# IJCAR'14

## 7th International Joint Conference on Automated Reasoning

## Workshop

# Automated Deduction: Decidability, Complexity, Tractability (ADDCT)

Silvio Ghilardi, Ulrike Sattler,
Viorica Sofronie-Stokkermans

18 July 2014

Vienna Summer of Logic 2014

In the summer of 2014, Vienna hosted the largest scientific conference in the history of logic. The Vienna Summer of Logic (VSL, http://vsl2014.at) consisted of twelve large conferences and 82 workshops, attracting more than 2000 researchers from all over the world. This unique event was organized by the Kurt Gödel Society at Vienna University of Technology from July 9 to 24, 2014, under the auspices of the Federal President of the Republic of Austria, Dr. Heinz Fischer.

The conferences and workshops dealt with the main theme, logic, from three important angles: logic in computer science, mathematical logic, and logic in artificial intelligence. They naturally gave rise to respective streams gathering the following meetings:

**Logic in Computer Science / Federated Logic Conference (FLoC)**

- 26th International Conference on Computer Aided Verification (CAV)
- 27th IEEE Computer Security Foundations Symposium (CSF)
- 30th International Conference on Logic Programming (ICLP)
- 7th International Joint Conference on Automated Reasoning (IJCAR)
- 5th Conference on Interactive Theorem Proving (ITP)
- Joint meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th ACM/IEEE Symposium on Logic in Computer Science (LICS)
- 25th International Conference on Rewriting Techniques and Applications (RTA) joint with the 12th International Conference on Typed Lambda Calculi and Applications (TLCA)
- 17th International Conference on Theory and Applications of Satisfiability Testing (SAT)
- 76 FLoC Workshops
- FLoC Olympic Games (System Competitions)

**Mathematical Logic**

- Logic Colloquium 2014 (LC)
- Logic, Algebra and Truth Degrees 2014 (LATD)
- Compositional Meaning in Logic (GeTFun 2.0)
- The Infinity Workshop (INFINITY)
- Workshop on Logic and Games (LG)
- Kurt Gödel Fellowship Competition

**Logic in Artificial Intelligence**

- 14th International Conference on Principles of Knowledge Representation and Reasoning (KR)
- 27th International Workshop on Description Logics (DL)
- 15th International Workshop on Non-Monotonic Reasoning (NMR)
- 6th International Workshop on Knowledge Representation for Health Care 2014 (KR4HC)

The VSL keynote talks which were directed to all participants were given by Franz Baader (Technische Universität Dresden), Edmund Clarke (Carnegie Mellon University), Christos Papadimitriou (University of California, Berkeley) and Alex Wilkie (University of Manchester); Dana Scott (Carnegie Mellon University) spoke in the opening session. Since the Vienna Summer of Logic contained more than a hundred invited talks, it is infeasible to list them here.

The program of the Vienna Summer of Logic was very rich, including not only scientific talks, poster sessions and panels, but also two distinctive events. One was the award ceremony of the *Kurt Gödel Research Prize Fellowship Competition*, in which the Kurt Gödel Society awarded three research fellowship prizes endowed with 100.000 Euro each to the winners. This was the third edition of the competition, themed *Logical Mind: Connecting Foundations and Technology* this year.

The other distinctive event were the *1st FLoC Olympic Games* hosted by the Federated Logic Conference (FLoC) 2014. Intended as a new FLoC element, the Games brought together 12 established logic solver competitions by different research communities. In addition to the competitions, the Olympic Games facilitated the exchange of expertise between communities, and increased the visibility and impact of state-of-the-art solver technology. The winners in the competition categories were honored with Kurt Gödel medals at the FLoC Olympic Games award ceremonies.

Organizing an event like the Vienna Summer of Logic has been a challenge. We are indebted to numerous people whose enormous efforts were essential in making this vision become reality. With so many colleagues and friends working with us, we are unable to list them individually here. Nevertheless, as representatives of the three streams of VSL, we would like to particularly express our gratitude to all people who have helped to make this event a success: the sponsors and the honorary committee; the organization committee and the local organizers; the conference and workshop chairs and program committee members; the reviewers and authors; and of course all speakers and participants of the many conferences, workshops and competitions.

The Vienna Summer of Logic continues a great legacy of scientific thought that started in Ancient Greece and flourished in the city of Gödel, Wittgenstein and the Vienna Circle. The heroes of our intellectual past shaped the scientific world-view and changed our understanding of science. Owing to their achievements, logic has permeated a wide range of disciplines, including computer science, mathematics, artificial intelligence, philosophy, linguistics, and many more. Logic is everywhere – or in the language of Aristotle, πάντα πλήρη λογικῆς τέχνης.

Vienna, July 2014

Matthias Baaz, Thomas Eiter, Helmut Veith

# Preface

This volume contains the papers presented at the workshop ADDCT 2014: Automated Deduction: Decidability, Complexity, Tractability, held in Vienna on July 18, 2014, affiliated with IJCAR 2014 and RTA 2014. ADDCT 2014 is the fourth of the ADDCT workshops: the previous workshops were ADDCT 2007 (held in Bremen, together with CADE 21), ADDCT 2009 (held in Montreal together with CADE 22) and ADDCT 2013 (held in Lake Placid together with CADE 24).

The goal of ADDCT is to bring together researchers interested in

– *Decidability*, in particular decision procedures based on logical calculi such as: resolution, rewriting, tableaux, sequent calculi, or natural deduction, but also decidability in combinations of logical theories;
– *Complexity*, especially complexity analysis for fragments of first- (or higher) order logic and complexity analysis for combinations of logical theories (including parameterized complexity results);
– *Tractability* (in logic, automated reasoning, algebra, ...);
– *Application domains* for which complexity issues are essential (verification, security, databases, ontologies, ...).

With the development of computer science these problems are becoming extremely important. Although general logical formalisms (such as predicate logic or number theory) are undecidable, decidable theories or decidable fragments thereof (sometimes even with low complexity) often occur in mathematics, in program verification, in the verification of reactive, real time or hybrid systems, as well as in databases and ontologies. It is therefore important to identify such decidable fragments and design efficient decision procedures for them. It is equally important to have uniform methods (such as resolution, rewriting, tableaux, sequent calculi, ...) which can be tuned to provide algorithms with optimal complexity.

The programme of ADDCT 2014 includes one invited talk, by Nicolas Peltier, entitled "A Superposition-Based Approach to Abductive Reasoning in Equational Clausal Logic" and 5 contributed papers. We allowed the possibility of submitting to ADDCT not only original papers, but also presentation-only papers, describing work presented in papers which are already published. We thank the programme committee and the additional reviewers for their careful referee reports.

July 2014

***Silvio Ghilardi***
Università degli Studi di Milano

***Viorica Sofronie-Stokkermans***
University Koblenz-Landau and
Max-Planck-Institut für Informatik, Saarbrücken

***Ulrike Sattler***
University of Manchester

*The workshop organizers greatly benefited from using the EasyChair system.*

## Program Committee

| | |
|---|---|
| Carlos Areces | FaMAF - Universidad Nacional de Córdoba |
| Franz Baader | TU Dresden |
| Peter Baumgartner | NICTA |
| Maria Paola Bonacina | Università degli Studi di Verona |
| Christian Fermüller | TU Wien |
| Silvio Ghilardi | Università degli Studi di Milano |
| Rajeev Gore | The Australian National University |
| Matthias Horbach | MPII Saarbrücken and University Koblenz-Landau |
| Ullrich Hustadt | The University of Liverpool |
| Felix Klaedtke | NEC Europe Ltd. |
| Carsten Lutz | Universität Bremen |
| Christopher Lynch | Clarkson University |
| Silvio Ranise | FBK-Irst |
| Ulrike Sattler | The University of Manchester |
| Renate A. Schmidt | The University of Manchester |
| Viorica Sofronie-Stokkermans | University Koblenz-Landau and MPII Saarbrücken |
| Ashish Tiwari | SRI International |
| Luca Viganò | King's College London |

# Table of Contents

# A Superposition-Based Approach to Abductive Reasoning in Equational Clausal Logic

M. Echenim, N. Peltier, and S. Tourret

Univ. Grenoble Alpes, F-38000 Grenoble, France
CNRS, LIG

**Abstract.** Abduction can be defined as the process of inferring plausible explanations (or hypotheses) from observed facts (conclusions). This form of reasoning has the potential of playing a central role in system verification, particularly for identifying bugs and providing hints to correct them. We describe an approach to perform abductive reasoning that is based on the superposition calculus. The formulas we consider are sets of first-order clauses with equality, and the explanations that are allowed to be inferred are boolean combinations of equations constructed over a given finite set of ground terms. By duality, abduction can be reduced to a consequence-finding problem. We show how the inference rules of the superposition calculus can be adapted to obtain a calculus that is deductive complete for ground clauses built on the considered sets of ground terms, thus guaranteeing that all required explanations can be generated. This calculus enjoys the same termination properties as the superposition calculus: in particular, it is terminating on ground extensions of decidable theories of interest in software verification.
The number of implicates of a given equational formula is usually huge. We describe techniques for storing sets of abduced clauses efficiently, and show how usual trie-based approaches for representing sets of propositional clauses in a compact way can be adapted and extended in order to denote equational clauses up to equivalence (modulo the axioms of equality). We provide algorithms for performing redundancy pruning in an efficient way on such representations. We also identify hints for improvements and provide lines of on-going and future research.

## 1 Introduction

The verification of complex systems is often based on proving the validity, or, dually, the satisfiability of a logical formula. The behavior of the system to be verified and its intended properties are both translated into logical formulas $\phi$ and $\psi$ respectively, and the verification task boils down to proving that $\phi \Rightarrow \psi$ holds, i.e., that the formula $\phi \wedge \neg\psi$ is unsatisfiable. If the formula turns up to be satisfiable, then any model of $\phi \wedge \neg\psi$ can be viewed as a trace that generates an error. Such models help system designers locate the origin of the errors and provide hints to correct them, and most state-of-the-art SMT solvers (see for instance `http://www.smtlib.org/`) feature automated model building tools (see

[7] for more details about general-purpose model building procedures in first-order logic). However, this approach is not always satisfactory. First, there is the risk of an information overkill: indeed, the generated model may be very large and complex, and discovering the origin of the error may require a long and difficult analysis. Second, the model may be too specific, in the sense that it only corresponds to one particular execution of the system and that dismissing this single execution may not be sufficient to fix the system. There are generally many interpretations on different domains that satisfy the formula. In order to understand where the error(s) may come from, it is generally necessary to analyze many of these models and to identify common patterns. This leaves the user with the burden of having to infer the general property that can rule out all the undesired behaviors.

We present what is, to the best of our knowledge, a novel approach to this debugging problem. Rather than studying one or several models of a formula, more valuable information can be extracted from the properties that hold in *all* the models of the formula. Our goal is thus to directly infer the missing axioms, or hypotheses, that can be added in order to ensure the unsatisfiability of the input formula. These axioms can be viewed as sufficient conditions ensuring that the system is valid. Such conditions must be *plausible* and *economical*: for instance, explanations that contradict the axioms of the considered theories are obviously irrelevant. We distinguish a specific set of ground terms on which additional hypotheses are allowed to be made. These terms may be represented by a particular set of constant symbols, called *abducible constants* or simply *abducibles*. The problem boils down to determining what ground clauses containing only abducible constants are logically entailed by the formula under consideration. The negation of any of these clauses can then be viewed as a set of additional hypotheses that make the formula unsatisfiable. Indeed, by duality, computing implicants (or explanations) of a formula $\phi$ is equivalent to computing implicates (i.e., logical consequences) of $\neg\phi$.

The generation of implicants (or, by duality, of implicates) of logical formulas has many applications in system verification and artificial intelligence, and this problem has been thoroughly investigated in the context of *propositional* logic. The earlier approaches use refinements of the resolution method [32, 20, 9, 31], while more recent and more efficient proposals use decomposition-based procedures [19, 8, 18, 25, 26]. These methods mainly focus on the efficient representation of information, and develop compact ways of storing and manipulating huge sets of implicates. In contrast, the approaches handling abductive reasoning in first-order or equational logic are very scarce [24, 21, 27, 11, 29]. When dealing with equational clause sets, the addition of equality axioms leads to inefficiency and divergence in almost all but trivial cases (note that the transitivity axiom alone already admits infinitely many implicates!).

In order to compute equational implicates in an efficient way, we devise a variant of the superposition calculus [4, 30] that is deductive-complete for the considered set of abducible constants, i.e., that can generate all the clauses built on abducible constants that are logical consequences of the input clause set, up to redundancy. Our procedure is defined by enriching the standard calculus with some new mechanisms allowing the assertion of relevant hypotheses during the proof search. These additional hypotheses are stored as constraints associated with the clauses and are propagated along the derivations. If the empty clause can be generated under a conjunction of hypotheses $\mathcal{X}$, then the conjunction of the original formula and $\mathcal{X}$ is unsatisfiable. An essential feature of this approach is that the conditions are not asserted arbitrarily or eagerly, using a generate-and-test approach (which would be inefficient): instead they are *discovered* on a need basis.

Due to a lack of space, proofs are omitted. Some of the proofs can be found in [12, 16, 14].

## 2 Preliminaries

The set of *terms* $T$ is built as usual on a set of *function symbols* $\mathcal{F}$ and a set of *variables* $\mathcal{V}$. Every symbol $f \in \mathcal{F}$ is mapped to a unique *arity* $ar(f) \in \mathbb{N}$. An element of $\mathcal{F}$ of arity 0 is a *constant*. An *atom* (or *equation*) is an unordered pair of terms, written $t \simeq s$, where $t$ and $s$ are terms. A *literal* is either an atom or the negation of an atom (i.e., a *disequation*), written $t \not\simeq s$. For every literal $l$, we denote by $l^{\mathrm{c}}$ the complementary literal to $l$, which is defined as follows: $(t \simeq s)^{\mathrm{c}} \stackrel{\text{def}}{=} t \not\simeq s$ and $(t \not\simeq s)^{\mathrm{c}} \stackrel{\text{def}}{=} t \simeq s$. As usual, a non-equational atom $p(\boldsymbol{t})$ is encoded as an equation $p(\boldsymbol{t}) \simeq \mathtt{true}$ (where $\mathtt{true}$ is a special constant symbol). For readability, such an equation is sometimes written $p(\boldsymbol{t})$, and $p(\boldsymbol{t}) \not\simeq \mathtt{true}$ is written $\neg p(\boldsymbol{t})$. A *clause* is a finite multiset of literals, sometimes written as a disjunction. The empty clause is denoted by $\Box$. For every clause $C = \{l_1, \ldots, l_n\}$, $C^{\mathrm{c}}$ denotes the set of unit clauses $\{\{l_i^{\mathrm{c}}\} \mid i \in [1, n]\}$ and for every set of unit clauses $S = \{\{l_i\} \mid i \in [1, n]\}$, $S^{\mathrm{c}}$ denotes the clause $\{l_1^{\mathrm{c}}, \ldots, l_n^{\mathrm{c}}\}$.

Let $\mathcal{F}_i \subseteq \mathcal{F}$ be a set of *interpreted function symbols*, and let $\mathcal{A}$ be a set of constants, called the *abducible constants*. The set $\mathcal{F}_i$ contains symbols whose interpretation is fixed according to the considered theory (e.g. $\mathcal{F}_i$ may contain symbols $0$, $1$,$+$ ... interpreted as natural numbers and operations on them), whereas the symbols in $\mathcal{F} \setminus \mathcal{F}_i$ are interpreted arbitrarily. The set $\mathcal{A}$ is fixed by the user and contains all constants on which the abducible formulas can be constructed. Such constants are used as names for ground terms, and equations of the form $c \simeq t$ can be added to encode the fact that an abducible constant $c$ denotes the ground term $t$.

The set of variables occurring in an expression (term, atom, literal, clause) $E$ is denoted by $\mathrm{var}(E)$. If $\mathrm{var}(E) = \emptyset$ then $E$ is *ground*. Let $\mathcal{V}_a, \mathcal{V}_i \subseteq \mathcal{V}$ be special sets of variables which are allowed to be instantiated only by constants in $\mathcal{A}$ and by terms built on $\mathcal{F}_i$, respectively. A *substitution* $\sigma$ is a function mapping every variable to a term, such that if $x \in \mathcal{V}_a$ then $\sigma(x) \in \mathcal{A} \cup \mathcal{V}_a$ and if $x \in \mathcal{V}_i$ then $\sigma(x)$ is a term built on $\mathcal{F}_i \cup \mathcal{V}_i$. For every term $t$ and for every substitution $\sigma$, we denote by $t\sigma$ the term obtained from $t$ by replacing every variable $x$ by its image w.r.t. $\sigma$. The *domain* of a substitution is the set of variables $x$ such that $x\sigma \neq x$. A substitution $\sigma$ is *ground* if for every $x$ in the domain of $\sigma$, $x\sigma$ is ground.

Throughout the paper, we assume that $\prec$ denotes some fixed reduction ordering on terms such that $\texttt{true} \prec t$, for all terms $t \neq \texttt{true}$. The ordering $\prec$ is extended to atoms, literals and clauses as usual (see, e.g., [4]). We assume that $f(\boldsymbol{t}) \succ a$, for all $a \in \mathcal{A}$ and all functions and constants $f \notin \mathcal{A} \cup \mathcal{F}_i$, and, similarly, that $f(\boldsymbol{s}) \succ t$, for all terms $t$ built on $\mathcal{F}_i$ and $\mathcal{V}_i$ and for all functions and constants $f \notin \mathcal{F}_i$. This entails that we can also assume that $f(\boldsymbol{s}) \succ x$, for every $x \in \mathcal{V}_a$ and $f \notin \mathcal{A} \cup \mathcal{F}_i$ or $x \in \mathcal{V}_i$ and $f \notin \mathcal{F}_i$.

A *position* is a finite sequence of natural numbers. A position $p$ *occurs* in a term $t$ if either $p = \varepsilon$ or if $t = f(t_1, \ldots, t_n)$, $p = i.q$ with $i \in [1, n]$ and $q$ is a position in $t_i$. If $p$ is a position in $t$, the terms $t|_p$ and $t[s]_p$ are defined as follows: $t|_\varepsilon \overset{\text{def}}{=} t$, $t[s]_\varepsilon \overset{\text{def}}{=} s$, $f(t_1, \ldots, t_n)|_{i.q} \overset{\text{def}}{=} (t_i)|_q$ and $f(t_1, \ldots, t_n)[s]_{i.q} \overset{\text{def}}{=} f(t_1, \ldots, t_{i-1}, t_i[s]_q, t_{i+1}, \ldots, t_n)$.

Given a set of constants $E$, a literal $t \bowtie s$ (with $\bowtie \in \{\simeq, \not\simeq\}$) is *$E$-flat* if $t, s \in \mathcal{V} \cup E$. A clause is *$E$-flat* if all its literals are $E$-flat. The set of $E$-flat clauses is denoted by $\mathcal{C}_{\mathit{flat}}(E)$. An expression is *flat* if it is $E$-flat for some set of constants $E$.

An *interpretation* is a congruence relation on ground terms, where the interpretation of the symbols in $\mathcal{F}_i$ is fixed according to the considered theory (for instance $0 + 1 \simeq 1$ is to be interpreted as true if $0, 1$ and $+$ denote the usual interpreted symbols on natural numbers, and $n \simeq \underbrace{1 + \cdots + 1}_{k \text{ times}}$ is to be interpreted as true for some $k \in \mathbb{N}$ if $n$ is intended to denote a non-fixed natural number). An interpretation $I$ *validates* a clause $C$ if for all ground substitutions $\sigma$ of domain $\text{var}(C)$ there exists $l \in C$ such that either $l = (t \simeq s)$ and $(l, s)\sigma \in I$, or $l = (t \not\simeq s)$ and $(l, s)\sigma \notin I$.

Note that $\mathcal{F}_i$ possibly contains interpreted functions whose value is not completely specified (for instance constants denoting arbitrary natural numbers), in which case the unsatisfiability problem becomes non-semi-decidable. The interpretation of the symbols in $\mathcal{F} \setminus \mathcal{F}_i$ is fully arbitrary (as in first-order logic). In the following, completeness results will be given only for the special case in which $\mathcal{F}_i = \emptyset$.

## 3 A Constrained Superposition Calculus

In this section we define an extension of the standard superposition calculus [4, 30] with which it is possible to generate all $\mathcal{A}$-flat implicates of a considered clause set. The results in this section are extensions of [13] (see Section 3.2 for a more detailed comparison). The calculus handles constrained clauses, or *c-clauses*, the constraint part of a c-clause being a set containing all the equations and disequations needed to derive the corresponding non-constraint part from the original clause set. The calculus uses the usual rules of the superposition calculus [4]; furthermore, an additional inference rule, called the *$\mathcal{A}$-Assertion* rule, is introduced in order to add disequations to the constraints.

**Definition 1.** *A* c-clause *is a pair $[C \mid \mathcal{X}]$ where $C$ is a clause containing no symbol in $\mathcal{A} \cup \mathcal{F}_i$ and $\mathcal{X}$ is a set of literals of the form $u \bowtie v$ where both $u$ and*

*$v$ are terms built on $\mathcal{A} \cup \mathcal{F}_i$ and $\mathcal{V}_a \cup \mathcal{V}_i$. If $\mathcal{X} = \emptyset$, then we may write $C$ instead of $[C \,|\, \emptyset]$.*

A clause containing symbols in $\mathcal{A} \cup \mathcal{F}_i$ can be transformed into an equivalent c-clause by *abstracting away* such symbols and replacing them by variables in $\mathcal{V}_a$ or $\mathcal{V}_i$. For instance, the clause $p(a, 1)$ where $a \in \mathcal{A}$ and $1 \in \mathcal{F}_i$ is written $[p(x, y) \,|\, \{x \simeq a, y \simeq 1\}]$, with $x \in \mathcal{V}_a$, $y \in \mathcal{V}_i$.

Let *sel* be a *selection function*, mapping every clause $C$ to a set of literals in $C$ such that $\mathit{sel}(C)$ either contains a negative literal or contains all literals that are $\preceq$-maximal in $C$. We assume that *sel* is stable under substitutions, i.e., that for every clause $C$, for every literal $l \in C$ and for every substitution $\eta$, if $l\eta \in \mathit{sel}(C\eta)$, then $l \in \mathit{sel}(C)$.

## 3.1 Inference Rules

The calculus $\mathcal{SA}^{\prec}_{sel}$ is defined by the rules below. The standard superposition calculus (denoted by $\mathcal{SP}^{\prec}_{sel}$) coincides with $\mathcal{SA}^{\prec}_{sel}$ if $\mathcal{A} = \mathcal{F}_i = \emptyset$.

### $\mathcal{A}$-Superposition

$$\frac{[C \vee t \bowtie s \,|\, \mathcal{X}],\ \ [D \vee u \simeq v \,|\, \mathcal{Y}]}{[C \vee D \vee t[v]_p \bowtie s \,|\, \mathcal{X} \cup \mathcal{Y}]\sigma}$$

If $\bowtie \in \{\simeq, \not\simeq\}$, $\sigma$ is a most general unifier of $u$ and $t|_p$, $v\sigma \not\succeq u\sigma, s\sigma \not\succeq t\sigma$, $(t \bowtie s)\sigma \in \mathit{sel}((C \vee t \bowtie s)\sigma)$, $(u \simeq v)\sigma \in \mathit{sel}((D \vee u \simeq v)\sigma)$ and if $t|_p$ is a variable then $t|_p \in \mathrm{var}(\mathcal{X}) \cap \mathcal{V}_a$.

The main difference with the usual superposition rule is that superposition into a variable is permitted, provided the considered variable occurs in the constraint part of the clause. The reason is that these symbols do not actually represent variables in the usual sense, but rather placeholders for (unknown) constants.

### $\mathcal{A}$-Reflection

$$\frac{[C \vee t \not\simeq s \,|\, \mathcal{X}]}{[C \,|\, \mathcal{X}]\sigma}$$

If $\sigma$ is a most general unifier of $t$ and $s$ and $(t \not\simeq s)\sigma \in \mathit{sel}((C \vee t \not\simeq s)\sigma)$.

### Equational $\mathcal{A}$-Factorization

$$\frac{[C \vee t \simeq s \vee u \simeq v \,|\, \mathcal{X}]}{[C \vee s \not\simeq v \vee t \simeq s \,|\, \mathcal{X}]\sigma}$$

If $\sigma$ is a most general unifier of $t$ and $u$, $s\sigma \not\succeq t\sigma$, $v\sigma \not\succeq u\sigma$ and $(t \simeq s)\sigma \in \mathit{sel}((C \vee t \simeq s \vee u \simeq v)\sigma)$.

**$\mathcal{A}$-Assertion**

$$\frac{[x \simeq y \vee C \,|\, \mathcal{X}]}{[C \,|\, \mathcal{X} \cup \{x \not\simeq y\}]}$$

If $x, y \in \mathcal{V}_a \cup \mathcal{V}_i$, $x \simeq y \in \mathit{sel}(x \simeq y \vee C)$ and $\mathcal{A} \cup \mathcal{F}_i \neq \emptyset$.

If $\mathcal{F}_i = \emptyset$, then the calculus is essentially equivalent to the one defined in the research report [14] although the presentation is slightly different (the calculus in [14] does not abstract away the constant symbols in $\mathcal{A}$, instead it performs unification modulo equations between such constants – in other words the replacement of constants in $\mathcal{A}$ by variables is directly handled by the unification algorithm).

**Redundancy** We now adapt the standard redundancy criterion to c-clauses.

**Definition 2.** *A c-clause $[C \,|\, \mathcal{X}]$ is* $\mathcal{A}$-redundant *in a set of c-clauses $S$ if for every ground substitution $\theta$ of the variables in $[C \,|\, \mathcal{X}]$ such that $\mathcal{X}\theta$ is satisfiable, there exist c-clauses $[D_i \,|\, \mathcal{Y}_i]$ and substitutions $\sigma_i$ ($1 \leq i \leq n$), such that:*

- *$\mathcal{Y}_i\sigma_i \models \mathcal{X}\theta$ for all $i = 1, \ldots, n$;*
- *$D_1\sigma_1, \ldots, D_n\sigma_n \models C\theta$;*
- *$D_1\sigma_1, \ldots, D_n\sigma_n \preceq C\theta$.*

**Definition 3.** *A set $S$ is* $\mathcal{SA}^{\prec}_{sel}$-saturated *if every c-clause that can be derived from c-clauses in $S$ by a rule in $\mathcal{SA}^{\prec}_{sel}$ is redundant in $S$.*

## 3.2 Soundness and Deductive Completeness

The interpretation of a c-clause is defined from its set of ground instances, viewing constraints as logical implications:

**Definition 4.** *An interpretation $I$* validates *a c-clause $[C \mid \mathcal{X}]$ iff for every ground substitution $\sigma$ of domain $\mathit{var}(C) \cup \mathit{var}(\mathcal{X})$, either $I \not\models \mathcal{X}\sigma$ or $I \models C\sigma$.*

The following theorem states the soundness of $\mathcal{SA}^{\prec}_{sel}$.

**Theorem 1.** *Let $S$ be a set of c-clauses. If $C$ is deducible from $S$ by one of the rules of $\mathcal{SA}^{\prec}_{sel}$ then $S \models C$.*

In the remainder of this section, we assume that $\mathcal{F}_i$ is empty, i.e., that the considered clause set contains no interpreted symbol. Note that the logic is not semi-decidable in general in the presence of interpreted symbols, thus no procedure can be refutationally complete if $\mathcal{F}_i \neq \emptyset$.

**Definition 5.** *Let $S$ be a set of c-clauses. A clause $C$ is an* $\mathcal{A}$-implicate *of $S$ if it satisfies the following conditions.*

- *$C$ is $\mathcal{A}$-flat and ground.*
- *$C$ is not a tautology.*

– $S \models C$.

*The clause $C$ is a* prime $\mathcal{A}$-implicate *of $S$ if, moreover, $C \models D$ holds for every $\mathcal{A}$-implicate $D$ of $S$ such that $D \models C$. We denote by $I_{\mathcal{A}}(S)$ the set of $\mathcal{A}$-implicates of $S$.*

**Definition 6.** *We denote by $\mathcal{C}_{\mathcal{A}}(S)$ the set of clauses of the form $(\mathcal{X}\sigma)^{\mathrm{c}}$, where $[\Box \,|\, \mathcal{X}] \in S$ and $\mathcal{X}\sigma$ is satisfiable. We write $S \sqsubseteq S'$ if for every clause $C' \in S'$, there exists $C \in S$ such that $C \models C'$.*

The main result we have is that $\mathcal{C}_{\mathcal{A}}(S) \sqsubseteq I_{\mathcal{A}}(S)$ holds for all saturated sets $S$. However, in practice it is important not only to be able to generate the sets of *all* prime $\mathcal{A}$-implicates but also to be able to generate only some *specific* $\mathcal{A}$-implicates, satisfying some additional property (e.g., to generate unit or positive implicates, etc.). To compute such sets of implicates efficiently, it is essential to block inferences generating candidates not satisfying the desired property. We thus introduce the following definition.

**Definition 7.** *A set of clauses $\mathfrak{P}$ is* closed under subsumption *if for every $C \in \mathfrak{P}$ and for every clause $D$ such that $D$ subsumes $C$, we have $D \in \mathfrak{P}$. A c-clause $[C \,|\, \mathcal{X}]$ is* $\mathfrak{P}$-compatible *if $\mathcal{X}^{\mathrm{c}} \in \mathfrak{P}$. $\mathcal{SA}^{\prec}_{sel}(\mathfrak{P})$ denotes the calculus $\mathcal{SA}^{\prec}_{sel}$ in which all inferences that generate non-$\mathfrak{P}$-compatible c-clause are blocked.*

Examples of classes of clauses that are closed under subsumption include the following sets that are of some practical interest:

– The set of clauses $C$ such that there exists a substitution $\sigma$ such that $C\sigma$ is equivalent to a clause of length at most $k$.
– The set of positive (resp. negative) clauses.
– The set of implicants of some formula $\phi$.

Note also that the class of clause sets that are closed under subsumption is closed under union and intersection, which entails that these criteria can be combined easily.

**Theorem 2.** *Let $S_{init}$ be a set of standard clauses and let $S$ be a set of c-clauses obtained from $S_{init}$ by $\mathcal{SA}^{\prec}_{sel}(\mathfrak{P})$-saturation. If $\mathfrak{P}$ is closed under subsumption then $\mathcal{C}_{\mathcal{A}}(S) \sqsubseteq I_{\mathcal{A}}(S) \cap \mathfrak{P}$.*

In particular, if $\mathfrak{P}$ is the set of all clauses constructed over $\mathcal{A}$, then $\mathcal{SA}^{\prec}_{sel}$-saturation permits to obtain the set of all prime $\mathcal{A}$-implicates of a given set of clauses

**A concise representation of $\mathcal{A}$-implicates.** It is interesting to investigate what happens if one prevents inferences on $\mathcal{A}$-literals. The obtained calculus then essentially simulates the one introduced in [13]. It is not complete since it does not generate all $\mathcal{A}$-implicates in general, but it is complete in a restricted sense: every $\mathcal{A}$-implicate is a logical consequence of the set of $\mathcal{A}$-flat clauses generated by the calculus (note that [13] established this result only in the particular case of variable-inactive clauses [2]). More precisely, we denote by $\mathcal{SAR}^{\prec}_{sel}$ the calculus $\mathcal{SA}^{\prec}_{sel}$ in which no inference upon $\mathcal{A}$-literals is allowed, except for the $\mathcal{A}$-assertion and $\mathcal{A}$-reflection rules.

**Theorem 3.** *For all $\mathcal{SAR}^{\prec}_{sel}$-saturated sets of c-clauses $S$, we have $\mathcal{C}_{\mathcal{A}}(S) \models I_{\mathcal{A}}(S)$.*

The difference between the calculi $\mathcal{SA}^{\prec}_{sel}$ and $\mathcal{SAR}^{\prec}_{sel}$ can be summarized as follows.

– The calculus $\mathcal{SA}^{\prec}_{sel}$ explicitly generates all prime implicates in $I_{\mathcal{A}}(S)$, whereas $\mathcal{SAR}^{\prec}_{sel}$ only generates a finite representation of these implicates, in the form of an $\mathcal{A}$-flat implicant $S'$ of $I_{\mathcal{A}}(S)$. The formula $S'$ can still contain redundancies and some additional post-processing step is required to generate explicitly the prime implicates of $S'$ if needed. Any algorithm for generating prime implicates of propositional clause sets can be used for this purpose, since flat ground equational clause sets can be reduced into equivalent sets of propositional clauses by adding equality axioms. From a practical point of view, the set $I_{\mathcal{A}}(S)$ can be very large, thus $S'$ can also be viewed as a concise and suitable representation of this set.
– The calculus $\mathcal{SAR}^{\prec}_{sel}$ restricts inferences on $\mathcal{A}$-flat literals to those that actually delete such literals, possibly by transferring them to the constraint part of the clauses (the $\mathcal{A}$-Assertion and $\mathcal{A}$-Reflection rules). From a practical point of view, this entails that these literals do not need to be considered anymore in the clausal part of the c-clause: they can be transferred *systematically* in the constraints. This can reduce the number of generated clauses by an exponential factor, since a given $\mathcal{A}$-flat clause $l_1 \vee \ldots \vee l_n$ can be in principle represented by $2^n$ distinct c-clauses depending on whether $l_i$ is stored to the clausal or constraint part of the c-clause (for instance $a \simeq b$ can be represented as $[a \simeq b \,|\, \emptyset]$ or $[\square \,|\, a \not\simeq b]$). Furthermore, the number of applicable inferences is also drastically reduced, since the rules usually apply in many different ways on (selected) $\mathcal{A}$-literals.

It is possible to combine the two calculi $\mathcal{SA}^{\prec}_{sel}$ and $\mathcal{SAR}^{\prec}_{sel}$. This can be done as follows.

– Starting from a set of clauses $S$, $\mathcal{SAR}^{\prec}_{sel}$ is applied first until saturation, yielding a new set $S'$. By Theorem 3 we have $\mathcal{C}_{\mathcal{A}}(S') \equiv I_{\mathcal{A}}(S)$.
– Then $\mathcal{SA}^{\prec}_{sel}(\mathfrak{P})$ is applied on $\mathcal{C}_{\mathcal{A}}(S')$ until saturation yielding a set $S''$, where $\mathfrak{P}$ denotes the set of clauses that logically entail at least one clause in $\mathcal{C}_{\mathcal{A}}(S')$. It is clear that this set of clauses is closed under subsumption, hence by Theorem 2, we eventually obtain a set of clauses $\mathcal{C}_{\mathcal{A}}(S'') \sqsubseteq I_{\mathcal{A}}(\mathcal{C}_{\mathcal{A}}(S')) \cap \mathfrak{P}$. But $I_{\mathcal{A}}(\mathcal{C}_{\mathcal{A}}(S')) \cap \mathfrak{P} \sqsubseteq \mathcal{C}_{\mathcal{A}}(S')$, hence $\mathcal{C}_{\mathcal{A}}(S'') \sqsubseteq \mathcal{C}_{\mathcal{A}}(S')$, and $\mathcal{C}_{\mathcal{A}}(S'') \equiv I_{\mathcal{A}}(S)$. The set of clauses $\mathcal{C}_{\mathcal{A}}(S'')$ can therefore be considered as a concise representation of $I_{\mathcal{A}}(S)$. This approach is appealing since $\mathcal{C}_{\mathcal{A}}(S'')$ is in generally much smaller than $I_{\mathcal{A}}(S)$, and contrary to $\mathcal{C}_{\mathcal{A}}(S')$, this set is free of redundancies.

Another straightforward method to eliminate redundant literals from the clauses in $\mathcal{C}_{\mathcal{A}}(S')$ without having to explicitly compute the set $I_{\mathcal{A}}(S')$ is to test, for every clause $l \vee C \in \mathcal{C}_{\mathcal{A}}(S')$, whether the relation $\mathcal{C}_{\mathcal{A}}(S') \models C$, holds, if which case the literal $l$ can be safely removed. The test can be performed by using any decision procedure for ground equational logic (see for instance [28]). See [10] for a similar approach.

Another calculus was proposed in [15] to generate implicates of ground flat clauses. This calculus is very similar to the restriction of $\mathcal{SA}^{\prec}_{sel}$ to abstracted flat clauses, except that constraints are attached to the clauses as additional literals (e.g., a c-clause $[x \simeq y \,|\, \{x \not\simeq a, y \not\simeq b, c \simeq d\}]$ is represented as the clause $a \simeq b \vee c \not\simeq d$). As a consequence, several additional inferences are possible, because superposition inferences can be performed on literals in the constraints, but the total number of clause representatives is reduced as explained above. Furthermore, the redundancy criterion used in [15] is more liberal than the one considered in the present paper: in order to keep the clause sets as compact as possible, all clauses that are logically entailed by an existing clause are considered as redundant[1]. An important consequence is that a new inference rule had to be devised to perform superposition inferences simultaneously on several negative literals, so that completeness can be enforced despite this weaker redundancy criterion.

## 4 Termination Results

In this section we relate the termination behavior of $\mathcal{SA}^{\prec}_{sel}$ to that of the usual superposition calculus in the case where $\mathcal{F}_i = \emptyset$. We first introduce restricted ordering and redundancy criteria.

**Definition 8.** *For all expressions (terms, atoms, literals or clauses) $t$ and $s$, we write $t \succ_{\mathcal{A}} s$ if $t' \succ s'$ holds for all expressions $t', s'$ that are identical to $t$ and $s$ respectively, up to a renaming of constants in $\mathcal{A}$. We denote by $sel_{\mathcal{A}}$ the selection function defined as follows: for every clause $l \vee C$, $l \in sel_{\mathcal{A}}(l \vee C)$ if there exists $l', C'$ such that $l' \in sel(l' \vee C')$ and $l'$, $C'$ are identical to $l$ and $C$ respectively, up to a renaming of constant symbols in $\mathcal{A}$.*

We show that most termination results for the calculus $\mathcal{SP}^{\prec_{\mathcal{A}}}_{sel_{\mathcal{A}}}$ also apply to $\mathcal{SA}^{\prec}_{sel}$. To this purpose, we consider a restricted form of redundancy testing.

**Definition 9.** *An ordinary clause $C$ is* strongly redundant *in a set of ordinary clauses $S$ iff for every clause $C'$ that is identical to $C$ up to a renaming of constants in $\mathcal{A}$, $C'$ is $\mathcal{A}$-redundant in $S$.*

We denote by $\mathcal{E}_{\mathcal{A}}$ the set of unit clauses $a \simeq b$ or $a \not\simeq b$, with $a, b \in \mathcal{A}$. For any set of clauses $S$, let $S^{\star}$ be the set of clauses inductively defined as follows.

– $S \subseteq S^{\star}$.

[1] The redundancy criterion used in [15] is actually not exactly logical entailment, but a slight restriction of it, so that the factors of a clause are not redundant w.r.t. their premise.

– If $C$ is not strongly redundant in $S$ and is deducible from $S^\star \cup \mathcal{E}_\mathcal{A}$ by applying the rules in $\mathcal{SP}^{\prec_\mathcal{A}}_{sel_\mathcal{A}}$ (in one step) then $C \in S^\star$.

**Theorem 4.** *Let $S$ be a set of clauses. If $S^\star$ is finite then $\mathcal{SA}^{\prec}_{sel}$ terminates on $S$ (up to redundancy).*

In order to prove that $\mathcal{SA}^{\prec}_{sel}$ terminates on some class of clause sets $\mathfrak{S}$, it suffices to prove that $S^\star$ is finite, for every $S \in \mathfrak{S}$. The calculus $\mathcal{SP}^{\prec_\mathcal{A}}_{sel}$ is slightly less restrictive than the usual superposition calculus $\mathcal{SP}^{\prec}_{sel}$, since $\prec_\mathcal{A}$ is a stronger relation than $\prec$ and $sel_\mathcal{A}(C) \supseteq sel(C)$. However, most of the usual termination results for the superposition calculus still hold for $\mathcal{SP}^{\prec_\mathcal{A}}_{sel}$, because they are closed under the addition of equalities between constants and do not depend on the order of constant symbols. Similarly, redundancy testing is usually restricted to subsumption and tautology detection. In particular, all the termination results in [3] are preserved (it is easy to check that $S^\star$ is finite for the considered sets of axioms).

An interesting continuation of the present work would be to devise formal (automated) proofs of the termination of $\mathcal{SA}^{\prec}_{sel}$ on the usual theories of interest in program verification (enriched by arbitrary ground clauses). This could be done by using existing schematic calculi [22, 23] to compute a symbolic representation of the set of c-clauses $S^\star$.

## 5 Representation of Ground Flat Clauses Modulo Equality

The number of ground flat implicates of a given formula is often huge, thus it is essential in practice to provide compact representations for such sets, as well as efficient algorithms for detecting and discarding redundant clauses. In propositional logic, detecting redundant clauses is an easy task, because, for all clauses $C, D$, we have $D \models C$ iff either $C$ is a tautology or $D \subseteq C$. Thus a non-tautological clause $C$ is redundant in a clause set $S$ iff either $C$ contains two complementary literals or if there exists a clause $D \in S$ such that $D \subseteq C$. The clause set $S$ can thus be conveniently represented as a trie (a tree-based data-structure commonly used to represent sets of strings, see for instance [17]), and inclusion can be tested efficiently using standard algorithms (the literals are totally ordered and sorted to handle commutativity of $\vee$). However, in equational logic, the above equivalence does not hold: for example we have $a \simeq c \models a \not\simeq b \vee b \simeq c$ and $a \simeq c \not\subseteq a \not\simeq b \vee b \simeq c$. Representing clause sets as tries would therefore yield many undesired redundancies: for instance the clauses $a \not\simeq b \vee b \simeq c$ and $a \not\simeq b \vee a \simeq c$ would be both stored, although they are equivalent. In this section (adapted from [15]), we provide a new redundancy criterion that generalizes subsumption, together with a new technique for representing clauses, that takes into account the special properties of the equality predicate. Unless stated otherwise, we assume that all the clauses considered in this section are $\mathcal{A}$-flat and ground.

## 5.1 Testing Logical Entailment

In what follows, we introduce the notion of $C$-representativity, which is used to define the normalized form of a clause in such a way that equivalent clauses have the same normalized form, and to define a syntactic test for logical entailment. Let $C$ be a clause.

**Definition 10.** *The $C$-*representative *of a constant $a$ is the constant $a_{\downharpoonleft C} \overset{def}{=} \min_{\prec}\{b \in \mathcal{C} \mid \neg C \models b \simeq a\}$ (it is clear that all constants have a representative, since $a \not\simeq a \models C$). This notion extends to literals and clauses as follows: $(a \bowtie b)_{\downharpoonleft C} \overset{def}{=} a_{\downharpoonleft C} \bowtie b_{\downharpoonleft C}$ and $D_{\downharpoonleft C} \overset{def}{=} \{l_{\downharpoonleft C} \mid l \in D\}$. The expression $E_{\downharpoonleft C}$ is called the* projection of $E$ on $C$. *We write $E \equiv_C E'$ if $E_{\downharpoonleft C} = E'_{\downharpoonleft C}$.*

By definition, $\equiv_C$ is an equivalence relation and the following equivalences hold:

$$(a \equiv_C b) \Leftrightarrow (a \not\simeq b \models C) \Leftrightarrow (\neg C \models a \simeq b)$$

The next proposition introduces a notion of normal form for equational clauses. Intuitively, all constants $a$ occurring in a clause $C$ are replaced by their representatives $a_{\downharpoonleft C}$, and all inequations $a \simeq a_{\downharpoonleft C}$ where $a \neq a_{\downharpoonleft C}$ are appended to the clause. This normal form will permit to test efficiently whether a clause is tautological and whether two clauses are equivalent.

**Proposition 1.** *Every clause $C$ is equivalent to the clause:*

$$C_{\downarrow} \overset{def}{=} \bigvee_{a \in \mathcal{C}, a \neq a_{\downharpoonleft C}} a \not\simeq a_{\downharpoonleft C} \vee \bigvee_{a \simeq b \in C} a_{\downharpoonleft C} \simeq b_{\downharpoonleft C}$$

*Furthermore, $C$ is a tautology iff $C_{\downarrow}$ contains a literal $a \simeq a$. A non-tautological clause $C$ is in* normal form *if $C = C_{\downarrow}$ and if, moreover, all literals occur at most once in $C$.*

The following definition introduces conditions that will permit to design efficient methods to test if a given clause is redundant w.r.t. those stored in the database (forward subsumption) and conversely to delete from the database all clauses that are redundant w.r.t. a newly generated clause (backward subsumption).

**Definition 11.** *Let $C, D$ be two clauses. The clause $D$* eq-subsumes *$C$ (written $D \leq_{eq} C$) iff the two following conditions hold.*

- *$\equiv_D \subseteq \equiv_C$ (i.e. every negative literal in $D_{\downharpoonleft C}$ is a contradiction).*
- *For every positive literal $l \in D$, there exists a literal $l' \in C$ such that $l \equiv_C l'$.*

*If $S, S'$ are sets of clauses, we write $S \leq_{eq} C$ if there exists $D \in S$ such that $D \leq_{eq} C$ and we write $S \leq_{eq} S'$ if $\forall C \in S', S \leq_{eq} C$. A clause $C$ is* eq-redundant *in $S$ if either $C$ is a tautology or if there exists a clause $D \in S$ such that $D \not\equiv C$ and $D \models C$. A clause set $S$ is* eq-subsumption-minimal *if it contains no eq-redundant clause.*

Intuitively, eq-subsumption consists in verifying that $\neg C \models \neg D$. This is done by first checking that all equations in $\neg D$ are logical consequences of those in $\neg C$, which can be easily done by checking that the relation $\equiv_D \subseteq \equiv_C$ holds. Now, consider a negative literal $l$ in $\neg D$. The literal $l$ can only be entailed by $\neg C$ if $\neg C$ contains a literal $l'$ which can be reduced to $l$ by the relation $\equiv_C$.

**Theorem 5.** *Let $C$ and $D$ be two clauses. If $C$ is not a tautology then $D \models C$ iff $D \leq_{eq} C$.*

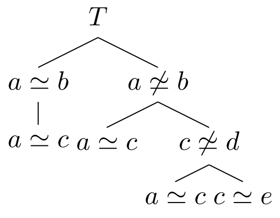
## 5.2 Clausal Trees

We define data-structures for storing and retrieving generated clauses, in such a way that the redundancy criterion introduced in Section 5.1 can be tested efficiently using Theorem 5. We use for this purpose a tree data-structure, called a *clausal tree*, specifically tailored to store sets of literals while taking into account the specific properties of the equality predicate (among these properties only transitivity is important, since reflexivity and commutativity are easy to handle by using normalization w.r.t. some ordering among constants). As in tries, the edges of the tree are labeled by literals and the leaves are either $\square$ (representing the empty clause) or $\emptyset$ (the failure node, representing an empty set of clauses). Each branch leading to a leaf $\square$ represents a clause defined as the disjunction of the literals labeling the edges in the branch. Failure nodes are useful mainly to represent empty sets – in fact they can always be eliminated by straightforward simplification rules, except if the root itself is labeled by $\emptyset$.

**Definition 12.** *A* clausal tree *is inductively defined as either $\square$, or a set of pairs of the form $(l, T')$ where $l$ is a literal and $T'$ a clausal tree. The set of clauses represented by a clausal tree $T$ is denoted by $S_c(T)$ and defined inductively as follows:*

$$S_c(T) = \begin{cases} \{\square\} & \textit{if } T = \square \\ \displaystyle\bigcup_{(l,T') \in T} \left( \bigcup_{D \in S_c(T')} l \vee D \right) & \textit{otherwise.} \end{cases}$$

*Example 1.* The structure $T$ below is a clausal tree. There is no failure node, and for readability the labels are associated with the nodes rather than with the edges leading to them.



The represented clauses $S_c(T)$ are:

$$\begin{array}{l} a \simeq b \vee a \simeq c \\ a \not\simeq b \vee a \simeq c \\ a \not\simeq b \vee c \not\simeq d \vee a \simeq c \\ a \not\simeq b \vee c \not\simeq d \vee c \simeq e \end{array}$$

We impose additional conditions on the clausal tree, in order to ensure that the represented clauses are in normal form and that sharing is maximal (for instance to ensure that there are no two edges starting from the same node and labeled by the same literal). Furthermore, the literals occurring along a given branch are ordered using the usual multiset extension of $\prec$, with the additional constraint that negative literals always occur before positive ones. The intuition behind this definition is that the negative literals occurring in a clause $C$ specify in some sense the "value" of the constant symbols in $C$ (i.e. their $C$-representative). Since the representatives of the positive literals depend on these values, it is necessary to ensure that every negative literal is known before positive ones can be stored. More formally, we define an ordering $\lhd$ on literals as follows.

– If $l$ is a negative literal and $l'$ is a positive literal, then $l \lhd l'$.
– If $l$ and $l'$ have the same sign, with $l = (b \bowtie a)$, $l' = (d \bowtie c)$, $b \succeq a$ and $d \succeq c$ then $l \lhd l'$ iff either $b \prec d$ or ($b = d$ and $a \prec c$).

**Definition 13.** *A clausal tree $T$ is* normal *if for any pair $(l, T')$ in $T$, the following conditions hold.*

– *$T' \neq \emptyset$.*
– *There is no $T'' \neq T'$ such that $(l, T'') \in T$.*
– *The literal $l$ is not of the form $a \simeq a$ or $a \not\simeq a$.*
– *All literals occurring in $T'$ are strictly greater than $l$ w.r.t. $\lhd$.*
– *If $l = a \not\simeq b$ with $a \prec b$ then $b$ does not occur in $T'$.*
– *The clausal tree $T'$ is a normal clausal tree.*

It is easy to verify that if $T$ is a normal clausal tree then all the clauses in $S_c(T)$ are in normal form.

Normal clausal trees are used to store the set of $\mathcal{A}$-implicates generated so far, and this set has to be updated every time a more general $\mathcal{A}$-implicate is generated: all the stored $\mathcal{A}$-implicates that it entails must be deleted. We now introduce two algorithms for testing whether a newly generated $\mathcal{A}$-implicate is entailed by one already stored in a clausal tree, and for deleting from a clausal tree all $\mathcal{A}$-implicates logically entailed by the newly generated one. The first algorithm (isEntailed) is invoked on a clause $C$ and a tree $T$, and returns *true* if and only if there exists a clause $D$ in $S_c(T)$ such that $D$ eq-subsumes $C$. To test this entailment, the algorithm performs a depth-first traversal of $T$ and attempts to project every encountered literal on $C$ (see Definition 10). If a literal cannot be projected, the exploration of the subtree associated to this literal is useless, so the algorithm switches to the following literal. As soon as a clause entailing $C$ is found, the traversal halts and *true* is returned. For the sake of readability we use the following notations. For any expression $E$, $E[a := b]$ denotes the expression obtained from $E$ by replacing all occurrences of $a$ by $b$. For any clausal tree $T$ and literal $l$, we denote by $l.T$ the clausal tree $l.T \overset{\text{def}}{=} \{(l, T)\}$.

**Theorem 6.** *The procedure* isEntailed *terminates in* $\mathcal{O}(\text{size}(S_c(T)) + |C| \times |S_c(T)|)$. *Moreover,* isEntailed$(C, T)$ *is true iff* $S_c(T)$ *contains a clause* $D$ *such that* $D \leq_{eq} C$.

**Algorithm 1** isEntailed$(C, T)$

```
if $T = \square$ then
    return true
end if
if $C = \square$ then
    return false
end if
$l_1 \leftarrow \min_{\lhd} \{l \in C\}$
for all $(l, T') \in T$ such that $l \geq l_1$ do
    if $l_1 = a \not\simeq b$, with $a \succ b$ then
        if $l = l_1$ then
            if isEntailed$(C \setminus \{l_1\}, T')$ then
                return true
            end if
        else if $\neg(l = a \not\simeq c)$, with $a \succ c$ then
            if isEntailed$(C \setminus \{l_1\}, (l.T')[a := b])$ then
                return true
            end if
        end if
    else if $l \in C$ then
        if isEntailed$(C \setminus \{l\}, T')$ then
            return true
        end if
    end if
end for
return false
```

The second algorithm (pruneEntailed) deletes from a tree $T$ all clauses that are eq-subsumed by $C$. It performs a depth-first traversal of $T$ and attempts to project $C$ on every clause in $S_c(T)$, deleting those on which such a projection succeeds. As soon as a projection is identified as impossible, the exploration of the associated subtree halts and the algorithm moves on to the next clause. When every literal in $C$ has been projected, all the clauses represented in the current subtree are entailed by $C$, and are therefore deleted. Afterward, the clause $C$ can itself be added in the tree (the insertion algorithm is straightforward and is omitted).

**Theorem 7.** *The procedure* PRUNEENTAILED *terminates in* $\mathcal{O}(\text{size}(S_c(T)))$. *Moreover,* PRUNEENTAILED$(C,T)$ *is a normal clausal tree and* $S_c(\text{PRUNEENTAILED}(C,T))$ *contains exactly the clauses* $D \in S_c(T)$ *such that* $C \not\leq_{eq} D$.

**Algorithm 2** PRUNEENTAILED$(C,T)$

```
if $C = \square$ then
    return $\emptyset$
end if
if $T = \square$ then
    return $T$
end if
$l_1 \leftarrow \min_{\triangleleft} \{l_i \in C\}$
for all $(l, T') \in T$ such that $l \leq l_1$ do
    if $l_1 = l$ then
        $T''$ := PRUNEENTAILED$(C \setminus \{l_1\}, T')$
    else
        if $l = a \simeq b$ then
            $T''$ := PRUNEENTAILED$(C, T')$
        else if $l = a \not\simeq b$, with $a \succ b$
          and $\nexists c$, $l_1 = a \not\simeq c$, with $a \succ c$ then
            $T''$ := PRUNEENTAILED$(C[a := b], T')$
        end if
    end if
    $T := (T \setminus \{(l, T')\}) \cup \{(l, T'')\}$
end for
return $T$
```

The presented techniques can be extended straightforwardly to store ground flat c-clauses instead of ordinary clauses: it suffices to insert a second index at every leaf in order to store the constraints. Both data-structures are handled similarly, the only difference is that a comparison must be added to verify that the clausal part of the subsuming c-clause is indeed smaller than the subsumed one.

## 6 Conclusion

Although the superposition calculus is not deductive-complete in general, we have shown that it can be adapted in order to make it able to generate all implicates defined over a given *finite* set of *ground* terms denoted by constant symbols. Furthermore, this is done in such a way that the usual termination properties of the calculus are preserved. By duality, the procedure can be used to generate abductive explanations of first-order formulas.

Our calculus shares some similarities with the constrained superposition calculi of [5, 1, 6]. However in our case the constraint and clausal parts are not defined over disjoint signatures: in contrast the Assertion rules allow one to transfer literals from the clausal part to the constraints. In [5, 1, 6] the constraints are used to store formulas that cannot be handled by the superposition calculus, whereas in our case they are used to store properties that are *asserted* instead of being proved.

A drawback with this calculus is that the user has to explicitly declare the set of abducible terms (i.e., the constants in $\mathcal{A}$). This set must be finite and must contain built-in constants (such as `true` or 0). Note that, thanks to Theorem 2, unsatisfiable or irrelevant implicates (such as $0 \simeq 1$) can be easily detected and discarded on the fly during proof search. Handling infinite (but recursive) sets of terms is possible from a theoretical point of view: it suffices to add an inference rule generating clauses of the form $a \simeq t$, where $t$ is an abducible ground terms and $a$ is a fresh constant symbol. It is straightforward to check that completeness is preserved, but of course termination is lost. A way to recover termination is to develop additional techniques to restrict the application of this rule by selecting the terms $t$. This could be done either statically, from the initial set of clauses, or dynamically, from the information deduced during proof search. Roughly speaking, the addition of an equation $a \simeq t$ is useless if it cannot generate $\mathcal{A}$-flat-clauses, and this could be proven automatically at least in some particular cases. Using existing calculi for *symbolically* computing sets of consequences of a given schema of clauses could be used for this purpose (see, e.g., [23]). Such a criterion could be useful also if the considered set of terms is finite, since it can reduce the search space.

Another possible extension is to assume that the set of abducibles is itself defined by a first-order formula (provided by the user). In this case, proving that a term is abducible becomes part of the derivation generating the corresponding implicate.

## References

1. E. Althaus, E. Kruglov, and C. Weidenbach. Superposition modulo linear arithmetic sup(la). In S. Ghilardi and R. Sebastiani, editors, *FroCoS 2009*, volume 5749 of *LNCS*, pages 84–99. Springer, 2009.
2. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic*, 10(1):129–179, January 2009.
3. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.
4. L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 3(4):217–247, 1994.
5. L. Bachmair, H. Ganzinger, and U. Waldmann. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing*, 5(3):193–212, 1994.

6. P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. In M. P. Bonacina, editor, *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2013.
7. R. Caferra, A. Leitsch, and N. Peltier. *Automated Model Building*, volume 31 of *Applied Logic Series*. Kluwer Academic Publishers, 2004.
8. T. Castell. Computation of prime implicates and prime implicants by a variant of the davis and putnam procedure. In *ICTAI*, pages 428–429, 1996.
9. J. De Kleer. An improved incremental algorithm for generating prime implicates. In *Proceedings of the National Conference on Artificial Intelligence*, pages 780–780. John Wiley & Sons ltd, 1992.
10. I. Dillig, T. Dillig, and A. Aiken. Small formulas for large programs: On-line constraint simplification in scalable static analysis. In R. Cousot and M. Martel, editors, *SAS*, volume 6337 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2010.
11. I. Dillig, T. Dillig, K. L. McMillan, and A. Aiken. Minimum satisfying assignments for smt. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 394–409. Springer, 2012.
12. M. Echenim and N. Peltier. A Calculus for Generating Ground Explanations. Technical report, CoRR, abs/1201.5954, 2012.
13. M. Echenim and N. Peltier. A Calculus for Generating Ground Explanations. In *Proceedings of the International Joint Conference on Automated Reasoning (IJCAR'12)*, volume 7364, pages 194–209. Springer LNCS, 2012.
14. M. Echenim and N. Peltier. A Superposition Calculus for Abductive Reasoning. Technical report, CoRR, abs/1406.0303, 2014.
15. M. Echenim, N. Peltier, and S. Tourret. An approach to abductive reasoning in equational logic. In *Proceedings of IJCAI'13 (International Conference on Artificial Intelligence)*, pages 3–9. AAAI, 2013.
16. M. Echenim, N. Peltier, and S. Tourret. An Approach to Abductive Reasoning in Equational Logic (long version). Technical report, LIG, 2013. http://membres-lig.imag.fr/peltier/EPT13.pdf.
17. E. Fredkin. Trie memory. *Commun. ACM*, 3(9):490–499, 1960.
18. L. Henocque. The prime normal form of boolean formulas. *Technical report at http://www.Isis.org/fiche.php*, 2002.
19. P. Jackson and J. Pais. Computing prime implicants. In *10th International Conference on Automated Deduction*, pages 543–557. Springer, 1990.
20. A. Kean and G. Tsiknis. An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, 9(2):185–206, 1990.
21. E. Knill, P. Cox, and T. Pietrzykowski. Equality and abductive residua for horn clauses. *Theoretical Computer Science*, 120:1–44, 1992.
22. C. Lynch and B. Morawska. Automatic Decidability. In *Proc. of 17th IEEE Symposium on Logic in Computer Science (LICS'2002)*, pages 7–16, Copenhagen, Denmark, July 2002. IEEE Computer Society.
23. C. Lynch, S. Ranise, C. Ringeissen, and D.-K. Tran. Automatic decidability and combinability. *Information and Computation*, 209(7):1026 – 1047, 2011.
24. P. Marquis. Extending abduction from propositional to first-order logic. In P. Jorrand and J. Kelemen, editors, *FAIR*, volume 535 of *Lecture Notes in Computer Science*, pages 141–155. Springer, 1991.
25. A. Matusiewicz, N. Murray, and E. Rosenthal. Prime implicate tries. *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 250–264, 2009.

26. A. Matusiewicz, N. Murray, and E. Rosenthal. Tri-based set operations and selective computation of prime implicates. *Foundations of Intelligent Systems*, pages 203–213, 2011.
27. M. C. Mayer and F. Pirri. First order abduction via tableau and sequent calculi. *Logic Journal of the IGPL*, 1(1):99–117, 1993.
28. O. Meir and O. Strichman. Yet another decision procedure for equality logic. In *Proceedings of the 17th International Conference on Computer Aided Verification*, CAV'05, pages 307–320, Berlin, Heidelberg, 2005. Springer-Verlag.
29. H. Nabeshima, K. Iwanuma, and K. Inoue. Solar: A consequence finding system for advanced reasoning. In M. C. Mayer and F. Pirri, editors, *TABLEAUX*, volume 2796 of *Lecture Notes in Computer Science*, pages 257–263. Springer, 2003.
30. R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.
31. L. Simon and A. Del Val. Efficient consequence finding. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 359–370, 2001.
32. P. Tison. Generalization of consensus theory and application to the minimization of boolean functions. *Electronic Computers, IEEE Transactions on*, 4:446–456, 1967.

# Satisfiability Modulo Non-Disjoint Combinations of Theories Connected via Bridging Functions

**Work in progress**

Paula Chocron[1,3], Pascal Fontaine[2], and Christophe Ringeissen[3]*

[1] Universidad de Buenos Aires, Argentina
[2] INRIA, Université de Lorraine & LORIA, Nancy, France
[3] INRIA & LORIA, Nancy, France

## 1 Introduction

Solving the satisfiability problem modulo a theory given as a union of decidable sub-theories naturally calls for combination methods. The Nelson-Oppen combination method [11] has been developed more than 30 years ago, and is now ubiquitous in SMT (Satisfiability Modulo Theories) solvers. However, this technique imposes strong assumptions on the theories in the combination; in the classical scheme [11,20], the theories notably have to be signature-disjoint and stably infinite. Many recent advances aim to go beyond these two limitations.

The design of a combination method for non-disjoint unions of theories is clearly a hard task [21,9]. To stay within the frontiers of decidability, it is necessary to impose restrictions on the theories in the combination; and at the same time, those restrictions should not be such that there is no hope of concrete applications for the combination scheme. For this reason, it is worth exploring specific classes of non-disjoint combinations of theories that appear frequently in software specification, and for which it would be useful to have a simple combination procedure. An example is the case of shared sets, where sets are represented by unary predicates [22,19,6]. In this context, the cardinality operator can also be considered; notice that this operator is a bridging function from sets to natural numbers [25]. In this paper, we investigate the case of bridging functions between data structures and a target theory (e.g. a fragment of arithmetic). Here, non-disjointness arises from connecting two disjoint theories via a third theory defining the bridging function. This problem has attracted a lot of interest in the last few years [26,8,3,16,17] due to its importance for solving verification problems expressed in a combination of data structures and arithmetic. With this work, we want to provide a synthesis of several previous contributions by different authors based on different techniques and frameworks. We mainly focus on the following papers that are closely related in terms of the considered data structures:

* This work has been partially supported by the project ANR-13-IS02-0001 of the Agence Nationale de la Recherche, by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS), and by the STIC AmSud MISMT

– Zarba presents a procedure for checking satisfiability of lists with length by using a reduction to the arithmetic [23]. The same kind of reduction is applied to multisets with multiplicity [24]. A goal was to relax the stably-infiniteness assumption in Nelson-Oppen's procedure; it is indeed possible to consider for instance multisets with a finite domain for elements. In his work, Zarba is able to reduce the problem into one expressed only in the theory of elements; the theory of lists (multisets) completely vanishes.
– Sofronie-Stokkermans [16] uses a locality property to show that the definition of the function connecting the theories can be eliminated (thanks to its instantiation by ground terms). She also considers the delicate problem of restricting models to standard ones (for data structures). A drawback of her solution is that it excludes cases where cardinality problems might arise from lack of elements to build structures that must be different.
– In [17], Suter and al. present a procedure to solve cardinality problems that is also based on a procedure for reducing bridging functions. As future work, they suggest to study relations between their work and the one in [16].

We investigate here an approach by reduction from non-disjoint to disjoint combination. The outcome of our approach is very close to that of the locality-based approach [16]. It is an alternative to a non-disjoint combination approach à la Ghilardi [9], for which some assumptions on the shared (target) theory are required. Ghilardi's approach has been applied to combine data structures with fragments of arithmetic, like Integer Offsets [13] and then Abelian groups [12]; it is however difficult to go beyond Abelian groups and consider for instance any decidable fragment of arithmetic as a shared theory. The approach by reduction does not impose such limitations, and any (decidable) fragment of arithmetic is suitable for the target (shared) theory.

The superposition calculi provide elegant and uniform ways to build satisfiability procedures for (combinations of) data-structures [2,1], possibly with bridging functions [13,12,10,4]. It appears that the approach by reduction is applicable to many data structures for which the standard superposition calculus can be used as an off-the-shelf underlying satisfiability procedure [2,1]. This approach by reduction leads to a combination procedure (see Section 3) which is indeed correct for a large class of data structure theories, ranging from the theory of equality to the theory of absolutely free data structures. Our correctness proof is not (directly) based on locality principles, but we rely on the form of Herbrand models we can expect from the data structure theories we are interested in.

When considering data structures, it is quite natural to restrict to *standard* interpretations. For instance, the standard interpretation for lists corresponds to the case where lists are interpreted as finite lists of elements. We show how to adapt the combination procedure to get a satisfiability procedure on standard interpretations, when the bridging function is *stable*. The notion of stable function encompasses both bijectivity and infinite surjectivity (defined in [17]). Moreover, we propose an enumeration procedure which has similarities with the procedure studied in [17,18,14]. This enumeration procedure allows to revisit the satisfiability problem in the standard interpretation of lists with length [8]. More generally, we conjecture that this procedure can be applied to data structures satisfying some gentle properties as defined in [7].

The work presented in this short paper corresponds to a part of [5], where the enumeration procedure mentioned above is detailed in the case of lists with a length function and its correctness is proven. We are now working on a full paper extending the short presentation given below.

## 2 The Combination Problem

We assume the reader is familiar with the classical notions and notations used in first-order logic with equality. By a slight abuse of notation, we write that a sort occurs in a signature if the sort belongs to the set of sorts of the signature.

Consider a many-sorted $\Sigma_s$-theory $T_s$ and a many-sorted $\Sigma_t$-theory $T_t$ ($s$ and $t$ stand for source and target respectively) such that $\Sigma_s$ and $\Sigma_t$ have no shared function symbols and no shared predicate symbols except the equality predicates: we have a shared equality predicate for each shared sort occurring in both $\Sigma_s$ and $\Sigma_t$. Roughly speaking, we consider a function $f$ mapping elements from $T_s$ to elements in $T_t$. This function is defined by some axioms expressed in the signature $\Sigma_s \cup \Sigma_t \cup \{f\}$. The set of axioms defining $f$ is called $T_f$.

The difficulty of building a decision procedure for the theories connected with the bridging function depends on many factors, for example, how $f$ is defined. In some cases there exists a very simple solution: since we are dealing with first order logic, $f$ always occurs applied to the appropriate number of terms. In all those occurrences $f$ could be substituted by its definition. The result may then be a disjoint problem. This is possible when $f$ is defined by an equality like $f(x) = e$, for some $\Sigma_t$-term $e$. This naive approach is not suitable for more complicated definitions. Particularly, it cannot be used for recursively defined bridging functions, like those commonly found for data structures. We introduce a procedure dedicated to that problem. The idea is to eliminate the function symbol $f$, expressing its definition using just $\Sigma_s \cup \Sigma_t$. If this maintains satisfiability, our problem has been reduced to a disjoint one, and any combination method we know for this problem can be used.

Let us now introduce the theories $T_s$, $T_t$ and $T_f$ we focus on. The theory $T_s$ is the theory of Absolutely Free Data Structures [16] (AFDS, for short) as defined below, and $T_f$ is a bridging theory connecting it to another theory $T_t$.

**Definition 1.** *Consider a set of sorts* `Elem`*, and a sort* `struct` $\notin$ `Elem`*. Let* $\Sigma$ *be a signature whose set of sorts is* $\{\texttt{struct}\} \cup \texttt{Elem}$ *and whose function symbols* $c \in \Sigma$ *(called* constructors*) have arities of the form:*

$$c : s_1 \times \cdots \times s_m \times \texttt{struct} \times \cdots \times \texttt{struct} \to \texttt{struct}$$

*where* $s_1, \ldots, s_m \in \texttt{Elem}$*. Consider the following axioms (where upper case letters denote implicitly universally quantified variables)*

$$\begin{cases} (Inj_c) & c(X_1, \ldots, X_n) = c(Y_1, \ldots, Y_n) \Rightarrow \bigwedge_{i=1}^{n} X_i = Y_i \\ (Clash_{c,d}) & c(X_1, \ldots, X_n) \neq d(Y_1, \ldots, Y_m) \\ (Acyc_{\Sigma}) & X \neq t[X] \text{ if } t \text{ is a non-variable } \Sigma\text{-term} \end{cases}$$

*The theory of Absolutely Free Data Structures over $\Sigma$ is*

$$AFDS_{\Sigma} = \Big(\bigcup_{c \in \Sigma} Inj_c\Big) \cup \Big(\bigcup_{c,d \in \Sigma, c \neq d} Clash_{c,d}\Big) \cup Acyc_{\Sigma}$$

*Example 1.* The theory of lists is an example of AFDS where the constructors are $cons : \texttt{elem} \times \texttt{list} \to \texttt{list}$ and $nil : \texttt{list}$. The theory of pairs (of numbers) is another example of AFDS where the constructor is $pair : \texttt{num} \times \texttt{num} \to \texttt{struct}$.

For sake of simplicity, we only consider Absolutely Free Data Structures without selector functions, but it would be easy to consider these additional functions in the presented framework.

Given a tuple $\boldsymbol{e}$ of terms of sorts in `Elem` and a tuple $\boldsymbol{t}$ of terms of sort `struct`, the tuple $\boldsymbol{e}, \boldsymbol{t}$ may be written $\boldsymbol{e}; \boldsymbol{t}$ to distinguish terms of sort `struct` from the other ones.

**Definition 2.** *Let $\Sigma$ be a signature as given in Definition 1 and let $\Sigma_t$ be a signature such that $\Sigma$ and $\Sigma_t$ have distinct function symbols, and may share sorts, except* `struct`*. A* bridging *function $f \notin \Sigma \cup \Sigma_t$ has arity* $\texttt{struct} \to \texttt{t}$ *where* `t` *is a sort in $\Sigma_t$. A* bridging *theory $T_f$ associated to a bridging function $f$ is has the form:*

$$T_f = \bigcup_{c \in \Sigma} \Big\{ \ \forall \boldsymbol{e} \forall t_1, \dots, t_n \ . \ f(c(\boldsymbol{e}; t_1, \dots, t_n)) = f_c(\boldsymbol{e}; f(t_1), \dots, f(t_n)) \ \Big\}$$

*where $f_c(\boldsymbol{x}; \boldsymbol{y})$ denotes a $\Sigma_t$-term.*

Remark that the notation $f_c(\boldsymbol{x}; \boldsymbol{y})$ does not mean that all elements of $\boldsymbol{x}; \boldsymbol{y}$ must occur in the term $f_c(\boldsymbol{x}; \boldsymbol{y})$, as shown in the first case of the example below.

*Example 2.* (Example 1 continued). Many useful bridging theories fall into the above definition such as:

- Length of lists: $\ell(cons(e, y)) = 1 + \ell(y), \ \ell(nil) = 0$
- Sum of lists of numbers: $lsum(cons(e, y)) = e + lsum(y), \ lsum(nil) = 0$
- Sum of pairs of numbers: $psum(pair(e, e')) = e + e'$

## 3 A Combination Procedure for Bridging Functions

We introduce a combination method for a particular non-disjoint union of theories made of a source theory, a target theory, and a third one defining the bridging theory. Let $T$ be the union of $T_s = AFDS_{\Sigma_s}$, $T_t$ and $T_f$ as given in Definition 2. For simplicity, we assume that $T_t$ is stably infinite for sorts in $\Sigma_s \cap \Sigma_t$: any $T_t$-satisfiable set of literals is satisfiable in a model of $T_t$ such that the domain associated to each sort in $\Sigma_s \cap \Sigma_t$ is infinite. The Nelson-Oppen combination method in its simplest presentation can then be reused. More generally, we could consider an arbitrary target theory $T_t$ and rely on a property of the data structure theory $T_s$ that may be stronger than stably infiniteness [15,7]. We describe below a decision procedure for checking the $T$-satisfiability of sets of ground literals.

*First phase: Variable Abstraction and Partition.* The first phase of our decision procedure takes an input set of mixed literals $\varphi$, and converts it into sets of flat (and so pure) literals. As usual, a *flat* equality is an equality $t_0 = f(t_1, \ldots, t_n)$ where each term $t_i$ is of depth 0 for $i = 0, \ldots, n$ with $n \geq 0$ (a term of depth 0 is either a constant or a variable[4]); a *flat* disequality is a disequality between two terms of depth 0. The output of this phase is an equisatisfiable formula $\varphi_{struct} \cup \varphi_{elem} \cup \varphi_t \cup \varphi_f$ such that:

- $\varphi_{struct}$ contains only flat literals of the following forms:
  - $x = y$, where $x$ and $y$ are of sort `struct`
  - $x \neq y$, where $x$ and $y$ are of sort `struct`
  - $x = k$, where $k$ is an atomic constructor
  - $x = c(\boldsymbol{e}; x_1, \ldots, x_n)$, where $c$ is a non-atomic constructor
- $\varphi_{elem}$ contains only flat literals of sorts in $\Sigma_s \backslash (\Sigma_t \cup \{\texttt{struct}\})$
- $\varphi_t$ contains only flat $\Sigma_t$-literals
- $\varphi_f$ contains only flat literals of the form $u = f(x)$

The procedure uses flattening: it introduces fresh variables to define sub-terms in compound terms as a mean to obtain pure literals.

*Second phase: Decomposition.* In this phase, we make use of the following notion: an *arrangement* over a set of variable symbols $S$ is a maximal satisfiable set of well-sorted equalities and inequalities $a = b$ or $a \neq b$, with $a, b \in S$. We build two sets of literals $\Gamma_{struct}$ and $\Gamma_t$ that will be necessary to maintain satisfiability: $\Gamma_{struct}$ and $\Gamma_t$ are initialized with the same arrangement (guessed non-deterministically) over the shared elements of sorts in $\Sigma_s \cap \Sigma_t$ occurring in both $\varphi_{struct}$ and $\varphi_t \cup \varphi_f$.

$\Gamma_{struct}$ will keep the information of equivalence between elements of sort `struct`. To do this, non-deterministically guess an arrangement over elements of sort `struct`, and add it to $\Gamma_{struct}$.

In $\Gamma_t$, add the collection of literals obtained by replacing all literals in $\varphi_{struct} \cup \varphi_f \cup \Gamma_{struct}$ with the following replacements:

1. $x = y \rightarrow f_x = f_y$, where $x, y$ are of sort `struct`
2. $u = f(x) \rightarrow u = f_x$
3. $x = k \rightarrow f_x = f_k$, where $k$ is an atomic constructor
4. $x = c(\boldsymbol{e}; x_1, \ldots, x_n) \rightarrow f_x = f_c(\boldsymbol{e}; f_{x_1}, \ldots, f_{x_n})$, where $c$ is a non-atomic constructor

*Third phase: Check.* The satisfiability check then reduces to two satisfiability check for the disjoint decision procedures, thanks to the following lemma[5]

[4] A variable can be considered as an uninterpreted constant if the satisfiability problem is viewed as a consistency problem in an expansion of the signature with fresh constants.

[5] The non-deterministic choices in the second phase have all to be checked before concluding to unsatisfiability.

**Lemma 1.** *Let $\varphi = \varphi_{struct} \cup \varphi_{elem} \cup \varphi_t \cup \varphi_f$ be a set of literals in separate form. The combination procedure described above computes $\Gamma_{struct}$ and $\Gamma_t$ such that $\varphi$ is $T$-satisfiable if and only if*

- *$\varphi_{struct} \cup \varphi_{elem} \cup \Gamma_{struct}$ is $T_s$-satisfiable, and*
- *$\varphi_t \cup \Gamma_t$ is $T_t$-satisfiable.*

*Example 3.* Consider the theory of (acyclic) lists with a length function $\ell$, and suppose we want to check the satisfiability of the set of literals $\varphi$:

$$\{x = cons(a, cons(b, z)),\ \ell(x) + 1 = \ell(z)\}$$

1. **Variable Abstraction and Partition.** $\varphi$ will be divided into:
   - $\varphi_{list} : \{y = cons(b, z), x = cons(a, y)\}$
   - $\varphi_{elem} : \emptyset$
   - $\varphi_{\mathbb{Z}} : \{c + 1 = d\}$
   - $\varphi_{\ell} : \{\ell(x) = c, \ell(z) = d\}$, where $c$ and $d$ are new variables.
2. **Decomposition.** We create the variables $\ell_x$, $\ell_y$ and $\ell_z$, and the sets:
   - $\Gamma_{list}$: we need to guess an arrangement between the list variables. Let us choose the one in which they are all different, so we will add to $\Gamma_{list}$ the set of literals: $\{x \neq y, y \neq z, z \neq x\}$ . This is the only arrangement that is satisfiable together with $\varphi_{list}$, so it is the only choice that may lead to satisfiability.
   - $\Gamma_{\mathbb{Z}}$: after performing the replacements, we will have the set of literals $\{\ell_y = \ell_z + 1, \ell_x = \ell_y + 1, \ell_x = c, \ell_z = d\}$.
3. **Check.** The set $\varphi_{list} \cup \varphi_{elem} \cup \Gamma_{list}$ is satisfiable in the theory of lists. However $\varphi_{\mathbb{Z}} \cup \Gamma_{\mathbb{Z}}$ is unsatisfiable in the theory of linear arithmetic (over the integers). The original set of literals $\varphi$ is thus unsatisfiable.

## 4 Conclusion

We briefly described a Nelson-Oppen like combination procedure for bridging functions. This procedure is not only restricted to absolutely free data structures (even if the current presentation only refers to this special case), but is also suitable for any theory in the spectrum between uninterpreted symbols and absolutely free data structures. A natural follow-up is to consider extensionality and the restriction to standard interpretations [26,8,16,17,5]. For simplicity, the presentation here is non-deterministic. However, just like in the classical Nelson-Oppen scheme, implementations will be based on deterministic, and thus more practical, approaches of the same procedure.

Several powerful and successful frameworks [3,12,16,17,4] have already been provided to handle bridging functions. We believe our approach is more lightweight, and is thus more amenable to implementation inside SMT solvers, just like superposition calculi [2,1,4] are perfectly suited for saturation-based provers.

*Acknowledgments*: we would like to thank the reviewers for their insightful comments. These will help us to complete the work briefly described in this short version.

## References

1. A. Armando, M. P. Bonacina, S. Ranise, and S. Schulz. New results on rewrite-based satisfiability procedures. *ACM Trans. Comput. Log.*, 10(1), 2009.
2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140–164, 2003.
3. F. Baader and S. Ghilardi. Connecting many-sorted theories. *J. Symb. Log.*, 72(2):535–583, 2007.
4. P. Baumgartner and U. Waldmann. Hierarchic superposition with weak abstraction. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA*, volume 7898 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2013.
5. P. Chocron. A study of the Combination Problem: dealing with multiple theories in SMT solving. Master's thesis, Universidad de Buenos Aires, Mar. 2014.
6. P. Chocron, P. Fontaine, and C. Ringeissen. A Gentle Non-Disjoint Combination of Satisfiability Procedures. In *Proc. of the 7th International Joint Conference on Automated Reasoning, IJCAR*. Springer, 2014. Extended version available as Inria Research Report, cf. `http://hal.inria.fr/hal-00985135`.
7. P. Fontaine. Combinations of theories for decidable fragments of first-order logic. In S. Ghilardi and R. Sebastiani, editors, *Frontiers of Combining Systems (FroCoS)*, volume 5749 of *LNCS*, pages 263–278. Springer, 2009.
8. P. Fontaine, S. Ranise, and C. G. Zarba. Combining lists with non-stably infinite theories. In F. Baader and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'04)*, volume 3452 of *LNCS*, pages 51–66. Springer-Verlag, 2005.
9. S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *Journal of Automated Reasoning*, 33(3-4):221–249, 2004.
10. E. Kruglov and C. Weidenbach. Superposition decides the first-order logic fragment over ground theories. *Mathematics in Computer Science*, 6(4):427–456, 2012.
11. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. on Programming Languages and Systems*, 1(2):245–257, Oct. 1979.
12. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combinable extensions of Abelian groups. In R. A. Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *LNCS*, pages 51–66. Springer, 2009.
13. E. Nicolini, C. Ringeissen, and M. Rusinowitch. Combining satisfiability procedures for unions of theories with a shared counting operator. *Fundam. Inform.*, 105(1-2):163–187, 2010.
14. T.-H. Pham and M. W. Whalen. An improved unrolling-based decision procedure for algebraic data types. In E. Cohen and A. Rybalchenko, editors, *Verified Software: Theories, Tools, Experiments - 5th International Conference, VSTTE 2013, Menlo Park, CA, USA, Revised Selected Papers*, volume 8164 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2014.
15. S. Ranise, C. Ringeissen, and C. G. Zarba. Combining data structures with non-stably infinite theories using many-sorted logic. In B. Gramlich, editor, *Frontiers of Combining Systems (FroCoS)*, volume 3717 of *LNCS*, pages 48–64. Springer, 2005.
16. V. Sofronie-Stokkermans. Locality results for certain extensions of theories with bridging functions. In R. A. Schmidt, editor, *Proc. Conference on Automated Deduction (CADE)*, volume 5663 of *LNCS*, pages 67–83. Springer, 2009.

17. P. Suter, M. Dotta, and V. Kuncak. Decision procedures for algebraic data types with abstractions. In M. V. Hermenegildo and J. Palsberg, editors, *Principles of Programming Languages (POPL)*, pages 199–210. ACM, 2010.
18. P. Suter, A. S. Köksal, and V. Kuncak. Satisfiability modulo recursive programs. In E. Yahav, editor, *Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy*, volume 6887 of *Lecture Notes in Computer Science*, pages 298–315. Springer, 2011.
19. P. Suter, R. Steiger, and V. Kuncak. Sets with cardinality constraints in satisfiability modulo theories. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference, VMCAI 2011, Austin, TX, USA*, volume 6538 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2011.
20. C. Tinelli and M. T. Harandi. A new correctness proof of the Nelson–Oppen combination procedure. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems (FroCoS)*, Applied Logic, pages 103–120. Kluwer Academic Publishers, Mar. 1996.
21. C. Tinelli and C. Ringeissen. Unions of non-disjoint theories and combinations of satisfiability procedures. *Theoretical Comput. Sci.*, 290(1):291–353, Jan. 2003.
22. T. Wies, R. Piskac, and V. Kuncak. Combining theories with shared set operations. In S. Ghilardi and R. Sebastiani, editors, *Frontiers of Combining Systems (FroCoS)*, volume 5749 of *LNCS*, pages 366–382. Springer, 2009.
23. C. G. Zarba. Combining lists with integers. In *International Joint Conference on Automated Reasoning (Short Papers), Technical Report DII 11/01*, pages 170–179. University of, 2001.
24. C. G. Zarba. Combining multisets with integers. In A. Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark*, volume 2392 of *Lecture Notes in Computer Science*, pages 363–376. Springer, 2002.
25. C. G. Zarba. Combining sets with cardinals. *J. Autom. Reasoning*, 34(1):1–29, 2005.
26. T. Zhang, H. B. Sipma, and Z. Manna. Decision procedures for term algebras with integer constraints. *Inf. Comput.*, 204(10):1526–1574, 2006.

# Finding Minimum Type Error Sources

Zvonimir Pavlinovic, Tim King, and Thomas Wies

New York University

**Abstract.** Automatic type inference is a popular feature of functional programming languages. If a program cannot be typed, the compiler typically reports a single program location in its error message. This location is the point where the type inference failed, but not necessarily the actual source of the error. Other potential error sources are not even considered. Hence, the compiler often misses the true error source, which increases debugging time for the programmer. In this paper, we present a general framework for automatic localization of type errors. Our algorithm finds all minimum error sources, where the exact definition of minimum is given in terms of a compiler-specific ranking criterion. Compilers can use minimum error sources to produce more meaningful error reports, and for automatic error correction. Our approach works by reducing type inference to constraint satisfaction. We then formulate the problem of computing minimum error sources in terms of weighted maximum satisfiability modulo theories (MaxSMT). Ranking criteria are incorporated by assigning weights to typing constraints. The reduction to MaxSMT allows us to build on decision procedures to support rich type systems.

## 1 Introduction

In functional programming languages such as OCaml and Haskell, programmers are not obliged to provide type annotations. Nevertheless, these languages guarantee strong static typing by automatically inferring types based on how expressions are used in the program. Unfortunately, the convenience of type inference comes at a cost: if the program cannot be typed, the compiler-generated error message often does not help to fix the error. Consequently, confusing error messages increase the debugging time. In this paper, we present a general framework for producing more meaningful type error messages.

A typical type inference algorithm immediately stops and reports an error at the current program location if the inferred type of the current program expression conflicts the inferred type of its context. Although fast in practice, this approach also produces poor error diagnostics. For example, consider the following simple OCaml program taken from the student benchmarks in [4]:

```
1 type 'a lst = Null | Cons of 'a * 'a lst
2 let x = Cons(3, Null)
3 let _ = print_string x
```

The standard OCaml compiler reports a type mismatch error for expression `x` on line 3, as the code before that expression is well typed. However, perhaps the programmer defined `x` incorrectly on line 2 or misused the `print_string` function. The student author of this code confirmed that the latter is the real source of the error. This simple example suggests that in order to generate useful error reports, compilers can consider several possible error causes and rank them by their relevance. In this work, we propose a general algorithm based on constraint solving that supplies compilers with error sources best matching their relevance criteria.

## 2 Overview of the Approach

Unlike typical type inference algorithms, we do not simply report the location of the first observed type inconsistency. Instead, we compute all minimum sets of expressions each of which, once corrected, yields a type correct program. The considered notion of minimality is controlled by the compiler. For example, the compiler may only be interested in those error causes that require the fewest changes to fix the program.

The crux of our approach is to reduce type error localization to the maximum satisfiability modulo theory (MaxSMT) problem. Each program expression is assigned a type variable and typing information is captured in terms of constraints over those variables. If an input program has a type error, then the corresponding set of typing constraints is unsatisfiable. We encode the compiler-specific ranking criterion by assigning weights to the generated typing constraints. A weighted MaxSMT solver then computes the satisfiable subsets of the constraints that have maximum cumulative weight. As constraints directly map to program expressions, the complements of these maximum sets represent minimum sets of program expressions that may have caused the type error.

We explain our reduction using the following OCaml program as an example:

```
let x = "hi" in not x
```

Clearly, the program is not well typed as the operation `not` on Booleans is applied to a variable `x` of type `string`. Our constraint generation procedure takes the program and generates a set of typing constraints using the OCaml type inference rules. For our example program, the constraint generation produces the following set of assertions:

$$\alpha_{not} = \mathsf{fun}(\mathsf{bool}, \mathsf{bool}) \qquad \text{[Def. of } \texttt{not}\text{]} \qquad (1)$$

$$\alpha_{app} = \mathsf{fun}(\alpha_i, \alpha_o) \qquad \texttt{not x} \qquad (2)$$

$$\alpha_{app} = \alpha_{not} \qquad \texttt{not} \qquad (3)$$

$$\alpha_i = \alpha_x \qquad \texttt{x} \qquad (4)$$

$$\alpha_x = \mathsf{string} \qquad \texttt{x = "hi"} \qquad (5)$$

Each assertion comes from a particular program expression shown to the right of the assertion. For instance, the assertion (1) is generated from the definition of the function `not` in OCaml's standard library. It specifies the type $\alpha_{not}$ of `not` as a function type from bool to bool. The generated type constraint is interpreted in the theory of inductive data types, where type variables stand for variables and other expressions, like fun and bool, are injective constructors. The generated type constraint is unsatisfiable, confirming that there is a type error.

It is easy to see that removing one assertion from the generated typing constraints makes the remaining set of assertions satisfiable. The expression corresponding to a removed constraint is regarded as an error source, i.e., correcting that expression makes the whole program well typed. In general, an *error source* is a set of program expressions that, once corrected, yield a well typed program. A *minimal error source* is an error source such that none of its proper subsets is also an error source. In our running example, each program expression is a minimal error source.

Compilers incorporate ranking criteria by assigning weights to program expressions. Smaller weights indicate that the corresponding expression is more likely contributing to the type error. Given a ranking criterion, a *minimum error source* is an error source with a minimum cumulative weight (i.e., it is also minimal). For example, consider a ranking criterion that assigns to each program expression the weight equal to the expression size in its abstract syntax tree form. Then, the expression corresponding to the assertion (2) is not a minimum error source as it has weight 2, while other minimal error sources have weight 1.

To find a minimum error source subject to a given a ranking criterion, our constraint generation procedure propagates weights from expressions to associated assertions. Then, we use a weighted MaxSMT procedure to compute a maximum satisfiable subset of these assertions. The program expressions that correspond to the complement of these assertions constitute the minimum error source. We have implemented this algorithm and applied it to the OCaml benchmarks from [4]. Our experiments showed that our approach can find minimum error sources subject to useful ranking criteria. For a detailed discussion of the algorithm and implementation we refer the reader to the full paper, which is available at `http://cs.nyu.edu/wies/publ/finding_minimum_type_error_sources.pdf`.

*Complexity and Tractability.* The decision problem that asks whether a given program is well-typed with respect to the Hindley-Milner type system is EXPTIME-complete [5,3]. Nevertheless, actual implementations of type checkers for OCaml and other languages that are based on this type system achieve good performance in practice. This is possible because type checking can be done compositionally by computing principle types using most general unifiers [6]. As explained above, we reduce the complement of the type checking problem for Hindley-Milner to satisfiability modulo the theory of inductive data types. This reduction preserves the complexity of the problem. However, a naive handling of polymorphism in the constraint generation results in an exponential explosion in the size of the generated constraints. This explosion quickly leads to intractable performance, even for moderately sized programs.

The question is then how this exponential explosion can be deferred to take advantage of the heuristics in the SMT solver that prune the search space and achieve good performance in practice. In the extended version of this abstract, we discuss a possible answer to this question. Our approach combines two ideas. The first idea is to use an encoding of type constraints for polymorphic functions that is based on stratified quantified constraints. The resulting constraints are linear in the size of the input program and remain decidable. The second idea is to make the optimistic assumption that most let-bound variables do not contribute to minimum type error sources. This idea leads to an iterative algorithm in which the types of let-bound variables are summarized by their principle types, i.e., the most general solutions of the associated typing constraints. Only if such a most general type occurs in a minimum type error source do we expand its associated constraints and reiterate. While we have not yet implemented this improved algorithm, we are confident that it will achieve considerable performance improvements in practice.

## 3 Related Work and Conclusions

Closely related to our approach is the `Seminal` [4] tool, which computes several possible error sources by repeated calls to the type checker. However, the search for error causes is based on heuristics and provides no formal guarantees that all error sources are found, respectively, that they are ranked according to some criterion. Zhang and Myers [8] encode typing information for Hindley-Milner type systems in terms of constraint graphs. The generated graphs are then analyzed to find most likely error sources by using Bayesian inference. It is unclear how this approach would support more expressive type systems. Previous approaches based on constraint solving [1,7] produce minimal but not minimum error sources and consider specific ranking criteria for specific type systems. Our approach is in part inspired by the `Bug-Assist` tool [2], which uses a MaxSAT procedure for fault localization in imperative programs. However, the problem we are solving is quite different.

In summary, we propose a novel framework for type error localization based on constraint solving. Our framework enables compilers to search for error sources of particular interest and supports rich type systems by relying on SMT solvers.

## References

1. C. Haack and J. B. Wells. Type error slicing in implicitly typed higher-order languages. *Sci. Comput. Program.*, pages 189–224, 2004.
2. M. Jose and R. Majumdar. Bug-Assist: Assisting Fault Localization in ANSI-C Programs. In *CAV*, pages 504–509. Springer-Verlag, 2011.
3. A. J. Kfoury, J. Tiuryn, and P. Urzyczyn. ML Typability is DEXTIME-Complete. In *CAAP*, pages 206–220, 1990.
4. B. Lerner, M. Flower, D. Grossman, and C. Chambers. Searching for type-error messages. In *PLDI*. ACM Press, 2007.
5. H. G. Mairson. Deciding ML Typability is Complete for Deterministic Exponential Time. In *POPL*, pages 382–401, 1990.
6. J. A. Robinson. Computational logic: The unification computation. *Machine intelligence*, 6(63-72):10–1, 1971.
7. P. J. Stuckey, M. Sulzmann, and J. Wazny. Improving type error diagnosis. In *ACM SIGPLAN Workshop on Haskell*, pages 80–91. ACM, 2004.
8. D. Zhang and A. C. Myers. Toward general diagnosis of static errors. In *POPL*, pages 569–581. ACM, 2014.

# Decidability of Iteration-free PDL with Parallel Composition

Philippe Balbiani and Joseph Boudou

Institut de recherche en informatique de Toulouse, Université de Toulouse
Joseph.Boudou@irit.fr, Philippe.Balbiani@irit.fr

**Abstract.** PRSPDL is a highly undecidable propositional dynamic logic with an operator for parallel composition of programs. This operator has a separation semantic such that a multiplicative conjunction similar to the one found in the logic of Boolean bunched implications is definable. The present work identifies an iteration-free decidable fragment of PRSPDL in which the multiplicative conjunction is still definable. A NEXPTIME complexity upper bound for the fragment is given.

## 1 Introduction

The propositional dynamic logic (PDL) is a multi-modal logic designed for reasoning about the behaviour of programs [7, 10, 9]. With each program $\alpha$ is associated the modal operator $[\alpha]$, formulas $[\alpha]\varphi$ being read "all executions of $\alpha$ from the current state lead to a state where $\varphi$ holds". The set of modal operators is inductively extended by some compound constructs: composition $(\alpha\ ;\ \beta)$ of programs $\alpha$ and $\beta$ corresponds to the composition of the accessibility relations $R(\alpha)$ and $R(\beta)$; test $\varphi?$ on formula $\phi$ corresponds to the partial identity relation in the subsets of the Kripke models in which the formula $\varphi$ is true; iteration $\alpha^{\star}$ corresponds to the reflexive and transitive closure of $R(\alpha)$. The problem with PDL is that the states of the Kripke models in which formulas are evaluated have no internal structure.

The logics of Boolean bunched implications (BBI) extend the classical propositional logic by adding a multiplicative intuitionistic conjunction operator $*$, formulas $(\varphi * \psi)$ being read "the current state can be split into two states respectively satisfying $\varphi$ and $\psi$" and a multiplicative intuitionistic implication operator $-\!\!*$, formulas $(\varphi -\!\!* \psi)$ being read "if the current state is combined with a state satisfying $\varphi$, then the resulting state satisfies $\psi$" [15]. This logic can be viewed as a modal logic with two related binary modalities. In the corresponding Kripke semantic, frames have a ternary relation $\lhd$, $x \lhd (y, z)$ denoting both that $x$ can be split into $y$ and $z$ and that $y$ and $z$ can be combined to produce $x$. BBI is semidecidable [8, 12] and undecidable [13, 6]. It forms the base ground for separations logics [16], some of them being decidable. One of the most significant application of separation logics, proposed by O'Hearn and Brookes [14, 5], consists in a Hoare-style logic for the verification of concurent programs with shared ressources. It would be interesting to have a dynamic logic related to O'Hearn and Brookes's logic as PDL is related to Hoare's logic.

The propositional dynamic logic with storing, recovering and parallel composition (PRSPDL), introduced by Benevides and al. [3], extends the syntax of PDL by adding the storing programs $s_1$ and $s_2$, the recovering programs $r_1$ and $r_2$, and the parallel composition of programs binary operator $\|$. In the corresponding Kripke semantic, all these new constructs are interpreted by means of a ternary operator $\lhd$ playing the same role as in BBI's Kripke semantic. Whenever $x \lhd (y, z)$, $y$ is related to $x$ by $s_1$, $z$ is related to $x$ by $s_2$ and $x$ is related to $y$ and $z$ by respectively $r_1$ and $r_2$. The parallel composition $(\alpha \parallel \beta)$ corresponds to the fork $R(\alpha)\nabla R(\beta)$ of $R(\alpha)$ and $R(\beta)$ defined as follows: whenever $w_1 \lhd (w_2, w_3)$, $w_6 \lhd (w_4, w_5)$, $w_2$ and $w_4$ are related by $R(\alpha)$ and $w_3$ and $w_5$ by $R(\beta)$, then $w_1$ and $w_6$ are related by $R(\alpha)\nabla R(\beta)$. A multiplicative conjonction similar to the one found in separation logics can be defined in PRSPDL by $(\varphi * \psi) \doteq \langle \varphi? \parallel \psi? \rangle \top$. Hence this logic is both a dynamic logic and a separation logic. Unfortunately, PRSPDL has been proved to be highly undecidable [2].

The purpose of this paper is to present a decidable fragment of PRSPDL in which the multiplicative conjunction is still definable. More precisely, we prove that the satisfiability problem for this fragment is in NEXPTIME. The method used is the classical selection of a finite model [4]. The main difficulty is that no comprehensive set of subformulas can be expressed in the langage for a formula of the form $[\alpha \parallel \beta]\,\varphi$. This difficulty is overcome by adding placeholders on the syntactic side and markers on the semantic side.

The next section presents the language and the semantic of the studied fragment. Section 3 adapts the usual unfolding model operation [4] to the studied frame. Section 4 presents the placeholders and markers. Section 5 proves decidabilty of the fragment, giving a complexity upper bound.

## 2 Language and Semantic

We consider the fragment $\mathrm{PPDL}_0^{\mathrm{det}}$ of PRSPDL without program iterations and the special programs $r_1$, $r_2$, $s_1$, $s_2$. This corresponds to the iteration-free PDL language with sequential compositions, tests and parallel compositions. Formaly, the language of $\mathrm{PPDL}_0^{\mathrm{det}}$ is defined as follows.

Let $\Phi_0$ be a set of propositional variables and $\Pi_0$ a countable set of atomic program variables. The sets $\Phi$ and $\Pi$ of formulas and programs are languages over the alphabet of symbols $\Phi_0 \cup \Pi_0 \cup \{;, ?, \|, \bot, [, ], (, )\}$ defined by:

$$\begin{aligned} \alpha, \beta &:= a \mid (\alpha\ ;\ \beta) \mid \phi? \mid (\alpha \parallel \beta) \\ \varphi &:= p \mid \bot \mid [\alpha]\varphi \end{aligned}$$

where $p$ range over $\Phi_0$, $a$ over $\Pi_0$, $\varphi$ over $\Phi$ and $\alpha$ and $\beta$ over $\Pi$.

The usual common operators like implication and diamond can easily be defined, respectively by $\varphi \to \psi \doteq [\varphi?]\,\psi$ and $\langle\alpha\rangle\,\varphi \doteq [[\alpha\ ;\ \varphi?]\,\bot?]\,\bot$. Moreover, the multiplicative conjunction related to BBI may be defined by $\varphi * \psi \doteq [[\varphi? \parallel \psi?]\bot?]\bot$. Althought these operators may be usefull in applications, they are not needed here and for the sake of simplicity, they will not be used in the remaining of the paper.

A model is a tuple $\mathcal{M} = (W, R, \lhd, V)$ where $W$ is a non-empty set of worlds, $R : \Pi_0 \longrightarrow \mathcal{P}\left(W^2\right)$ is a function from atomic programs to corresponding accessibility relations, $\lhd \subseteq W^3$ is the separation relation and $V : \Phi_0 \longrightarrow \mathcal{P}(W)$ is the valuation function. The language $\Phi$ is interpreted over $\lhd$-separated $\lhd$-deterministic models, i.e. models such that $\forall w, w_1, w_2, v, v_1, v_2 \in W$:

$$w \lhd (w_1, w_2) \wedge w \lhd (v_1, v_2) \Rightarrow w_1 = v_1 \wedge w_2 = v_2 \qquad (\lhd\text{-separated})$$
$$w \lhd (w_1, w_2) \wedge v \lhd (w_1, w_2) \Rightarrow w = v \qquad (\lhd\text{-deterministic})$$

The forcing relation $\vDash$ is defined by parallel induction along with the extension of $R$ to all programs:

$$\begin{array}{ll}
\mathcal{M}, w \vDash p & \text{iff } w \in V(p) \\
\mathcal{M}, w \vDash \bot & \text{never} \\
\mathcal{M}, w \vDash [\alpha]\varphi & \text{iff } \forall w',\ wR(\alpha)w' \Rightarrow \mathcal{M}, w' \vDash \varphi \\
w\ R(\alpha\ ;\ \beta)\ w' & \text{iff } \exists w'',\ wR(\alpha)w'' \wedge w''R(\beta)w' \\
w\ R(\varphi?)\ w' & \text{iff } w = w' \wedge \mathcal{M}, w \vDash \varphi \\
w\ R(\alpha \parallel \beta)\ w' & \text{iff } \exists w_1, w_2, w_3, w_4, \\
 & \qquad w \lhd (w_1, w_2) \wedge w_1R(\alpha)w_3 \wedge w_2R(\beta)w_4 \wedge w' \lhd (w_3, w_4)
\end{array}$$

As usual, a formula $\varphi \in \Phi$ is said to be *satisfiable* iff there exists a model $\mathcal{M}$ and a world $w$ such that $\mathcal{M}, w \vDash \varphi$.

In order to ease inductive reasoning about this logic, the length of both formulas and programs is defined.

**Definition 1 (Length).** *The lengths $|\varphi| \in \mathbb{N}$ and $|\alpha| \in \mathbb{N}$ of each formula $\varphi \in \Phi$ and each program $\alpha \in \Pi$ is inductively defined as:*

$$\begin{array}{cc}
|p| = 0 & |a| = 1 \\
|\bot| = 0 & |\alpha\ ;\ \beta| = |\alpha| + |\beta| + 1 \\
|[\alpha]\varphi| = |\alpha| + |\varphi| & |\varphi?| = |\varphi| + 1 \\
 & |\alpha \parallel \beta| = |\alpha| + |\beta| + 1
\end{array}$$

**Lemma 1.** *The length of any formula $\varphi \in \Phi$ is bounded by the number of occurences of symbols in $\varphi$.*

*Proof.* We prove simultaneously the corresponding property for programs: the length of any program $\alpha \in \Pi$ is bounded by the number of occurences of symbols in $\alpha$. The proof is by parallel induction on the number of occurences of symbols in $\varphi$ and $\alpha$ respectively and is left to the reader. □

**Definition 2 (Size).** *The size* size$(\alpha)$ *of any program $\alpha \in \Pi$ is inductively defined as:*

$$\begin{aligned}
\text{size}(a) &= 1 \\
\text{size}(\varphi?) &= 0 \\
\text{size}(\alpha\ ;\ \beta) &= \text{size}\,\alpha + \text{size}\,\beta \\
\text{size}(\alpha \parallel \beta) &= \text{size}\,\alpha + \text{size}\,\beta
\end{aligned}$$

**Lemma 2.** *For all $\alpha \in \Pi$,* $\text{size}(\alpha) \leq |\alpha|$.

*Proof.* By induction on the number of occurences of symbols in $\alpha$, left to the reader. □

**Lemma 3.** *Given any $\lhd$-deterministic model $\mathcal{M} = (W, R, \lhd, V)$, for all $\alpha \in \Pi$ and $w, v \in W$, if $wR(\alpha)v$ and* $\text{size}(\alpha) = 0$*, then $w = v$.*

*Proof.* By induction on the length of $\alpha$. The cases for atomic programs and tests are trivial. For sequential composition, the property holds by induction. Let suppose $wR(\alpha \parallel \beta)v$ and $\text{size}(\alpha \parallel \beta) = 0$. Then there exists $w_1, w_2, w_3, w_4 \in W$ such that $w \lhd (w_1, w_2)$, $w_1R(\alpha)w_3$, $w_2R(\beta)w_4$ and $v \lhd (w_3, w_4)$. By induction, $w_1 = w_3$ and $w_2 = w_4$. Since $\mathcal{M}$ is $\lhd$-deterministic, $w = v$. □

## 3 Model Unfolding

**Definition 3 (Bounded morphism).** *Given two $\lhd$-separated $\lhd$-deterministic models $\mathcal{M} = (W, R, \lhd, V)$ and $\mathcal{M}' = (W', R', \lhd', V')$, a mapping $f : \mathcal{M} \longrightarrow \mathcal{M}'$ is called* a bounded morphism *iff it satisfies the following conditions for all $v, w, w_1, w_2 \in W$, $w', w_1', w_2' \in W'$ and $a \in \Pi_0$:*

$$w \text{ and } f(w) \text{ satisfy the same propositional variables} \tag{1}$$
$$vR(a)w \Rightarrow f(v)R'(a)f(w) \tag{2}$$
$$f(v)R'(a)w' \Rightarrow \exists w,\ f(w) = w' \text{ and } vR(a)w \tag{3}$$
$$w \lhd (w_1, w_2) \Rightarrow f(w) \lhd' (f(w_1), f(w_2)) \tag{4}$$
$$f(w) \lhd' (w_1', w_2') \Rightarrow \exists w_1, w_2, f(w_1) = w_1', f(w_2) = w_2' \text{ and } w \lhd (w_1, w_2) \tag{5}$$
$$w' \lhd' (f(w_1), f(w_2)) \Rightarrow \exists w,\ f(w) = w' \text{ and } w \lhd (w_1, w_2) \tag{6}$$

**Proposition 1.** *If $f$ is a bounded morphism from $\mathcal{M}$ to $\mathcal{M}'$, then for all $w \in W$ and $\varphi \in \Phi$, $\mathcal{M}, w \vDash \varphi$ iff $\mathcal{M}', f(w) \vDash \varphi$.*

*Proof.* By simultaneous induction on the length of both $\varphi \in \Phi$ and $\alpha \in \Pi$, the following properties can be proved:

$$\mathcal{M}, w \vDash \varphi \Leftrightarrow \mathcal{M}', f(w) \vDash \varphi$$
$$vR(\alpha)w \Rightarrow f(v)R'(\alpha)f(w)$$
$$f(v)R'(\alpha)w' \Rightarrow \exists w,\ f(w) = w' \text{ and } vR(\alpha)w$$
□

Given a $\lhd$-separated $\lhd$-deterministic countable model $\mathcal{M}' = (W', R', \lhd', V')$ and a world $w_0' \in W'$, we will construct the *unfolding* of $\mathcal{M}'$ at $w_0'$ as follows. Let $W_\infty$ be a countably infinite set. For all $k \in \mathbb{N}$ we will construct the tuple $T_k = (W_k, R_k, \lhd_k, h_k, d_k, p_k)$ such that $W_k \subseteq W_\infty$, $\mathcal{M}_k = (W_k, R_k, \lhd_k, V_k)$ is a model, $h_k : W_k \longrightarrow W'$ is a function satisfying the conditions (2) and (4) of Definition 3, $d_k : W_k \longrightarrow \mathbb{Q}$ gives the *degree* of worlds in $W_k$ and $p_k : W_k \longrightarrow \mathbb{Z}$ gives the *depth* of worlds in $W_k$.

We define the following defects:

1. A tuple $(v, a, w') \in W_\infty \times \Pi_0 \times W'$ is a defect of type 1 for $T_k$ iff $v \in W_k$, $h_k(v) R'(a) w'$ and $\forall w \in W_k$, $h_k(w) = w'$ implies $(v, w) \notin R_k(a)$;
2. A tuple $(v, w'_1, w'_2) \in W_\infty \times W' \times W'$ is a defect of type 2 for $T_k$ iff $v \in W_k$, $h_k(v) \lhd' (w'_1, w'_2)$ and $\forall w_1, w_2 \in W_k$, $h_k(w_1) = w'_1 \wedge h_k(w_2) = w'_2$ implies $(v, w_1, w_2) \notin \lhd_k$;
3. A tuple $(w', w_1, w_2) \in W' \times W_\infty \times W_\infty$ is a defect of type 3 for $T_k$ iff $w_1, w_2 \in W_k$, $w' \lhd' (h_k(w_1), h_k(w_2))$ and $\forall w \in W_k$, $h_k(w) = w'$ implies $(w, w_1, w_2) \notin \lhd_k$.

As all sets $W_\infty, W'$ and $\Pi_0$ are countable, there exists an enumeration $\delta_0, \delta_1 \ldots$ of tuples belonging to $(W_\infty \times \Pi_0 \times W') \cup (W_\infty \times W' \times W') \cup (W' \times W_\infty \times W_\infty)$ where each tuple appears infinitely often.

As a first step, let $w_0 \in W_\infty$, $W_0 = \{w_0\}$, $R_0(a) = \emptyset$ for all $a \in \Pi_0$, $\lhd_0 = \emptyset$, $h_0(w_0) = w'_0$, $d_0(w_0) = 0$ and $p_0(w_0) = 0$.

Next, given the $k$-tuple $T_k$, if $\delta_k$ is not a defect for $T_k$ then $T_{k+1} = T_k$. Otherwise, depending on the type of $\delta_k$ one of the following rule is applied.

1. When $\delta_k$ is of type 1, let $\delta_k = (v, a, w')$ and $w^+$ be a fresh element from $W_\infty$

$$\begin{aligned} W_{k+1} &= W_k \cup \{w^+\} \\ R_{k+1}(a) &= R_k(a) \cup \{(v, w^+)\} \\ R_{k+1}(b) &= R_k(b) \text{ for all } b \neq a \\ \lhd_{k+1} &= \lhd_k \end{aligned}$$

$$h_{k+1}(w) = \begin{cases} w' & \text{if } w = w^+ \\ h_k(w) & \text{otherwise} \end{cases}$$

$$d_{k+1}(w) = \begin{cases} d_k(v) + 1 & \text{if } w = w^+ \\ d_k(w) & \text{otherwise} \end{cases}$$

$$p_{k+1}(w) = \begin{cases} p_k(v) & \text{if } w = w^+ \\ p_k(w) & \text{otherwise} \end{cases}$$

2. When $\delta_k$ is of type 2, let $\delta_k = (v, w'_1, w'_2)$ and $w_1^+$ and $w_2^+$ be fresh elements from $W_\infty$

$$\begin{aligned} W_{k+1} &= W_k \cup \{w_1^+, w_2^+\} \\ R_{k+1}(a) &= R_k(a) \text{ for all } a \\ \lhd_{k+1} &= \lhd_k \cup \{(v, w_1^+, w_2^+)\} \end{aligned}$$

$$h_{k+1}(w) = \begin{cases} w'_1 & \text{if } w = w_1^+ \\ w'_2 & \text{if } w = w_2^+ \\ h_k(w) & \text{otherwise} \end{cases}$$

$$d_{k+1}(w) = \begin{cases} \frac{1}{2} d_k(v) & \text{if } w \in \{w_1^+, w_2^+\} \\ d_k(w) & \text{otherwise} \end{cases}$$

$$p_{k+1}(w) = \begin{cases} p_k(v) + 1 & \text{if } w \in \{w_1^+, w_2^+\} \\ p_k(w) & \text{otherwise} \end{cases}$$

3. When $\delta_k$ is of type 3, let $\delta_k = (w', w_1, w_2)$ and $w^+$ be a fresh element from $W_\infty$

$$\begin{aligned} W_{k+1} &= W_k \cup \{w^+\} & h_{k+1}(w) &= \begin{cases} w' & \text{if } w = w^+ \\ h_k(w) & \text{otherwise} \end{cases} \\ R_{k+1}(a) &= R_k(a) \text{ for all } a \\ \lhd_{k+1} &= \lhd_k \cup \{(w^+, w_1, w_2)\} & d_{k+1}(w) &= \begin{cases} d_k(w_1) + d_k(w_2) & \text{if } w = w^+ \\ d_k(w) & \text{otherwise} \end{cases} \end{aligned}$$

$$p_{k+1}(w) = \begin{cases} p_k(w_1) - 1 & \text{if } w = w^+ \text{ and } p_k(w_1) = p_k(w_2) \\ -1 & \text{if } w = w^+ \text{ and } p_k(w_1) \neq p_k(w_2) \\ p_k(w) & \text{otherwise} \end{cases}$$

Then let $W$, $R$ and $\lhd$ be the union of respectively $W_k$, $R_k$ and $\lhd_k$ on all $k \in \mathbb{N}$. We further define the functions $h(w) = h_{k_w}(w)$, $d(w) = d_{k_w}(w)$ and $p(w) = p_{k_w}(w)$ for all $w \in W$, $k_w$ being the smallest $k$ such that $w \in W_k$. Finaly, let $V(p) = \{w \in W \mid h(w) \in V'(p)\}$ for all $p \in \Phi_0$. The model $\mathcal{M} = (W, R, \lhd, V)$ is the *unfolding* of $\mathcal{M}'$ at $w'_0$. The initial world $w_0$ is called the *root* of the unfolding.

**Lemma 4.** *For all $w, v, w_1, w_2 \in W$, $k \in \mathbb{N}$, $a \in \Pi_0$, $w' \in W'$, $r \in \mathbb{Q}$ and $z \in \mathbb{Z}$, the following implications hold:*

$$w \in W_k \Rightarrow w \in W_{k+1} \tag{7}$$
$$vR_k(a)w \Rightarrow vR_{k+1}(a)w \tag{8}$$
$$w \lhd_k (w_1, w_2) \Rightarrow w \lhd_{k+1} (w_1, w_2) \tag{9}$$
$$h_k(w) = w' \Rightarrow h_{k+1}(w) = w' \tag{10}$$
$$d_k(w) = r \Rightarrow d_{k+1}(w) = r \tag{11}$$
$$p_k(w) = z \Rightarrow p_{k+1}(w) = z \tag{12}$$

*Proof.* Each implication is easily checked for each type of the defect $\delta_k$. □

**Lemma 5.** *The model $\mathcal{M}$ is $\lhd$-separated and $\lhd$-deterministic.*

*Proof.* It suffices to check that for all $k \in \mathbb{N}$, the model $(W_k, R_k, \lhd_k, V\lfloor_{W_k})$ is $\lhd$-separated and $\lhd$-deterministic, which is left to the reader. □

**Lemma 6.** *The map $h$ is a bounded morphism from $\mathcal{M}$ to $\mathcal{M}'$.*

*Proof.* Condition (1) holds by definition of $V$. The fact that for all $k \in \mathbb{N}$, $h_k$ satisfies the conditions (2) and (4) is easily checked. Hence the map $h$ satisfies the conditions (2) and (4). By our step-by-step construction, it also satisfies the conditions (3), (5) and (6). □

Moreover, the degree $d$ and the depth $p$ as constructed above have the followings properties which will be useful in the next sections.

**Property 1.** *For all $w, w_1, w_2 \in W$, if $w \lhd (w_1, w_2)$ then $d(w) = d(w_1) + d(w_2)$.*

*Proof.* Each tuple $(w, w_1, w_2) \in \lhd$ has been added by case 2 or 3 and in both cases the property holds. $\square$

**Property 2.** *For all $v, w \in W$ and $\alpha \in \Pi$, if $vR(\alpha)w$ then $d(w) = d(v) + \text{size}(\alpha)$.*

*Proof.* The proof is by induction on the length of $\alpha$, left to the reader. $\square$

**Property 3.** *For all $w, w_1, w_2 \in W$, if $w \lhd (w_1, w_2)$ and $p(w) \geq 0$ then $p(w_1) = p(w_2) = p(w) + 1$.*

*Proof.* Each tuple $(w, w_1, w_2) \in \lhd$ has been added by case 2 or 3. The former case is trivial. In the latter case, as $p(w) \neq -1$, the property holds too. $\square$

**Property 4.** *For all $v, w \in W$ and $\alpha \in \Pi$, if $vR(\alpha)w$ and $p(v) \geq 0$ then $p(v) = p(w)$.*

*Proof.* By induction on the length of $\alpha$, left to the reader. $\square$

## 4 Placeholders and Markers

### 4.1 Subformulas with Placeholders

Using the same sets $\Phi_0$ and $\Pi_0$ of propositional variables and atomic programs as before, the sets $\Phi^+$ and $\Pi^+$ of formulas and programs with indices and placeholders are defined by parallel induction:

$$\alpha, \beta := a \mid (\alpha \, ; \beta) \mid \varphi? \mid (\alpha \parallel_i \beta)$$
$$\varphi := p \mid (i, j) \mid \bot \mid [\alpha]\varphi$$

where $p$ range over $\Phi_0$, $a$ over $\Pi_0$, $i$ over $\mathbb{N}$, $j$ over $\{1, 2\}$, $\varphi$ over $\Phi^+$ and $\alpha$ and $\beta$ over $\Pi^+$. The integers below the parallel composition symbols are called *indices*. The atomic formulas of the form $(i, j)$ are called *placeholders*. The definitions of lengths and size of Section 2 are extended to $\Phi^+$ and $\Pi^+$ by considering placeholders as new propositional variables and ignoring indices.

A formula $\varphi \in \Phi^+$ with indices and placeholders is an *annotated formula with placeholders* if each integer appears at most once as an index in it. The subset $\Phi_{PH} \subset \Phi^+$ of all annotated formula with placeholders is called the *annotated language with placeholders*, and $\Pi_{PH}$ is the corresponding set of annotated programs with placeholders. For each annotated formula with placeholders $\varphi \in \Phi_{PH}$, $I_\varphi \subseteq \mathbb{N}$ denotes the set of indices appearing in $\varphi$.

An annotated formula with placeholders $\varphi \in \Phi_{PH}$ is a *pure annotated formula* if it contains no placeholders. The subset $\Phi_{\mathbb{N}} \subset \Phi_{PH}$ of all pure annotated formula is called the *pure annotated language*, and $\Pi_{\mathbb{N}}$ is the corresponding set of pure annotated programs. There exists a forgetful epimorphism $\overline{\cdot} : \Phi_{\mathbb{N}} \longrightarrow \Phi$ associating to each pure annotated formula $\varphi_{\mathbb{N}}$ the formula $\overline{\varphi_{\mathbb{N}}}$ obtained by removing all indices in $\varphi_{\mathbb{N}}$. $\overline{\varphi_{\mathbb{N}}}$ is called the unannotated formula of $\varphi_{\mathbb{N}}$ and $\varphi_{\mathbb{N}}$ an annotation of $\overline{\varphi_{\mathbb{N}}}$.

**Definition 4 (Subformulas with placeholders).** *The function* $\mathrm{sf} : \mathbb{N} \times \mathbb{N} \times \Phi_{PH} \longrightarrow \mathcal{P}(\mathbb{N} \times \mathbb{N} \times \Phi_{PH})$ *is inductively defined by:*

$$\begin{aligned}
\mathrm{sf}\,(d,p,q) &= \{(d,p,q)\} \text{ for all } q \in \Phi_0 \\
\mathrm{sf}\,(d,p,(i,j)) &= \{(d,p,(i,j))\} \text{ for all } (i,j) \in \mathbb{N} \times \{1,2\} \\
\mathrm{sf}\,(d,p,\bot) &= \{(d,p,\bot)\} \\
\mathrm{sf}\,(d,p,[a]\varphi) &= \{(d,p,[a]\varphi)\} \cup \mathrm{sf}(d+1,p,\varphi) \\
\mathrm{sf}\,(d,p,[\varphi?]\psi) &= \{(d,p,[\varphi?]\psi)\} \cup \mathrm{sf}(d,p,\varphi) \cup \mathrm{sf}(d,p,\psi) \\
\mathrm{sf}\,(d,p,[\alpha\,;\,\beta]\varphi) &= \{(d,p,[\alpha\,;\,\beta]\varphi)\} \cup \mathrm{sf}(d,p,[\alpha][\beta]\varphi) \\
\mathrm{sf}\,(d,p,[\alpha \parallel_i \beta]\,\varphi) &= \{(d,p,[\alpha \parallel_i \beta]\,\varphi)\} \cup \mathrm{sf}(d,p+1,[\alpha](i,1)) \cup \\
&\quad\ \mathrm{sf}(d,p+1,[\beta](i,2)) \cup \mathrm{sf}(d+\mathrm{size}(\alpha \parallel \beta),p,\varphi)
\end{aligned}$$

*For all pure annotated formula* $\varphi \in \Phi_{\mathbb{N}}$ *and* $(d,p,\psi) \in \mathrm{sf}(0,0,\varphi)$*,* $\psi$ *is called a* subformula with placeholders *of* $\varphi$ *of degree* $d$ *and depth* $p$*. The set of all subformulas with placeholders of* $\varphi$ *is denoted by* $\mathrm{SF}(\varphi)$*.*

**Lemma 7.** *For all* $d_1, d_2, p_1, p_2 \in \mathbb{N}$ *and all* $\varphi_1, \varphi_2 \in \Phi_{PH}$*:*

$$(d_1,p_1,\varphi_1) \in \mathrm{sf}(d_2,p_2,\varphi_2) \Leftrightarrow \mathrm{sf}(d_1,p_1,\varphi_1) \subseteq \mathrm{sf}(d_2,p_2,\varphi_2)$$

*Proof.* The right to left direction is trivial. The left to right direction is by induction on $|\varphi_2|$. When $\varphi_2 \in \Phi_0 \cup (\mathbb{N} \times \{1,2\})$ or $\varphi_2 = \bot$, $\mathrm{sf}(d_2,p_2,\varphi_2)$ is a singleton hence $(d_1,p_1,\varphi_1) = (d_2,p_2,\varphi_2)$. When $\varphi_2 = [a]\varphi$ then either $(d_1,p_1,\varphi_1) = (d_2,p_2,\varphi_2)$ or $(d_1,p_1,\varphi_1) \in \mathrm{sf}(d_2+1,p_2,\varphi)$ and by induction $\mathrm{sf}(d_1,p_1,\varphi_1) \subseteq \mathrm{sf}(d_2+1,p_2,\varphi) \subseteq \mathrm{sf}(d_2,p_2,\varphi_2)$. The others cases are similar and left to the reader. □

**Corollary 1.** *For all* $d,p,i \in \mathbb{N}$*,* $\alpha, \beta \in \Pi_{PH}$ *and* $\varphi, \psi, \varphi_0 \in \Phi_{PH}$*,*

$$(d,p,[\varphi?]\psi) \in \mathrm{sf}(0,0,\varphi_0) \Rightarrow (d,p,\varphi) \in \mathrm{sf}(0,0,\varphi_0) \tag{13}$$

$$(d,p,[\alpha\,;\,\beta]\varphi) \in \mathrm{sf}(0,0,\varphi_0) \Rightarrow (d,p,[\alpha][\beta]\varphi) \in \mathrm{sf}(0,0,\varphi_0) \tag{14}$$

$$(d,p,[\alpha \parallel_i \beta]\,\varphi) \in \mathrm{sf}(0,0,\varphi_0) \Rightarrow (d,p+1,[\alpha](i,1)) \in \mathrm{sf}(0,0,\varphi_0) \tag{15}$$

$$(d,p,[\alpha \parallel_i \beta]\,\varphi) \in \mathrm{sf}(0,0,\varphi_0) \Rightarrow (d,p+1,[\beta](i,2)) \in \mathrm{sf}(0,0,\varphi_0) \tag{16}$$

*Proof.* The proof is given for (14) only. The other implications are similar and left to the reader. By Lemma 7, $(d,p,[\alpha\,;\,\beta]\varphi) \in \mathrm{sf}(0,0,\varphi_0) \Rightarrow \mathrm{sf}(d,p,[\alpha\,;\,\beta]\varphi) \subseteq \mathrm{sf}(0,0,\varphi_0)$. By construction, $\mathrm{sf}(d,p,[\alpha][\beta]\varphi) \subseteq \mathrm{sf}(d,p,[\alpha\,;\,\beta]\varphi)$. And by Lemma 7 again, $(d,p,[\alpha][\beta]\varphi) \in \mathrm{sf}(d,p,[\alpha\,;\,\beta]\varphi)$. □

**Lemma 8.** *For all* $d,p \in \mathbb{N}$*,* $\alpha \in \Pi_{PH}$ *and* $\varphi, \psi \in \Phi_{PH}$*, if* $(d,p,[\alpha]\varphi) \in \mathrm{sf}(0,0,\psi)$ *then* $(d+\mathrm{size}(\alpha),p,\varphi) \in \mathrm{sf}(0,0,\psi)$*.*

*Proof.* The proof is by induction on the length of $\alpha$. Each case is similar to the proof of Corollary 1, using Lemma 7 twice. They are all left to the reader. □

**Lemma 9.** *For all pure annotated formula* $\varphi \in \Phi_{\mathbb{N}}$*, the cardinality of* $\mathrm{sf}(0,0,\varphi)$ *is linear in the number of occurences of symbols in the unannotated formula* $\overline{\varphi}$*.*

*Proof.* A value is assigned to each symbol in any annotated formula with placeholders: 3 for $\|$; 1 for propositional variables, commas, $\bot$, atomic programs, semicolons and question marks; 0 for anything else (braces, parentheses and integers). For all annotated formula with placeholders $\varphi \in \Phi_{PH}$, let $L(\varphi)$ be the sum of those values for each occurence of symbols in $\varphi$. Obviously, for all pure annotated formula $\varphi \in \Phi_{\mathbb{N}}$, $L(\varphi)$ is less or equal to three times the number of occurences of symbols in the unannotated formula $\overline{\varphi}$. Moreover, it can be easily proved by induction on $L(\varphi)$, that for all $d, p \in \mathbb{N}$ and all annotated formula with placeholders $\varphi \in \Phi_{PH}$, the cardinality of $\mathrm{sf}(d, p, \varphi)$ is equal to $L(\varphi)$. $\square$

**Lemma 10.** *For all $d, p \in \mathbb{N}$, $\varphi \in \Phi_{\mathbb{N}}$ and $(d', p', \varphi') \in \mathrm{sf}(d, p, \varphi)$:*

$$d' \leq d + |\varphi| \tag{17}$$

$$p' \leq p + |\varphi| \tag{18}$$

*Proof.* The proof is by induction on $|\varphi|$. The base cases, for propositional variables, placeholders and $\bot$, are trivial. If $(d', p', \varphi') \in \mathrm{sf}(d, p, [a]\varphi)$, then either $(d', p', \varphi') = (d, p, [a]\varphi)$ or $(d', p', \varphi') \in \mathrm{sf}(d + 1, p, \varphi)$ and by induction $d' \leq d+1+|\varphi| = d+|[a]\varphi|$ and $p' \leq p+|\varphi| \leq p+|[a]\varphi|$. For tests and sequential compositions, the proof is direct by induction and left to the reader.

If $(d', p', \varphi') \in \mathrm{sf}\,(d, p, [\alpha \parallel_i \beta]\,\varphi)$, then either $(d', p', \varphi') = (d, p, [\alpha \parallel_i \beta]\,\varphi)$ or one of the following holds:

$$(d', p', \varphi') \in \mathrm{sf}(d, p + 1, [\alpha](i, 1)) \tag{19}$$

$$(d', p', \varphi') \in \mathrm{sf}(d, p + 1, [\beta](i, 2)) \tag{20}$$

$$(d', p', \varphi') \in \mathrm{sf}(d + \mathrm{size}\,(\alpha \parallel_i \beta)\,, p, \varphi) \tag{21}$$

If (19) holds, as $|[\alpha](i, 1)| < |[\alpha \parallel_i \beta]\,\varphi| = |\alpha| + 1 + |\beta| + |\varphi|$, properties (17) and (18) are verified by induction. The proof is identical if (20) holds. If (21) holds, by induction, $d' \leq d+\mathrm{size}\,(\alpha \parallel_i \beta)+|\varphi|$ and $p' \leq p+|\varphi| < p+|[\alpha \parallel_i \beta]\,\varphi|$. And by Lemma 2, $\mathrm{size}\,(\alpha \parallel_i \beta) + |\varphi| \leq |[\alpha \parallel_i \beta]\,\varphi|$. $\square$

**Definition 5 (Annotated subprograms).** *Given a pure annotated formula $\varphi_0 \in \Phi_{\mathbb{N}}$, the set $\mathrm{SP}\,(\varphi_0)$ of $\varphi_0$'s* annotated subprograms *is defined as*

$$\mathrm{SP}\,(\varphi_0) = \{\alpha \in \Pi_{PH} \mid \exists \varphi \in \Phi_{PH},\ [\alpha]\varphi \in \mathrm{SF}(\varphi_0)\}$$

**Lemma 11.** *No placeholders appear in any annotated subprogram.*

*Proof.* Let $\Phi_S$ be the smallest language containing $\bot$, all propositional variables and placeholders from $\Phi_{PH}$ and such that for all $\alpha \in \Pi_{\mathbb{N}}$ and $\varphi \in \Phi_S$, $[\alpha]\varphi \in \Phi_S$. Obviously, $\Phi_{\mathbb{N}} \subset \Phi_S \subset \Phi_{PH}$. By induction on $|\varphi|$, the following property can be easily proved:

$$\forall \varphi \in \Phi_S, \forall d, p \in \mathbb{N}, \forall (d', p', \psi) \in \mathrm{sf}(d, p, \varphi),\ \ \psi \in \Phi_S$$

Therefore, for all $[\alpha]\varphi \in \mathrm{SF}(\varphi_0)$, $\alpha \in \Pi_{\mathbb{N}}$. $\square$

## 4.2 Model extension with Markers

Let $\mathcal{M} = (W, R, \lhd, V)$ be a model on $\Phi$. The sets $\Phi^+_{\mathcal{M}}$ and $\Pi^+_{\mathcal{M}}$ of formulas and programs with indices and markers from $\mathcal{M}$ are defined by parallel induction:

$$\begin{aligned} \alpha, \beta &:= a \mid (\alpha \,;\, \beta) \mid \varphi? \mid (\alpha \parallel_i \beta) \\ \varphi &:= p \mid w \mid \bot \mid [\alpha]\varphi \end{aligned}$$

where $p$ range over $\Phi_0$, $a$ over $\Pi_0$, $i$ over $\mathbb{N}$, $w$ over $W$, $\varphi$ over $\Phi^+_{\mathcal{M}}$ and $\alpha$ and $\beta$ over $\Pi^+_{\mathcal{M}}$. The atomic formulas belonging to $W$ are called *markers*. The definitions of lengths and size of Section 2 are extended to $\Phi^+_{\mathcal{M}}$ and $\Pi^+_{\mathcal{M}}$ by considering markers as new propositional variables and ignoring indices.

A formula $\varphi \in \Phi^+_{\mathcal{M}}$ is an *annotated formula with markers from* $\mathcal{M}$ if each integer appears at most once as index in it. The subset $\Phi_{\mathcal{M}} \subset \Phi^+_{\mathcal{M}}$ of all annotated formula with markers from $\mathcal{M}$ is called the *annotated language with markers from* $\mathcal{M}$, and $\Pi_{\mathcal{M}}$ is the corresponding set of annotated programs with markers from $\mathcal{M}$. A formula $\varphi \in \Phi_{\mathcal{M}}$ is *pure* iff it contains no markers. It is worth noting that the subset of pure formulas from $\Phi_{\mathcal{M}}$ is the language of pure annotated formula $\Phi_{\mathbb{N}}$.

The *extension of* $\mathcal{M}$ *with markers* is the model $\mathcal{M}^+ = (W, R, \lhd, V^+)$ on the language $\Phi_{\mathcal{M}}$ with the new valuation $V^+$ defined as follows:

$$\begin{aligned} V^+(p) &= V(p) && \text{if } p \in \Phi_0 \\ V^+(w) &= W \setminus \{w\} && \text{if } w \in W \end{aligned}$$

**Lemma 12.** *For all $\varphi \in \Phi_{\mathcal{M}}$, if $\varphi$ is pure then for all $w \in W$, $\mathcal{M}, w \vDash \overline{\varphi} \Leftrightarrow \mathcal{M}^+, w \vDash \varphi$.*

*Proof.* By induction on $\varphi$, left to the reader.

**Definition 6.** *Given a pure annotated formula $\varphi \in \Phi_{\mathbb{N}}$ and a model $\mathcal{M} = (W, R, \lhd, V)$,* a binding *for $\varphi$ on $\mathcal{M}$ is a partial function $m : I_\varphi \times \{1, 2\} \rightharpoonup W$. The set of all such bindings is denoted by $B^{\mathcal{M}}_{\varphi}$.*

Given a pure annotated formula $\varphi_0 \in \Phi_{\mathbb{N}}$, a model $\mathcal{M} = (W, R, \lhd, V)$ and a binding $m \in B^{\mathcal{M}}_{\varphi_0}$, the partial function $F_{\varphi_0,\mathcal{M},m} : \mathrm{SF}(\varphi_0) \rightharpoonup \Phi_{\mathcal{M}}$ is defined inductively by:

$$\begin{aligned} F_{\varphi_0,\mathcal{M},m}(p) &= p && \text{if } p \in \Phi_0 \\ F_{\varphi_0,\mathcal{M},m}(\bot) &= \bot && \\ F_{\varphi_0,\mathcal{M},m}((i,k)) &= m(i,k) && \text{if } m(i,k) \text{ is defined} \\ F_{\varphi_0,\mathcal{M},m}([\alpha]\varphi) &= [\alpha]\, F_{\varphi_0,\mathcal{M},m}(\varphi) && \text{if } F_{\varphi_0,\mathcal{M},m}(\varphi) \text{ is defined} \\ F_{\varphi_0,\mathcal{M},m}(\varphi) & \text{ is undefined} && \text{otherwise} \end{aligned}$$

When it does not lead to confusion we will write $F_m$ instead of $F_{\varphi_0,\mathcal{M},m}$.

**Procedure 1:** Selection of a finite submodel

```
Input: An annotated formula $\varphi_a \in \Phi_{\mathcal{M}_u}$ satisfiable in $\mathcal{M}$ at $w_u \in W$; a
       function $E : W \times B^{\mathcal{M}_u}_{\varphi_a} \longrightarrow \mathcal{P}(\Phi_{\mathcal{M}_u})$.
Result: $\mathcal{S}$ a tree with nodes belonging to $W \times B^{\mathcal{M}_u}_{\varphi_a}$.
1  initialisation
2      $\mathcal{S} =$ the one (unmarked) node tree $\{(w_u, \emptyset)\}$
3  while some nodes in $\mathcal{S}$ are not marked do
4      choose an unmarked node $(w, m)$ from $\mathcal{S}$
5      mark $(w, m)$
6      foreach $[a]\varphi \in E(w, m)$ s.t. $\mathcal{M}, w \nvDash [a]\varphi$ do
7          choose $w'$ s.t. $wR(a)w'$ and $\mathcal{M}, w' \nvDash \varphi$
8          add $(w', m)$ as $(w, m)$'s child in $\mathcal{S}$
9      foreach $[\alpha \parallel_i \beta]\varphi \in E(w, m)$ s.t. $\mathcal{M}, w \nvDash [\alpha \parallel_i \beta]\varphi$ do
10         choose $w_1, w_2, w_3, w_4, w_5$ s.t.
11             $w \lhd (w_1, w_2) \wedge w_1 R(\alpha) w_3 \wedge w_2 R(\beta) w_4 \wedge w_5 \lhd (w_3, w_4) \wedge \mathcal{M}, w_5 \nvDash \varphi$
12         add $(w_1, m \uplus \{((i, 1), w_3)\})$ as $(w, m)$'s child in $\mathcal{S}$
13         add $(w_2, m \uplus \{((i, 2), w_4)\})$ as $(w, m)$'s child in $\mathcal{S}$
14         if $w_5 \neq w$ then
15             add $(w_5, m)$ as $(w, m)$'s child in $\mathcal{S}$
```

## 5 Finite Model Property by Selection

Let us consider a formula $\varphi_0 \in \Phi$. If $\varphi_0$ is satisfiable, thanks to the Löwenheim-Skolem theorem, there exists a countable model $\mathcal{M}_0 = (W_0, R_0, \lhd_0, V_0)$ and a world $w_0 \in W_0$ such that $\mathcal{M}_0, w_0 \vDash \varphi_0$. Let $\mathcal{M}_u = (W_u, R_u, \lhd_u, V_u)$ be the unfolding of $\mathcal{M}_0$ at $w_0$ with root $w_u$, degree $d : W_u \longrightarrow \mathbb{Q}$ and depth $p : W_u \longrightarrow \mathbb{Z}$ as defined in Section 3. Let $\mathcal{M} = (W, R, \lhd, V)$ be the extension with markers of $\mathcal{M}_u$. As $W = W_u$, degree and depth apply to worlds of $\mathcal{M}$ too. Let $\varphi_a \in \Phi_{\mathbb{N}}$ be an annotation of $\varphi_0$. As $\varphi_a$ is pure, by Lemma 12, $\mathcal{M}, w_u \vDash \varphi_a$.

Procedure 1 constructs a tree $\mathcal{S}$ whose nodes are tuples from $W \times B^{\mathcal{M}_u}_{\varphi_a}$. The function $E : W \times B^{\mathcal{M}_u}_{\varphi_a} \longrightarrow \mathcal{P}(\Phi_{\mathcal{M}_u})$ associates to each node from $\mathcal{S}$ the set of $\varphi_a$'s annotated subformulas with markers which have to be considered to add children to this node. It is defined as:

$$E(w, m) = \{F_m(\varphi) \mid \exists d', p',\ (d', p', \varphi) \in \mathrm{sf}(0, 0, \varphi_a) \wedge d(w) \leq d' \wedge p(w) = p'\}$$

**Lemma 13.** *For all $(w, m) \in \mathcal{S}$, $p(w) \geq 0$.*

*Proof.* The proof is by induction on $\mathcal{S}$: the root's depth is 0 and for each child $(w', m')$ of $(w, m)$, if $p(w) \geq 0$ then $p(w') \geq 0$. If $(w', m')$ has been added at line 8, then $wR(a)w'$ and by Property 4, $p(w') = p(w)$. The proof is identical if $(w', m')$ has been added at line 15. If $(w', m')$ has been added at line 12, there exists $w_2 \in W$ such that $w \lhd (w', w_2)$ and by Property 3, $p(w') = p(w) + 1$. The proof is identical if $(w', m')$ has been added at line 13. □

**Lemma 14.** *The size of $\mathcal{S}$ is bounded by an exponential in the number of occurences of symbols in $\varphi_0$.*

*Proof.* The vertex degree of $\mathcal{S}$ is bounded by the cardinality of $\mathrm{sf}(0, 0, \varphi_a)$ multiplied by three. Lemma 9 proved the cardinality of $\mathrm{sf}(0, 0, \varphi_a)$ is linear in the number of occurences of symbols in $\varphi_0$. The remaining of the proof is devoted to demonstrate the length of the path from the root $(w_u, \emptyset)$ to any leaf is bounded by a quadratic function on the number of occurences of symbols in $\varphi_0$.

We first define the following maximal elements:

$$d_{\max} = \max\{d \in \mathbb{N} \mid \exists p, \varphi,\ (d, p, \varphi) \in \mathrm{sf}(0, 0, \varphi_a)\}$$
$$p_{\max} = \max\{p \in \mathbb{N} \mid \exists d, \varphi,\ (d, p, \varphi) \in \mathrm{sf}(0, 0, \varphi_a)\}$$

Clearly, if $d(w) > d_{\max}$ or $p(w) > p_{\max}$ for a given node $(w, m) \in \mathcal{S}$, then $E(w, m) = \emptyset$ and $(w, m, t)$ has no children in $\mathcal{S}$. Moreover, by Lemmas 10 and 1, both $d_{\max}$ and $p_{\max}$ are less or equal than the number of occurences of symbols in $\varphi_0$.

We define the function $f : \mathcal{S} \longrightarrow \mathbb{Q}$ by:

$$f(w, m) = (d_{\max} + 1).p(w) + d(w)$$

Obviously, if $f(w, m) \geq (d_{\max} + 1)(p_{\max} + 1)$ for a given node $(w, m) \in \mathcal{S}$, then this node has no children in $\mathcal{S}$. We will prove that for any child $(w', m')$ of $(w, m)$ in $\mathcal{S}$, $f(w', m') \geq f(w, m) + 1$.

If $(w', m')$ has been added as $(w, m)$'s child at line 8, then $wR(a)w'$ and by Lemma 13 and Properties 2 and 4, $d(w') = d(w) + 1$ and $p(w') = p(w)$. Hence $f(w', m') = f(w, m) + 1$.

If $(w', m')$ has been added as $(w, m)$'s child at line 12, then $\exists w_2 \in W$ such that $w \lhd (w', w_2)$. By Lemma 13 and Property 3, $p(w') = p(w) + 1$, thus $f(w', m') \geq (d_{\max} + 1).p(w) + d_{\max} + 1$. And since $(w, m)$ has children, $d(w) \leq d_{\max}$. The proof is identical if $(w', m')$ has been added at line 13.

If $(w', m')$ has been added as $(w, m)$'s child at line 15, then $wR\,(\alpha \parallel_i \beta)\, w'$ and by Lemma 13 and Properties 2 and 4, $d(w') = d(w) + \mathrm{size}\,(\alpha \parallel_i \beta)$ and $p(w') = p(w)$, hence $f(w', m') = f(w, m) + \mathrm{size}\,(\alpha \parallel_i \beta)$. Moreover, $\mathrm{size}\,(\alpha \parallel_i \beta) \geq 1$ because otherwise, by Lemmas 3 and 5, $w' = w$ which is impossible by the condition at line 14. □

From the tree $\mathcal{S}$ produced by Procedure 1, the model $\mathcal{M}_f = (W_f, R_f, \lhd_f, V_f)$ is defined with $W_f$ being the subset $\{w \in W \mid \exists m,\ (w, m) \in \mathcal{S}\}$ and $R_f$, $\lhd_f$ and $V$ the restriction of $R$, $\lhd$ and $V$ to $W_f$. Obviously, $\mathcal{M}_f$ is finite and $w_u \in W_f$. Let $E^+(w) = \bigcup_{m \mid (w,m) \in \mathcal{S}} E(w, m)$ for all $w \in W_f$.

**Lemma 15.** *The model $\mathcal{M}_f$ is $\lhd$-separated and $\lhd$-deterministic.*

*Proof.* By Lemma 5, $\mathcal{M}_u$ is $\lhd$-separated and $\lhd$-deterministic. Since both this conditions are universal and $\mathcal{M}_f$ is a submodel of $\mathcal{M}_u$, $\mathcal{M}_f$ is $\lhd$-separated and $\lhd$-deterministic. □

**Lemma 16 (Truth lemma).** *$\forall w \in W_f$:*

$$\forall \varphi \in E^+(w), \varphi \text{ pure},\ \mathcal{M}, w \vDash \varphi \Rightarrow \mathcal{M}_f, w \vDash \varphi \quad (22)$$
$$\forall \varphi \in E^+(w),\ \mathcal{M}, w \vDash \varphi \Leftarrow \mathcal{M}_f, w \vDash \varphi \quad (23)$$
$$\forall v \in W_f, \forall (d', p(v), [\alpha]\varphi) \in \mathrm{sf}(0, 0, \varphi_a), d(v) \leq d',\ vR(\alpha)w \Leftarrow vR_f(\alpha)w \quad (24)$$

*Proof.* The proof is by parallel induction on the length of $\varphi$ for (22) and (23) and on the length of $\alpha$ for (24).

*Hypothesis* (22)*.* The base cases for $\bot$ and propositional variables are trivial.

Suppose $\mathcal{M}, w \vDash [a]\varphi$, $[a]\varphi \in E^+(w)$, $[a]\varphi$ is pure and $wR_f(a)v$. Then there exists $d'$ and $p'$ such that $(d', p', [a]\varphi) \in \mathrm{sf}(0, 0, \varphi_a)$, $d(w) \leq d'$ and $p(w) = p'$. By hypothesis (24), $wR(a)v$ and hence $\mathcal{M}, v \vDash \varphi$. By Property 2, $d(v) = d(w) + 1$ and by Lemma 13 and Property 4, $p(v) = p(w)$. By Lemma 8, $(d' + 1, p', \varphi) \in \mathrm{sf}(0, 0, \varphi_a)$. As $d(v) \leq d' + 1$, $p(v) = p'$ and $\varphi$ is pure, $\varphi \in E^+(v)$. By induction, $\mathcal{M}_f, v \vDash \varphi$.

Suppose $\mathcal{M}, w \vDash [\alpha\ ;\ \beta]\varphi$, $[\alpha\ ;\ \beta]\varphi \in E^+(w)$ and $[\alpha\ ;\ \beta]\varphi$ is pure. Then there exists $d' \geq d(w)$ such that $(d', p(w), [\alpha\ ;\ \beta]\varphi) \in \mathrm{sf}(0, 0, \varphi_a)$. By Corollary 1, $(d', p(w), [\alpha][\beta]\varphi) \in \mathrm{sf}(0, 0, \varphi_a)$ and thus $[\alpha][\beta]\varphi \in E^+(\varphi_a)$. Since $|[\alpha][\beta]\varphi| < |[\alpha\ ;\ \beta]\varphi|$ and $\mathcal{M}, w \vDash [\alpha][\beta]\varphi$, by induction, $\mathcal{M}_f, w \vDash [\alpha][\beta]\varphi$.

Suppose $\mathcal{M}, w \vDash [\varphi?]\psi$, $[\varphi?]\psi \in E^+(w)$ and $[\varphi?]\psi$ is pure. Then there exists $d' \geq d(w)$ such that $(d', p(w), [\varphi?]\psi) \in \mathrm{sf}(0, 0, \varphi_a)$. By Corollary 1 and Lemma 8 $(d', p(w), \varphi)$ and $(d', p(w), \psi)$ belong to $\mathrm{sf}(0, 0, \varphi_a)$. Thus $\varphi$ and $\psi$ belong to $E^+(w)$. If $\mathcal{M}, w \nvDash \varphi$ then by induction hypothesis (23), $\mathcal{M}_f, w \nvDash \varphi$. If $\mathcal{M}, w \vDash \psi$, by induction hypothesis (22), $\mathcal{M}_f, w \vDash \psi$.

Suppose $\mathcal{M}, w \vDash [\alpha \parallel_i \beta]\, \varphi$, $[\alpha \parallel_i \beta]\, \varphi \in E^+(w)$, $wR_f\, (\alpha \parallel_i \beta)\, v$ and $[\alpha \parallel_i \beta]\, \varphi$ is pure. Then there exists $d' \geq d(w)$ such that $(d', p(w), [\alpha \parallel_i \beta]\, \varphi) \in \mathrm{sf}(0, 0, \varphi_a)$. By hypothesis (24), $wR\, (\alpha \parallel_i \beta)\, v$ and hence $\mathcal{M}, w \vDash \varphi$. By Property 2, $d(v) = d(w) + \mathrm{size}\, (\alpha \parallel_i \beta)$ and by Lemma 13 and Property 4, $p(v) = p(w)$. By Lemma 8, $(d' + \mathrm{size}\, (\alpha \parallel_i \beta)\, , p(w), \varphi) \in \mathrm{sf}(0, 0, \varphi_a)$. As $d(v) \leq d' + \mathrm{size}\, (\alpha \parallel_i \beta)$, $p(v) = p(w)$ and $\varphi$ is pure, $\varphi \in E^+(v)$. By induction, $\mathcal{M}_f, v \vDash \varphi$.

*Hypothesis* (23)*.* The base cases for $\bot$, propositional variables and markers are trivial.

Suppose $\mathcal{M}, w \nvDash [a]\varphi$ and $[a]\varphi \in E^+(w)$. Then there exists $(w, m) \in \mathcal{S}$ such that $[a]\varphi \in E(w, m)$. By construction of $\mathcal{S}$ there exists $w' \in W$ such that $wR(a)w'$, $\mathcal{M}, w' \nvDash \varphi$ and $(w', m) \in \mathcal{S}$, thus $wR_f(a)w'$. Moreover, there exists $\psi \in \mathrm{SF}(\varphi_a)$ and $d' \geq d(w)$ such that $F_m(\psi) = \varphi$ and $(d', p(w), [a]\psi) \in \mathrm{sf}(0, 0, \varphi_a)$. By Lemma 8, $(d' + 1, p(w), \psi) \in \mathrm{sf}(0, 0, \varphi_a)$ and by Lemma 13 and Properties 2 and 4, $d(w') = d(w) + 1$ and $p(w') = p(w)$. Hence $\varphi \in E^+(w')$ and by induction $\mathcal{M}_f, w' \nvDash \varphi$. Consequently $\mathcal{M}_f, w \nvDash [a]\varphi$.

Suppose $\mathcal{M}, w \nvDash [\alpha\ ;\ \beta]\varphi$ and $[\alpha\ ;\ \beta]\varphi \in E^+(w)$. Then there exists $(w, m) \in \mathcal{S}$, $\psi \in \mathrm{SF}(\varphi_a)$ and $d' \geq d(w)$ such that $F_m(\psi) = \varphi$ and $(d', p(w), [\alpha\ ;\ \beta]\psi) \in \mathrm{sf}(0, 0, \varphi_a)$. By Corollary 1, $(d', p(w), [\alpha][\beta]\psi) \in \mathrm{sf}(0, 0, \varphi_a)$. Thus $[\alpha][\beta]\psi \in E^+(w)$ and by induction $\mathcal{M}_f, w \nvDash [\alpha][\beta]\varphi$.

Suppose $\mathcal{M}, w \nvDash [\varphi?]\psi$ and $[\varphi?]\psi \in E^+(w)$. Then both $\mathcal{M}, w \vDash \varphi$ and $\mathcal{M}, w \nvDash \psi$ hold. Therefore there exists $(w, m) \in \mathcal{S}$, $\psi' \in \mathrm{SF}(\varphi_a)$ and $d' \geq d(w)$ such that $F_m(\psi') = \psi$ and $(d', p(w), [\varphi?]\psi') \in \mathrm{sf}(0, 0, \varphi_a)$. By Corollary 1 and Lemma 8, $(d', p(w), \varphi)$ and $(d', p(w), \psi')$ both belong to $\mathrm{sf}(0, 0, \varphi_a)$. And by induction hypothesis (22) and (23), $\mathcal{M}_f, w \vDash \varphi$ and $\mathcal{M}_f, w \nvDash \psi$.

Suppose $\mathcal{M}, w \nvDash [\alpha \parallel_i \beta]\, \varphi$ and $[\alpha \parallel_i \beta]\, \varphi \in E^+(w)$. Then there exists $\psi \in \mathrm{SF}(\varphi_a)$, $(w, m) \in \mathcal{S}$ and $d' \geq d(w)$ such that $(d', p(w), [\alpha \parallel_i \beta]\, \psi) \in \mathrm{sf}(0, 0, \varphi_a)$

and $F_m(\psi) = \varphi$. By construction, there exists $w_1, w_2, w_3, w_4, w_5 \in W$ such that $w \lhd (w_1, w_2)$, $w_1 R(\alpha) w_3$, $w_2 R(\beta) w_4$, $w_5 \lhd (w_3, w_4)$, $\mathcal{M}, w_5 \nvDash \varphi$, $(w_1, m_1) \in \mathcal{S}$, $(w_2, m_2) \in \mathcal{S}$ and $(w_5, m) \in \mathcal{S}$, with $m_1 = m \uplus \{((i,1), w_3)\}$ and $m_2 = m \uplus \{((i,2), w_4)\}$. By Lemma 8, $(d' + \text{size}(\alpha \parallel_i \beta), p(w), \varphi) \in \text{sf}(0, 0, \varphi_a)$ and by Lemma 13 and Properties 2 and 4, $d(w_5) = d(w) + \text{size}(\alpha \parallel_i \beta)$ and $p(w_5) = p(w)$, then by induction hypothesis (23), $\mathcal{M}_f, w_5 \nvDash \varphi$. It remains to prove that $w R_f (\alpha \parallel_i \beta) w_5$. By Corollary 1, both $(d', p(w) + 1, [\alpha](i,1))$ and $(d', p(w) + 1, [\alpha](i,2))$ belong to $\text{sf}(0, 0, \varphi_a)$. By Lemma 13 and Properties 1 and 3, $d(w_1) \leq d(w)$, $d(w_2) \leq d(w)$ and $p(w_1) = p(w_2) = p(w) + 1$. Therefore $[\alpha] w_3 \in E(w_1, m_1)$ and $[\beta] w_4 \in E(w_2, m_2)$. Thus by induction hypothesis (23), $w_3$ and $w_4$ belongs to $W_f$, $w_1 R_f(\alpha) w_3$ and $w_2 R_f(\beta) w_4$. Therefore, $w R_f(\alpha \parallel_i \beta) w_5$.

*Hypothesis* (24). The base case for atomic programs is trivial.

Suppose $v R_f(\alpha \,;\, \beta) w$, $(d', p(v), [\alpha \,;\, \beta]\varphi) \in \text{sf}(0, 0, \varphi_a)$ and $d' \geq d(v)$. Then there exists $w' \in W_f$ such that $v R_f(\alpha) w'$ and $w' R_f(\alpha) w$. By Corollary 1, $(d', p(v), [\alpha][\beta]\varphi) \in \text{sf}(0, 0, \varphi_a)$. By induction hypothesis (24), $v R(\alpha) w'$. Thanks to Lemma 13 and Properties 2 and 4, $d(w') = d(v) + \text{size}(\alpha)$ and $p(w') = p(v)$. By Lemma 8, $(d' + \text{size}(\alpha), p(v), [\beta]\varphi) \in \text{sf}(0, 0, \varphi_a)$ and by induction hypothesis (24), $w' R(\beta) w$. Since $v R(\alpha) w'$, $v R(\alpha \,;\, \beta) w$.

Suppose $v R_f(\psi?) w$, $(d', p(v), [\psi?]\varphi) \in \text{sf}(0, 0, \varphi_a)$ and $d' \geq d(v)$. Then $w = v$ and as $\psi$ is pure by Lemma 11, $\mathcal{M}_f, v \vDash \psi$. By Corollary 1, $(d', p(v), \psi) \in \text{sf}(0, 0, \varphi_a)$. As $|\psi?| > |\psi|$, by induction hypothesis (23), $\mathcal{M}, v \vDash \psi$. Since $v = w$, $v R(\psi?) w$.

Suppose $v R_f (\alpha \parallel_i \beta) w$, $(d', p(v), [\alpha \parallel_i \beta] \varphi) \in \text{sf}(0, 0, \varphi_a)$ and $d' \geq d(v)$. Then there exists $w_1, w_2, w_3, w_4 \in W_f$ such that $v \lhd_f (w_1, w_2)$, $w_1 R_f(\alpha) w_3$, $w_2 R_f(\beta) w_4$ and $w \lhd_f (w_3, w_4)$. Obviously, both $v \lhd (w_1, w_2)$ and $w \lhd (w_3, w_4)$ hold. By Corollary 1, both $(d', p(v)+1, [\alpha](i,1))$ and $(d', p(v)+1, [\beta](i,2))$ belong to $\text{sf}(0, 0, \varphi_a)$. By Lemma 13 and Properties 1 and 3, $d(w_1) \leq d(v)$, $d(w_2) \leq d(v)$ and $p(w_1) = p(w_2) = p(v) + 1$. Thus, by induction hypothesis (24), $w_1 R(\alpha) w_3$ and $w_2 R(\beta) w_4$. Then $v R(\alpha \parallel_i \beta) w$. □

**Proposition 2.** *The satisfiability problem of PPDL$_0^{det}$'s formulas interpreted in $\lhd$-separated $\lhd$-deterministic models is in NEXPTIME.*

*Proof.* Lemmas 14, 15 and 16 prove that whenever a formula $\varphi$ is satisfiable in a $\lhd$-separated $\lhd$-deterministic model, $\varphi$ is satisfiable in a $\lhd$-deterministic $\lhd$-separated model with size exponential in the number of occurences of symbols.

## 6 Conclusion

We have proved the fragment PPDL$_0^{\text{det}}$ of PRSPDL is decidable, giving a NEXPTIME complexity upper bound. Because of the subtleties brought about by the parallel composition, we have used placeholders and markers in the selection procedure. We expect this technique could be reused to express subformulas in other fragments of PRSPDL.

Still, the exact complexity of $\mathrm{PPDL}_0^{\mathrm{det}}$ is unknown. The only known lower bound is given by the straightforward embeding of the modal logic **K**, resulting in $\mathrm{PPDL}_0^{\mathrm{det}}$ being PSPACE-hard [11].

Althought the $\lhd$-separation condition is needed for the axiomatization of $\mathrm{PRSPDL}_0$ [1], we believe this condition makes the satisfiability problem harder. We conjecture the satisfiability problem of the same language interpreted on $\lhd$-deterministic models to be PSPACE-complete, leaving the proof as future work.

## References

1. Balbiani, P., Boudou, J.: Iteration-free PDL with storing, recovering and parallel composition: a complete axiomatization, submitted
2. Balbiani, P., Tinchev, T.: Definability and computability for PRSPDL. In: Advances in Modal Logic (2014), to appear
3. Benevides, M.R.F., de Freitas, R.P., Viana, J.P.: Propositional dynamic logic with storing, recovering and parallel composition. Electr. Notes Theor. Comput. Sci. 269, 95–107 (2011)
4. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic, Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press (2001)
5. Brookes, S.: A semantics for concurrent separation logic. Theor. Comput. Sci. 375(1-3), 227–270 (2007)
6. Brotherston, J., Kanovich, M.I.: Undecidability of propositional separation logic and its neighbours. In: LICS. pp. 130–139. IEEE Computer Society (2010)
7. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. 18(2), 194–211 (1979)
8. Galmiche, D., Larchey-Wendling, D.: Expressivity properties of boolean BI through relational models. In: FSTTCS. Lecture Notes in Computer Science, vol. 4337, pp. 357–368. Springer Berlin Heidelberg (2006)
9. Goldblatt, R.: Logics of Time and Computation. Center for the Study of Language and Information (1987)
10. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT press (2000)
11. Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM J. Comput. 6(3), 467–480 (1977)
12. Larchey-Wendling, D., Galmiche, D.: Exploring the relation between Intuitionistic BI and Boolean BI: an unexpected embedding. Mathematical Structures in Computer Science 19(3), 435–500 (2009)
13. Larchey-Wendling, D., Galmiche, D.: The undecidability of boolean BI through phase semantics. In: LICS. pp. 140–149. IEEE Computer Society (2010)
14. O'Hearn, P.W.: Resources, concurrency, and local reasoning. Theor. Comput. Sci. 375(1-3), 271–307 (2007)
15. Pym, D.J.: The semantics and proof theory of the logic of bunched implications, vol. 26. Springer (2002)
16. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: LICS. pp. 55–74. IEEE Computer Society (2002)

# Axiomatic and Tableau-Based Reasoning for Kt(H,R)

Renate A. Schmidt[†], John G. Stell[‡], and David Rydeheard[†]

[†]University of Manchester, UK
[‡]University of Leeds, UK

The context of this work is automated reasoning for modal logics. Here, we explore a variety of different deduction approaches in the spectrum between the purely axiomatic approach and the explicitly semantic approach. Our investigation is focussed on a propositional modal logic with non-trivial interactions between the modalities. The logic, called $Kt(H, R)$, has forward and backward looking modal operators defined by two accessibility relations $H$ and $R$. The frame conditions are reflexivity and transitivity of $H$, and *stability* of $R$ with respect to $H$. The stability condition is defined as $H\,;R;H \subseteq R$, where ; denotes relational composition. This means in a Kripke frame, for any two states $u$ and $v$, whenever there is an $H$-transition from $u$, followed by an $R$-transition and an $H$-transition, to $v$, then there is also an $R$-transition from $u$ to $v$.

The logic $Kt(H, R)$ originates with recent work on a bi-intuitionistic tense logic, called BISKT, which is studied with the motivation to develop a theory of relations on graphs and applications to spatial reasoning [11]. Using the standard embedding of intuitionistic logics into modal logic, BISKT can be embedded into $Kt(H, R)$ and properties such as decidability, the finite model property and complexity of $Kt(H, R)$ carry over to BISKT. Moreover, deduction methods for $Kt(H, R)$ and implementations can be used for BISKT.

$Kt(H, R)$ is of independent interest because the modal axiom(s) corresponding to the stability condition can be used to ascribe levels of awareness to agents in a multi-agent setting. The standard model for formalising knowledge and actions performed by agents, or events happening in an agent environment, uses the *S5*-modality as knowledge operator and $K$-modalities as action operators. In $Kt(H, R)$, the $[H]$-modality and the $[R]$-modality can be seen as modelling knowledge and action operators. (The formalisation is slightly more general, because the negative introspecion axiom is not assumed for the $[H]$-modality but this is not critical because it can be easily added to the logic.) $[H]\phi$ is read to mean 'the agent knows $\phi$' and $[R]\phi$ is read to mean 'always after executing action $R$, $\phi$ holds'. In this context, the axiom

$$S = [R]\phi \to [H][R][H]\phi$$

corresponding to the stability condition $H\,;R\,;H \subseteq R$, can be viewed as saying 'the agent knows that, after performing an action $R$, it knows the effects of the action'. Thus, it states the agent has (strong) awareness of performing action $R$ and its effects.

The logic $Kt(H, R)$ has an alternative axiomatisation in which the stability axiom $S$ is equivalent to the two axioms

$$A = [R]\phi \to [H][R]\phi \quad \text{and} \quad P = [R]\phi \to [R][H]\phi.$$

From an agent perspective, Axiom $A$ says 'the agent knows, when action $R$ is performed, then $\phi$ necessarily holds'; in other words, the agent is aware of action $R$. Axiom $P$ says 'after performing action $R$ the agent knows $\phi$ holds', i.e., it knows the post-condition has been realised. In some sense, Axioms $A$ and $P$ can be viewed as weak forms of no learning and perfect recall. No learning is typically formalised as $[R][H]\phi \to [H][R]\phi$, and perfect recall as $[H][R]\phi \to [R][H]\phi$.

A contribution of this work is a series of labelled semantic tableau calculi for the logic $Kt(H,R)$. Labelled semantic tableau calculi of the pure semantic kind explicitly and directly construct Kripke models during the inference process. They use *structural rules* which are direct reflections of the background theory given by a set of characterising frame conditions. For example, for axiom

$$4 = [H]\phi \to [H][H]\phi$$

the structural rule is $H(s,t),\ H(t,u)\ /\ H(s,u)$ and ensures $H$ will be a transitive relation. Transitivity is the frame condition of Axiom 4. For logics with semantic characterisations, labelled tableau calculi using structural rules may be developed by systematic methods. A general method is described in [10, 12].

Alternatively, the background theory can be accommodated as *propagation rules* [3]. The propagation rule for Axiom 4 is $s : [H]\phi,\ H(s,t)\ /\ t : [H]\phi$. Propagation rules accommodate the background theory not by representations of the frame conditions but by representations of inferences with the Hilbert axioms [6]. Propagation rules can be seen to attempt to speed up the inference process by not returning complete concrete models but only skeleton models and performing just enough inferences to determine both satisfiability and unsatisfiability.

We also explore the extreme case of basing the tableau rules of the background theory on direct representations of Hilbert axioms, e.g., using the rule $s : [H]\phi\ /\ s : [H][H]\phi$ for Axiom 4. This is an example of what we call an *axiomatic rule*. Calculi with such rules are seldom seen in the literature, and some authors have suggested completeness and termination cannot be guaranteed with such rules. We show however complete and terminating tableau calculi based on such rules can be obtained.

In total we present fifteen different tableau systems for $Kt(H,R)$ which we prove sound and complete. These emerge in systematic ways from the two axiomatisations and corresponding semantic characterisations of the logic. Only in one case a cut rule is needed when using propagation rules. For the Axioms $A$ and $P$, no cut is needed.

Each of the calculi is terminating when endowed with the unrestricted blocking mechanism. The unrestricted blocking mechanism is based on the use of the following (ub) rule.

$$(\mathsf{ub})\ \frac{}{s \approx t \mid s \not\approx t}$$

Adding the unrestricted blocking mechanism to a sound and constructively complete labelled tableau calculus forces termination, when the logic has the (effective) finite model property [8, 9].

Each of the tableau calculi can serve as a basis for an effective translation of satisfiability in $Kt(H, R)$ into first-order logic. The soundness and completeness of these translations is a consequence of the soundness and completeness of the calculi, and the fact that derivations are defined over a bounded number of modal formulae. The encoding of one of the calculi has the property that it maps satisfiability of a formula in $Kt(H, R)$ into the guarded fragment [1, 5]. This enables us to show: (i) $Kt(H, R)$ is decidable and has the effective finite model property, and (ii) satisfiability in $Kt(H, R)$ is PSPACE-complete. Important is the effective finite model property; using the results in [9] guarantees termination for all our tableau calculi, when unrestricted blocking is used.

The guarded fragment can be decided by ordered resolution [4]. Therefore, both ordered resolution and ordered resolution with selection as defined in [4] decide the axiomatic translation of satisfiability problems in $Kt(H, R)$.

To get insight into the relative performances of different approaches and the properties of different techniques, we implemented the tableau calculi by encoding them into first-order logic and using the SPASS-YARRALUMLA system. SPASS-YARRALUMLA is a bottom-up model generator based on the SPASS theorem prover [13]. SPASS-YARRALUMLA emulates the behaviour of semantic labelled tableau provers [2]. We used four forms of blocking available in SPASS-YARRALUMLA: (i) sound ancestor blocking; (ii) unrestricted blocking; (iii) sound ancestor blocking on non-disjoint worlds; and (iv) sound anywhere blocking on non-disjoint worlds. The encodings of the tableau calculi are implemented as an extension of the `ml2dfg` tool used for the empirical evaluation of the axiomatic translation principle in [6].

The results of the experiments with implementations of the tableau calculi with SPASS-YARRALUMLA showed, on the whole, the encoding based on only propagation rules fared best for all forms of blocking tested. Similarly good results were obtained for the encodings involving a mixture of propagation and axiomatic rules based on the rules for Axioms $A$ and $P$. For satisfiable problems the encodings based on only structural rules fared well, too, in two cases giving best results. In terms of blocking, for all encodings unrestricted blocking was most expensive on unsatisfiable problems. Sound ancestor blocking produced best results for all encodings on both satisfiable and unsatisfiable problems.

We also tested SPASS in auto mode, which uses a form of ordered resolution with dynamic selection, on the encodings. As expected many timeouts were observed for encodings based on correspondence properties, as these do not provide decision procedures. For all other encodings the performances were very close for unsatisfiable problems. It is interesting how much faster these performances were than the best performances for SPASS-YARRALUMLA. For unsatisfiable problems, tableau-like approaches need to construct a complete derivation tree in which every formula is grounded, and this is generally larger than the non-ground clause set derived with ordered resolution.

Though the focus has been on $Kt(H, R)$, the techniques and ideas used in this research are of general nature and provide a useful methodology for developing practical decision procedures for modal logics. Some aspects are completely routine. The main aspect for which creativity is required and is specific to $Kt(H, R)$ is the development of the tableau calculi based on propagation or axiomatic rules and the axiomatic translation to the guarded fragment. Here the contribution of the paper has been to extend the ideas of the axiomatic translation principle from [6]. Key is finding effective refinements and showing completeness and termination, which is in general non-trivial and will not always be possible. All in all, because of the ubiquity of modal logics, we believe this kind of systematic research of decidability, proof theory, refinements and relative efficiency is widely applicable and useful, and should be extended to more logics, more types of tableau approaches, other deduction approaches, different provers and more problem sets.

Full details can be found in [7].

## References

1. H. Andréka, I. Németi, and J. van Benthem. Modal languages and bounded fragments of predicate logic. *J. Philos. Logic*, 27(3):217–274, 1998.
2. P. Baumgartner and R. A. Schmidt. Blocking and other enhancements for bottom-up model generation methods. In *Proc. IJCAR 2006*, volume 4130 of *LNAI*, pages 125–139. Springer, 2006.
3. M. A. Castilho, L. Fariñas del Cerro, O. Gasquet, and A. Herzig. Modal tableaux with propagation rules and structural rules. *Fund. Inform.*, 3–4(32):281–297, 1997.
4. H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In *Proc. LICS 1999*, pages 295–303. IEEE Computer Society Press, 1999.
5. E. Grädel. On the restraining power of guards. *J. Symbolic Logic*, 64:1719–1742, 1999.
6. R. A. Schmidt and U. Hustadt. The axiomatic translation principle for modal logic. *ACM Trans. Comput. Log.*, 8(4):1–55, 2007.
7. R. A. Schmidt, J. G. Stell, and D. Rydeheard. Axiomatic and tableau-based reasoning for Kt(H,R). In R. Goré and A. Kurucz, editors, *Advances in Modal Logic, Volume 10.* College Publications, 2014.
8. R. A. Schmidt and D. Tishkovsky. Using tableau to decide expressive description logics with role negation. In *Proc. ISWC 2007 + ASWC 2007*, volume 4825 of *LNCS*, pages 438–451. Springer, 2007.
9. R. A. Schmidt and D. Tishkovsky. A general tableau method for deciding description logics, modal logics and related first-order fragments. In *Proc. IJCAR 2008*, volume 5195 of *LNCS*, pages 194–209. Springer, 2008.
10. R. A. Schmidt and D. Tishkovsky. Automated synthesis of tableau calculi. *Logical Methods in Comput. Sci.*, 7(2):1–32, 2011.
11. J. G. Stell, R. A. Schmidt, and D. Rydeheard. Tableau development for a bi-intuitionistic tense logic. In *Proc. RAMiCS 14*, volume 8428 of *LNCS*, pages 412–428. Springer, 2014.
12. D. Tishkovsky and R. A. Schmidt. Refinement in the tableau synthesis framework. arXiv e-Print 1305.3131v1, 2013.
13. C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: SPASS version 3.0. In *Proc. CADE-21*, volume 4603 of *LNAI*, pages 514–520. Springer, 2007.

# On dual tableau-based decision procedures for relational fragments

Domenico Cantone[1], Joanna Golińska-Pilarek[2], and Marianna Nicolosi-Asmundo[1]

[1] University of Catania, Dept. of Mathematics and Computer Science
email: cantone@dmi.unict.it, nicolosi@dmi.unict.it
[2] University of Warsaw, Institute of Philosophy
email: j.golinska@uw.edu.pl

**Abstract.** We give a brief survey of the main results on dual tableau-based decision procedures for fragments of the logic of binary relations and we outline our current work in the design of relational decision procedures for logics for order-of-magnitude reasoning. Then we shortly hint at some further research plans on other relational decision procedures.

## 1 Introduction

In the last decades, the relational representation of many non-classical propositional logics has been studied and analyzed in a systematic way [20]. A homogeneous relational framework grounded on the logic of binary relations $\mathsf{RL}(\mathbf{1})$, presented in [19] and called *relational dual tableau*, turned out to be an effective logical means to represent in a modular way the three fundamental parts of a formal proof system, namely its syntax, semantics, and deduction system. Proof systems based on the logic of relations have been proposed for many non-classical logics such as modal and intuitionistic logics, relevant and many-valued logics, etc. Relational proof systems have also been defined for reasoning in logics of information and data analysis, for reasoning about time and space, for fuzzy set- and rough set-based reasoning, for order-of-magnitude and qualitative reasoning, for dynamic reasoning about programs, etc.

Formulae of $\mathsf{RL}(\mathbf{1})$ have the form $xRy$, where $x, y$ are individual variables and $R$ is a relational term of $\mathsf{RL}(\mathbf{1})$. We recall that the set of relational terms of $\mathsf{RL}(\mathbf{1})$ is the smallest set (with respect to inclusion) built from relational variables and constants (considering also the constant $\mathbf{1}$) with the operators '$\cap$', '$\cup$', '$;$' (binary) and '$-$', '$^{-1}$' (unary).

The possibility of representing non-classical logics in $\mathsf{RL}(\mathbf{1})$ relies on the fact that once the Kripke-style semantics of the logic under consideration is known, formulae can be treated as relations. In particular, since in Kripke-style semantics formulae are interpreted as collections of objects, in their relational representation they are seen as right ideal relations. In the case of binary relations this means that $(R\,;\mathbf{1}) = R$ is satisfied, where '$;$' is the composition operation on binary relations and '$\mathbf{1}$' is the universal relation.

One of the most useful features of the relational methodology is that, given a logic with a relational formalization, it is possible to construct its relational dual tableau in a systematic and modular way. On the other hand, using a uniform formalism to represent different logics could allow one to detect similarities among the logics which might be hidden by other formalisms.

Though the relational logic $\mathsf{RL}(\mathbf{1})$ is undecidable, it contains several decidable fragments. In many cases, however, dual tableau proof systems are not by themselves decision procedures for decidable fragments of $\mathsf{RL}(\mathbf{1})$. This is mainly due to the way decomposition and specific rules are defined and to the strategy of proof construction.

Over the years, great efforts have been spent in the construction of dual tableau proof systems for various logics known to be decidable; little care has been taken, however, in the design of dual tableau-based decision procedures for them. On the other hand, it is well known that when a proof system is designed and implemented, it is important to have decision procedures for decidable logics. In [11], for instance, an optimized relational dual tableau for $\mathsf{RL}(\mathbf{1})$, based on Binary Decision Graphs, has been implemented; however, such an implementation turns out not to be effective for decidable fragments.

In this paper we give a short survey of the main results on dual tableau-based decision procedures for fragments of the logic of binary relations. In particular, we outline a deterministic dual tableau-based decision procedure $\mathsf{RDT}_{\mathsf{RL_L}}$, presented in [7], for a class of multimodal logics whose accessibility relations may satisfy the properties of reflexivity, transitivity, and heredity. Next, we hint at our current work of designing a dual tableau decision procedure in the flavour of $\mathsf{RDT}_{\mathsf{RL_L}}$ for a logic for order-of-magnitude reasoning, called $\mathsf{OMR}_N$. We recall that logics for order-of-magnitude reasoning are non-classical logics used for qualitative reasoning. In general they are multimodal logics over a strict linear order with some additional accessibility relations satisfying certain constraints. Finally, we briefly hint at some further research plans in the ambit of relational decision procedures and draw our conclusions.

## 2 Dual tableau-based decision procedures for fragments of the logic of binary relations

In [20], relational dual tableau-based decision procedures have been constructed for fragments of $\mathsf{RL}$ corresponding to well-known classes of first-order formulae in prenex normal form, namely the class of existential formulae ($\exists^*$), of universal formulae ($\forall^*$), and of formulae with universal quantifiers followed by existential quantifiers ($\forall^*\exists^*$). A dual tableau decision procedure for each of these fragments has been obtained from the general dual tableau system defined in [20] by simply restricting the applicability of the ($\mathbf{;}$)-rule only in case (a) the variable used in the ($\mathbf{;}$)-decomposition occurs on the current node of the dual tableau, (b) the application of the ($\mathbf{;}$)-rule produces some new formulae not already occurring on the current node, and (c) no other rule is applicable.

In [18], a relational dual tableau decision procedure has been provided for the relational logic corresponding to the modal logic $\mathsf{K}$. The system was constructed in the framework of the original methodology of relational proof systems, determined only by axioms and inference rules, without any external techniques. Its main feature is *uniqueness*, i.e., given a formula, the system generates in a deterministic way only one proof tree for it. Thus, the dual tableau presented in [18] is not only a base for an algorithm verifying validity of a formula, but is *itself* a deterministic decision procedure. Furthermore, due to a special dual clause representation of modal formulae, the system is very simple as it consists only of two rules. The system in [18] has been implemented in Prolog in [17].

Fragments of $\mathsf{RL}(\mathbf{1})$, characterized by some restrictions in terms of type $(R \mathbin{;} S)$, have been presented in [8, 9] and, more recently, in [10]. Specifically, in [8], two fragments of $\mathsf{RL}(\mathbf{1})$, respectively called $(\mathsf{r} \mathbin{;} \_)$ and $(\cup, \cap \mathbin{;} \_)$, have been introduced. The $(\mathsf{r} \mathbin{;} \_)$-fragment is the collection of the $\mathsf{RL}(\mathbf{1})$-formulae $xPy$ in which the composition operator '$;$' can occur only in the following restricted way. For each subterm of $P$ of the form $R \mathbin{;} S$, $R$ must belong to a designated nonempty proper subset $\mathbb{RV}_1$ of $\mathbb{RV}$, whereas $S$ can involve all the relational operators used to construct $\mathsf{RL}(\mathbf{1})$-formulae, with the exception of the converse operator '${}^{-1}$'. In the relational interpretation of logics, the elements of $\mathbb{RV}_1$ are meant to denote accessibility relations (resp., roles) in modal (resp., description) logics. This fragment allows one to express the multimodal logic $\mathsf{K}$ and, therefore, also the description logic $\mathcal{ALC}$ [2, 1]. The $(\cup, \cap; \_)$-fragment is an extension of the $(\mathsf{r}; \_)$-fragment in which the constraints on the composition operator '$;$' are more relaxed. In particular, the first argument in a term of type $R; S$ of the $(\cup, \cap \mathbin{;} \_)$-fragment can be any term constructed from the relational variables of a proper nonempty subset of $\mathbb{RV}$, say again $\mathbb{RV}_1$, by applying only the operators '$\cup$' and '$\cap$'. The restriction on the second argument is the same of the $(\mathsf{r} \mathbin{;} \_)$-fragment. The $(\cup, \cap \mathbin{;} \_)$-fragment can express the description logic $\mathcal{ALC}(\cup, \cap)$ [2]. For each of these fragments, a dual tableau-based decision procedure is given differing from the general proof system described in [20] in the definition of the $(;)$-rule and in some strictness conditions on the applicability of the rules. (These, in fact, are the two sources of non termination in dual tableaux for formulae of such fragments.) The versions of $(;)$-rule introduced in [8] select the individual variables to decompose the $(;)$-formula by analyzing the literals occurring on the current node of the dual tableau. No blocking condition is needed to guarantee termination.

In [9] a dual tableau-based decision procedure is defined for a fragment, called $(\cup, \cap, ^{-1}; \_)$, of $\mathsf{RL}(\mathbf{1})$ which extends the $(\cup, \cap \mathbin{;} \_)$-fragment by permitting a more liberal application of the composition operator '$;$'. In fact, the first argument, $R$, in a term of type $R \mathbin{;} S$ is allowed to be any term obtained from the relational variables of a proper nonempty subset of $\mathbb{RV}$, by applying the operators '${}^{-1}$', '$\cup$', and '$\cap$'. The $(\cup, \cap, ^{-1}; \_)$-fragment of $\mathsf{RL}(\mathbf{1})$ expresses the description logic $\mathcal{ALCI}(\cup, \cap)$ [2]. The dual tableau decision procedure for this fragment differs from the ones introduced in [8] in the definition of the $(;)$-rule modified to handle more expressive $(;)$-formulae containing the '${}^{-1}$' operator.

We recently proved that decidability holds also for the extended versions of the $(\mathsf{r}\,;\,\_)$-fragment and of the $(\cup,\cap\,;\,\_)$-fragment where the distinction between variables in $\mathbb{RV}_1$ and in $\mathbb{RV}\setminus\mathbb{RV}_1$ is dropped and, therefore, the same relational variable can occur both on the left and on the right hand side of compositional terms. Then, in [10], a fragment of $\mathsf{RL}(\mathbf{1})$, called $(\{\mathbf{1},\cup,\cap\}\,;\,\_)$, has been introduced admitting a restricted form of composition where the left subterm, $R$, of any term of type $(R\,;S)$ is allowed to be either the constant $\mathbf{1}$ or any term constructed from the relational variables by applying only the operators of relational intersection and union. Similarly, terms of type $(R\,;\mathbf{1})$ are admitted only if $R$ is a Boolean term containing neither the complement operator nor the constant '$\mathbf{1}$'. Decidability of the $(\{\mathbf{1},\cup,\cap\}\,;\,\_)$-fragment is proved by defining a dual tableau-based decision procedure where a suitable blocking mechanism has been introduced and rules for compositional and complemented compositional formulae have been appropriately modified to deal with the constant $\mathbf{1}$ while preserving termination. The fragment $(\{\mathbf{1},\cup,\cap\}\,;\,\_)$ is a first result towards the use of entailment inside relational dual tableau-based decision procedures. We recall that relational entailment allows one to deal with properties of relational constants and of relational variables in dual tableau proofs without adding any specific rule to the basic set of decomposition rules. The $(\{\mathbf{1},\cup,\cap\}\,;\,\_)$-fragment allows one also to express some simple forms of inclusion between relations.

The paper [13] presents relational decision procedures in dual tableaux style for a class of relational logics admitting just one relational constant $R$ with the properties of reflexivity, transitivity, and heredity (i.e., if $xRy$ and $xPz$, then $yPz$, for all atomic relations $P$). These results are extended to multimodal logics in [7], where a class $\mathcal{RDL}^m$ of fragments of the logic $\mathsf{RL}$, allowing an unbounded number of relational constants which may enjoy the properties of reflexivity, transitivity, and heredity, is defined, and a dual tableau decision procedure for logics belonging to $\mathcal{RDL}^m$, called $\mathsf{RDT}_{\mathsf{RL}_\mathsf{L}}$, is introduced.

The $\mathsf{RDT}_{\mathsf{RL}_\mathsf{L}}$ system is constructed along the lines of the dual tableau methodology described in [20]. It consists of decomposition rules to analyze the structure of the formulae to be proved, of specific rules to deal with properties that can be enjoyed by the relational constants occurring in the formulae to be proved (such as reflexivity, transitivity, and heredity), and of axiomatic sets which specify the closure conditions. Each rule and axiom is applied in a deterministic way without resorting to external tools. Determinism of the decision procedure is guaranteed by enforcing an order of application of the rules obtained by associating with each rule a specific formula of the current node, called *pivot formula*, and by introducing a suitable order on the set of relational formulae. An important feature of the dual tableau procedure is a rule to handle the relational composition operator, that permits to decompose in a single step compositional formulae and negative compositional formulae sharing the same left object variable. In Table 1 we report the decomposition rule $(\mathsf{glob}\,;)$ to handle compositional and negative-compositional formulae sharing the same left object variable, and in Table 2 the specific rules $(\mathsf{ref}_i)$, $(\mathsf{tran}_i)$, and $(\mathsf{her}_i)$ to deal with relational constants $R_i$ satisfying the properties of reflexivity, transitivity, and heredity, respectively. The

$\mathsf{RDT}_{\mathsf{RL_L}}$ proof system is correct, complete, and terminating. Termination of the procedure is guaranteed by introducing some restrictions in rule applications to prevent infinite loops.

The class $\mathcal{RDL}^m$ can express several multimodal logics whose accessibility relations are reflexive, transitive, and satisfy the heredity property. Among them we recall the multimodal logic with reflexive and transitive frames and the description logic $\mathcal{ALC}_{R^+}$ [21].

**Table 1.** Decomposition rule for compositional formulae in $\mathsf{RDT}_{\mathsf{RL_L}}$.

$$(\mathsf{glob}\ ;)\ \frac{X \cup Y \cup \bigcup_{i \in I_\Phi} \{z_n(-(R_i\,;S_{q_i}))z_0, z_n(R_i\,;T_{j_i})z_0 : q_i \in Q_i, j_i \in J_i\}}{X \cup \bigcup_{i \in I_\Phi} \{z_{n_{q_i}}(-S_{q_i})z_0, z_{n_{q_i}}T_{j_i}z_0 : q_i \in Q_i, j_i \in J_i\}}$$

where:

- $n \geq 1$ and $Y$ is the set of literals with left variable $z_n$ occurring in the current node;
- $I_\Phi$ is the set of indices of constants of $\mathbb{RC}$ occurring in the current node $\Phi$;
- for all $T \in \mathbb{RT}_{\mathcal{RDL}^m}$, $z_nTz_0 \notin X$ (the only formulae in the premise that are neither compositional nor negative-compositional and have $z_n$ as left variable are in $Y$);
- $Q = \bigcup_{i \in I_\Phi} Q_i$ and $J = \bigcup_{i \in I_\Phi} J_i$ are sets of indices such that $Q_i \neq \emptyset$, for some $i \in I_\Phi$ (by this condition, if in the current node there is a formula $z_n(R_i\,;T_{j_i})z_0$, for some $i \in I_\Phi$, and no formula of type $z_n(-(R_i\,;S_{q_i}))z_0$ occurs in the current node, then $z_n(R_i\,;T_{j_i})z_0$ cannot be decomposed anymore and therefore it is not repeated in the successive nodes of the dual tableau);
- $S_{q_i}, T_{j_i} \in \mathbb{RT}_{\mathcal{RDL}^m}$, for all $q_i \in Q_i$, $j_i \in J_i$, with $i \in I_\Phi$;
- the set $N = \{n_{q_i} : q_i \in Q_i, i \in I_\Phi\}$ satisfies the following conditions:
  - the elements of $N$ are consecutive natural numbers,
  - $\min(N) = k+1$, where $k$ is the largest number such that $z_k$ occurs in the premise,
  - for all $n_{q_i}, n_{q'_{i'}} \in N$, we have $n_{q_i} < n_{q'_{i'}}$ if and only if $\langle R_i, S_{q_i}\rangle < \langle R_{i'}, S_{q'_{i'}}\rangle$;
- the pivot of $(\mathsf{glob}\ ;)$ is the formula $z_n(-(R_i\,;S_{q_i}))z_0$ with the minimal pair $\langle R_i, S_{q_i}\rangle$.

## 3 Current and future developments in qualitative reasoning and related settings

Qualitative reasoning (QR) is an area of Artificial Intelligence especially useful for handling situations in which purely numeric methods are too complex or the available knowledge is vague or incomplete. QR methods are alternatives to quantitative ones. Although quantitative approaches usually provide precise answers to precise questions and assumptions, their applicability is very limited. Indeed, most problems of the physical world cannot be solved by purely quantitative methods. Moreover, even if numeric methods are used in solving a problem, often we have to neglect some factors as unimportant in the given context. Qualitative representation make only as many distinctions (qualitative classes) as necessary to identify objects and to set essential relationships between them. Thus, the main aim of QR is to design methods for dealing with

**Table 2.** Reflexivity, transitivity, and heredity rules for $\mathsf{RDT}_{\mathsf{RL_L}}$.

Reflexivity rule:

$$(\mathsf{ref}_i)\ \frac{X \cup \{z_n(R_i^s\,;T)z_0\}}{X \cup \{z_n(R_i^s\,;T)z_0\} \cup \{z_n(R_i^j\,;T)z_0 : j \in \{0,\dots,s-1\}\}}\,,$$

where $n, s \geq 1$ and $T \in \mathbb{RT}_{\mathcal{RDL}^m}$ is either a non-compositional term or a compositional term $(R_j\,;T')$, with $j \neq i$, and $z_n(R_i^t\,;T) \notin X$, for all $t > s$.

Transitivity rule:

$$(\mathsf{tran}_i)\ \frac{X \cup \{z_n(R_i\,;T)z_0\}}{X \cup \{z_n(R_i\,;T)z_0\} \cup \{z_n(R_i^2\,;T)z_0\}}\,,$$

where $n \geq 1$ and $T \in \mathbb{RT}_{\mathcal{RDL}^m}$ is either a non-compositional term or a compositional term $(R_j\,;T')$, with $j \neq i$.

Heredity rule:

$$(\mathsf{her}_i)\frac{X \cup \{z_n(-(R_i\,;T))z_0\} \cup \{z_n(-\mathsf{p}_j)z_0 : j \in J_\Phi\}}{X \cup \{z_n(-(R_i^s\,;T))z_0\} \cup \{z_n(-\mathsf{p}_j)z_0 : j \in J_\Phi\} \cup \{z_n(R_i\,;(-\mathsf{p}_j))z_0\ : j \in J_\Phi\}}$$

where: $n \geq 1$, $T \in \mathbb{RT}_{\mathcal{RDL}^m}$, and $z_n(-\mathsf{p}_j)z_0 \notin X$, for any $\mathsf{p}_j \in \mathbb{RV}$.
The pivot of the rule $(\mathsf{her}_i)$ is $z_n(-(R_i\,;T))z_0$.

commonsense knowledge without using numerical computation. Over the years, QR has developed various techniques of qualitative representation and proved to have a wide variety of potential applications.

An approach in QR concerned with the analysis of physical systems in terms of relative magnitudes is order-of-magnitude reasoning (OMR). Note that in purely qualitative reasoning the lack of quantitative information may often lead to ambiguity. One of the aims of OMR is to avoid this problem. Thus, in a sense, order-of-magnitude methods are situated midway between numerical and qualitative formalisms. OMR approaches are usually based on a family of binary order-of-magnitude relations, among which are comparability, negligibility, and closeness relations. Such a relative order-of-magnitude paradigm is rich enough to capture many types of commonsense reasoning. For instance, as presented in [5], using OMR we may represent and reason in a qualitative way about the behaviour of a device to control automatically the temperature in a building. In this representation, temperature is assumed to be either VERY HOT, HOT, OK, COLD, or VERY COLD. Next, some general and specific axioms about the behaviour of the system are introduced. The axioms are expressed in terms of order-of-magnitude relations, for example: 'If the temperature is OK and it is incremented in some value with respect to which the actual value is negligible, then humidification or extra humidification system must operate'. A detailed example of order-of-magnitude representation is discussed in [5].

The simplest logic for OMR, $\mathsf{OMR}_N$ for short, has been introduced in [5]. It is a multimodal logic with five propositional constants and four necessity operators determined by a strict linear order $<$, the negligibility relation $N$, and their converses. The intended meaning of the negligibility relation defined on the set of real numbers is as follows: 0 is negligible with respect to any real number and each number from the qualitative class of sufficiently small numbers is negligible with respect to any number from the qualitative class of sufficiently large numbers. Models of $\mathsf{OMR}_N$ are Kripke structures with a strict linear order, the negligibility relation, and five landmarks which represent propositional constants and divide the universe of a model into seven qualitative classes. Sound and complete relational proof systems for $\mathsf{OMR}_N$ have been presented in [16]. However, the system presented in [16] is not a decision procedure for $\mathsf{OMR}_N$, even though the logic is decidable, as proved in [12].

Recently, it has been shown that the logic $\mathsf{OMR}_N(C)$, which is the extension of $\mathsf{OMR}_N$ with the comparability relation (see [4]), is also decidable, as shown in [3]. Its relational proof system, which, however, is not a decision procedure, can be found in [20, Chapter 15].

Thus, a natural question is how to modify the relational dual tableau systems for the logics $\mathsf{OMR}_N$ and $\mathsf{OMR}_N(C)$ into their relational decision procedures. Our first step is the construction of a decision procedure along the lines of the one presented in [7] for a multimodal logic over a strict linear order. Since there are relational decision procedures for logics with transitive relations, to design a relational decision procedure for logics over a strict linear order we need, in particular, a rule for the property of connectedness (for all $x$ and $y$, either $xRy$ or $yRx$ or $x = y$) and irreflexivity (for all $x$, not $xRx$). Recall that these properties are not definable by a standard modal formula, while they are definable by a relational formula. This fact can be very useful for the construction of a desired decision procedure.

In [6], a more complex logic for order-of-magnitude reasoning has been introduced. It includes modal operators determined by the relations of negligibility, non-closeness, and distance. In particular, a simplified version of it, called $\mathsf{OMR_D}$, has been considered. This is the logic determined by a class of Kripke frames with a strict linear order and a distance relation. As it turns out, the relations of non-closeness and negligibility are definable in $\mathsf{OMR_D}$. It has been shown that the logic $\mathsf{OMR_D}$ is sound and complete over Kripke models with a strict total order and distance relation.

A sound and complete dual tableau system for the relational logic associated with $\mathsf{OMR_D}$ has been constructed in [15]. Such a system can be used to verify whether a formula is $\mathsf{OMR_D}$-provable. However, it does not provide any decision procedure for $\mathsf{OMR_D}$, as it may generate infinite trees. In fact, the decision problem for the logic $\mathsf{OMR_D}$ is still an open question, which we plan to study. In case of a positive solution, we also intend to construct a dual tableau-based decision procedure for the relational logic $\mathsf{OMR_D}$.

In addition, we intend to construct relational decision procedures for fragments of multimodal logics whose accessibility relations satisfy properties such as seriality, symmetry, euclideanity, partial functionality, and weak density. We have already designed specific rules for each of these properties to be integrated in the procedure presented in [7]. However, in order to preserve termination of the procedure, we still have to define suitable conditions and restrictions on their application.

**Acknowledgments.** This work was supported by the Polish National Science Centre research project DEC-2011/02/A/HS1/00395.

## References

1. F. Baader. Description logics. In: *Reasoning Web: Semantic Technologies for Information Systems, 5th International Summer School.* Lecture Notes in Computer Science 5689, 2009, pp. 1–39.
2. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
3. Burrieza, A.: Decidability of a logic for order of magnitude qualitative reasoning with comparability and negligibility relations. CAEPIA 2013: 101-110.
4. Burrieza, A., Ojeda-Aciego, M.: A multimodal logic approach to order of magnitude qualitative reasoning with comparability and negligibility relations. Fundamenta Informaticae. 68 (1-2), 21-46 (2005).
5. Burrieza, A., Muñoz-Velasco, E., Ojeda-Aciego, M.: Order of magnitude qualitative reasoning with bidirectional negligibility. In: Marín, R., Onaindia, E., Bugarín, A., Santos, J. (eds.) CAEPIA 2005. LNCS, vol. 4177, pp. 370–378. Springer (2005).
6. Burrieza, A., Muñoz-Velasco, E., Ojeda-Aciego, M.: A logic for order-of-magnitude reasoning with negligibility, non-closeness and distance. In: Borrajo, D., Castillo, L., Corchado, J.M. (eds.) CAEPIA 2007. LNCS, vol. 4788, pp. 210–219 (2007).
7. D. Cantone, J. Golińska-Pilarek, M. Nicolosi-Asmundo. A Relational Dual Tableau Decision Procedure for Multimodal and Description Logics. In: *Proceedings of the 9th International Conference on Hybrid Artificial Intelligence Systems*, Salamanca, Spain, June 11-13, 2014. LNCS, vol. 8480, pp. 466-477 (2014).
8. D. Cantone, M. Nicolosi-Asmundo, E. Orłowska. Dual tableau-based decision procedures for some relational logics. In: *Proceedings of the 25th Italian Conference on Computational Logic*, Rende, Italy, July 7-9, 2010, pp. 1–16. CEUR Workshop Proceedings vol. 598.
9. D. Cantone, M. Nicolosi-Asmundo, E. Orłowska. Dual tableau-based decision procedures for relational logics with restricted composition operator. *Journal of Applied Non-classical Logics* 21, No 2, 2011, 177-200.
10. D. Cantone, M. Nicolosi-Asmundo, E. Orłowska.A Dual Tableau-based Decision Procedure for a Relational Logic with the Universal Relation. Submitted (2014).
11. A. Formisano and M. Nicolosi-Asmundo. An efficient relational deductive system for propositional non-classical logics. *Journal of Applied Non-Classical Logics*, vol. 16(3-4), pp. 367-408 (2006).
12. Golińska-Pilarek, J.: On decidability of a logic for order of magnitude qualitative reasoning with bidirectional negligibility. Lecture Notes in Computer Science 7519, 2012, pp. 255–266, doi: 10.1007/978-3-642-33353-8.
13. J. Golińska-Pilarek, T. Huuskonen, E. Munoz-Velasco, Relational dual tableau decision procedures and their applications to modal and intuitionistic logics. *Annals of Pure and Applied Logics* vol. 165 (2), pp. 409–427 (2014).
14. Golińska-Pilarek, J., Mora, A., Muñoz-Velasco, E.: An ATP of a relational proof system for order-of-magnitude reasoning with negligibility, non-closeness and distance. In: Ho, T-B., Zhou, Z-H. (eds.) PRICAI 2008. LNAI, vol. 5351, pp. 128–139 (2008).

15. Golińska-Pilarek, J., Muñoz-Velasco, E.: Relational approach for a logic for order-of-magnitude qualitative reasoning with negligibility, non-closeness and distance. Logic Journal of IGPL. 17(4), 375–394 (2009).
16. Golińska-Pilarek, J., Muñoz-Velasco, E.: Dual tableau for a multimodal logic for order-of-magnitude qualitative reasoning with bidirectional negligibility. International Journal of Computer Mathematics. 86(10-11), 1707–1718 (2009).
17. J. Golińska-Pilarek, E. Munoz-Velasco, and A. Mora. Implementing a relational theorem prover for modal logic K. *International Journal of Computer Mathematics*, 88(9):1869–1884 (2011).
18. J. Golińska-Pilarek, E. Munoz-Velasco, and A. Mora. A new deduction system for deciding validity in modal logic K. *Logic Journal of IGPL* 19(2):425–434 (2011).
19. E. Orłowska. Relational interpretation of modal logics. In: *H. Andreka, D. Monk, and I. Nemeti eds., Algebraic Logic. Colloquia Mathematica Societatis Janos Bolyai*, vol. 54, pp. 443–471, North Holland, 1988.
20. Orłowska, E., Golińska-Pilarek, J.: Dual Tableaux: Foundations, Methodology, Case Studies. Trends in Logic 36, Springer, 2011.
21. U. Sattler. A concept language extended with different kinds of transitive roles. In: *G. Görz and S. Hölldobler, eds., 20 Deutsche Jahrestagung für Künstliche Intelligenz, n. 1137 LNAI*, pp. 333–345. Springer-Verlag, 1996.

# Keyword Index

# Author Index