivity of $\mathcal{EL}$, FOIL-$\mathcal{DL}$ has turned out to be robust mainly due to the refinement operator and to the fuzzification facilities. Note that a fuzzy OWL 2 version of the trains' problem (ontology *fuzzytrains_v1.5.owl*) [12] has been developed by Iglesias for testing the fuzzy extension of CELOE proposed in [7]. However, FOIL-$\mathcal{DL}$ can not handle fuzzy OWL 2 constructs such as fuzzy classes obtained by existential restriction of fuzzy datatypes, fuzzy concept assertions, and fuzzy role assertions. Therefore, it has been necessary to prepare an *ad-hoc* ontology (*trains3*) for comparing FOIL-$\mathcal{DL}$ and DL-Learner.

For the future, we intend to conduct a more extensive empirical evaluation of FOIL-$\mathcal{DL}$, which could suggest directions of improvement of the method towards more effective formulations of, *e.g.*, the information gain function and the refinement operator as well as of the search strategy and the halt conditions employed in LEARN-ONE-AXIOM. Also, it can be interesting to analyse the impact of the different fuzzy logics on the learning process. Eventually, we shall investigate about learning fuzzy GCI axioms from FuzzyOWL 2 ontologies, by coupling the learning algorithm to the *fuzzyDL* reasoner, instead of learning from crisp OWL 2 data by using a classical DL reasoner.

[11] `http://www.w3.org/TR/owl2-manchester-syntax/`

[12] Available at `http://wiki.aksw.org/Projects/DLLearner/fuzzyTrains`.

## References


1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications (2nd ed.). Cambridge University Press (2007)
2. Bobillo, F., Straccia, U.: fuzzyDL: An expressive fuzzy description logic reasoner. In: 2008 Int. Conf. on Fuzzy Systems. pp. 923–930. IEEE Computer Society (2008)
3. Borgida, A.: On the relative expressiveness of description logics and predicate logics. Artificial Intelligence Journal 82, 353–367 (1996)
4. Cerami, M., Straccia, U.: On the (un)decidability of fuzzy description logics under Łukasiewicz t-norm. Information Sciences 227, 1–21 (2013)
5. Dubois, D., Prade, H.: Possibility theory, probability theory and multiple-valued logics: A clarification. Annals of Mathematics and Artificial Intelligence 32(1-4), 35–66 (2001)
6. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer (1998)
7. Iglesias, J., Lehmann, J.: Towards integrating fuzzy logic capabilities into an ontology-based inductive logic programming framework. In: Proc. of the 11th Int. Conf. on Intelligent Systems Design and Applications. IEEE Press (2011)
8. Klir, G.J., Yuan, B.: Fuzzy sets and fuzzy logic: theory and applications. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1995)
9. Konstantopoulos, S., Charalambidis, A.: Formulating description logic learning as an inductive logic programming task. In: Proc. of the 19th IEEE Int. Conf. on Fuzzy Systems. pp. 1–7. IEEE Press (2010)
10. Lehmann, J.: DL-Learner: Learning Concepts in Description Logics. Journal of Machine Learning Research 10, 2639–2642 (2009)
11. Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. Journal of Web Semantics 9(1), 71–81 (2011)
12. Lehmann, J., Haase, C.: Ideal Downward Refinement in the $\mathcal{EL}$ Description Logic. In: De Raedt, L. (ed.) ILP 2009. Revised Papers. Lecture Notes in Computer Science, vol. 5989, pp. 73–87. Springer (2010)
13. Lisi, F.A.: A formal characterization of concept learning in description logics. In: Kazakov, Y., Lembo, D., Wolter, F. (eds.) Proc. of the 2012 Int. Workshop on Description Logics. CEUR Workshop Proceedings, vol. 846. CEUR-WS.org (2012)
14. Lisi, F.A., Straccia, U.: A logic-based computational method for the automated induction of fuzzy ontology axioms. Fundamenta Informaticae 124, 1–17 (2013)
15. Lukasiewicz, T., Straccia, U.: Managing Uncertainty and Vagueness in Description Logics for the Semantic Web. Journal of Web Semantics 6(4), 291–308 (2008)
16. Michalski, R.: Pattern recognition as a rule-guided inductive inference. IEEE Transactions on Pattern Analysis and Machine Intelligence 2(4), 349–361 (1980)
17. Motik, B., Rosati, R.: A faithful integration of description logics with logic programming. In: Veloso, M. (ed.) IJCAI 2007, Proc. of the 20th Int. Joint Conf. on Artificial Intelligence. pp. 477–482 (2007)
18. Quinlan, J.R.: Learning logical definitions from relations. Machine Learning 5, 239–266 (1990)
19. Reiter, R.: Equality and domain closure in first order databases. Journal of ACM 27, 235–249 (1980)
20. Straccia, U.: Reasoning within fuzzy description logics. Journal of Artificial Intelligence Research 14, 137–166 (2001)
21. Straccia, U.: Description logics with fuzzy concrete domains. In: Bachus, F., Jaakkola, T. (eds.) 21st Conf. on Uncertainty in Artificial Intelligence. pp. 559–567. AUAI Press (2005)

# Constraint and Optimization techniques for supporting Policy Making*

Marco Gavanelli[1], Fabrizio Riguzzi[1], Michela Milano[2], and Paolo Cagnoli[3]

[1] University of Ferrara
Via Saragat 1, 44122 Ferrara Italy
[2] University of Bologna
V.le Risorgimento 2, 40136, Bologna, Italy
[3] ARPA Emilia-Romagna
Bologna, Italy

**Abstract.** Modeling the policy making process is a very challenging task. To the best of our knowledge the most widely used technique in this setting is agent-based simulation. Each agent represents an individual entity (e.g., citizen, stakeholder, company, public association, public body). The agent interaction enables emerging behaviours to be observed and taken into account in the policy making process itself. We claim that another perspective should be considered in modeling policy issues, that is the global perspective. Each public body has global objectives, constraints and guidelines that have to be combined to take decisions. The policy making process should be at the same time consistent with constraints, optimal with respect to given objectives and assessed to avoid negative impacts on the environment, economy and society. We propose in this paper a constraint-based model for the global policy making process and we apply the devised model to the regional planning activity. A case study in the field of energy plan is used to evaluate the proposed model. Clearly an interaction with agent-based simulation is desirable and could provide important feedback to the global model. This aspect is the subject of current research.

## 1 The problem

Public policy issues are extremely complex, occur in rapidly changing environments characterized by uncertainty, and involve conflicts among different interests. Our society is ever more complex due to globalisation, enlargement and the changing geo-political situation. This means that political activity and intervention become more widespread, and so the effects of its interventions become more difficult to assess, while at the same time it is becoming ever more important to ensure that actions are effectively tackling the real challenges that this increasing complexity entails. Thus, those responsible for creating, implementing, and enforcing policies must be able to reach decisions about ill-defined problem situations that are not well understood, have no single correct answer, involve

* An extended version of this paper appeared in [9].

many competing interests and interact with other policies at multiple levels. It is therefore increasingly important to ensure coherence across these complex issues.

In this paper we consider policy issues related to regional planning, the science of the efficient placement of activities and infrastructures for the sustainable growth of a region. Regional plans are classified into types, such as Agriculture, Forest, Fishing, Energy, Industry, Transport, Waste, Water, Telecommunication, Tourism, Urban Development and Environment to name a few. Each plan defines activities that should be carried out during the plan implementation. On the regional plan, the policy maker has to take into account impacts on the environment, the economy and the society. The procedure aimed to assess the impacts of a regional plan is called Strategic Environmental Assessment [15] and relates activities defined in the plan to environmental and economic impacts. This assessment procedure is now manually implemented by environmental experts, but it is never applied during the plan/program construction. In addition, this procedure is applied on a given, already instantiated plan. Taking into account impacts a posteriori enables only corrective interventions that can at most reduce the negative effect of wrong planning decisions.

One important aspect to be considered for supporting policy makers with Computational Intelligence approaches is the definition of formal policy models. In the literature, the majority of policy models rely on agent based simulation [11, 14, 18] where agents represent the parties involved in the decision making and implementation process. The idea is that agent-based modeling and simulation is suitable for modeling complex systems. In particular, agent-based models permit carrying out computer experiments to support a better understanding of the complexity of economic, environmental and social systems, structural changes, and endogenous adjustment reactions in response to a policy change.

In addition to agent-based simulation models, which provide "individual level models", we claim that the policy planning activity needs a global perspective: in the case of regional planning, we need "a regional perspective" that faces the problem at a global level while tightly interacting with the individual level model. Thus rather than proposing an alternative approach with respect to simulation, we claim that the two approaches should be properly combined as they represent two different perspectives of the same problem: the individual and the global perspective. This integration is the subject of our current research activity. In this setting, regional planning activities can be cast into complex combinatorial optimization problems. The policy maker has to take decisions satisfying a set of constraints while at the same time achieving a set of (possibly conflicting) objectives such as reducing negative impacts and enhancing positive impacts on the environment, the society and the economy. For this reason, impact assessment should be integrated into the policy model so as to improve the current procedure performed a posteriori.

In previous work [8], we experimented two different technologies to address the Strategic Environmental Assessment (SEA) of a regional plan. The technologies we applied were Constraint Logic Programming (CLP) [13] and Causal

Probabilistic Logic Programming [19]; Logic Programming is common to both models, so the user could use one or the other from a same environment, and possibly hybridize them. In [10] we proposed a fuzzy model for the SEA. While being far more expressive than a traditional CLP approach, it is less usable within a Regional planning decision support system. We evaluated a previous regional plan with the two models, and proposed the outputs to an environmental expert. The expert compared the two outputs and chose the CLP model as the closest to a human-made assessment.

In this work, we extend the CLP model used for the assessment, and apply it to the planning problem, i.e., deciding which actions should be taken in a plan. In the model, decision variables represent political decisions (e.g., the magnitude of a given activity in the regional plan), potential outcomes are associated with each decision, constraints limit possible combination of assignments of decision variables, and objectives (also referred to as criteria) can be either used to evaluate alternative solutions or translated into additional constraints. The model has been solved with CLP [13] techniques, and tested on the Emilia-Romagna regional energy plan. The results have been validated by experts in policy making and impact assessment to evaluate the accuracy of the results.

Further constraint based approaches have been proposed for narrower problems in the field of energy, such as locating biomass power plants in positions that are both economically affordable [6, 2, 5] and environmentally sustainable [4]. Other approaches have been applied to wind turbine placement [12]. The problem faced in this paper is much broader, as the Region should decide which strategic investments to perform in the next two-three years (with a longer vision to 2020) in the energy field. All specific details are left to the implementation of the plan, but are not considered at the Regional Planning stage. To the best of our knowledge, this is the first time constraint-based reasoning is applied to such a wide and strategic perspective.

### 1.1 Regional Planning and Impact assessment

Regional Planning is the result of the main policy making activity of European regions. Each region has a budget distributed by the Operational Programme (OP): an OP sets out each region's priorities for delivering the funds. On the basis of these funds, the region has to define its priorities: in the field of energy, one example of priority is increasing the use of renewable energy sources. Then, a region should decide which activities to insert in the plan. Activities may be roughly divided into six types: infrastructures and plants; buildings and land use transformations; resource extraction; modifications of hydraulic regime; industrial transformations; environmental management. Also, a magnitude for each activity should be decided describing how much of a given activity is performed.

Each activity has an outcome (such as the amount of energy produced or consumed) and a cost. We have two vectors $\mathbf{O} = (o_1, \ldots, o_{N_a})$ and $\mathbf{C} = (c_1, \ldots, c_{N_a})$ where each element is associated to a specific activity and represents the outcome and cost per unit of an activity.

There are constraints linking activities: for instance if a regional plan decides to build three biomass power plants (primary activities for an energy plan), each of these plants should be equipped with proper infrastructures (streets, sewage or possibly a small village nearby, power lines) also called *secondary activities*. We thus have a matrix of dependencies between activities. In particular, we have a $N_a \times N_a$ square matrix $\mathcal{D}$ where each element $d_{ij}$ represents the magnitude of activity $j$ per unit of activity $i$.

Taking as an example the Emilia-Romagna Regional Energy Plan approved in 2007, some objectives of the policy makers are the production of a given amount of energy (400 additional MW from renewable energy sources), while reducing the current greenhouse gas emission percentage by 6.5% with respect to 2003. In addition, the budget constraint limiting the amount of money allocated to the energy plan by the Regional Operational Programme was 30.5M€ in 2007.

The policy maker also takes into account impacts on the environment, the economy and the society, as defined by a Strategic Environmental Assessment that relates activities defined in the plan to environmental and economic impacts. Each activity has impacts on the environment in terms of positive and negative pressures. An example of positive pressure is the increased availability of energy, while an example of a negative pressure is the production of pollutants. Pressures are further linked to environmental receptors such as the quality of the air or of surface water. On both pressures and receptors, there are constraints: for example the maximum amount of greenhouse gas emissions of the overall plan.

One of the instruments used for assessing a regional plan in Emilia-Romagna are the so called *coaxial matrices* [3], a development of the network method [17].

One matrix $\mathcal{M}$ defines the dependencies between the above mentioned activities *impacts* (also called *pressures*) on the environment. Each element $m^i_j$ of the matrix $\mathcal{M}$ defines a qualitative dependency between the activity $i$ and the impact $j$. The dependency can be *high*, *medium*, *low* or *null*. Examples of negative impacts are energy, water and land consumption, variation of water flows, water and air pollution and so on. Examples of positive impacts are reduction of water/air pollution, reduction of greenhouse gas emission, reduction of noise, natural resources saving, creation of new ecosystems etc.

The second matrix $\mathcal{N}$ defines the dependencies between the impacts and environmental receptors. Each element $n^i_j$ of the matrix $\mathcal{N}$ defines a qualitative dependency between the impact $i$ and an environmental receptor $j$. Again the dependency can be *high*, *medium*, *low* or *null*. Examples of environmental receptors are the quality of surface water and groundwater, the quality of landscapes, energy availability, wildlife wellness and so on.

The matrices used in Emilia-Romagna contain 93 activities, 29 negative impacts, 19 positive impacts and 23 receptors, and assess 11 types of plans.

## 2 Why constraint based approaches

The regional planning activity is now performed by human experts that build a single plan, considering strategic regional objectives that follow national and

EU guidelines. After the plan has been devised, the agency for environmental protection is asked to assess the plan from an environmental point of view. Typically, there is no feedback: the assessment can state that the devised plan is environmentally friendly or not, but it cannot change the plan. In rare cases, it can propose corrective countermeasures, that can only mitigate the negative impact of wrong planning decisions. Moreover, although regulations state that a significant environmental assessment should compare two or more options (different plans), this is rarely done in Europe, because the assessment is typically hand made and requires a long work. Even in the few cases in which two options are considered, usually one is the plan and the other is the absence of a plan.

Constraint based modeling overcomes the limitation of a hand made process for a number of reasons. First, it provides a tool that automatically performs planning decisions, considering both the budget allocated to the plan by the Regional Operative Plan, and national/EU guidelines.

Second, it takes environmental aspects into consideration during plan construction, avoiding trial-and-error schemes.

Third, constraint reasoning provides a powerful tool in the hand of a policy maker as the generation of alternative scenarios is extremely easy and their comparison and evaluation comes for free. Adjustments can be performed on-the-fly in the case that the results do not satisfy policy makers or environmental experts. For example, in the field of energy regional plan, by changing the bounds on the amount of energy each source can provide, we can adjust the plan by considering market trends and also the potential receptivity of the region.

## 3 A CLP model

To design a constraint-based model for the regional planning activity, we have to define variables, constraints and objectives. Variables represent decisions that have to be taken. Given a vector of activities $\mathbf{A} = (a_1, \ldots, a_{N_a})$, we associate to each activity a variable $G_i$ that defines its magnitude. The magnitude could be represented either in an absolute way, as the amount of a given activity, or in a relative way, as a percentage with respect to the existing quantity of the same activity. We use in this paper the absolute representation.

As stated above, we distinguish primary from secondary activities: let $A^P$ be the set of indexes of primary activities and $A^S$ the set of indexes of secondary activities. The distinction is motivated by the fact that some activities are of primary importance in a given plan. Secondary activities are those supporting the primary activities by providing the needed infrastructures. The dependencies between primary and secondary activities are considered by the constraint:

$$\forall j \in A^S \quad G_j = \sum_{i \in A^P} d_{ij} G_i$$

Given a budget $B_{Plan}$ available for a given plan, we have a constraint limiting the overall plan cost as follows

$$\sum_{i=1}^{N_a} G_i \; c_i \leq B_{Plan} \tag{1}$$

Such constraint can be imposed either on the overall plan or on parts of it. For example, if the budget is partitioned into chapters, we can impose constraint (1) on activities of a given chapter.

Moreover, given an expected outcome $o_{Plan}$ of the plan, we have a constraint ensuring to reach the outcome:

$$\sum_{i=1}^{N_a} G_i \; o_i \geq o_{Plan}.$$

For example, in an energy plan the outcome can be to have more energy available in the region, so $o_{Plan}$ could be the increased availability of electrical power (e.g., in kilo-TOE, Tonnes of Oil Equivalent). In such a case, $o_i$ will be the production in kTOE for each unit of activity $a_i$.

Concerning the impacts, we sum up the contributions of all the activities and obtain the estimate of the impact on each environmental pressure:

$$\forall j \in \{1, \ldots, N_p\} \quad p_j = \sum_{i=1}^{N_a} m_j^i \; G_i. \tag{2}$$

The qualitative values in the matrices have been converted into quantitative values $m_j^i$ in the $[0, 1]$ range for positive impacts and in the $[-1, 0]$ range for negative ones. The actual values were suggested by an environmental expert.

In the same way, given the vector of environmental pressures $\mathbf{P} = (p_1, \ldots, p_{N_p})$, one can estimate their influence on the environmental receptor $r_i$ by means of the matrix $\mathcal{N}$, that relates pressures with receptors:

$$\forall j \in \{1, \ldots, N_r\} \quad r_j = \sum_{i=1}^{N_p} n_j^i p_i. \tag{3}$$

Moreover we can have constraints on receptors and pressures. For example, *"Greenhouse gas emission"* (that is a negative pressure) should not exceed a given threshold.

Concerning objectives, there are a number of possibilities suggested by planning experts. From an economic perspective, one can decide to minimize the overall cost of the plan (that is anyway subject to budget constraints). Clearly in this case, the most economic energy sources are preferred, despite their potentially negative environmental effects (which could be anyway constrained). On the other hand, one could maintain a fixed budget and maximize the produced energy. In this case, the most efficient energy sources will be pushed forward.

Or the planner could prefer a *green* plan and optimize environmental receptors. For example, one can maximize, say, the air quality, or the quality of the surface water. In this case, the produced plan decisions are less intuitive and the system we propose is particularly useful. The link between decisions on primary and secondary activities and consequences on the environment are extremely complex to be manually considered. Clearly, more complex objectives can be pursued, by properly combining the above mentioned aspects.

### 3.1 The regional energy plan

We can now describe how to cast the general model for regional planning described above into the model for designing a regional energy plan. The first step is to identify primary and secondary activities. In the context of a regional energy plan, the environmental and planning experts defined the following distinction. Primary activities are those capable of producing energy, namely renewable and non-renewable power plants. Secondary activities are those supporting the energy production, such as activities for energy transportations (e.g., power lines), and infrastructures supporting the primary activities (e.g., dams, yards).

One important aspect to be taken into account when designing a regional energy plan is the energy source diversification: this means that funds should not be directed toward a single energy source, but should cover both renewable and non renewable energy sources. This requirement comes from fluctuations of the price and availability of the various resources. For this reason, we have constraints on the minimal fraction $F_i$ of the total energy produced by each source $i$:

$$\forall i \in A^P \quad G_i o_i \geq F_i T^o$$

where the total outcome $T^o$ is simply obtained as

$$T^o = \sum_{j \in A^P} G_j o_j.$$

In addition, each region has its own geo-physical characteristics. For instance, some regions are particularly windy, while some others are not. Hydroelectric power plants can be built with a very careful consideration of environmental impacts, the most obvious being the flooding of vast areas of land. This poses constraints on the maximum energy $U_i$ that can be produced by a given energy source $i$

$$\forall i \in A^P \quad G_i o_i \leq U_i.$$

Finally, the region priorities should be compliant with European guidelines, such as the 20-20-20 initiative, that aims at achieving three ambitious targets by 2020: reducing by 20% greenhouse gas emissions, having a 20% share of the final energy consumption produced by renewable sources, and improving by 20% its energy efficiency. For this reason, we can impose constraints on the minimum amount of energy $L_{ren}$ produced by renewable energy sources whose set of activities is

referred to as $A^P{}_{ren}$. The constraint that we can impose is

$$\sum_{i \in A^P{}_{ren}} G_i o_i \geq L_{ren}.$$

## 4 The Regional Energy Plan 2011-2013

The constraint-based model described in previous sections has been used in the planning of the regional energy plan for 2011-2013. The system is implemented in the Constraint Logic Programming language ECL$^i$PS$^e$ [1], and in particular uses its Eplex library [16], that interfaces ECL$^i$PS$^e$ with a (mixed-integer) linear programming solver. Nowadays, linear solvers are able to solve problems with millions of variables, while our problem is much smaller (see end of Section 1.1). In fact, the computation time was hardly measurable on a modern computer.

The regional energy plan had the objective of paving the way to reach the ambitious goal of the 20-20-20 directive, in particular having 20% of energy in 2020 produced by renewable sources. This amount does not consider only electric power, but the whole energy balance in the region, including thermal energy, and transports.

Transports can use renewable energy by using renewable fuels, like biogas (methane produced from the fermentation of vegetable or animal wastes) or oil produced from various types of crops. Currently, we do not consider this issue.

Thermal energy can be used e.g. for home heating; renewable sources in this case are thermal solar panels (that produce hot water for domestic use), geothermal pumps (that are used to heat or to refresh houses), biomass plants, that produce hot water used to heat neighboring houses during winter.

The considered electric power plants that produce energy from renewable sources are hydroelectric plants, photovoltaic plants, thermodynamic solar plants, wind generators and, again, biomass power plants.

For each energy source, the plan should provide: the installed power, in MW; the total energy produced in a year, in kTOE (TOE stands for Tonne of Oil Equivalent); the total cost, in M€. The ratio between installed power and total produced energy is mainly influenced by the availability of the source: while a biomass plant can (at least in theory) produce energy 24/7, the sun is available only during the day, and the wind only occasionally. For unreliable sources an average for the whole year is taken.

The cost of the plant, instead, depends mainly on the installed power: a solar plant has an installation cost that depends on the square meters of installed panels, which on their turn can provide some maximum power (peak power).

It is worth noting that the considered cost is the total cost of the plant for the regional system, which is not the same as the cost for the taxpayers of the Emilia-Romagna region. In fact, the region can enforce policies in many ways, convincing private stakeholders to invest in power production. This can be done with financial leverage, or by giving favorable conditions (either economic or other) to investors. Some power sources are economically profitable, so there is

no need for the region to give subsidies. For example, currently in Italy biomasses are economically advantageous for investors, so privates are proposing projects to build biomasses plants. On the other hand, biomasses also produce pollutants, they are not always sustainable (see [4] for a discussion) so local committees are rather likely to spawn a protest against the construction of new plants. For these reasons, there is a limit on the number of licenses the region gives to private stakeholders for building biomass-based plants.

Technicians in the region estimated (considering current energy requirements, growth trends, foreseen energy savings) the total energy requirements for 2020; out of this, 20% should be provided by renewable sources. They also proposed for this amount a percentage to be provided during the plan 2011-2013: about 177kTOE of electrical energy and 296kTOE of thermal energy.

Starting from these data, they developed a plan for electrical energy and one for thermal energy.

We used the model presented in Section 3 considering initially only "extreme" cases, in which only one type of energy source is used. The application provides the optimal plan, together with its environmental assessment, namely an evaluation of the environmental receptors used by the environmental protection agency.

In order to understand the individual contributions of the various energy forms, we plotted all the plans that use a single type of energy in Figure 1, together with the plan developed by the region's experts (we consider here electrical energy sources). On the $x$-axis, we chose the receptor *Air quality* because it is probably the most sensitive receptor in the Emilia-Romagna region. On the $y$-axis we plotted the cost of the plan. As explained previously, all plans provide the same energy in kTOE, while they can require different installation power (in MW).

First of all, we notice that some of the energy types improve the air quality (positive values on the $x$-axis), while others worsen it (negative values). Of course, no power plant can improve the air quality by itself (as it cannot remove pollutants from the air). The point is that the plant provides electrical energy without introducing new pollutants; if such energy would not have been provided to the electrical network, it would have been imported from neighboring regions. In such a case, the required energy would be produced with the same mixture of energy sources as in the national production, including those emitting pollutants, so the net contribution is positive for the air quality. Note also that the different energy sources have different impacts on the air quality not only due to the emissions of the power plants, but also to the impact of the secondary activities required by the various sources.

Finally, the "extreme" plans are usually not feasible, in the sense that the constraint on the real availability of the energy source in the region was relaxed. For example, wind turbines provide a very good air quality at a low cost, but the amount required in the corresponding extreme plan is not possible in the region considering the average availability of wind and of land for installing turbines.

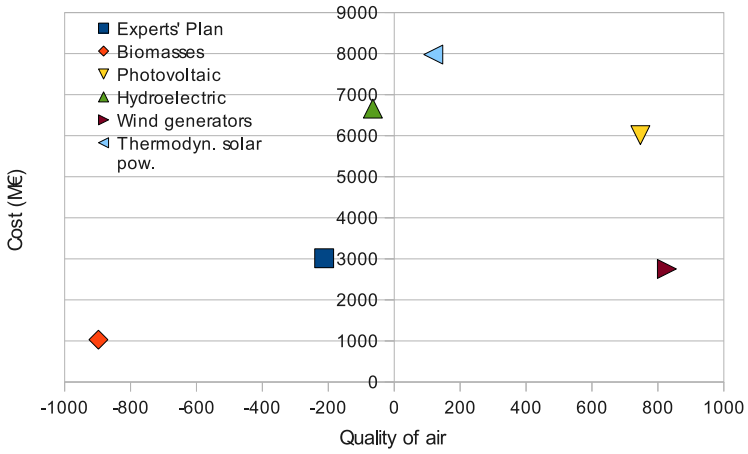



**Fig. 1.** Plot of the *extreme* plans using only one energy source, compared with the plan by the region's experts.

The plan proposed by the region's experts is more *balanced*: it considers the real availability of the energy source in the region, and provides a mixture of all the different renewable types of energy. This is very important in particular for renewable sources, that are often discontinuous: wind power is only available when the wind is blowing at a sufficient speed, solar power is only available during sunny days, etc., so having a mixture of different sources can provide an energy availability more continuous during the day.

Beside assessing the plan proposed by the experts, we also provided new, alternative plans. In particular, we searched for optimal plans, both with respect to the cost, and to the *air quality*. Since we have two objective functions, we plotted the Pareto-optimal frontier: each point of the frontier is a point such that one cannot improve one of the objectives without sacrificing the other. In our case, the air quality cannot be improved without raising the cost, and, vice-versa, it is impossible to reduce the cost without sacrificing the air quality. The Pareto frontier is shown in Figure 2, together with the experts' plan. The objective function is a weighted sum of single criteria so our formulation of the problem is linear and we can compute the Pareto frontier by changing coefficients in the weighted sum.

The picture shows that, although the plan devised by the experts is close to the frontier, it can be improved. In particular, we identified on the frontier two solutions that dominate the experts' plan: one has the same cost, but better air quality, while the other has same air quality, but a lower cost.

Table 1 contains the plan developed by the region's experts, while Table 2 shows the plan on the Pareto curve that has the same air quality as the plan of the experts. The energy produced by wind generators is almost doubled (as they provide a very convenient ratio (air quality)/cost, see Figure 1), we have a

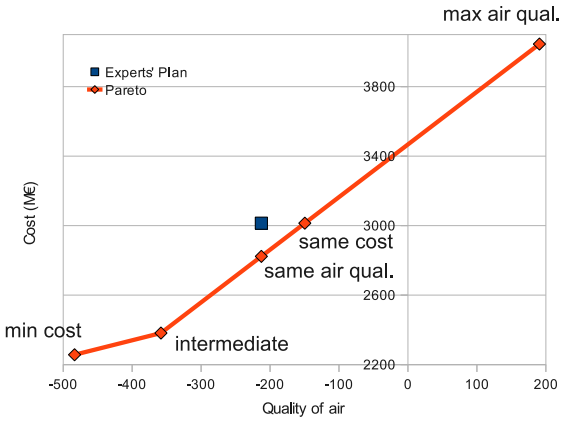



**Fig. 2.** Pareto frontier of the air quality against cost.

| Electrical power plants | Power 2010 (MW) | Power 2013 (MW) | Energy 2013 (kTOE) | Investments (M€) |
|---|---|---|---|---|
| Hydroelectric | 300 | 310 | 69.3 | 84 |
| Photovoltaic | 230 | 850 | 87.7 | 2170 |
| Thermodyn. solar | 0 | 10 | 1 | 45 |
| Wind generators | 20 | 80 | 10.3 | 120 |
| Biomasses | 430 | 600 | 361.2 | 595 |
| Total | 980 | 1850 | 529.5 | 3014 |

**Table 1.** Energy plan developed by the region's experts

slight increase in the cheap biomass energy, while the other energy sources reduce accordingly. Extracting plans from the solution of the CLP model is trivial: we simply extract the values assigned to decision variables by the linear solver.

Concerning the environmental assessment, we plot in Figure 3 the value of the receptors in significant points of the Pareto front. Each bar represents a single environmental receptor for a specific plan on the Pareto frontier of Figure 2. In this way it is easy to compare how receptors are impacted by different plans. In the Figure, the white bar is associated to the plan, on the frontier, that has the highest air quality, while bars with dark colors are associated to plans that have a low cost (and, thus, a low quality of the air). Notice that the receptors have different trends: some of them improve as we move in the frontier towards higher air quality (like *climate quality*, *mankind wellness*, *value of material goods*), while others improve when moving to less expensive solutions (like *quality of sensitive landscapes*, *wellness of wildlife*, *soil quality*). This is due to several reasons, depending both on the type of power plants installed and on the secondary activities.

| Electrical power plants | Power 2010 (MW) | Power 2013 (MW) | Energy 2013 (kTOE) | Investments (M€) |
|---|---|---|---|---|
| Hydroelectric | 300 | 303 | 67.74 | 25.2 |
| Photovoltaic | 230 | 782.14 | 80.7 | 1932.51 |
| Thermodyn. solar | 0 | 5 | 0.5 | 22.5 |
| Wind generators | 20 | 140 | 18.03 | 240 |
| Biomasses | 430 | 602.23 | 362.54 | 602.8 |
| Total | 980 | 1832.37 | 529.5 | 2823 |

**Table 2.** Energy plan that dominates the experts' plan, retaining same air quality but with lower cost

## 5 Added value of CLP

The application (including both the assessment and the planning) was developed in few person-months by a CLP expert. It does not have a graphical user interface yet, and it is currently usable only by CLP experts; however it produces spreadsheet files with tables having the same structure as those used for years by the region's experts, so the output is easily understandable by the end user. We are currently developing a web-based application, to let users input the relevant data, and try themselves producing plans on-the-fly.

The assessment module [8] was first tested on a previously developed plan, then used during the planning of the 2011-2013 regional energy plan. The various alternatives have been submitted to the regional council, that could choose one of them, instead of accepting/rejecting the only proposal, as in previous years.

One of the results is the ability to generate easily alternative plans with their assessment; this is required by the EU regulations, but it is widely disregarded.

Another result is the possibility to provide plans that are optimal; the optimization criteria can include the cost, or one of the various environmental receptors. The user can select two objectives, and in this case the application generates a Pareto front. This helps the experts or the regional council in doing choices that are more grounded.

We still do not know which plan the regional council will choose, neither we know if and how the directives given in the regional plan will be indeed implemented. More refined plans (at the province or municipality level) should follow the guidelines in the regional plan, but it is also possible to introduce modifications during the plan execution. However, in a perfect world, in which everything is implemented as expected, the added value of CLP in monetary terms could be the difference of the *investment* columns in the plans in Tables 1 and 2: 191M€ saved (by the various actors, public and private, in the whole region) in three years.

Finally, the choice of Constraint Programming greatly enables model flexibility. Discussing with experts, it is often the case that they change their mind on some model constraints or on objectives. Therefore, the flexibility in dealing with side constraints and in dealing with non linear constraints facilitates

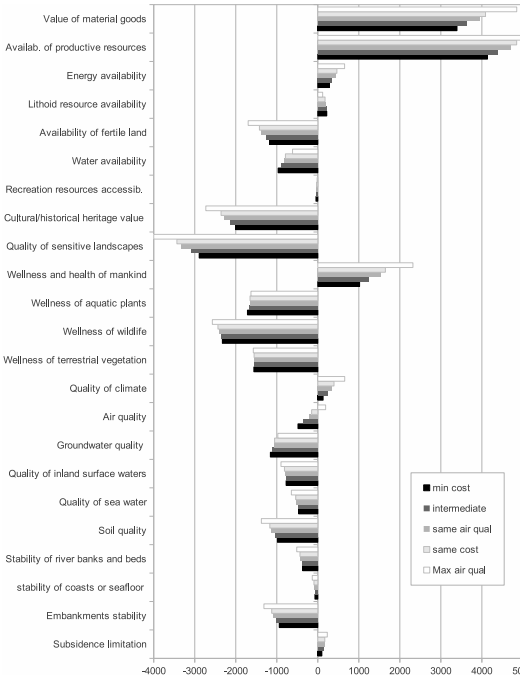



**Fig. 3.** Value of the receptors on the Pareto front

knowledge acquisition making Constraint Programming the technique of choice for the problem and its future extensions.

## 6 Conclusion and Future Open Issues

Global public policy making is a complex process that is influenced by many factors. We believe that the use of constraint reasoning techniques could greatly increase the effectiveness of the process, by enabling the policy maker to analyze various aspects and to play with parameters so as to obtain alternative solutions along with their environmental assessment. Given the amount of financial, human and environmental resources that are involved in regional plans, even a small improvement can have a huge effect.

Important features of the system are: its efficiency, as a plan is returned in few milliseconds, its wide applicability to many regional plans, to provincial and urban plans and also to private and public projects. The system was used for the environmental assessment of the regional energy plan of the Emilia-Romagna region of Italy. Beside performing automatically the assessment (that was performed by hand in previous years), the assessment for the first time includes the evaluation of alternative plans: this is a requirement of EU regulations that is largely disregarded in practice. Moreover, the alternative plans were produced

by optimizing the quality of the environmental receptors, together with the cost for the community of the plan itself.

This work is a first step towards a system that fully supports the decision maker in designing public policies. To achieve this objective, the method must be extended to take into account the individual level, by investigating the effect of a policy over the parties affected by it. This can be achieved by integrating constraint reasoning with simulation models that reproduce the interactions among the parties. In our current research, we are studying how the region can choose the form of incentives and calibrate them in order to push the energy market to invest in the directions foreseen by the Regional Energy plan [7].

In turn these models can be enriched by adopting e-Participation tools that allow citizens and stakeholders to voice their concerns regarding policy decisions. To fully leverage e-Participation tools, the system must also be able to extract information from all the available data, including natural language. Thus, opinion mining techniques will be useful in this context.

At the moment the system can only be used by IT expert people. In order to turn it into a practical tool that is routinely used by decision makers, we must equip it with a user-friendly interface. In particular, we are in the process of developing a web interface to the constraint solver, in order to make it easy to use and widely accessible.

Finally, economic indicators will be used to assess the economic aspect of the plan. Up to now, only budget and few economic pressures and receptors are considered. We believe that a comprehensive system should fully incorporate this aspect. We will integrate a well established approach (UN and Eurostat Guidelines) and robust data from official statistics into the system to combine economic accounts (measured in monetary terms) and environmental accounts (measured in physical units) into a single framework useful for the evaluation of the integrated economic, environmental-social performance of regions. If the relations with economic indicators are linear, then the solution time should not increase significantly.

## 7 Acknowledgements

This work was partially supported by EU project *ePolicy*, FP7-ICT-2011-7, grant agreement 288147. Possible inaccuracies of information are under the responsibility of the project team. The text reflects solely the views of its authors. The European Commission is not liable for any use that may be made of the information contained in this paper.

## References

1. Krzysztof R. Apt and Mark Wallace. *Constraint logic programming using Eclipse*. Cambridge University Press, 2007.
2. Maurizio Bruglieri and Leo Liberti. Optimal running and planning of a biomass-based energy production process. *Energy Policy*, 36(7):2430–2438, July 2008.

3. Paolo Cagnoli. *VAS valutazione ambientale strategica*. Dario Flaccovio, 2010.
4. Massimiliano Cattafi, Marco Gavanelli, Michela Milano, and Paolo Cagnoli. Sustainable biomass power plant location in the Italian Emilia-Romagna region. *ACM Transactions on Intelligent Systems and Technology*, 2(4), July 2011.
5. Damiana Chinese and Antonella Meneghetti. Design of forest biofuel supply chains. *International Journal of Logistics Systems and Management*, 5(5):525–550, 2009.
6. Davide Freppaz, Riccardo Minciardi, Michela Robba, Mauro Rovatti, Roberto Sacile, and Angela Taramasso. Optimizing forest biomass exploitation for energy supply at regional level. *Biomass and Bioenergy*, 26:15–24, 2003.
7. Marco Gavanelli, Michela Milano, Alan Holland, and Barry O'Sullivan. What-if analysis through simulation-optimization hybrids. In K. G. Troitzsch, M. Möhring, and U. Lotzmann, editors, *Proceedings* $26^{th}$ *European Conference on Modelling and Simulation ECMS 2012*, pages 624–630. Digitaldruck Pirrot GmbH, 2012.
8. Marco Gavanelli, Fabrizio Riguzzi, Michela Milano, and Paolo Cagnoli. Logic-Based Decision Support for Strategic Environmental Assessment. *Theory and Practice of Logic Programming*, 10(4-6):643–658, July 2010.
9. Marco Gavanelli, Fabrizio Riguzzi, Michela Milano, and Paolo Cagnoli. Constraint and optimization techniques for supporting policy making. In Ting Yu, Nitesh Chawla, and Simeon Simoff, editors, *Computational Intelligent Data Analysis for Sustainable Development*, Data Mining and Knowledge Discovery Series, chapter 12. Chapman & Hall/CRC, 2013.
10. Marco Gavanelli, Fabrizio Riguzzi, Michela Milano, Davide Sottara, Alessandro Cangini, and Paolo Cagnoli. An application of fuzzy logic to strategic environmental assessment. In R. Pirrone and F. Sorbello, editors, *XIIth International Conference of the Italian Association for Artificial Intelligence*, volume 6934 of *Lecture Notes in Computer Science*, pages 324–335. Springer, 2011.
11. Nigel Gilbert. *Computational Social Science*. SAGE, 2010.
12. S. A. Grady, M. Y. Hussaini, and Makola M. Abdullah. Placement of wind turbines using genetic algorithms. *Renewable Energy*, 30(2):259 – 270, 2005.
13. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
14. Robin Matthews, Nigel Gilbert, Alan Roach, Gary Polhill, and Nick Gotts. Agent-based land-use models: a review of applications. *Landscape Ecology*, 22(10), 2007.
15. B. Sadler, R. Aschemann, J. Dusik, T. Fischer, M. Partidario, and R. Verheem, editors. *Handbook of Strategic Environmental Assessment*. Earthscan, 2010.
16. Kish Shen and Joachim Schimpf. Eplex: Harnessing mathematical programming solvers for constraint logic programming. In P. van Beek, editor, *CP 2005*, volume 3709 of *LNCS*. Springer-Verlag, 2005.
17. Jens C. Sorensen and Mitchell L. Moss. Procedures and programs to assist in the impact statement process. Technical report, Univ. of California, Berkely, 1973.
18. Klaus G. Troitzsch, Ulrich Mueller, Nigel Gilbert, and Jim Doran. Social science microsimulation. *J. Artificial Societies and Social Simulation*, 2(1), 1999.
19. Joost Vennekens, Sofie Verbaeten, and Maurice Bruynooghe. Logic programs with annotated disjunctions. In B. Demoen and V. Lifschitz, editors, *International Conference on Logic Programming*, volume 3131 of *LNCS*. Springer, 2004.

# Nondeterministic Programming in Java with JSetL

Gianfranco Rossi and Federico Bergenti

Dipartimento di Matematica e Informatica, Università di Parma
{gianfranco.rossi | federico.bergenti}@unipr.it

**Abstract.** JSetL is a Java library that endows Java with a number of facilities that are intended to support declarative and constraint (logic) programming. In this paper we show how JSetL can be used to support general forms of nondeterministic programming in an object-oriented framework. This is obtained by combining different but related facilities such as logical variables, set data structures, unification, along with a constraint solver that allows the user to solve nondeterministic constraints, as well as to define new constraints using the nondeterminism handling facilities provided by the solver itself. Thus, the user can define her/his own general nondeterministic procedures as new constraints, letting the constraint solver handle them. The proposed solutions are illustrated by showing a number of concrete Java implementations using JSetL, including the implementation of simple Definite Clause Grammars.

**Keywords**. Nondeterministic programming, Constraint Programming, Set-based Programming, Java language.

## 1 Introduction

The problem of incorporating constructs to support nondeterminism into programming languages has been discussed at length in the past. Early references to this topic are [4] for a general overview, and [11] for an analysis of the problem in the context of functional programming languages. Logic programming languages, notably Prolog, strongly rely on nondeterminism. Their computational model is inherently nondeterministic (at each computation step, one of the clauses unifying a given goal is selected nondeterministically) and the programmer can exploit and control nondeterminism using the language features when defining her/his own procedures.

As regards imperative programming, however, only relatively few languages provide primitive constructs to support nondeterminism. An early example is SETL [10], a Pascal-like language endowed with sets, which provides, among others, a few built-in features to support backtracking (e.g., the `ok` and `fail` primitives). More recently, the programming language Alma-0 [1] [2] provides a comprehensive collection of primitive constructs to support nondeterministic

programming, such as the statements `orelse`, `some`, `forall`, `commit`, for creating choice points and handling backtracking. Also Python's `yield` mechanism—and, more generally, the coroutining mechanisms present in various programming languages—can be used as a way to explore the computation tree associated with a nondeterministic program.

Our goal in this paper is to explore the feasibility of a *library-based* approach to support nondeterministic programming in an object-oriented language. Specifically, our proposal is to exploit the nondeterministic constraint solver provided by JSetL [9], a Java library that combines the object-oriented programming paradigm of Java with valuable concepts of CLP languages, such as logical variables, partially specified list data structures, unification, constraint solving. Using this library the programmer can define nondeterministic procedures by exploiting either the nondeterminism embedded in the predefined constraints (in particular, in set constraints), or the possibility to define new user-defined nondeterministic constraints and handle them through the built-in constraint solver. We will illustrate our solution with a number of simple examples using Java with JSetL.

The paper is organized as follows. In Section 2 we show how JSetL can support nondeterminism through the use of built-in nondeterministic features, such as set constraints and the labeling mechanism. Section 3 briefly introduces general nondeterministic control structures and the relevant language constructs. In Section 4 we show how different nondeterministic control structures can be implemented in Java using the facilities for defining new constraints provided by JSetL. Finally, in Section 5 we show a more complete example of application of the facilities offered by JSetL to support nondeterministic control structures: the implementation of Definite Clause Grammars.

## 2 Embedded Nondeterminism

A convenient way to express nondeterminism in JSetL is by means of *set constraints*. As a matter of fact, nondeterminism is strongly related to the notion of set and set operations (see, e.g., [13] and [7]).

Sets can be defined in JSetL as instances of the class `LSet`. Elements of a `LSet` object can be of any type, including other `LSet` objects (i.e., *nested* sets are allowed). Moreover, sets denoted by `LSet` (also referred to as *logical sets*) can be *partially specified*, i.e., they can contain unknown elements, as well as an unknown part [3]. Single unknown elements are represented by unbound *logical variables* (i.e., uninitialized objects of the class `LVar`), whereas the unknown part of the set is represented by an unbound logical set (i.e., an uninitialized object of the class `LSet`). For example, the three statements:

```
LVar x = new LVar();
LSet r = new LSet();
LSet s = r.ins(x);
```

create an unbound logical variable `x`, an unbound logical set `r`, and a partially specified logical set `s` with an unknown element `x` and an unknown rest `r` (i.e., {`x` | `r`}, using a Prolog-like notation).

JSetL provides the basic operations on this kind of sets, such as equality (viz., *set unification* [6]), inequality, membership, cardinality, union, etc., in the form of primitive constraints, similarly to what provided by the Constraint Logic Programming language CLP($\mathcal{SET}$) [5]. A JSetL *constraint solver* is an instance of the class `SolverClass`. Basically, it provides methods for adding constraints to its *constraint store* (e.g., the method `add`) and to prove satisfiability of a given constraint (methods `check` and `solve`). If `solver` is a solver, $\Gamma$ is the constraint stored in its constraint store (possibly empty), and `c` is a constraint, then `solver.check(c)` returns `false` if and only if $\Gamma \wedge$ `c` is unsatisfiable.

Solving equalities, as well as other basic set-theoretical operations, over partially specified sets may involve nondeterminism. For example, the equation $\{x, y\} = \{1, 2\}$, where $x$ and $y$ are unbound logical variables, admits two distinct solutions: $x = 1 \wedge y = 2$ and $x = 2 \wedge y = 1$. In JSetL, these solutions are computed nondeterministically by the constraint solver, using choice points and backtracking.

In the following example we exploit the nondeterminism embedded in set operations to provide a nondeterministic solution to the problem of printing all permutations of a set of integer numbers $s$. The problem can be modelled as the problem of unifying a (partially specified) set of $n = |s|$ logical variables $\{x_1, \ldots, x_n\}$ with the set $s$, i.e., $\{x_1, \ldots, x_n\} = s$. Each solution to this problem yields an assignment of (distinct) values to variables $x_1, \ldots, x_n$ that represents a possible permutation of the integers in $s$.

*Example 1.* (Permutations)

```
public static void allPermutations(LSet s) {
    int n = s.getSize();                // the cardinality of s
    LSet r = LSet.mkLSet(n);            // r = {x1,x2,...,xn}
    solver.check(r.eq(s));              // r = s
    do {
        r.printElems(' ');
        System.out.println();
    } while (solver.nextSolution());
}
```

The invocation `LSet.mkLSet(n)` creates a set consisting of `n` unbound logical variables. This set is unified, through the constraint `eq`, with the set of `n` integers `s`. This is done by invoking the method `check` of the current constraint solver `solver` (`solver` is assumed to be created outside the method `allPermutations`). The invocation `check(r.eq(s))` causes a viable assignment of values from `s` to variables in `r` to be computed. Values in `r` are then printed on the standard output by calling the method `printElems`.

Calling the method `nextSolution` allows checking whether the current constraint admits further solutions and possibly computing the next one. This

method exploits the backtracking mechanism embedded in the constraint solver: calling `nextSolution` forces the computation to go back until the nearest open choice point is encountered. Specifically, in the above example, solving `r.eq(s)` nondeterministically computes a solution to the set unification problem involving the two sets `r` and `s`. Thus, all possible rearrangements of the values in the given sets (i.e., all possible permutations) are computed and printed, one at a time.

The example illustrates also how the nondeterminism of the JSetL solver interacts with the usual features of the underlying imperative Java language.

Note that in this example nondeterminism is implemented simply by operations on sets and the nondeterministic search is completely embedded in the constraint solver. Since the semantics of set operations is usually well understood and quite intuitive, making nondeterministic programming the same as programming with sets can contribute to make the (non-trivial) notion of nondeterminism easier to understand and to use.

Whenever the problem at hand can be formulated as a Constraint Satisfaction Problem (CSP) over Finite Domains, solutions can be computed nondeterministically by exploiting the so-called *labeling* mechanism. Values to be assigned to variables of the CSP are picked up nondeterministically from the variable domains: if the selected assignment turns out to be not suitable, another alternative is then explored.

In JSetL this is obtained by using the constraint `label`. Solving the constraint `s.label()`, where `s` is a collection of logical variables, forces the program to nondeterministically generate an admissible assignment of values to variables in `s`, starting from the first variable in `s` and the first value in its domain (default labeling strategy in JSetL). This assignment is propagated to all the constraints in the constraint store and if none of them turns out to be unsatisfiable, then an assignment for the next variable in `s` is computed and propagated, and so on. As soon as a constraint in the store turns out to be unsatisfiable, backtracking occurs and a new assignment for the lastly assigned variable is computed. If a viable assignment for all the variables in `s` is finally found, then it represents a solution for the given CSP.

For example, the well-known *n-queens* problem, very often used as a sample problem for illustrating nondeterministic programming, can be easily modelled as a CSP and solved using constraints over Finite Domains and a final labeling phase to nondeterministically generate all possible solutions.

Unfortunately, not all problems whose solutions are naturally formulated as nondeterministic algorithms are also easily modelled as CSP. There are situations in which, in particular, the variable domains are difficult to bring under those supported by existing CP solvers, making the programming effort to model them in terms of the existing ones too cumbersome and sometimes quite *ad hoc*. On the other hand, the use of sets and set operations to model nondeterministic computations, as shown in this section, is not always feasible and/or convenient.

In conclusion, there are cases in which some more general programming abstractions to express and handle nondeterminism are required. We address this problem in the next sections.

## 3 Nondeterministic Control Structures

Dealing with general nondeterministic control structures requires primarily the ability to express and handle *choice points* and *backtracking*. This implies, first of all, that the notion of program computation is extended to allow distinguishing between computations that terminate with success and computations that terminate with failure. Basically, a computation *fails* whenever it executes, either implicitly or explicitly, a `fail` statement. Conversely, a finite, error-free computation *succeeds* if it does not fail. In response to a failure, the computation backtracks to the last open choice point. Choice points may be created by the programmer using suitable language constructs, such as the following `orelse` statement (borrowed from [1]):

$$\texttt{either } S_1 \texttt{ orelse } S_2 \ldots \texttt{orelse } S_n \texttt{ end}$$

which expresses a nondeterministic choice from among $n$ statements $S_1 \ldots S_n$. More precisely, the computation of the `orelse` goes as follows: statement $S_1$ is executed first; if, at some point of the computation (possibly beyond the end of the `orelse` statement) a failure occurs, then backtracking takes place and the computation resumes with $S_2$ in the state it had when entering $S_1$; if a new failure occurs, then the computation backtracks and it resumes with $S_3$, and so on; if a failure occurs after executing $S_n$ and no other open choice points do exist, then the computation definitively fails.

Let us briefly illustrate how to deal with general nondeterministic control structures with a simple example written using a C-like pseudo-language endowed with the `orelse` statement and a few other facilities to support nondeterministic programming. In the next section we will show how the same control structures can be implemented in Java with JSetL.

Given a list $l$ of strings, split (all) the elements of $l$ into two lists $l1$ and $l2$, such that the total length of the strings in $l1$ is equal to the total length of the strings in $l2$. For example, if $l$ is `["I","you","she","we","you","they"]` a possible splitting of $l$ is $l1$ = `["I","they","you"]` (total length = 8) and $l2$ = `["she","we","you"]` (total length = 8), whereas if `"she"` is replaced by `"he"` no splitting is feasible. Note that we are assuming that $l$ can contain repeated elements and that strings can be picked up from $l$ in any order. The problem can be solved by defining a function `split` that nondeterministically splits $l$ into two lists $l1$ and $l2$, and then forcing `split` to generate (via backtracking) all possible pairs of lists $l1$ and $l2$ until a pair respecting the given condition is found. An implementation of this algorithm written in pseudo-code using the `orelse` construct is shown in Example 2.

*Example 2.* (List splitting—in pseudo-code)

```
split(l):
   either
      l is [];
      return ⟨[],[]⟩;
   orelse
      x is the first element and r the rest of l;
      ⟨r1, l2⟩ = split(r);
      return ⟨[x | r1], l2⟩;
   orelse
      x is the first element and r the rest of l;
      ⟨l1, r2⟩ = split(r);
      return ⟨l1, [x | r2]⟩;
   end;
```

where $[x \mid r1]$ (resp., $[x \mid r2]$) represents the list which is obtained by adding $x$ as the first element to the list $r1$ (resp., $r2$). Therefore, if $l$ is not the empty list, its first element is nondeterministically added to either the first sublist (second `orelse` alternative) or to the second sublist (third `orelse` alternative). If `sumLength(`$l$`)` is a function that computes the total length of all the strings in the list $l$, then the given problem is simply solved by calling `split(`$l$`)` and then requiring that the results of `sumLength(`$l1$`)` and `sumLength(`$l2$`)` are equal, that is:

```
⟨l1, l2⟩ = split(l);
sumLength(l1) == sumLength(l2);
```

Note that we are assuming that, in our pseudo-language, whenever an expression $e$ is used as a statement, such as, for instance, `sumLength(`$l1$`) == sumLength(`$l2$`)` or $l$ *is* `[]`, the following operational semantics is enforced: if $e$ evaluates to `true` then continue; else fail.[1] Therefore, if the pair $\langle l1, l2\rangle$ computed by `split(`$l$`)` does not satisfy the condition `sumLength(`$l1$`) == sumLength(`$l2$`)`, then the computation backtracks to `split` and tries another open alternative created by the `orelse` statement, as long as at least one such alternative does exist.

This example shows also the typical interleaving between nondeterminism and recursion: each recursive call to `split` opens three branches in the nondeterministic computation of `split`. Executing the first `orelse` alternative, which represents the base of the recursion, corresponds to reaching a leaf in the computation tree, i.e. a possible solution.

The domain of discourse, in this example, is that of lists of strings. Moreover we do not make any assumption on the length of the lists, on the presence of duplicated elements in them, and on the length of the strings composing them. Trying to encode this domain in terms of the usual constraint domains and trying to restate the problem as a CSP, e.g. over the domain of integer numbers, though feasible in principle, may lead to rather involved programs in practice. On the other hand, trying to restate that problem as a set-theoretical one, in order to

[1] This goes much like the "boolean expressions as statement" feature of Alma-0 [1].

exploit the nondeterminism involved in set constraints as shown in Section 2, may be hindered, at least, by the presence of duplicates in the input sequence.

As mentioned in Section 1, very few programming languages support the above mentioned nondeterministic constructs as *primitive* features. Extending the language with *primitive* constructs that offer such support is, indeed, quite demanding in general. Our goal in this paper is to explore the feasibility of a *library-based* approach, where features to support nondeterministic programming are implemented on the top of an high-level language, namely Java, by exploiting the language abstraction mechanisms, without requiring any modification to the language itself.

We will illustrate this solution in the next sections with a number of simple examples, using the Java library JSetL.

## 4 Implementing Nondeterministic Control Structures in JSetL

As shown in Section 2, JSetL embeds nondeterminism at various levels. In particular, set constraints, as well as the labeling mechanism, are inherently nondeterministic. Availability of built-in nondeterministic constraints, however, is not sufficient to ensure the general kind of nondeterminism we would like to have.

The solution that we propose in order to circumvent such difficulties is based on the availability in JSetL of a nondeterministic constraint solver and the possibility for the programmer to define her/his own (nondeterministic) constraints. Those methods that require the use of nondeterminism are defined as new user-defined constraints. Within these methods the programmer can exploit facilities offered by JSetL for creating and handling choice-points. When solving these constraints the solver will explore the different alternatives using backtracking.

User-defined constraints in JSetL are defined as part of a user class that extends the abstract class `NewConstraintsClass`. The actual implementation of user-defined constraints requires some programming conventions to be respected, as shown in the following example.

*Example 3.* (Implementing new constraints) Define a class `MyOps` which offers two new constraints `c1(o1,o2)` and `c2(o3)`, where `o1`, `o2`, `o3` are objects of type `t1`, `t2`, and `t3`, respectively.

```
public class MyOps extends NewConstraintsClass {
   public MyOps(SolverClass currentSolver) {
       super(currentSolver);
   }
   public Constraint c1(t1 o1, t2 o2) {
           return new Constraint("c1", o1, o2);
   }
   public Constraint c2(t3 o3) {
           return new Constraint("c2", o3);
```

```
    }
    protected void user_code(Constraint c)
    throws Failure, NotDefConstraintException {
        if (c.getName().equals("c1")) c1(c);
        else if(c.getName().equals("c2")) c2(c);
        else throw new NotDefConstraintException();
    }
    private void c1(Constraint c) {
        t1 x = (t1)c.getArg(1);
        t2 y = (t2)c.getArg(2);
        //implementation of constraint c1 over objects x and y
    }
    private void c2(Constraint c) {
        t3 x = (t3)c.getArg(1);
        //implementation of constraint c2 over object x
    }
}
```

The one-argument constructor of the class `MyOps` initializes the field `solver` of the super class `NewConstraintsClass` with (a reference to) the solver currently in use by the user program.

The other public methods simply construct and return new objects of class `Constraint`. This class implements the *atomic constraint* data type. All built-in constraint methods implemented by JSetL (e.g., `eq`, `neq`, `in`, etc.) return an object of class `Constraint`. Each different constraint is identified by a string name (e.g., `"c1"`), which can be specified as a parameter of the constraint constructor.

The method `user_code`, which is defined as abstract in `NewConstraintsClass`, implements a "dispatcher" that associates each constraint name with the corresponding user-defined constraint method. It will be called by the solver during constraint solving.

Finally, the private methods, such as `c1` and `c2`, provide the implementation of the new constraints. These methods must, first of all, retrieve the constraint arguments, whose number and type depend on the constraint itself. We will show possible implementations of such methods (using nondeterminism) in Examples 4 and 6.

Once objects of the class containing user-defined constraints have been created, one can use these constraints in the same way as the built-in ones: user-defined constraints can be added to the constraint store using the method `add` and solved using the `SolverClass` methods for constraint solving. For example, executing the statements

```
MyOps myOps = new MyOps(solver);
solver.solve(myOps.c1(o1,o2));
```

first creates an object of type `MyOps`, called `myOps`, then it creates the constraint `"c1"` over two objects `o1` and `o2` by calling `myOps.c1(o1,o2)`, finally it adds this constraint to the constraint store and solves it by calling the method `solve`

of `solver`. Solving the constraint `"c1"` will cause the solver to call the concrete implementation of the method `user_code` provided by `myOps`, and consequently to execute its method `c1`.[2]

User-defined constraints in JSetL can implement nondeterministic procedures by exploiting special features offered by the JSetL constraint solver. Defining nondeterministic constraints in JSetL, however, requires some additional considerations to be taken into account.

First of all note that methods defining user-defined constraints must necessarily return an object of type `Constraint`. Thus, any other result possibly computed by the method must be returned through parameters. The use of unbound *logical objects*, i.e., logical variables as well as logical sets and lists, as arguments of the user-defined constraints provides a simple solution to this problem.

More generally, the use of logical objects is fundamental in JSetL when dealing with nondeterminism. As a matter of fact, if an object is involved in a nondeterministic computation then it is necessary to restore the status it had before the last choice point whenever the computation backtracks and tries a different alternative. In JSetL this is obtained by allowing the solver to automatically save and restore the global status of all *logical objects* involved in the computation. Since a logical object is characterized by the fact that its value, if any, can not be changed through side-effects, saving and restoring the status of logical objects is a relatively simple task for the solver. Hence, we will always use logical objects, in particular logical variables, for all those objects that are involved in nondeterministic computations.

As a consequence of this choice we can not manipulate logical objects by using the usual imperative statements (e.g., the assignment), but we will always need to use constraints to deal with them. In particular, the equality constraint $l$`.eq(`$v$`)` is used to *unify* a logical variable $l$ with a value $v$. If $l$ is unbound, this simply amounts to binding $l$ to $v$. Once assigned, however, the value $v$ is no longer modifiable.

As an example let us consider the implementation of the nondeterministic function `split(`$l$`)` shown in Example 2. This function can be implemented in JSetL by a user-defined constraint `split(l,l1,l2)`. Note that, here we are replacing a function with the corresponding *relation*: in fact, `split(l,l1,l2)` defines a ternary relation whose elements are those triples $\langle$`l,l1,l2`$\rangle$, with `l`, `l1` and `l2` belonging to the domain of lists, such that all elements of `l` are split into two lists `l1` and `l2`.

As noted above, variables dealt with by nondeterministic constraints are conveniently represented in JSetL as logical objects. Thus, we represent the lists `l`, `l1`, and `l2` of the constraint `split` as JSetL's logical lists (i.e., objects of the class `LList`) and we manipulate them through constraints over lists.

[2] Note that the constructor of the super class `NewConstraintsClass`, which is invoked when `myOps` is created, provides for storing (a reference to) itself within the specified solver, so making the latter able to invoke the method `user_code` of the created object `myOps`.

*Example 4.* (`split` in JSetL) Define a constraint `split(l,l1,l2)` which is true if all elements of the list `l` are split into two lists `l1` and `l2`.

```
public Constraint split(LList l,LList l1,LList l2) {
   return new Constraint("split",l,l1,l2);
}
private void split(Constraint c) throws Failure {
   LList l = (LList)c.getArg(1);
   LList l1 = (LList)c.getArg(2);
   LList l2 = (LList)c.getArg(3);
   switch(c.getAlternative()) {
   case 0:
      solver.addChoicePoint(c);
      solver.add(l.eq(LList.empty()));     // l is []
      solver.add(l1.eq(LList.empty()));    // l1 is []
      solver.add(l2.eq(LList.empty()));    // l2 is []
      break;
   case 1: {
      solver.addChoicePoint(c);
      LVar  x = new LVar();
      LList r = new LList();
      LList r1 = new LList();
      solver.add(l.eq(r.ins(x)));   // 1st element (x) and rest (r) of l
      solver.add(split(r,r1,l2));   // split r into two lists r1 and l2
      solver.add(l1.eq(r1.ins(x))); // l1 is [x|r1]
      break; }
   case 2: {
      LVar  x = new LVar();
      LList r = new LList();
      LList r2 = new LList();
      solver.add(l.eq(r.ins(x)));   // 1st element (x) and rest (r) of l
      solver.add(split(r,l1,r2));   // split r into two lists l1 and r2
      solver.add(l2.eq(r2.ins(x))); // l2 is [x|r2]
     break; }
    }
}
```

`split` implements the nondeterministic construct `orelse` by using the methods `getAlternative` and `addChoicePoint`. The invocation `c.getAlternative()` returns an integer associated with the constraint `c` that can be used to count the nondeterministic alternatives within this constraint. Its initial value is 0. Each time the constraint `c` is re-considered due to backtracking, the value returned by `c.getAlternative()` is automatically incremented by 1. The invocation `solver.addChoicePoint(c)` adds a choice point to the alternative stack of the current solver. This allows the solver to backtrack and re-consider the constraint `c` if a failure occurs subsequently.

Note that the JSetL implementation of the method `split` closely resembles the abstract definition in pseudo-code of the function `split` given in Section 3. In particular, lists are implemented as `LList` objects, and the addition and extraction of elements from such lists is performed through the JSetL constraint `eq`.

Specifically, `l.eq(r.ins(x))` is true if `l` is the list composed by an element `x` and a remaining part `r`. If `l` is known and `x` and `r` are not, solving `l.eq(r.ins(x))` amounts to computing the first element `x` and the rest `r` of the list `l`, whereas if `x` and `r` are known and `l` is not, `l.eq(r.ins(x))` can be used to build the list `l` out of its parts `x` and `r`.

Finally, note that the recursive call to `split(`$r$`)` in the abstract definition of the function `split` is replaced by the (recursive) posting of the constraint `split(r,r1,l2)` (or `split(r,l1,r2)`) in the above concrete implementation.

As a sample use of `split`, if `l` is the `LList` with value `["I","you","she","we","you","they"]`, `sumLength(l,n)` is a user-defined (deterministic) constraint that implements the function `sumLength(`$l$`)` of Example 2, `listOps` is an instance of the class that extends `NewConstraintsClass` containing `split` and `sumLength`, and `l1`, `l2` are unbound logical lists and `n`, `m` are unbound logical variables, then executing the following fragment of code

```
solver.add(listOps.split(l,l1,l2));
solver.add(listOps.sumLength(l1,n));
solver.add(listOps.sumLength(l2,m));
solver.check(m.eq(n));
```

will bind `l1` to `["you","I","they"]`, `l2` to `["she","you","we"]`, and `n` and `m` to 8.

*Remark 1.* The use of relations in place of functions, along with the use in their implementation of a number of specific features provided by JSetL, have another important consequence on the usability of user-defined constraint methods.

Let us consider a function $y = f(x)$ and a possible call to it, $z = f(a)$. In JSetL one can define a constraint $c_f(x, y)$ which represents the relation $R_f = \{\langle x, y\rangle : y = f(x)\}$ and then solve the constraint $c_f(a, z)$. Solving this constraint actually amounts to checking whether $\langle a, z\rangle \in R_f$, for some $z$. While calling $f(x)$ to compute $y$ implies assuming $x$ to be the input parameter and $y$ the output, solving $c_f(x, y)$ does not make any assumption on the "direction" of their parameters. Thus, one can compute $y$ out of a given $x$, or, vice versa, $x$ out of a given $y$, or one can test whether the relation among two given values $x$ and $y$ holds, or one can compute any of the pairs $\langle x, y\rangle$ belonging to $R_f$. Hence, the same method can be used to implement different, though related, functions.[3]

This general use of user-defined constraints in JSetL is made possible thanks to the availability of a number of different facilities to be used in the constraint implementation. Specifically,

- the use of logical variables as parameters
- the use of unification in place of equality and assignment
- the use of nondeterminism to compute multiple solutions for the same constraint.

[3] It worth emphasizing here the similarity with Prolog or, more to the point, with Prolog-based CP languages.

Note that the fact that a logical variable acts as an input or as an output parameter depends on the fact it is bound or not when the method is called. In particular, unbound variables can be easily used to obtain output parameters.

Moreover, if the value bound to a variable is a partially specified aggregate, e.g. a logical list, then it can act simultaneously as input and as output, i.e. as an input-output parameter. For example, let us consider the fragment of code shown at the end of Example 4. If we want to say, for instance, that `l2` must contain two repeated elements, then the above statements can be preceded by the following declarations

```
LVar x = new LVar();
LList l2 = new LList().ins(x).ins(x);
```

In this way `split` is called with its third argument bound to the partially specified list `[x,x|_]` instead of being left unbound. Thus, solving `split(l,l1,l2)` will bind `l1` to `["I","she","they"]` and `l2` to `["you","you","we"]`.

## 5 Implementing DCGs

In this section we show a more complete example of application of the facilities offered by JSetL to support nondeterminism: the implementation of Definite Clause Grammars [8].

A Definite Clause Grammar (DCG) is a way to represent a context-free grammar as a set of first-order logic formulae in the form of definite clauses. As such, DCGs are closely related to Logic Programming, and tools for dealing with DCGs are usually provided by current Prolog systems. Given the DCG representation of a grammar one can immediately obtain a parser for the language it describes by viewing the DCG as a set of Prolog clauses and using the Prolog interpreter to execute them.

In this section we show how DCGs can be conveniently used also in the context of more conventional languages, such as Java, provided the language is equipped with a few features that are fundamental to support DCGs processing, namely (logical) lists and nondeterminism. We prove this claim by showing how DCGs can be encoded and processed using Java with JSetL.

Consider the following excerpt of a grammar of constant arithmetic expressions

$$\langle expr \rangle ::= \langle num \rangle | \langle num \rangle + \langle expr \rangle | \langle num \rangle - \langle expr \rangle$$

Assume that input to be parsed is represented as a list of numerals and symbols. For example, `[8, +, 2, -, 7]` is a valid $\langle expr \rangle$.

This grammar may be encoded in terms of first-order logic formulae in clausal form in the following way: create one predicate for each non-terminal in the grammar and define each predicate using one clause for each alternative form of the corresponding non-terminal. Each predicate takes two arguments, the first being the list representation of the input stream, and the second being instantiated to the list of input elements that remain after a complete syntactic

structure has been found. As an example, the above grammar can be written as a DCG as follows (using a pure Prolog notation).

*Example 5.* (A DCG for $\langle expr \rangle$)

```
expr(L, Remain) :-
   num(L, Remain).
expr(L, Remain) :-
   num(L, L1), L1 = [+|L2], expr(L2, Remain).
expr(L, Remain) :-
   num(L, L1), L1 = [-|L2], expr(L2, Remain).
num(L, Remain) :-
   L = [D|Remain], number(D).
```

where the predicate `number(D)` is true if `D` is a numeric constant.[4]

This grammar representation constitutes an executable Prolog program that can be immediately used as a top-down parser for the denoted language. Using this program we can prove that, for example,
`expr([1, +, 2, -, 3], [])`
is true (i.e., `1+2-3` is a valid arithmetic expression), while
`expr([1, +, 2, -], [])`
is false.

The DCG shown in Example 5, that we have written as a Prolog program, can be implemented with a relatively little effort as a JSetL program as well. Each predicate corresponding to a non-terminal in the grammar is implemented as a new JSetL constraint, that is a method of a class extending the class `NewConstraintsClass`. These methods exploit the nondeterministic features provided by JSetL to support the nondeterministic choice from among different clauses for the same predicate. List data structures are implemented using JSetL logical lists, that is objects of the class `LList`. In particular, partially specified lists with an unknown rest (i.e., $[o \,|\, l]$, $l$ unbound) can be constructed using the method `ins` and accessed through unification. The complete JSetL implementation of the DCG shown above is given in Example 6.

*Example 6.* (Implementing the DCG for $\langle expr \rangle$ in JSetL)

```
public class ExprParser extends NewConstraintsClass {
      public ExprParser(SolverClass currentSolver) {
         super(currentSolver);
      }
      public Constraint expr(LList L, LList Remain) {
         return new Constraint("expr", L, Remain);
      }
```

[4] Special syntax exists in current Prolog systems that allows EBNF-like specification of DCGs. For instance, the second clause of Example 5 can be written in Prolog alternatively as `expr --> num, [+], expr`. The Prolog interpreter automatically translates this special form to the pure clausal form used in Example 5.

```
public Constraint num(LList L, LList Remain) {
   return new Constraint("num", L, Remain);
}
public Constraint number(LVar n) {
   return new Constraint("number", n);
}

protected void user_code(Constraint c)
throws NotDefConstraintException {
  if (c.getName().equals("expr"))
     expr(c);
  else if (c.getName().equals("num"))
     num(c);
  else if (c.getName().equals("number"))
     number(c);
  else {
     throw new NotDefConstraintException();
  }
}

private void expr(Constraint c) {
  LList L = (LList)c.getArg(1);
  LList Remain = (LList)c.getArg(2);
  switch (c.getAlternative()) {
    // expr(L, Remain) :- num(L, Remain).
    case 0: {
      solver.addChoicePoint(c);
      solver.add(num(L, Remain));
      break;
    }
    // expr(L, Remain) :- num(L, L1), L1 = [+|L2], expr(L2, Remain).
    case 1: {
      solver.addChoicePoint(c);
      LList L1 = new LList();
      LList L2 = new LList();
      solver.add(num(L, L1));
      solver.add(L1.eq(L2.ins('+')));
      solver.add(expr(L2, Remain));
      break;
    }
    // expr(L, Remain) :- num(L, L1), L1 = [-|L2], expr(L2, Remain).
    case 2: {
      LList L1 = new LList();
      LList L2 = new LList();
      solver.add(num(L, L1));
      solver.add(L1.eq(L2.ins('-')));
      solver.add(expr(L2, Remain));
    }
  }
}
```

```
      private void num(Constraint c) {
        LList L = (LList)c.getArg(1);
        LList Remain = (LList)c.getArg(2);
        LVar D = new LVar();
        solver.add(L.eq(Remain.ins(D)));
        solver.add(number(D));
      }

      private void number(Constraint c) {
        LVar n = (LVar)c.getArg(1);
        if (n.getValue() instanceof Integer)
              return;
        else
              c.fail();
      }
}
```

If, for example, the expression to be parsed is $5 + 3 - 2$, which is represented by a logical list `tokenList` with value `['5','+','3','-','2']`, and `sampleParser` is an instance of the class `ExprParser`, then the invocation

```
solver.check(sampleParser.expr(tokenList,LList.empty()))
```

will return `true`, while, if `tokenList` has value `['5','+','3','-']`, the same invocation to `sampleParser.expr` will return `false`.

Actions to be performed when a non-terminal has been successfully reduced (e.g., in order to evaluate the parsed expression or to generate the corresponding target code) can be easily added to a DCG by adding new arguments to predicates defining non-terminals and new atoms at the end of the body of the corresponding clauses. Accordingly, the JSetL implementation of a DCG can be easily extended by adding new arguments and suitable statements to the user-defined constraints implementing the non-terminals.

## 6 Conclusions and future work

In this paper we have made evident, through a number of simple examples, that nondeterministic programming is conveniently exploitable also within conventional O-O languages such as Java. We have obtained this by combining a number of different features offered by the Java library JSetL: set data abstractions, nondeterministic constraint solving, logical variables, unification, user-defined constraints. In particular, general nondeterministic procedures can be defined in JSetL as new user-defined constraints, taking advantage of the facilities for expressing and handling nondeterminism provided by the solver.

The JSetL library, along with the source code of all the sample programs shown in this paper, can be downloaded from the JSetL's home page at `http://cmt.math.unipr.it/jsetl.html`.

As a future work we plan to identify the minimal extensions to be made to the JSetL's solver to make it capable of supporting, with the same approach outlined in this paper, other nondeterministic control structures e.g. the ones described in [1] and [12].

## Acknowledgments

This work has been partially supported by the G.N.C.S. project "Specifica e verifica di algoritmi tramite strumenti basati sulla teoria degli insiemi". Special thanks to Luca Chiarabini who took part and stimulated the preliminary discussions on this work, and to Andrea Longo who contributed to the development of the JSetL based implementation of DCGs.

## References

1. K.R. Apt, J. Brunekreef, V. Partington, A. Schaerf (1998) Alma-0: An Imperative Language that Supports Declarative Programming. ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 20, No. 5, 1014–1066.
2. K.R. Apt, A. Schaerf (1999) The Alma Project, or How First-Order Logic Can Help Us in Imperative Programming. In E.-R. Olderog, B. Steffen, Eds., *Correct System Design*, *LNCS*, Springer-Verlag, v. 1710, 89–113.
3. F. Bergenti, L. Chiarabini, and G. Rossi (2011) Programming with Partially Specified Aggregates in Java. *Computer Languages, Systems & Structures*, Elsevier, 37(4), 178–192.
4. J. Cohen (1979) Non-Deterministic Algorithms. Computing Surveys, Vol. 11, No. 2, 79–94.
5. A. Dovier, C. Piazza, E. Pontelli, and G. Rossi (2000) Sets and Constraint Logic Programming. *ACM Transactions on Programming Languages and Systems*, 22(5):861–931.
6. A. Dovier, E. Pontelli, and G. Rossi (2006) Set unification. *Theory and Practice of Logic Programming*, 6:645–701.
7. S.H. Mirian-HosseinaAbadi, M.R. Mousavi (2002) Making Nondeterminism Explicit in Z. Proceedings of the Iranian Computer Society Annual Conference (CSICC 02), Tehran, Iran.
8. F.C.N. Pereira, D.H.D. Warren (1980) Definite clause grammars for language analysis - A survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* 13, 3, 231-278.
9. G. Rossi, E. Panegai, and E. Poleo (2007) JSetL: A Java Library for Supporting Declarative Programming in Java. *Software-Practice & Experience*, 37:115-149.
10. J. T. Schwartz, R. B. K. Dewar, E. Dubinsky, and E. Schonberg (1986) *Programming with sets, an introduction to SETL*. Springer-Verlag.
11. H. Sondergaard, P. Sestoft (1992) Non-Determinism in Functional Languages. The Computer Journal, 35(5), 514-523.
12. P. Van Hentenryck, L. Michel (2006) Nondeterministic Control for Hybrid Search. Constraints, Vol. 11, No. 4, 353– 373.
13. M. Walicki, S. Meldal (1993) Sets and Nondeterminism. Presented at ICLP'93 Post-Conference Workshop on Logic Programming with Sets (`http://people.math.unipr.it/gianfranco.rossi/sets/body_workshop.html`), Budapest.

# A proof-checking experiment on representing graphs as membership digraphs*

Pierpaolo Calligaris[1], Eugenio G. Omodeo[2], Alexandru I. Tomescu[3]

[1] Università di Trieste (laureando),
email: frenkyo@gmail.com
[2] Dipartimento di Matematica e Geoscienze, Università di Trieste,
Via Valerio 12/1, I-34127 – Trieste, Italy
email: eomodeo@units.it
[3] Helsinki Institute for Information Technology HIIT,
Department of Computer Science, University of Helsinki,
P.O. 68 (Gustaf Hällströmin katu 2b), FI-00014 – Helsinki, Finland
email: tomescu@cs.helsinki.fi

**Abstract.** We developed, and computer-checked by means of the Ref verifier, a formal proof that every weakly extensional, acyclic (finite) digraph can be decorated injectively *à la* Mostowski by finite sets so that its arcs mimic membership. We managed to have one sink decorated with $\emptyset$ by this injection.
We likewise proved that a graph whatsoever admits a weakly extensional and acyclic orientation; consequently, and in view of what precedes, one can regard its edges as membership arcs, each deprived of the direction assigned to it by the orientation.
These results will be enhanced in a forthcoming scenario, where every connected claw-free graph $G$ will receive an extensional acyclic orientation and will, through such an orientation, be represented as a transitive set $T$ so that the membership arcs between members of $T$ will correspond to the edges of $G$.

**Key words:** Theory-based automated reasoning; proof checking; Referee aka ÆtnaNova; graphs and digraphs; Mostowski's decoration.

## 1 Can graphs be represented as membership digraphs?

One usually views the edges of a graph as vertex doubletons;[4] but various ways of representing graphs can be devised (as quickly surveyed in [5, Sec. 2]). Thanks to a convenient choice on how to represent connected claw-free graphs, Milanič and Tomescu [2] proved with relative ease two classical results on graphs of that kind, namely that any such graph owns a near-perfect matching and has a Hamiltonian cycle in its square. Those results are, in fact, legitimately transferred to a special class of digraphs, whose vertices are hereditarily finite sets and whose

* Work partially supported by the INdAM/GNCS 2013 project *"Strumenti basati sulla teoria degli insiemi per la verifica di algoritmi"*

[4] We call undirected graphs simply *graphs*, and directed graphs, *digraphs*.

arcs reflect the membership relation. Under this change of perspective, a fully formal reconstruction of those results became affordable and, once carried out, was certified correct with the Ref proof-checker [3,4].

Can we, with equal ease, formalize in Ref the Milanič-Tomescu representation result *per se*? That result is the claim that for every connected claw-free graph $G$ there exist a set $\nu_G \supseteq \bigcup \nu_G$ and an injection $f$ from the vertices of $G$ onto $\nu_G$ such that $\{x, y\}$ is an edge of $G$ if and only if either $fx \in fy$ or $fy \in fx$ holds. The proof articulates as follows:

1. One shows that for any graph $G = (V, E)$ as said, there is a $D \subseteq V \times V$ such that $E = \big\{\{x,\ y\} : [x, y] \in D\big\}$ and $(V, D)$ is an acyclic digraph which is EXTENSIONAL: i.e., no two vertices in $V$ have the same out-neighbors.
2. One DECORATES vertices by putting $f\,v = \{f\,w : [v, w] \in D\}$ *à la* Mostowski, for all $v \in V$. Acyclicity ensures that this recursion makes sense; extensionality ensures the injectivity of $f$.

As a preparatory, simpler formalization task, we have proved with Ref that a graph $G$ whatsoever admits a set $\nu_G$ and an injection $f$ from the vertices of $G$ onto $\nu_G$ such that $\{x, y\}$ is an edge of $G$ if and only if either $fx \in fy$ or $fy \in fx$ holds. We *could not* have insisted on the transitivity condition $\nu_G \supseteq \bigcup \nu_G$ here, because we have nohow restrained $G$. The proof now articulates as follows:

1′. For any $G = (V, E)$, there is a $D \subseteq V \times V$ s.t. $E = \big\{\{x,\ y\} : [x, y] \in D\big\}$ and $(V, D)$ is an acyclic digraph which is WEAKLY EXTENSIONAL: i.e., any two vertices that share the same out-neighbors have *no* out-neighbors.
2′. We decorate vertices by putting: $f\,v = \{f\,w : [v, w] \in D\}$ for each $v \in V$ endowed with out-neighbors, $f\,z = \emptyset$ for one sink $z$, and $f\,u = \big\{\{V\} \cup V \setminus \{u\}\big\}$ for each sink $u \neq z$.

(Notice that 2′. subsumes 2. altogether, because an extensional digraph has exactly one sink.)

## 2 Ingredients of a Ref's scenario

What one submits to the Ref checker, to have its correctness verified, is a SCENARIO: namely, a script file consisting of definitions and of theorems endowed with their proofs; a construct, named THEORY , enables one to package definitions and theorems into reusable proofware components. A variant of the Zermelo-Fraenkel set theory, postulating global choice, regularity, and infinity, underlies the logical armory of Ref: this is apparent from the fifteen or so inference rules available in the proof-specification language (see [5, Sec. 3]), of which only a few sprout directly from first-order predicate calculus, while most embody some form of set-theoretic reasoning. Multi-level syllogistic [1] acts as a ubiquitous inference mechanism, while THEORYs add a touch of second-order reasoning ability to Ref's overall machinery.

Our figures offer a glimpse of the Ref's language. Fig. 1 shows the definitions of graph-theoretic notions relevant to the proof-checking experiment on which

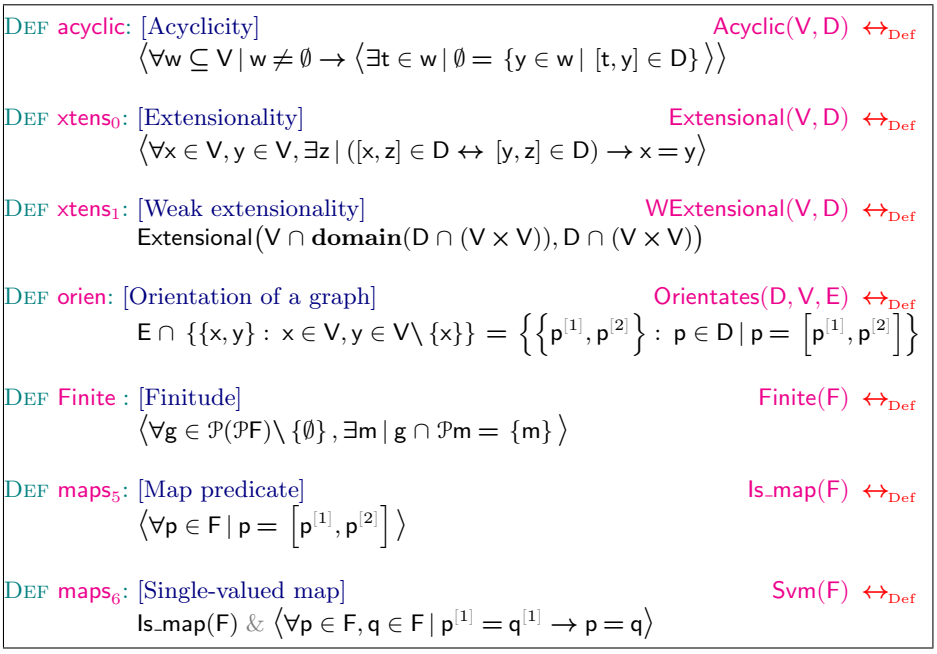



**Fig. 1.** Four properties refer to digraphs, the other three to generic sets

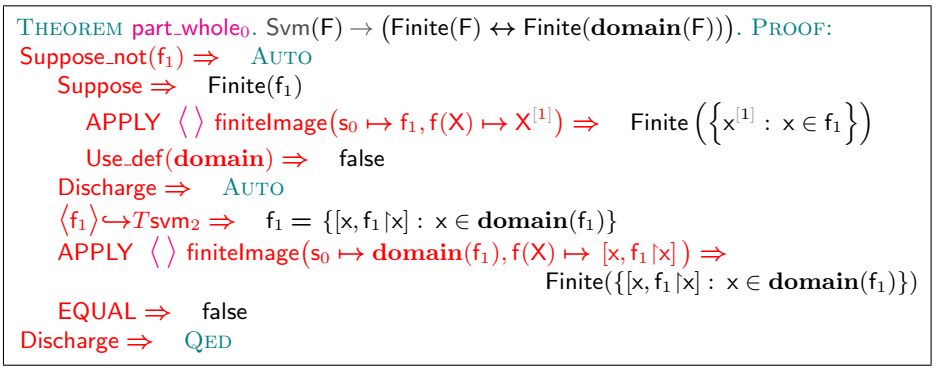



**Fig. 2.** Example of a theorem proved in the Ref language

we report, and introduces finitude and the notion of mapping ('Svm').[5] Fig. 2

[5] To enforce a useful distinction, we denote by $G(x)$ the application of a *global* function $G$ to an argument $x$ ('global' meaning that the domain of $G$ is the class of all sets), while denoting by $f\restriction x$ the application to $x$ of a *map* $f$ (typically single-valued), viewed as set of pairs.

shows the formal development of a proof, consisting of nine steps, each indicating which inference rule is employed to get the corresponding statement. This proof invokes twice a THEORY named finiteImage, whose interface is displayed in Fig. 3. While finiteImage does not return any symbol, the other, subtler THEORY displayed in the same figure, namely finiteInduction, returns a symbol, $\mathsf{fin}_\Theta$, representing an $\subseteq$-minimal set which meets P—given that at least one finite set satisfying property P exists. Likewise, the THEORY finiteAcycLabeling shown in Fig. 4 returns a labeling of a given acyclic digraph, thereby furnishing the technique for decorating the graph *à la* Mostowski.

| | |
|---|---|
| THEORY finiteImage$(s_0, f(X))$ | THEORY finiteInduction$(s_0, P(S))$ |
| Finite$(s_0)$ | Finite$(s_0)$ & $P(s_0)$ |
| $\Rightarrow$ | $\Rightarrow$ $(\mathsf{fin}_\Theta)$ |
| Finite$(\{f(x) : x \in s_0\})$ | $\langle \forall S \mid S \subseteq \mathsf{fin}_\Theta \to \mathsf{Finite}(S)$ & $(P(S) \leftrightarrow S = \mathsf{fin}_\Theta)\rangle$ |
| END finiteImage | END finiteInduction |

**Fig. 3.** Interfaces of two THEORYs regarding finitude

THEORY finAcycLabeling$(v_0, d_0, h(s, x))$
Acyclic$(v_0, d_0)$ & Finite$(v_0)$
$\Rightarrow$ $(\mathsf{lab}_\Theta)$
Svm$(\mathsf{lab}_\Theta)$ & **domain**$(\mathsf{lab}_\Theta) = v_0$
$\langle \forall x \in v_0 \mid \mathsf{lab}_\Theta \restriction x = h\left(\left\{\mathsf{lab}_\Theta \restriction p^{[2]} : p \in d_{0\mid\{x\}} \mid p^{[2]} \in v_0\right\}, x\right)\rangle$
END finAcycLabeling

**Fig. 4.** Interface of a THEORY usable to label an acyclic digraph

## 3 Our experiment in a nutshell

The two THEORYs in which our experiment culminates are shown in Fig. 5; the key theorem which makes the second of them derivable from the first was stated in Ref as follows:

THEOREM $\mathsf{weaXtensionalization}_0$. Finite$(V)$ & $S \in V \to$
$\langle \exists d \mid \mathsf{Orientates}(d, V, E)$ & Acyclic$(V, d)$ & WExtensional$(V, d)$ & $S \notin$ **range**$(d)\rangle$.

Due to its centrality in our scenario, we wish to briefly sketch the proof of the orientability theorem just cited. Arguing by contradiction, suppose that there

THEORY finMostowskiDecoration($v_0, d_0$)
$v_0 \times v_0 \supseteq d_0$ & $v_0 \neq \emptyset$ & Finite($v_0$) & Acyclic($v_0, d_0$) & WExtensional($v_0, d_0$)
$\Rightarrow$ ($mski_\Theta$)
Svm($mski_\Theta$) & **domain**($mski_\Theta$) $= v_0$
$\langle \forall w \mid w \in \mathbf{domain}(d_0) \rightarrow mski_\Theta{\restriction}w = \{ mski_\Theta{\restriction}p^{\restriction 2} : p \in d_{0\restriction\{w\}} \}$ & $mski_\Theta{\restriction}w \neq \emptyset \rangle$
$\emptyset \in \mathbf{range}(mski_\Theta)$ & $\langle \forall y \mid y \in \mathbf{range}(mski_\Theta) \rightarrow$ Finite(y)$\rangle$
$\langle \forall x, y \mid \{x, y\} \subseteq v_0$ & $mski_\Theta{\restriction}x = mski_\Theta{\restriction}y \rightarrow x = y \rangle$
$\langle \forall y \mid y \in v_0 \rightarrow (mski_\Theta{\restriction}y \in mski_\Theta{\restriction}x \leftrightarrow [x, y] \in d_0) \rangle$
END finMostowskiDecoration

THEORY finGraphRepr($v_0, e_0$)
$e_0 \subseteq \{\{x, y\} : x \in v_0, y \in v_0 \setminus \{x\}\}$ & $v_0 \neq \emptyset$ & Finite($v_0$)
$\Rightarrow$ ($wski_\Theta$)
Svm($wski_\Theta$) & **domain**($wski_\Theta$) $= v_0$ & $\emptyset \in \mathbf{range}(wski_\Theta)$
$\langle \forall y \mid y \in \mathbf{range}(wski_\Theta) \rightarrow$ Finite(y)$\rangle$
$\langle \forall x, y \mid \{x, y\} \subseteq v_0$ & $wski_\Theta{\restriction}x = wski_\Theta{\restriction}y \rightarrow x = y \rangle$
$\langle \forall x, y \mid \{x, y\} \subseteq v_0 \rightarrow ((wski_\Theta{\restriction}y \in wski_\Theta{\restriction}x \vee wski_\Theta{\restriction}x \in wski_\Theta{\restriction}y) \leftrightarrow \{x, y\} \in e_0) \rangle$
$\langle \forall x \mid wski_\Theta{\restriction}x \cap \mathbf{range}(wski_\Theta) \neq \emptyset \rightarrow wski_\Theta{\restriction}x \subseteq \mathbf{range}(wski_\Theta) \rangle$
END finGraphRepr

**Fig. 5.** THEORYs on Mostowski's decoration and on graph representation

is a counterexample; then, exploiting the finiteness hypothesis, take a *minimal* counterexample $v_1, s_1, e_0$. We are supposing that there is no acyclic, weakly extensional orientation of the graph $\big(v_1, e_0 \cap \big\{\{x, y\} : x \in v_1,\ y \in v_1 \setminus \{x\}\big\}\big)$ having $s_1$ as a source; whereas, for every $v_0 \subsetneq v_1$, one can orient $\big(v_0, e_0 \cap \big\{\{x, y\} : x \in v_0,\ y \in v_0 \setminus \{x\}\big\}\big)$ in an acyclic and weakly extensional way, for any vertex $t \in v_0$, so that $t$ plays the role of a source. Let, in particular, $v_0 = v_1 \setminus \{s_0\}$. Unless $s_1$ is an isolated vertex, an acyclic and weakly extensional orientation of $v_0$ exists that has as a source a chosen neighbor $t_1$ of $s_1$. However, that orientation could trivially be extended to the graph with vertices $v_1$ so that $s_1$ becomes a source; this contradiction shows that $s_1$ cannot have neighbors in $v_1$, which is also untenable: if so, any orientation for $v_0$ would in fact work also as an orientation for $v_1$ and, as such, would have $s_0$ as a source.

The full Ref scenario can be seen at `http://www2.units.it/eomodeo/wERS.pdf` (cf. also `http://www2.units.it/eomodeo/ClawFreeness.html`).

## 4 Planned work on representing claw-free graphs

The larger experiment we have in mind will associate with each connected claw-free graph $G = (V, E)$ an injection $f$ from $V$ onto a transitive, hereditarily finite set $\nu_G$ so that $\{x, y\} \in E$ if and only if either $f\,x \in f\,y$ or $f\,y \in f\,x$.

The new notions entering into play can be rendered formally as follows:

$$\begin{aligned}
\mathsf{ClawFreeG}(V,E) &\leftrightarrow_{\mathrm{Def}} \big\langle \forall w,x,y,z \mid \{w,x,y,z\} \subseteq V \;\&\; \{\{w,y\},\{y,x\},\{y,z\}\} \subseteq E \rightarrow \\
&\qquad (x=z \vee w \in \{z,x\} \vee \{x,z\} \in E \vee \{z,w\} \in E \vee \{w,x\} \in E)\big\rangle\,, \\
\mathsf{Connected}(E) &\leftrightarrow_{\mathrm{Def}} \emptyset \notin E \wedge \\
&\qquad \big\langle \forall p \mid \Big(\bigcup p = E \wedge \big\langle \forall b \in p,\, c \in p \mid \bigcup b \cap \bigcup c \neq \emptyset \leftrightarrow b = c\big\rangle\Big) \rightarrow p = \{E\}\big\rangle\,, \\
\mathsf{HerFin}(S) &\leftrightarrow_{\mathrm{Def}} \mathsf{Finite}(S) \;\&\; \big\langle \forall x \in S \mid \mathsf{HerFin}(x)\big\rangle\,.
\end{aligned}$$

Here, the first *definiens* requires that no subgraph of $(V, E)$ induced by four vertices has the shape of a 'Y'. The second definiens requires that the set $E$ of edges can nohow be split into multiple disjoint blocks so that no edge acts as a 'bridge' by sharing endpoints with edges in distinct blocks. Hereditary finitude is a recursive notion.

We aim at getting the analogue, shown in Fig. 6, of THEORY finGraphRepr (cf. Fig. 5). For that, we must again exploit THEORY finMostowskiDecoration; in addition, a key theorem will ensure the acyclic extensional orientability of a connected and claw-free graph:

$$\begin{aligned}
&\textsc{Theorem}\ \mathsf{cClawFreeG}_2.\ \mathsf{Finite}(V) \;\&\; \mathsf{Connected}(V,E) \;\&\; \\
&\quad \mathsf{ClawFreeG}(V,E) \;\&\; E \subseteq \{\{x,y\} : x \in V, y \in V \setminus \{x\}\} \rightarrow \\
&\qquad \big\langle \exists d \subseteq V \times V \mid \mathsf{Orientates}(d,V,E) \;\&\; \mathsf{Acyclic}(V,d) \;\&\; \mathsf{Extensional}(V,d)\big\rangle.
\end{aligned}$$

THEORY herfinCCFGraphRepr($v_0, e_0$)
$e_0 \subseteq \{\{x,y\} : x \in v_0, y \in v_0 \setminus \{x\}\}$ & Finite($v_0$)
Connected($v_0, e_0$) & ClawFreeG($v_0, e_0$)
⇒ ($\mathsf{trans}_\Theta$)
Svm($\mathsf{trans}_\Theta$) & **domain**($\mathsf{trans}_\Theta$) = $v_0$
$\langle \forall x,y \mid \{X,Y\} \subseteq v_0 \;\&\; \mathsf{trans}_\Theta|X = \mathsf{trans}_\Theta|Y \rightarrow X = Y\rangle$
$\langle \forall x,y \mid \{X,Y\} \subseteq v_0 \rightarrow (\mathsf{trans}_\Theta|Y \in \mathsf{trans}_\Theta|X \vee \mathsf{trans}_\Theta|X \in \mathsf{trans}_\Theta|Y \leftrightarrow \{X,Y\} \in e_0)\rangle$
$\{y \in \mathbf{range}(\mathsf{trans}_\Theta) \mid y \not\subseteq \mathbf{range}(\mathsf{trans}_\Theta)\} = \emptyset$
$\mathbf{range}(\mathsf{trans}_\Theta) \neq \emptyset$ & HerFin($\mathbf{range}(\mathsf{trans}_\Theta)$)
END herfinCCFGraphRepr

**Fig. 6.** THEORY on representing a connected claw-free graph via membership

Another fact we must exploit is that every connected graph has a vertex whose removal (along with all edges incident to it) does not disrupt connectivity. The existence of such a NON-CUT VERTEX is easily proved for a tree. So, in order to cheaply achieve our goal, we will define

$$\begin{aligned}
\mathsf{HankFree}(T) &\leftrightarrow_{\mathrm{Def}} \big\langle \forall e \subseteq T \mid e = \emptyset \vee \big\langle \exists a \in e \mid a \not\subseteq \bigcup(e \setminus \{a\})\big\rangle\big\rangle\,, \\
\mathsf{Is_tree}(T) &\leftrightarrow_{\mathrm{Def}} \mathsf{Connected}(T) \wedge \mathsf{HankFree}(T)
\end{aligned}$$

and—if only provisionally—recast the connectivity assumption, $\mathsf{Connected}(\mathsf{v}_0, \mathsf{e}_0)$, of THEORY herfinCCFGraphRepr as the assumption that $(\mathsf{v}_0, \mathsf{e}_0)$ has a 'spanning tree':

$$\left\langle \exists \mathsf{t} \mid \mathsf{ls\_tree}(\mathsf{t}) \;\&\; \bigcup \mathsf{t} = \mathsf{v}_0 \;\&\; (\mathsf{v}_0 = \{\mathbf{arb}(\mathsf{v}_0)\} \;\vee\; \mathsf{t} \subseteq \mathsf{e}_0) \right\rangle .$$

This eases things: for, any vertex with fewer than 2 incident edges in the spanning tree of a connected graph will be a non-cut vertex of the graph.

## References


1. A. Ferro, E.G. Omodeo, and J.T. Schwartz. Decision procedures for elementary sublanguages of set theory. I. Multi-level syllogistic and some extensions. *Comm. Pure Applied Math.*, 33(5):599–608, 1980.
2. M. Milanič and A. I. Tomescu. Set graphs. I. Hereditarily finite sets and extensional acyclic orientations. *Discrete Applied Mathematics*, 161(4-5):677–690, 2013.
3. E. G. Omodeo. The Ref proof-checker and its "common shared scenario". In Martin Davis and Ed Schonberg, editors, *From Linear Operators to Computational Biology: Essays in Memory of Jacob T. Schwartz*, pages 121–131. Springer, 2012.
4. E. G. Omodeo and A. I. Tomescu. Appendix: Claw-free graphs as sets. In Martin Davis and Ed Schonberg, editors, *From Linear Operators to Computational Biology: Essays in Memory of Jacob T. Schwartz*, pages 131–167. Springer, 2012.
5. E.G. Omodeo and A.I. Tomescu. Set graphs. III. Proof Pearl: Claw-free graphs mirrored into transitive hereditarily finite sets. *Journal of Automated Reasoning*, In press. DOI: 10.1007/s10817-012-9272-3.

# Encodings of Sets and Hypersets

Alberto Policriti

[1] University of Udine, Department of Mathematics and Informatics, Udine, Italy
[2] Istituto di Genomica Applicata, Udine, Italy

**Abstract.** We will present some results and open problems on an extension of the Ackermann encoding of Hereditarily Finite Sets into Natural Numbers. In particular, we will introduce and discuss a simple modification of the above mentioned Ackermann encoding, that should naturally generalize from Hereditarily Finite Sets to Hereditarily Finite Hypersets.

**Keywords**: Ackermann encoding, Hereditarily Finite Sets, Hereditarily Finite Hypersets.

## Introduction

This work is related with an attempt to (sensibly) extend the following function (map) introduced by Ackermann in 1937 (see [Ack37]) :

**Definition 1.**

$$\mathbb{N}_A(x) = \Sigma_{y \in x} 2^{\mathbb{N}_A(y)} \tag{1}$$

The above map is a bijection between Hereditarily Finite Sets (denoted by $\mathsf{HF}$: the collection of sets obtained starting from $\emptyset$ and closing with respect to finite set formation) and Natural Numbers (denoted by $\mathbb{N}$).

The usage of 2 as base for the exponentiations in Definition 1 allows us to see the binary expansion of the natural number $\mathbb{N}_A(x)$ as a *full description*—that is the list of its elements—of $x$ in terms of $\mathbb{N}_A$: the presence of a 1 in position $i$ of the binary expansion of $\mathbb{N}_A(x)$ is equivalent to saying that the element $y$ such that $\mathbb{N}_A(y) = i$ belongs to $x$.

*Example 1.* $\mathbb{N}_A(\emptyset) = 0, \mathbb{N}_A(\{\emptyset\}) = 2^0 = 1, \mathbb{N}_A(\{\emptyset, \{\emptyset\}\}) = 2^0 + 2^1 = 3$, the binary code of $\mathbb{N}_A(\{\emptyset, \{\emptyset, \{\emptyset\}\}\})$ is 1001, that is 9.

$\mathbb{N}_A$ is a simple and natural encoding of sets, hence a powerful tool for representing and manipulating objects that are suitable to represent any kind of mathematical information.

Two sets are equal if and only if they have the same elements and the "translation" of this in terms of $\mathbb{N}_A$ corresponds to observing that two natural numbers are equal if and only if they have the same binary code. The previously mentioned set-theoretic principle for testing equality—a basic axiom in classical Set Theory called *extensionality*—can be applied only if the membership relation $\in$

does not admit cycles. A "set" $u$ such that $u$ belongs to itself, for example, cannot be tested for equality using extensionality against another set $v$: among the equalities to be checked we would need ... to take $u$ into account! Nevertheless, non well-founded set theories (whose elements are called *hypersets*) are useful and very expressive tools. Especially in Informatics. They add the ability to represent circular phenomena by blending the basic set-theoretic machinery with the notion of bisimulation used in place of extensionality (see [Acz88]). Therefore, for example, it becomes important to find (fast) algorithms to compute hyperset-equality. In [PP04] a number of examples of usages and extensions of the Ackermann map are given, exploring the possibility of computing hyeperset equality by comparing Ackermann-like encodings.

Here we discuss a possible extension of $\mathbb{N}_A$ whose aim is to maintain formal elegance while using as codomain a number system larger than $\mathbb{N}$.

We mention the fact that, along the same line, we already proposed extensions $\mathbb{Q}_A$ of $\mathbb{N}_A$ mapping the collection $\overline{\mathsf{HF}}$ of rational hereditarily finite hypersets into *dyadic rational numbers* (see [DOPT10]). Dyadic rational numbers are rational numbers that can be denoted by finitely many (binary) digits and our proposal can be illustrated by a simple example: if $\mathbb{Q}_A(z) = 1010,0111$, then $z$ is the hyperset whose well-founded elements are the second and the fourth (that is those having Ackermann code equal to 2 and 4), while $z$'s non well-founded elements are the second, the third, and the fourth. Clearly, to build $\mathbb{Q}_A$ an ordering of the hereditarily finite hypersets must enter into play. The set $\Omega = \{\Omega\}$ is—naturally—the first non well-founded set and, consequently, $\mathbb{Q}_A(\Omega) = 0,1$.

The extension of Ackermann map discussed below is more *direct* than $\mathbb{Q}_A$, as it does not require any (somehow arbitrary) ordering of hereditarily finite sets. This feature opens the way to an usage of the newly proposed map for bisimulation computation based on code comparison, as well as to a large array of numerically-based (hyper)set manipulation techniques.

## 1 Extending Ackermann map

Consider the following definition, obtained from Definition (1) by simply adding a minus sign at exponent.

**Definition 2.**

$$\mathbb{R}_A(x) = \Sigma_{y\in x} 2^{-\mathbb{R}_A(y)} \tag{2}$$

As a first and very basic motivation to consider the above map, notice that the above definition allows a (unique) solution to the following equation:

$$x = 2^{-x} \tag{3}$$

To see this, it suffices to observe that the two curves $y = x$ and $y = 2^{-x}$ are increasing and decreasing, respectively, and intersect in the first quadrant of $\mathbb{R}$. Let $\mathbf{\Omega}$ be the solution of (3) over $\mathbb{R}$. It is not difficult to see that $\mathbf{\Omega} \notin \mathbb{Q}$.

Our first objective would be to show that (2) is injective on the collection of Hereditarily Finite Sets.

*Conjecture 1.* The function $\mathbb{R}_A$ is injective on $\mathsf{HF}$.

A second, more challenging, point will consist in establishing the fact that $\mathbb{R}_A$ is injective on the full collection of rational hereditarily finite hypersets.

*Conjecture 2.* The function $\mathbb{R}_A$ is injective on $\overline{\mathsf{HF}}$.

We only have partial results related with the above conjectures that are mostly related with a study of the codes of the elements of the following sub-family of $\mathsf{HF}$:

**Definition 3.** *The elements of the family $\mathcal{S}$ of* super-singletons

$$\mathcal{S} = \left\{ \{\emptyset\}^i \mid i \in \mathbb{N} \right\},$$

*are defined recursively as follows:* $\{\emptyset\}^0 = \emptyset$ *and* $\{\emptyset\}^{n+1} = \{\{\emptyset\}^n\}$.

Super-singletons were first introduced by Zermelo in [Zer08] as a set-theoretic representation for ordinals and they have been recently discussed by Kirby in [Kir13].

The following figure shows the disposition of the first few code values of super-sigletons (let $s_i$ denote the code of the $i$-th super-singleton).

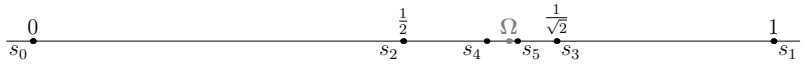



Fig. 1: The $\mathbb{R}_A$-code of the first 5 super-singletons

The $\mathbb{R}_A$-code of super-singletons is easily determined and seen to converge to the above defined value $\mathbf{\Omega}$.

**Proposition 1.** *The following hold:*

$$0 = s_0 < s_2 < \cdots s_{2i} < s_{2i+2} \cdots < \mathbf{\Omega} < \cdots s_{2i+3} < s_{2i+1} \cdots < s_3 < s_1 = 1,$$

*and*

$$\lim_{i\to\infty} s_{2i} = \lim_{i\to\infty} s_{2i+1} = \mathbf{\Omega}.$$

*Proof.* To see the above result it suffices to observe that $2^{-2^{-x}}$ is increasing and therefore, since $s_0 = 0 < s_2 = 1/2$ and since $s_1 = 1 > s_3 = 1/\sqrt{2}$, we have:

$$s_0 < s_2 < \cdots s_{2i} < 2^{-2^{-s_{2i}}} = s_{2i+2} \cdots$$

and

$$s_1 > s_3 > \cdots s_{2i+1} > 2^{-2^{-s_{2i+1}}} = s_{2i+3} \cdots .$$

Moreover, since $s_0 < \mathbf{\Omega} = 2^{-\mathbf{\Omega}}$ and since $s_{2i+2} = 2^{-2^{-s_{2i}}} < 2^{-2^{-\mathbf{\Omega}}} = \mathbf{\Omega}$, we have that all even-indexed super-singletons are smaller than $\mathbf{\Omega}$.

Similarly, all odd-indexed super-singletons are larger than $\mathbf{\Omega}$.

To conclude we must prove that both even and odd indexed sequences of super-singletons converge to $\mathbf{\Omega}$.

This is a consequence of the fact that $(2^{-x} - 2^{-y}) < (y - x)/2$, for all $x, y \in [1/2, 1]$. This, in turn, follows from Lagrange theorem stating that $(f(b) - f(a))/(b - a) = f'(z)$, for some $z \in (a, b)$. In fact, assuming $y > x$, the value $(2^{-x} - 2^{-y})/(y - x)$ is equal to $(2^{-z})' = -2^{-z} \ln(2)$, for some $z \in [x, y] \subseteq [1/2, 1]$, and this value is always smaller than $-1/2$. To conclude it is sufficient to consider the sequence for $i > 0$, with $x = s_{2i}$ and $y = s_{2i-1}$. □

With some extra observations and using the above result we can prove the injectivity of $\mathbb{R}_A$ on the codes of arbitrary unions of super-singletons.

**Definition 4.** *Given $j$ pairwise distinct indexes $i_1, \ldots, i_j$, let $s_{i_1,\ldots,i_j}$ be the code of $\{\emptyset\}^{i_1} \cup \cdots \cup \{\emptyset\}^{i_j}$, that is $s_{i_1} + \cdots + s_{i_j}$.*

*Moreover, let*

$$\mathcal{S}_{i_1,\ldots,\overline{i_j}} = \left\{ s_{i_1,\ldots,i_j,k} \mid k > i_j \right\}.$$

On the grounds of the above definition, we have that the codes of non-null super-singletons in $\mathcal{S}$ are in $\mathcal{S}_{\overline{0}}$. If we imagine (codes of) super-singletons in $\mathcal{S}_{\overline{0}}$ as obtained from the intersection of a *spiral* with the $x$-axis, as in the Figure 2,

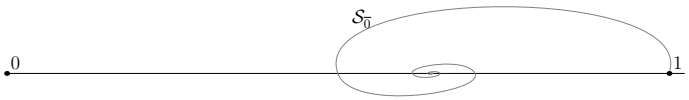



Fig. 2: The spiral of $\mathcal{S}_{\overline{0}}$

then the arrangement of the subsequent spirals can be seen to be a spiral of smaller and smaller spirals, as in Figure 3.

Notice that the points of convergence of all the above spirals—that is $\mathbf{\Omega} + s_i = \mathbb{R}_A(\Omega \cup \{\emptyset\}^i)$, for $i \geqslant 0$—are, in fact, codes of hypersets. This is not the case for the point of convergence of all the points of convergence, that turns out to be $2\mathbf{\Omega}$. Looking at the point of convergence of $2^{-(\mathbf{\Omega}+s_i)}$ for $i \geqslant 0$, one obtains the sequence of $s_{i+1}\mathbf{\Omega}$ that—no wonder—converges at $\mathbf{\Omega}^2$. Starting from the sequence of spirals $\mathcal{S}_{i,\overline{j}}$, whose points of convergence bring us at $3\mathbf{\Omega}$, by exponentiating we get to $\mathbf{\Omega}^3$, and so on.

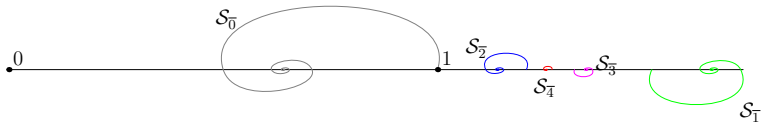



Fig. 3: The spirals of $\mathcal{S}_{\overline{0}}, \mathcal{S}_{\overline{1}}, \mathcal{S}_{\overline{2}}, \mathcal{S}_{\overline{3}}, \mathcal{S}_{\overline{4}}$

**Proposition 2.** *If* $\{i_1, \ldots, i_j\} \neq \{h_1, \ldots, h_k\}$, *then* $\mathcal{S}_{i_1,\ldots,\overline{i_j}} \cap \mathcal{S}_{h_1,\ldots,\overline{h_k}} = \emptyset$.

The above proposition is proved by first reducing the general case to the case in which $j = k$, since a padding of 0's on the left can always be performed. Then, proving that the leftmost difference among indexes in two elements belonging to $\mathcal{S}_{i_1,\ldots,\overline{i_j}}$ and $\mathcal{S}_{h_1,\ldots,\overline{h_j}}$, respectively, can never be compensated by the following differences.

By letting $h_i$ be the set belonging to $\mathsf{HF}$ whose Ackermann code is $i$, that is such that $\mathbb{N}_A(h_i) = i$, an ordering among the elements in $\mathsf{HF}$ is naturally (!) induced by $\mathbb{N}_A$[3].

Looking at indexes that do not necessarily belong to the collection of super-singletons or to sums of such sets, the following result holds.

**Proposition 3.** *For all* $i \in \mathbb{N}$*:*

1. $\mathbb{R}_A(h_i) \neq \mathbb{R}_A(h_{i+1})$;
2. $\mathbb{R}_A(h_i) \neq \mathbb{R}_A(h_{i+2})$.

We can prove the above proposition by rather *ad-hoc* arguments based on the specific value that a difference between two subsequent codes can assume.

## Conclusions

We proposed a new numerical encoding for hereditarily finite hypersets that is a natural extension of the celebrated Ackermann map establishing a bijection between natural numbers and hereditarily finite (well-founded) sets. Our proposed encoding differs from Ackermann's one only for a minus sign in the exponent. Such a small difference in the definition, however, radically changes the encoding that now maps the hypersets universe on real numbers. The map seems to have elegant analytical properties guaranteeing its injectivity on both well-founded and non well-founded hereditarily finite sets. Both injectivities, however, are

[3] Such ordering can be seen to correspond to the ordering holding on binary representation of natural numbers: given two strings of bits $\alpha$ and $\beta$, if the leftmost difference is such that a 1 appears in $\alpha$, then $\alpha > \beta$.

just conjectured here and our opinion is that once proved $\mathbb{R}_A$ to be 1-1 on well-founded sets, Conjecture 2 could/should be attacked by proving that the code of a hyperset is the unique accumulation point of the codes of the well-founded sets obtained by its unfolding. Notice that this is the case for $\mathbf{\Omega}$, as well as for other cases briefly mentioned above.

**Acknowledgments.** We warmly thank D. Cantone, G. D'Agostino, E. Omodeo, and A. I. Tomescu for discussions, corrections, stimuli, and patience greatly profuse during the past two years on this subject. Moreover, we wish to thank D. Cantone also for an elegant and simple proof of Proposition 2.

## References

[Ack37] W. Ackermann, *Die Widerspruchfreiheit der allgemeinen Mengenlehre*, Mathematische Annalen **114** (1937), 305–315.

[Acz88] P. Aczel, *Non-well-founded sets*, vol. 14 of CSLI Lecture Notes, Stanford, CA, 1988.

[DOPT10] G. D'Agostino, E. Omodeo, A. Policriti, and A. Tomescu, *Mapping hypersets into numbers (extended abstract)*, 12th Italian Conference on Theoretical Computer Science (ICTCS), 2010.

[Kir13] L. Kirby, *Ordinal operations on graph representations of sets*, Math. Log. Q. **59** (2013), no. 1-2, 19–26.

[PP04] C. Piazza and A. Policriti, *Ackermann Encoding, Bisimulations, and OBDDs*, Theory and Practice of Logic Programming **4** (2004), no. 5-6, 695–718.

[Zer08] E. Zermelo, *Untersuchungen über die Grundlagen der Mengenlehre. I*, Mathematische Annalen **65** (1908), no. 2, 261–281 (German).

# A First Comparison of Abstract Argumentation Systems: A Computational Perspective

Stefano Bistarelli[1,2], Fabio Rossi[1], and Francesco Santini[3]

[1] Department of Mathematics and Computer Science, University of Perugia, Italy
[bista,rossi]@dmi.unipg.it
[2] Institute of Informatics and Telematics (CNR), Pisa, Italy
stefano.bistarelli@iit.cnr.it
[3] EPI Contraintes, INRIA Rocquencourt
francesco.santini@inria.fr

**Abstract.** In this paper we introduce an initial comparison among three different implementations of Abstract Argumentation Systems: ASPARTIX, ConArg, and Dung-O-Matic. These tools are tested over four different kinds of interaction graphs, corresponding to Erdős-Rényi networks, scale-free Barabasi networks, Watts-Strogatz, and Kleinberg small-world networks. Our final goal is to thoroughly evaluate their performance (in this work we test complete and stable semantics only), and to find the most efficient one, but also, more in general, to better study this kind of problems from the computational point of view.

## 1 Introduction

An Abstract Argumentation Framework (AF), or System, as introduced in the seminal paper by Dung [5], is simply a pair $\langle \mathcal{A}, R \rangle$ consisting of a set $A$ whose elements are called arguments, and of a binary relation $R$ on $A$, called "attack" relation. Several different semantics (i.e., extensions) can be obtained over arguments by considering subsets of arguments with specific properties. An AF has an obvious representation as a directed graph where nodes are arguments and edges are drawn from attacking to attacked arguments.

The main aim of this work is to better understand this kind of systems under the perspective of their computational performance, in terms of networks with different properties and size. Indeed, existent and future applications [9] exploiting AFs need to efficiently behave and scale over "large" networks with properties close to real-world ones. In fact, in complex discussions it is not hard to find 50-100 arguments at least, especially if we consider on-line open fora, or discussion groups. When we make these digital tribunes correspond to well-known social networks, as *Twitter* or *Facebook*, but also to more structured debate-friendly tools, as DebateGraph[4], then the number of arguments can further increase due to the a high number of users participating to a discussion. The different intrinsic nature of these graphs leads to find more or less extensions of some kind,

[4] http://debategraph.org

e.g., stable ones [5]. In our tests we adopt two different types of small-world networks (*Kleinberg* [11] and *Watts-Strogatz* [13]), one class of scale-free networks (*Barabasi* [1]), and *Erdős-Rényi* [8] networks (see Sec. 3.2).

Our main goal is to compare three different tools that are more oriented to a straight computation of extensions, than on providing support to negotiation or decision-making. These tools correspond to ASPARTIX, ConArg, and Dung-O-Matic, introduced in Sec. 3.

## 2 Dung Argumentation

In [5], Dung has proposed an abstract framework for argumentation in which the focus is on the definition of the argument status.

**Definition 1.** *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 $c \in \mathcal{C}$.*

Dung gave several semantics to "acceptability". These various semantics produce none, one or several acceptable sets of arguments, called "extensions". In Def. 2 we define the concepts of conflict-free and stable extensions:

**Definition 2.** *A set $\mathcal{B} \subseteq \mathcal{A}$ is conflict-free in a given $\langle \mathcal{A}, R \rangle$ iff no two arguments $a$ and $b$ in $\mathcal{B}$ exist such that $a$ attacks $b$. A conflict-free set $\mathcal{B} \subseteq \mathcal{A}$ is a stable extension iff for each argument which is not in $\mathcal{B}$, there exists an argument in $\mathcal{B}$ that attacks it.*

The other semantics for "acceptability" rely upon the concept of defense:

**Definition 3.** *Given $\langle \mathcal{A}, R \rangle$, 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 $a$.*

An admissible set of arguments according to Dung must be a conflict-free set which defends all its elements. Formally:

**Definition 4.** *Given $\langle \mathcal{A}, R \rangle$, 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, three more semantics refining admissibility have been introduced in [5]:

**Definition 5.** *Given a $\langle \mathcal{A}, R \rangle$, a preferred extension is a maximal (w.r.t. set inclusion) admissible subset of $\mathcal{A}$. An admissible $\mathcal{B} \subseteq \mathcal{A}$ is a complete extension iff each argument which is defended by $\mathcal{B}$ is in $\mathcal{B}$. The least (w.r.t. set inclusion) complete extension is the grounded extension.*

## 3 Tools and Graphs

In the following of this section we describe our benchmark environment. We organize the content into two subsections: Sec. 3.1 concerns a more detailed description of the three tools we used in our comparison, while Sec. 3.2 describes in detail the random networks we generated as benchmark.

### 3.1 Tools

**ASPARTIX.** The *ASPARTIX*[5] system [7, 6] is an ASP-based tool for computing acceptable extensions for a broad range of formalizations of Dung's AFs and generalisations, e.g., value-based AFs [2]. *ASPARTIX* relies on a fixed disjunctive *Datalog* program (ASP also includes default negation) which takes an instance of an argumentation framework as input, and uses an ASP solver (as DLV) for computing the type of extension specified by the user. ASPARTIX is able to solve admissible, stable, complete, grounded, preferred, semi-stable, ideal, stage, cf2, resolution-based grounded and stage2 extensions.
**ConArg.** ConArg[6] [4, 3] is a constraint-based tool based on the *Java Constraint Programming* solver[7] (*JaCoP*), a Java library that provides the Java user with a *Finite Domain Constraint Programming* paradigm [12]. All the properties describing conflict-free, admissible, complete, stable, grounded and preferred extensions have been modeled and implemented with constraints. In this paper we consider a (faster) command-line version of ConArg. To model all the semantics presented in Sec. 2 we used MiniZinc[8], which is a medium-level constraint modelling language that can be mapped onto different existing solvers.
**Dung-O-Matic.** Dung-O-Matic[9] is an abstract argument computation engine implemented by the *javaDungAF* Java class. Dung-O-Matic supports Dung's AFs and several of their semantics, namely admissible, complete, eager, grounded, ideal, preferred, semi-stable, and finally stable. In order to find each of the proposed extensions, this tool implements a different algorithm presented in literature; for instance, the grounded semantics is computed with the original algorithms presented by Dung in [5].

### 3.2 Networks

Due to the lack of well-established benchmarks using real interaction graphs, we decided to randomly generate our test networks. To generate random graphs we adopted two different tools. The first one is *Java Universal Network/Graph Framework* (*JUNG*, `http://jung.sourceforge.net`), which is a Java software library for the modeling, generation, analysis and visualization of graphs. With

[5] `http://www.dbai.tuwien.ac.at/proj/argumentation/systempage/`
[6] `https://sites.google.com/site/santinifrancesco/tools/ConArg.zip`
[7] `http://www.jacop.eu`
[8] `http://www.minizinc.org`
[9] `http://www.arg.dundee.ac.uk/?page_id=279`

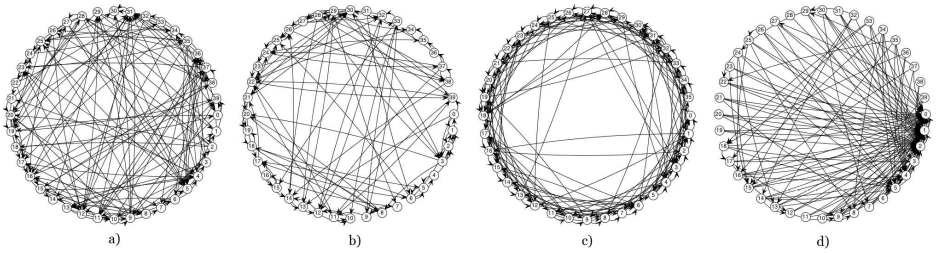



Fig. 1: An example of a *a)* Erdős-Rényi, *b)* Watts-Strogatz, *c)* Kleinberg, and *d)* a Barabasi graph, all with around 40 nodes.

JUNG we generated *Barabasi* [1] and *Kleinberg* [11] networks. The second library we used is named *NetworkX* (`http://networkx.github.io`), and it consists of a Python software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. With this tool we were able to generate *Erdős-Rényi* [8] and *Watts-Strogatz* [13]. We use two different libraries because of their different capabilities: with JUNG we are able to randomly generate directed Barabasi and Kleinberg networks, while NetworkX does not cover Kleinberg networks at all, and only provides undirected Barabasi ones. On the other side, NetworkX offers both Watts-Strogatz and Erdős-Rényi networks (not present in JUNG). Four examples of these networks are represented in Fig. 1. Our attention is more focused on testing small-world networks because we consider real large interaction-graphs as coming from social networks [10, 9].

## 4 Preliminary Tests and Conclusion

In Fig. 2 and Fig. 3 we show a first comparison among the three systems presented in Sec. 3.1, testing complete and stable extensions respectively. On the x-axis we report the exact number of nodes of each set of networks we use: in each set, we considered 50 random networks for each of the four classes reported in Sec. 3.2, for a total of 200 networks. On the y-axis we report the average CPU time to compute all the complete/stable extensions on that given set of networks. These first tests show that ConArg performs better than the other two systems, even if ASPARTIX has comparable results.

Performance has been collected using an Intel(R) Core(TM) i7 2.4Ghz processor, with 16Gb of RAM. To run ASPARTIX, we have used the last version of DLV (December 2012) with *Gringo* v3.0.5 and *ClaspD* v1.1.4 as extensions.

In this work we have commenced a study on the computational efficiency of nowadays tools dealing with AFs [5]. Our will is to extend this study under several lines. First, we will show tests on more computationally-demanding semantics (e.g., preferred extensions), we will test hard problems in Argumentation (e.g., deciding if a set is a preferred extension is CO-NP-complete), and, finally, we will better define the link with small-world networks, by finding the most appropriate random-network model. At last, we will test these tools over

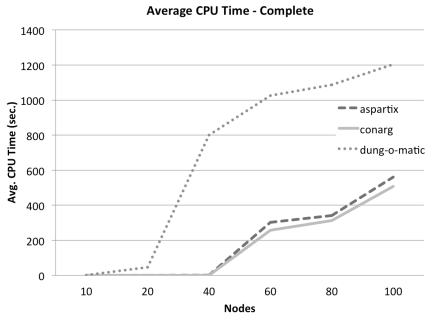



Fig. 2: Comparing complete extensions.

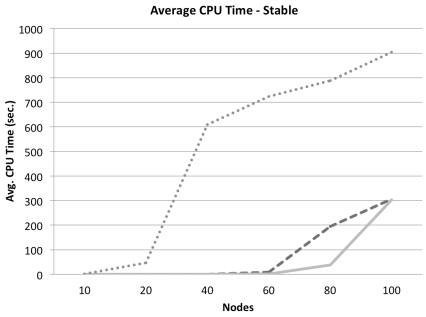



Fig. 3: Comparing stable extensions.

larger networks (e.g., 1,000 arguments), to check how feasible it is to use them in a real application, as, for instance, large-scale agreements via microdebates [9].

## References

1. A. L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
2. T. J. M. Bench-Capon. Persuasion in practical argument using value-based argumentation frameworks. *J. Log. Comput.*, 13(3):429–448, 2003.
3. S. Bistarelli and F. Santini. A common computational framework for semiring-based argumentation systems. In *European Conference on Artificial Intelligence*, volume 215 of *Frontiers in A.I. and Applications*, pages 131–136. IOS Press, 2010.
4. S. Bistarelli and F. Santini. Conarg: A constraint-based computational framework for argumentation systems. In *ICTAI*, pages 605–612. IEEE, 2011.
5. 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.
6. W. Dvorák, S. A. Gaggl, J. P. Wallner, and S. Woltran. Making use of advances in answer-set programming for abstract argumentation systems. *CoRR*, abs/1108.4942, 2011.
7. U. Egly, S. A. Gaggl, and S. Woltran. ASPARTIX: Implementing argumentation frameworks using answer-set programming. In *International Conference on Logic Programming (ICLP)*, pages 734–738. LNCS, Springer, 2008.
8. P. Erdős and A. Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4):343–347, 1961.
9. S. Gabbriellini and P. Torroni. Large scale agreements via microdebates. In *AT*, volume 918 of *CEUR Workshop Proceedings*, pages 366–377. CEUR-WS.org, 2012.
10. S. Gabbriellini and P. Torroni. Arguments in social networks. In *AAMAS*, pages 1119–1120. IFAAMAS, 2013.
11. C. Martel and V. Nguyen. Analyzing Kleinberg's (and other) small-world models. In *PODC '04*, pages 179–188, New York, NY, USA, 2004. ACM.
12. F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.
13. D. J. Watts and S. H. Strogatz. Collective dynamics of "small-world" networks. *nature*, 393(6684):440–442, 1998.

# BPAL: A Platform for Managing Business Process Knowledge Bases via Logic Programming*

Fabrizio Smith, Dario De Sanctis, Maurizio Proietti

National Research Council, IASI "Antonio Ruberti" - Viale Manzoni 30, 00185 Roma, Italy
{fabrizio.smith, dario.desanctis, maurizio.proietti}@iasi.cnr.it

## 1 Introduction

The adoption of structured approaches for the management of the Business Processes (BPs) operating within an organization is constantly gaining popularity. Nevertheless, their further automation is severely hampered by the fact that standard approaches are an insufficient means for capturing the complex process-related knowledge and making it available in a machine-accessible form [1]. As a result, many tasks, such as process analysis, verification, retrieval and composition, still require great manual efforts. In this scenario, the application of well-established techniques stemming from the area of Knowledge Representation has been shown as a promising approach for the enhancement of BP and Web Service [1, 2] management systems.

While several tools are today available for the modeling, verification, simulation, and execution (e.g., Intalio, Tibco, YAWL, Enhydra Shark), no commercial tool enables the semantic annotation of BP models, nor semantics-based reasoning services. Although several approaches have been proposed in literature to enable the exploitation of semantic facilities (see, e.g., the seminal work in [2, 3], and recent proposals [4, 5]), very few implemented tools (e.g., [6, 7]) give a (limited) support to the integrated management of the structural definition of a flow model, the formal definition of its behavior, and the domain knowledge related to the business scenario where it operates.

The BPAL platform implements a BP modeling and reasoning environment where the procedural knowledge of a BP can be enriched through ontology-based annotations. The theoretical basis of the tool is the Business Process Abstract Language [8], a language grounded in Logic Programming (LP) for representing and reasoning on various facets of process knowledge: (i) the meta-model of a BP schema (BPS), which covers a core of the BPMN notation, (ii) the BPS execution semantics, specified in a specialized version of the Fluent Calculus, a well-known LP-based action language, (iii) the behavioral properties of process executions, expressed by means of the CTL temporal logic, and (iv) the domain specific semantics of individual activities occurring in a BP, defined via OWL annotations (falling within the OWL 2 RL fragment) along the line of Semantic Web Services proposals.

The BPAL platform provides a graphical user interface to ease the definition of a BP Knowledge Base (BPKB) that collects the various pieces of process knowledge. BPAL also provides a reasoner implementing services for the enactment, verification, retrieval,

* A video demonstration is available at `http://www.youtube.com/watch?v=xQkapzjhO7g`

and composition of processes in the BPKB. Complex queries combining different aspects of process knowledge can be expressed in QuBPAL [9], a query language based on the SELECT-WHERE paradigm. QuBPAL queries are translated into clausal form and answered through an efficient, sound and complete LP query evaluation mechanism.

## 2 An Overview of the Functionalities of BPAL

**Management of BP Repositories.** The platform provides functionalities for managing BP repositories, such as: (1) creating a new BPS, (2) importing an existing BPS from an XML serialization of a BPMN diagram, and (3) editing a BPS via a graphical editor.
**Semantic Annotation.** Two kinds of annotations enable the enrichment of a BPS with domain related knowledge defined in a given reference ontology: (1) *terminological annotations*, which associate BPS elements with concept expressions, and (2) *functional annotations*, which define the conditions under which flow elements can be executed and the effects of their execution on the state of the world.
**Enactment.** The execution of a BP is modeled as an *execution trace*, corresponding to a plan in the Fluent Calculus, i.e., a sequence of actions of the form $[\texttt{begin}(\texttt{e}_1), \ldots, \texttt{complete}(\texttt{e}_\texttt{n})]$ where $\texttt{e}_\texttt{i}$ represents a flow elements. Execution traces correspond to process logs, which are commonly stored by BPM systems to record the enactment of BP instances. BPAL can verify whether a trace can be generated by a BP enactment (i.e., the compliance of a trace w.r.t. a given BPS) and, by exploiting the LP inference mechanism, the rules defining the trace semantics can also be used to *generate* the traces of a BPS satisfying some given (behavioral and/or ontological) property.
**Verification.** BPAL enables the verification of properties that depend on the interaction between the operational behavior of the process and the ontology-based semantic annotation. Thus, besides well-known correctness criteria typically addressed in the workflow community (e.g., *soundness*), the tool is also able to verify that, during a BP enactment, no semantics-related constraint is violated. For instance, given a BPS named p, we can define the following predicate:

$$\texttt{holds}(\texttt{not}(\texttt{ef}(\texttt{false})), \texttt{bps}(\texttt{p}))$$

meaning that no state is reachable (expressed by the temporal operator ef 'exists finally') where the false concept can be inferred from functional annotations specifying the effects of execution (e.g., o : approvedPO and o : rejectedPO) and execution-independent OWL axioms (e.g., approvedPO $\sqcap$ rejectedPO $\sqsubseteq$ false).
**Compliance.** Temporal queries can also be used for analyzing the compliance with *business rules*, i.e., directives expressing internal policies and regulations of an enterprise. In an eProcurement scenario, one such compliance rule may be that every *order* is eventually *closed*. This rule can be expressed by the following predicate meaning that it is not possible to reach the final state of the process where some order is not closed:

$$\texttt{holds}(\texttt{not}(\texttt{ef}(\texttt{final}(\texttt{p}) \texttt{ and } \texttt{nonclosedPO})), \texttt{bps}(\texttt{p}))$$

Here nonclosedPO holds in a given state if, for some O, the OWL assertion O : order holds and the OWL assertion O : closedPO does not hold.
**Retrieval.** The LP inference mechanism based on resolution can be also used for computing, via unification, substitutions for variables occurring in queries. BPAL exploits this query answering mechanism and provides a reasoning service for the retrieval of process fragments described in a declarative way. In particular, the WHERE clause of

a QuBPAL query can specify a combination of ontological, structural, and behavioral properties. For instance, if we want to retrieve all activities that must precede a delivery and require an authorization by the sales manager, then we may issue the following query (names prefixed by '?' denote variables):

`SELECT ?a`

`WHERE precedes(?a,delivering,p) AND requiresSalesMgrAuth(?a)`

where (i) `precedes(a,b,p)` is a predicate, defined by using the CTL temporal operators, which means that in any enactment of process `p`, activity `a` precedes activity `b`, and (ii) `requiresSalesMgrAuth(?a)` holds if the (terminological) annotation of *?a* is a concept subsuming the OWL assertion $\exists$ `requiresAuth.salesMgr`.

**Composition.** The tool allows the user to specify a *process skeleton*, which constitutes a high level definition of a new BP to be composed by retrieving subprocesses from a given BP repository [10]. Tasks appearing in the skeleton are associated with *local constraints*, which express requirements for the selection of the corresponding subprocesses to be retrieved, and *global constraints*, specifying the requirements on the composed BPS as a whole. Local and global constraints are expressed as QuBPAL queries and evaluated over the BPKB in order to compute possible compositions.

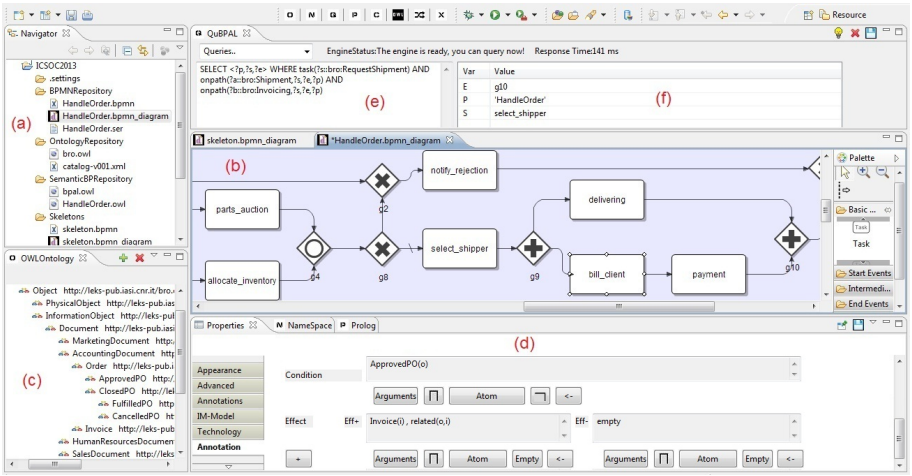



Fig. 1: GUI of the BPAL platform

## 3 Tool Description

The BPAL platform is implemented as an Eclipse Plug-in[1], whose main components are depicted in the functional view in Figure 2.

The **BPKB Editor** provides a graphical user interface to define a BPKB and to interact with the BPAL Reasoner. It encompasses: a tree view of the available resources (Fig. 1a), the STP[2] BPMN Modeler (Fig. 1b), a browser for the visualization of OWL

[1] `http://www.eclipse.org/`

[2] `http://www.eclipse.org/soa`

ontologies (Fig. 1c), an annotation panel (Fig. 1d), and finally a query prompt (Fig. 1e) to submit queries and collect the results (Fig. 1f).

The **BPAL Reasoner** provides the means to process and query the BPKB. Process schemas are imported into the BPKB from BPMN process models via the *BPMN2BPAL* interface. In order to ease the sharing and re-use of semantic meta-data, semantic information used and produced during the annotation process (i.e., reference ontologies and semantic annotations) can be exported and imported from OWL/RDF files by means of the *RDF I/O* module. The underlying rule-based reasoner can deal indifferently with RDF, RDFS and OWL (restricted to the RL profile). The *BPKB Manager* handles the set-up and the interaction with the LP engine by initializing and updating a BPKB. After populating the BPKB, inference is essentially performed by posing queries to the *XSB Prolog* engine[3], connected through a Java/Prolog interface. XSB extends conventional Prolog systems with an operational semantics based on tabling, i.e., a mechanism for storing intermediate results and avoiding to prove sub-goals more than once. In our setting, XSB has a crucial advantage with respect to other Prolog systems, because tabling ensures the termination of query evaluation over a BPKB. Finally, the *Query Manager* exposes functionalities to translate QuBPAL queries into LP queries, evaluate them, and collect the results in a textual form or export them in an XML serialization.

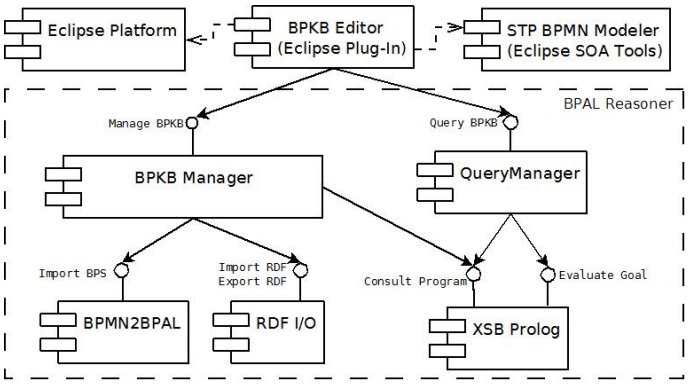



Fig. 2: Functional view of the BPAL platform

## 4 Discussion

The BPAL platform presented in this paper enables the combination of the procedural and ontological perspectives related to process knowledge in a very smooth and natural way. BPAL provides a uniform framework for modeling and semantically enriching BP models, in order to reason on properties that depend on the sequence of operations that occur during process enactment and also on the domain where the process operates. In doing this, our approach does not introduce a new BP modeling paradigm, but provides a framework where one can map and integrate knowledge represented by means of existing formalisms. This is very important from a pragmatic point of view, as one can express process-related knowledge by using standard modeling languages such as

[3] `http://xsb.sourceforge.net/`

BPMN for BP models and OWL for ontologies, and then automatically translate this knowledge into logic programs (see [8] for details), thus allowing the use of standard LP methods and tools to perform reasoning. This LP translation also enables the application of further, very sophisticated reasoning techniques recently developed in the field of logic programming. In this respect, interesting directions of future work include the enhancement of our framework with: (i) *process mining* facilities, by adopting Inductive Logic Programming techniques, such as the ones presented in [11], and (ii) verification techniques for BPs in the presence of data constraints, by following approaches based on Constraint Logic Programming such as, for instance, the one proposed in [12].

The approach has been applied to real-world scenarios coming from end-users involved in the European Project BIVEE[4] and from the pilot conducted within a collaboration between the italian CNR and SOGEI (ICT Company of the Italian Ministry of Finance). The former is related to the modeling of production processes in manufacturing oriented networked-enterprises, while the latter regards the procedural modeling of legislative decrees in the tax domain. The experiments we have conducted are encouraging and revealed the practical usability of the tool and its acceptance by business experts. On a more technical side, the LP reasoner based on the XSB system shown a significant efficiency, since very sophisticated reasoning tasks have been performed on BPs of small-to-medium size (about one hundred of activities and several thousands of reachable states) in an acceptable amount of time and memory resources.

## References

1. Hepp, M., et al. (2005). Semantic Business Process Management: A Vision Towards Using Semantic Web Services for BPM. In Proc. of Int. Conf. on e-Business Engineering, IEEE.
2. Fensel, D., et al. (2006). *Enabling Semantic Web Services: The Web Service Modeling Ontology*, Springer.
3. Burstein, M., et al. (2004). OWL-S: Semantic Markup for Web Services. W3C Member Submission, `http://www.w3.org/Submission/OWL-S/`.
4. Baryannis, G., Plexousakis, D. (2013). WSSL: A Fluent Calculus-Based Language for Web Service Specifications. In Proc. of the 25th CAiSE Conference, LNCS 7908, Springer.
5. Calvanese, D., et al. (2012). Ontology-Based Governance of Data-Aware Processes. In Proc. of the 6th Int. Conf. on Web Reasoning and Rule Systems, LNCS 7497, Springer.
6. Dimitrov, M., et al. (2007). WSMO Studio: A Semantic Web Services Modelling Environment for WSMO. In Proc. of the 4th European Conf. on the Semantic Web. LNCS 4519, Springer.
7. Born, M., et al. (2009). Supporting Execution-level Business Process Modeling with Semantic Technologies. In Proc. of the 14th DASFAA Conference. LNCS 5463, Springer.
8. Smith, F., Proietti, M. (2013). Rule-based Behavioral Reasoning on Semantic Business Processes. In Proc. of the 5th Int. Conf. on Agents and Artificial Intelligence, SciTePress.
9. Smith, F., Missikoff, M., Proietti, M. (2012). Ontology-Based Querying of Composite Services. In Business System Management and Engineering, BSME 2010, LNCS 7350, Springer.
10. Smith, F., Bianchini, D. (2012). Semi-Automatic Process Composition via Semantics-Enabled Sub-Process Selection and Ranking. Enterprise Interoperability V, I-ESA'12, Springer.
11. Lamma, E., et al. (2008). Applying Inductive Logic Programming to Process Mining. In Proc. of the 17th Int. Conf. on Inductive Logic Programming, LNCS 4894, Springer.
12. F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni. Generalization strategies for the verification of infinite state systems. *Theo. Pract. Log. Prog.*, 13(2):175–199, 2013.

[4] BIVEE: Business Innovation and Virtual Enterprise Environment (FoF-ICT-2011.7.3-285746)

# An ASP-based system for preference handling and planning.

Stefania Costantini, Giovanni De Gasperis, Niva Florio, and Claudia Zuppella

Dept. of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Via Vetoio, Coppito, L'Aquila
{stefania.costantini,giovanni.degasperis,niva.florio}@univaq.it
{claudia.zuppella}@hotmail.it

**Abstract.** Internet and mobile applications are becoming more and more helpful and widespread, while our daily lives are becoming increasingly busy and complicated. Preferences affect our actions, as well as those of intelligent agents. Answer Set Programming is a suitable framework for a decision making process which aims at supporting users by suitably planning their activities. Resourced Answer Set Programming (RASP) provides mechanisms for the optimization of answer sets according to preferences and resources. In this paper, an ASP-based system integrated with a mobile application able to plan the activities of a user is proposed, taking into account the context in which the user is located, the available resources, the geographical position and user's preferences.

**Keywords:** answer set programming, logic programming, preference handling, resource management, planning and scheduling

## 1 Introduction

Everyone's daily life is becoming increasingly complicated and busy, and preferences affect our daily actions and the way in which we try to achieve our goals. On the other side, the technology, and in particular intelligent agents, is potentially able to help us, since mobile applications and the web in particular are becoming more and more helpful and are available on affordable mobile devices. In this scenario, evaluation and adequate handling of user preferences (expressed either in an explicit or in an implicit way) is becoming increasingly useful. In fact, an adequate preference handling system can help the user in the organization of everyday life. In fact, preferences affect the way intelligent agents (including human) act, and their decision-making process.

Approaches concerning preferences in (constraint) logic programming and non-monotonic reasoning are widely studied (cf., e.g., [9], [11], [12] and [8]). More specifically, reasoning on preferences in relation to Answer Set Programming (ASP, cf., e.g., [2], [1], [10] and [13]) has been investigated (see among many [2], [4], [6] and [3]) These approaches introduce preferences either globally (e.g., [4]) or among rules (e.g., [14]). Particularly suitable to our purposes seems to be the Resourced Answer Set Programming (RASP), an extension of ASP that

supports declarative reasoning on consumption and production of resources and allows to model and plan preferences on these aspects in a very simple way (see [6], [5] and [7] for a comprehensive treatment about RASP).

In this paper, we illustrate the design of an integrated system able to plan activities of users, taking into account the context in which the user is located, the available resources, the geographical position and her/his preferences. This system, which is being implemented, is ASP- and RASP-based and can interact with the user through a mobile application. In Section 2 we briefly overview RASP, while in Section 3 the design of the system is described and in Section 4 we conclude.

## 2 RASP

Even resource production and consumption processes are connected with preferences: in fact, an agent may prefer to use some resources and not others in a given situation, or it may prefer to use available resources for obtaining a certain resource rather than others. o

Resourced Answer Set Programming (RASP) extends the ASP framework by introducing resources and preferences. With RASP we can easily specify available resources and the amount of resources needed to produce others: it supports reasoning about resource consumption and production, according to preferences. Resources are modelled by *amount-atoms* of the form $q\#a$ ($q$ is the kind of resource and $a$ its available quantity [1]). *Amount-atoms* are used in *r-facts* (RASP-facts) to represent the available quantities of resources. Thanks to *r-rules* (RASP-rules), a specific quantity of certain resources can be transformed into a specific quantity of another resource: *amount-atoms* in the body of a rule model the consumed resources, while in the head they model the resources produced by that rule. An *r-rule* can be fired several times thanks to the prefix $[N-M]$ (respectively the minimum and the maximum number of times a rule can be fired). In the following example (1), the rule can be fired from one to four times, producing from one to four portions of pasta according to available resources; to produce one portion of pasta with pesto the needed resources are 80 gr. of pasta and a little bottle of home made pesto, while to make that bottle of pesto we need some grams of many ingredients:

$$
\begin{aligned}
&[1-4]pasta\_with\_pesto\#1 \leftarrow pasta\#80, home\_made\_pesto\#1.\\
&home\_made\_pesto\#1 \leftarrow basil\#15, garlic\#1, pine\_nuts\#25,\\
&oil\#10, grated\_parmesan\#25. \qquad (1)\\
&basil\#300.\ pine\_nut\#250.\\
&garlic\#3.\ oil\#120.\ grated\_parmesan\#120.
\end{aligned}
$$

[1] For lack of space we do not consider management of quantities here, and in the example we just use integers.

If we want to clearly express which resource we prefer to use, we have to introduce *p-lists* (i.e. preference-lists), where the leftmost element of the list has high priority. For example, we can state that when we cook pasta with pesto we prefer to use home made pesto rather than pesto from supermarket:

$$[1-4]pasta\_with\_pesto\#1 \leftarrow \\ pasta\#80, (home\_made\_pesto\#1 > supermarket\_pesto\#1). \tag{2}$$

RASP provides also two kinds of conditional preference lists (*cp-list*): $pref\_when$ and *only_when* lists. Suppose that one prefers normal pasta instead of gluten free pasta, but not in case of *allergy*: the *only_when* condition is false and the *cp-list* is ignored. And suppose that when one has guests, one prefers to use home made pesto instead of supermarket pesto: if the condition *has_guest* does not hold, the *cp-list* becomes simply a disjunction.

$$[1-4]pasta\_with\_pesto\#1 \leftarrow \\ (pasta\#80 > gluten\_free\_pasta\#80\ only\_when\ notallergy), \\ (home\_made\_pesto\#1 > supermarket\_pesto\#1\ pref\_when\ has\_guest). \tag{3}$$

## 3 Framework Design

The design of an ASP-based integrated system for preference and resource management and planning is proposed here as a concrete application in real-world contexts. The main purpose of this system is to simplify our daily life, and for this reason the ASP-based system is integrated with web services to reach users everywhere, in every moment and situation via inexpensive mobile devices, typically smartphones. It is important to notice that the system is able to handle even conditional preferences and priorities among preferences. Furthermore, starting from the user's geographic location, the system optimizes the preferred answer sets according to her/his objectives, whether they are declared in an explicit or implicit way.

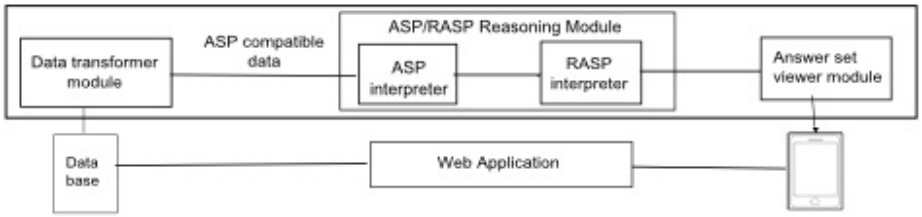



**Fig. 1.** The framework

Resources managed by the system are al least time and money, but the end-user can define others, through a web application interface, so to ensure a simple customization and personalization of the system. The framework is composed of different parts (Fig. 1): a database, a web application, an answer set viewer and the core of the system. In the database, the data needed by the system to make inferences are stored. It is a NoSQL database (http://nosql-database.org/), useful when we have to deal with a huge quantity of data that do not require a relational model. NoSQL is the *de facto* standard for mobile applications, and has bee chosen in view of efficiency and scalability, since the reasoning part is performed by the ASP module and thus no complex queries are needed. Data extracted from the database are processed by the data-transformer module, that transforms query results into an ASP-compatible format. At first, a pre-processing is needed: the system extracts the activities to plan and the preferences from the database, and then constructs suitable RASP rules and ASP facts and rules (needed to describe available resources and constraints). At this point, the files built in the previous step are processed by the ASP/RASP reasoning module, that consists of two parts: the ASP interpreter processes the data and grounds the program; the RASP interpreter processes the ground program and produces the preferred answer sets. Finally, these answer sets are transformed into a format understandable by a non academic user: the interaction between end-users and the system is made through the web application, via a smartphone. The end interface also performs user monitoring and profiling, feeding the system with new data, so as to elicit user needs, habits and preferences.

In the implementation which is being developed, the database and ASP/RASP parts are on a server, while the interface is on the mobile. In perspective (i.e., when a suitable deployment will be available) for the sake of scalability the ASP/RASP part can be moved on the mobile.

Let us provide an example of use. Assume that John wants to keep fit and hates gyms, but loves being outdoors and is very busy because of his new job: the system can produce a training program tailored for him in real time, that changes from time to time according to his preferences, resources, objectives and the place he actually is (the current user geographic and contextual location). The database has a catalogue of gym exercises, John's preferences (he hates gym, loves being outdoors, etc.), his available resources (the amount of his free time today, exercise equipment he has at home, parks nearby, and so on), resources he wants to be produced (e.g. loss of weight). If it is a sunny day, but John is very busy and has a lot of work to do: the system plans for him a run in the park near the office during his lunch break and exercises suitable for outdoor. However, if it is a rainy day and John had a rough day, the system plans for him training less hard exercises he can do while watching TV.

## 4 Concluding Remarks

The main purpose of the proposed framework is not to advance the state of the art of ASP and RASP approaches, but to realize an innovative application of logic

programming, by means of an effective integration with modern technologies affordable and understandable by everyone. The architecture includes in fact a suitable user interface, which is being designed so as to be understandable also by elder or impaired users, for which such a system can be particularly useful.

The novelty is the design of the framework itself; it aims at moving ASP and RASP outside the academic world, into the real world, and make them usable in concrete situations by ordinary people. In fact, we gave two small examples that can represent situations of our daily lives: in the first (see Section 1) the system can be a cooking teacher who offers us recipes suitable to our needs, while in the second (see Section 2) a personal trainer that offers custom workouts.

As mentioned, an implementation is under way, where the system will be able to automatically learn user preferences and objectives through machine learning mechanisms typical of proactive and adaptive agents.

## References

1. Anger, C., Schaub, T., Truszczynski, M.: Aspargus-the Dagstuhl initiative. (2004)
2. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press. (2003)
3. Bienvenu, M., Lang, J., Wilson, N., et al.: From preference logics to preference languages, and back. Proc. KR 2010.(2010)
4. Brewka, G.: Complex preferences for answer set optimization. In: Principles of Knowledge Representation and Reasoning: Proc. of the Ninth International Conference (KR2004). 213–223. (2004)
5. Costantini, S., Formisano, A., Petturiti, D.: Extending and implementing RASP. Fundamenta Informaticae, 105(1), 1–33. (2010)
6. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms, 64(1), 3–15. (2009)
7. Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation, 20(2), 53–571. (2010)
8. Cui, B., Swift, T.: Preference logic grammars: fixed point semantics and application to data standardization. Artif. Int., 138(1), 117–147. (2002)
9. Domshlak, C., Hllermeier, E., Kaci, S., Prade, H.: Preferences in AI: An overview. Artif. Intell. 175(7-8), 1037–1052. (2011)
10. Gelfond, M.: Answer sets. Foundations of Artificial Intelligence, 3, 285–316. (2008)
11. Govindarajan, K., Jayaraman, B., Mantha, S.: Preference queries in deductive databases. New Generation Computing, 19 (1), 57–86. (2001)
12. Guo, H.F., Jayaraman, B.: Mode-directed preferences for logic programs. In: Proc. of the 2005 ACM symposium on Applied computing. ACM Press, 1414–1418. (2005)
13. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In: Logic Programming and Nonmonotonic Reasoning. Springer, 1. (2007)
14. Van Nieuwenborgh, D., Vermeir, D.: Preferred answer sets for ordered logic programs. Theory and Practice of Logic Programming, 6(2), 107–167. (2006)