Lecture Notes in Artificial Intelligence 4790

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Nachum Dershowitz   Andrei Voronkov (Eds.)

Logic for Programming, Artificial Intelligence, and Reasoning

14th International Conference, LPAR 2007
Yerevan, Armenia, October 15-19, 2007
Proceedings

Springer

Series Editors

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

Volume Editors

Nachum Dershowitz
Tel Aviv University
School of Computer Science
Ramat Aviv, Tel Aviv, 69978 Israel
E-mail: nachumd@post.tau.ac.il

Andrei Voronkov
University of Manchester
School of Computer Science, Kilburn Building
Oxford Road, Manchester M13 9PL, UK
E-mail: voronkov@cs.man.ac.uk

Library of Congress Control Number: 2007937099

CR Subject Classification (1998): I.2.3, I.2, F.4.1, F.3, D.2.4, D.1.6

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-75558-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-75558-6 Springer Berlin Heidelberg New York

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

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2007
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12172481 06/3180 5 4 3 2 1 0

Preface

This volume contains the papers presented at the [illegible] (LPAR 2007), held in Yerevan, Armenia on October 15–19, 2007.

LPAR evolved out of the 1st and 2nd [illegible], held in Irkutsk, in 1990, and aboard the ship "Michail Lomonosov", in 1991. The idea of organizing a conference came largely from Robert Kowalski, who also proposed the creation of the Russian Association for Logic Programming. In 1992, it was decided to extend the scope of the conference. Due to considerable interest in automated reasoning in the former Soviet Union, the conference was renamed [illegible] (LPAR). Under this name three meetings were held in 1992–1994: on board the ship "Michail Lomonosov" (1992); in St. Petersburg, Russia (1993); and on board the ship "Marshal Koshevoi" (1994).

In 1999, the conference was held in Tbilisi, Georgia. At the suggestion of Michel Parigot, the conference changed its name again to [illegible] (preserving the acronym LPAR!), reflecting an interest in additional areas of logic. LPAR 2000 was held Reunion Island, France.

In 2001, the name (but not the acronym) changed again to its current form. The 8th to the 13th meetings were held in the following locations: Havana, Cuba (2001); Tbilisi, Georgia (2002); Almaty, Kazakhstan (2003); Montevideo, Uruguay (2004); Montego Bay, Jamaica (2005); and Phnom Penh, Cambodia (2006).

There were 78 submissions to LPAR 2007, each of which was reviewed by at least four programme committee members. The committee deliberated electronically via the EasyChair system and ultimately decided to accept 36 papers. The programme also included three invited talks by Johann Makowsky, Helmut Veith, and Richard Waldinger, and about 15 short papers, extended abstracts of which were distributed to participants.

We are most grateful to the 39 members of the programme committee who agreed to serve under short notice, and to them and the 123 additional reviewers who performed their duties most admirably under the tightest of time constraints.

August 2007

Nachum Dershowitz
Andrei Voronkov

Organization

Programme Chairs

Nachum Dershowitz
Andrei Voronkov

Programme Committee

Eyal Amir
Franz Baader
Matthias Baaz
Peter Baumgartner
Nikolaj Bjorner
Maria Paola Bonacina
Alessandro Cimatti
Michael Codish
Simon Colton
Byron Cook
Thomas Eiter
Christian Fermueller
Georg Gottlob
Reiner Haehnle
John Harrison
Brahim Hnich
Tudor Jebelean
Deepak Kapur
Delia Kesner
Helene Kirchner
Michael Kohlhase
Konstantin Korovin
Viktor Kuncak
Leonid Libkin
Christopher Lynch
Hrant Marandjian
Maarten Marx
Luke Ong
Peter Patel-Schneider
Brigitte Pientka
I.V. Ramakrishnan
Albert Rubio
Ulrike Sattler
Geoff Sutcliffe
Cesare Tinelli
Ralf Treinen
Toby Walsh
Christoph Weidenbach
Frank Wolter

Local Organization

Hrant Marandjian
Artak Petrosyan
Vladimir Sahakyan
Yuri Shoukourian

External Reviewers

Wolfgang Ahrendt
Jean-Marc Andreoli
Takahito Aoto
Lev Beklemishev
Christoph Benzmueller
Josh Berdine

Karthikeyan Bhargavan
Benedikt Bollig
Thierry Boy de la Tour
Sebastian Brand
Jan Broersen
Richard Bubel
Diego Calvanese
Agata Ciabattoni
Horatiu Cirstea
Hubert Comon-Lundh
Giuseppe De Giacomo
Stéphanie Delaune
Bart Demoen
Stephane Demri
Roberto Di Cosmo
Lucas Dixon
Phan Minh Dung
Mnacho Echenim
Michael Fink
Sergio Flesca
Pascal Fontaine
Carsten Fuhs
Isabelle Gnaedig
John Gallagher
Silvio Ghilardi
Martin Giese
Christoph Gladisch
Lluis Godo
Mayer Goldberg
Georges Gonthier
Gianluigi Greco
Yves Guiraud
Hannaneh Hajishirzi
Patrik Haslum
James Hawthorne
Emmanuel Hebrard
Joe Hendrix
Ian Hodkinson
Matthias Horbach
Paul Houtmann
Dominic Hughes
Ullrich Hustadt
Rosalie Iemhoff
Florent Jacquemard
Tomi Janhunen
Mouna Kacimi
George Katsirelos
Yevgeny Kazakov
Benny Kimelfeld
Boris Konev
Adam Koprowski
Robert Kosik
Pierre Letouzey
Tadeusz Litak
Thomas Lukasiewicz
Carsten Lutz
Michael Maher
Toni Mancini
Nicolas Markey
Annabelle McIver
François Métayer
George Metcalfe
Aart Middeldorp
Dale Miller
Sanjay Modgil
Angelo Montanari
Barbara Morawska
Boris Motik
Normen Mueller
K. Narayan Kumar
Alan Nash
Immanuel Normann
Albert Oliveras
Joel Ouaknine
Gabriele Puppis
David Parker
Nicolas Peltier
Ruzica Piskac
Toniann Pitassi
Nir Piterman
Femke van Raamsdonk
Christian Retoré
Philipp Rümmer
Florian Rabe
C.R. Ramakrishnan
Silvio Ranise
Mark Reynolds
Alexandre Riazanov
Christophe Ringeissen
Iemhoff Rosalie

Riccardo Rosati
Camilla Schwind
Stefan Schwoon
Niklas Sörensson
Volker Sorge
Lutz Strassburger
Peter Stuckey
Aaron Stump
S.P. Suresh
Deian Tabakov
Ernest Teniente
Sebastiaan Terwijn
Michael Thielscher
Nikolai Tillmann
Stephan Tobies
Emina Torlak
Dmitry Tsarkov
Antti Valmari
Ivan Varzinczak
Lionel Vaux
Luca Vigano
Uwe Waldmann
Dirk Walther
Claus-Peter Wirth
Richard Zach
Alessandro Zanarini
Calogero Zarba
Chang Zhao
Hans de Nivelle

Table of Contents

From Hilbert's Program to a Logic Toolbox (Invited Talk) 1
[illegible]

On the Notion of Vacuous Truth (Invited Talk) 2
[illegible]

Whatever Happened to Deductive Question Answering? (Invited Talk) 15
[illegible]

Decidable Fragments of Many-Sorted Logic 17
[illegible]

One-Pass Tableaux for Computation Tree Logic 32
[illegible]

Extending a Resolution Prover for Inequalities on Elementary Functions 47
[illegible]

Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic 62
[illegible]

Monadic Fragments of Gödel Logics: Decidability and Undecidability Results 77
[illegible]

Least and Greatest Fixed Points in Linear Logic 92
[illegible]

The Semantics of Consistency and Trust in Peer Data Exchange Systems 107
[illegible]

Completeness and Decidability in Sequence Logic 123
[illegible]

HORPO with Computability Closure: A Reconstruction 138
[illegible]

Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs 151
[illegible]

Matching in Hybrid Terminologies 166
[illegible]

Verifying Cryptographic Protocols with Subterms Constraints 181
[illegible]

Deciding Knowledge in Security Protocols for Monoidal Equational Theories 196
[illegible]

Mechanized Verification of CPS Transformations 211
[illegible]

Operational and Epistemic Approaches to Protocol Analysis: Bridging the Gap 226
[illegible]

Protocol Verification Via Rigid/Flexible Resolution 242
[illegible]

Preferential Description Logics 257
[illegible]
[illegible]

On Two Extensions of Abstract Categorial Grammars 273
[illegible]

Why Would You Trust [illegible]? 288
[illegible]

How Many Legs Do I Have? Non-Simple Roles in Number Restrictions Revisited 303
[illegible]

On Finite Satisfiability of the Guarded Fragment with Equivalence or Transitive Guards 318
[illegible]

Data Complexity in the $\mathcal{EL}$ Family of Description Logics 333
[illegible]

An Extension of the Knuth-Bendix Ordering with LPO-Like Properties 348
[illegible]

Retractile Proof Nets of the Purely Multiplicative and Additive Fragment of Linear Logic 363
[illegible]

Integrating Inductive Definitions in SAT 378
[illegible]

The Separation Theorem for Differential Interaction Nets 393
[illegible]

Complexity of Planning in Action Formalisms Based on Description Logics 408
[illegible]

Faster Phylogenetic Inference with MXG 423
[illegible]

Enriched μ–Calculus Pushdown Module Checking 438
[illegible]

Approved Models for Normal Logic Programs 454
[illegible]

Permutative Additives and Exponentials 469
[illegible]

Algorithms for Propositional Model Counting 484
[illegible]

Completeness for Flat Modal Fixpoint Logics (Extended Abstract) 499
[illegible]

𝔽𝔻ℕℂ: Decidable Non-monotonic Disjunctive Logic Programs with Function Symbols 514
[illegible]

The Complexity of Temporal Logic with Until and Since over Ordinals 531
[illegible]

ATP Cross-Verification of the Mizar MPTP Challenge Problems 546
[illegible]

Author Index 561

From Hilbert's Program to a Logic Toolbox

Johann A. Makowsky

Technion—Israel Institute of Technology
`janos@cs.technion.ac.il`

In this talk we discuss what, according to my long experience, every computer scientists should know from logic. We concentrate on issues of modelling, interpretability and levels of abstraction. We discuss how the minimal toolbox of logic tools should look like for a computer scientist who is involved in designing and analyzing reliable systems. We shall conclude that many classical topics dear to logicians are less important than usually presented, and that less known ideas from logic may be more useful for the working computer scientist.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, p. 1, 2007.
© Springer-Verlag Berlin Heidelberg 2007

On the Notion of Vacuous Truth

Marko Samer[1] and Helmut Veith[2]

[1] Department of Computer Science
Durham University, UK
marko.samer@durham.ac.uk
[2] Institut für Informatik (I-7)
Technische Universität München, Germany
veith@in.tum.de

Abstract. The model checking community has proposed numerous definitions of vacuous satisfaction, i.e., formal criteria which tell whether a temporal logic specification holds true on a system model *for the intended reason*. In this paper we attempt to study the notion of vacuous satisfaction from first principles. We show that despite the apparently vague formulation of the vacuity problem, most proposed notions of vacuity for temporal logic can be cast into a uniform and simple framework, and compare previous approaches to vacuity detection from this unified point of view.

1 Introduction

193. *What does this mean: the truth of a proposition is certain?*
L. Wittgenstein, On Certainty [35]

Modern model checkers are equipped with capabilities which go well beyond deciding the truth of a temporal specification φ_{Spec} on a system $\mathfrak{S}$. Most importantly, when a model checker determines that the specification is violated, i.e., $\mathfrak{S} \not\models \varphi_{\mathsf{Spec}}$, it will output a counterexample, for instance a program trace, which illustrates the failure of the specification φ_{Spec} on $\mathfrak{S}$. This counterexample is a piece of evidence which the user can analyze to understand and diagnose the problem. Since counterexamples should be perceptually and mathematically simple, counterexample generation has both algorithmic and psychological aspects [12,13,17].

In this paper, we are concerned with the dual situation when the model checker asserts $\mathfrak{S} \models \varphi_{\mathsf{Spec}}$. Industrial practice shows that 20% of successful model checking passes are *vacuous*, i.e., φ_{Spec} is satisfied for some trivial or unintended reason [4]. A classical example of vacuity is *antecedent failure*, where the model checker correctly asserts

$$\mathfrak{S} \models \mathbf{AG}\,(\mathsf{trigger\_event} \Rightarrow \varphi),$$

but a closer inspection shows that in fact trigger_event is always false, and thus the implication becomes vacuously true. The total absence of trigger_event may be an indicator of erroneous system behavior, and should be reported to the user.

A model checker with automated vacuity detection thus has three kinds of outputs:

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 2–14, 2007.
© Springer-Verlag Berlin Heidelberg 2007

	Model Theoretic Result	**Supporting Evidence**
(i)	$\mathfrak{S} \not\models \varphi_{\mathsf{Spec}}$	Counterexample
(ii)	$\mathfrak{S} \models \varphi_{\mathsf{Spec}}$ *vacuously*	Explanation of Vacuity
(iii)	$\mathfrak{S} \models \varphi_{\mathsf{Spec}}$ *non-vacuously*	Witness of Non-Vacuity

The central question in the vacuity literature concerns the line which separates cases (ii) and (iii). In other words: When is a specification vacuously satisfied? Similar as counterexample generation, vacuity detection also relies on algorithmic and psychological insights. A recent thread of papers [1,3,4,5,6,8,11,15,18,19,22,23,25,27,28,30,33] have given different, sometimes competing, definitions, including one by the present authors [28,30].

The controversial examples and discussions of vacuity in the literature have their origin in a principal limitation of formal vacuity detection: Declaring φ_{Spec} to be vacuous on $\mathfrak{S}$ means that the specification φ_{Spec} is inadequate to capture the desired system behavior. Adequacy of specifications however is a meta-logical property that cannot be addressed inside the temporal logic, because we need *domain knowledge* to distinguish adequate specifications from inadequate ones.

The current paper develops the line of thought started in [30] in that it focuses on the notion of *vacuity grounds* as the main principle in vacuity detection. Vacuity grounds are explanations of $\mathfrak{S} \models \varphi_{\mathsf{Spec}}$, which entail the specification, but are perceptionally simpler and logically stronger. Formally, a vacuity ground is a formula φ_{Fact} such that

$$\mathfrak{S} \models \varphi_{\mathsf{Fact}} \qquad \text{and} \qquad \varphi_{\mathsf{Fact}} \models \varphi_{\mathsf{Spec}}$$

where φ_{Fact} is simpler than φ_{Spec}; criteria for simplicity will be discussed below. Thus, vacuity grounds can be viewed as a form of *interpolants* between $\mathfrak{S}$ and φ_{Spec}.

Employing vacuity grounds, it is easy to resolve conflicts between different notions of vacuity: *the same specification may be tagged as vacuous or non-vacuous, depending on which vacuity grounds the verification engineer is willing to admit*. In the antecedent failure example mentioned above, the natural vacuity ground is $\mathbf{AG}\,\neg\mathsf{trigger\_event}$. Equipped with this feedback, the verification engineer can draw a well-informed conclusion about vacuous satisfaction. We thus arrive at a revised output scheme for model checkers which support vacuity detection:

	Model Theoretic Result	**Supporting Evidence**
(i)	$\mathfrak{S} \not\models \varphi_{\mathsf{Spec}}$	Counterexample
(ii)	$\mathfrak{S} \models \varphi_{\mathsf{Spec}}$	Vacuity grounds from which the engineer decides on vacuity

We believe that our approach yields the first genuinely semantical definition of vacuity. In the rest of the paper, we compare our notion of vacuity with existing definitions, and show how our approach subsumes and uniformly explains a significant part of previous work. Moreover, we show that our approach captures cases of vacuous satisfaction not covered in the literature.

In Section 2, we review the capabilities and limitations of the prevailing definitions of vacuity – which we call unicausal semantics – and argue that they have limited explanatory power. As they are intimately tied to the syntax of φ_{Spec}, two logically equivalent specifications may become vacuous in one case and non-vacuous in the other.

Section 3 introduces our interpolation-based vacuity framework. We show that unicausal semantics yield a specific class of vacuity grounds related to uniform interpolation, thus embedding unicausal semantics into our framework. We briefly review our previously published vacuity methodology based on temporal logic query solving [30], and derive basic complexity results for the general case presented here.

Technical Preliminaries. We assume the reader is familiar with the temporal logics CTL and LTL, Kripke structures, and other standard notions. For Kripke structures $\mathfrak{S}$ and $\mathfrak{S}'$ we write $\mathfrak{S} \cong_{cnt} \mathfrak{S}'$ to denote that $\mathfrak{S}$ and $\mathfrak{S}'$ are counting-bisimilar, and we write $\mathfrak{S} \cong \mathfrak{S}'$ to denote that $\mathfrak{S}$ and $\mathfrak{S}'$ are bisimilar. We write $\varphi(\psi)$ to denote that ψ occurs once or several times as subformula in φ; we write $\varphi(x)$ to denote the formula $\varphi[\psi \leftarrow x]$ obtained from φ by replacing all occurrences of ψ by x. The formula $\varphi(x)$ is called *monotonic* in x if $\alpha \Rightarrow \beta$ implies $\varphi(\alpha) \Rightarrow \varphi(\beta)$. It holds that φ is monotonic in x if all occurrences of x are positive, and dually for anti-monotonicity. We say φ is *(semantically) unipolar* in x if φ is either monotonic or anti-monotonic in x; otherwise, we call φ multipolar. In the following, when we speak about unipolar formulas, we shall without loss of generality assume monotonicity.

2 Unicausal Vacuity Semantics

> ***199.*** *The reason why the use of the expression "true or false" has something misleading about it is that it is like saying "it tallies with the facts or it doesn't", and the very thing that is in question is what "tallying" is here.* [35]

The *irrelevance of a subformula* for the satisfaction of a temporal specification φ_{Spec} is a natural indicator for vacuity: If on a model $\mathfrak{S}$, subformula ψ of φ_{Spec} can be arbitrarily modified without affecting the truth of the specification, φ_{Spec} is declared vacuous. This approach [4,5] has been the seminal paradigm for most of the research on vacuity. It has the obvious advantage that the vacuity of the specification can be (syntactically) traced back to a subformula, and that (non)-vacuity can be explained to the engineer on the grounds of the temporal logic. Syntactic vacuity however is usually hard to evaluate and not "robust" [1], i.e., dependent on the syntax of the specification and on changes in the system which are not related to the specification. The quest for efficiency and robustness has motivated new semantics for vacuity [1,8,18] which quantify over the allegedly vacuous subformulas. We shall refer to these semantics as *unicausal semantics*, since they all are attempts to obtain a (unique) formula $\forall x.\varphi(x)$ – which we call a *unicausal vacuity ground* – such that model checking $\mathfrak{S} \models \forall x.\varphi(x)$ determines the vacuity of φ on $\mathfrak{S}$ with respect to subformula ψ. The different possibilities to define the universal quantifier give rise to different unicausal semantics:

1. In the **formula semantics**, $\forall x$ ranges over all truth functions for x over the language L of $\mathfrak{S}$, i.e., $\forall x.\varphi(x)$ amounts to a (possibly infinitary) big conjunction of formulas $\quad_{\theta \in L} \varphi(\theta)$.

For the following definitions, let $\star$ be a new atomic proposition which occurs neither in $\mathfrak{S}$ nor in φ. Given a structure $\mathfrak{S}$, a $\star$-labeling ℓ labels some states of $\mathfrak{S}$ with $\star$, resulting in a structure $\ell(\mathfrak{S})$.

Table 1. Evidence for non-vacuity of $\mathfrak{S} \models \varphi(\psi)$ with respect to ψ

Formula Semantics	[4,5]
A formula θ over the language of $\mathfrak{S}$ such that $\mathfrak{S} \not\models \varphi(\theta)$.	
Structure Semantics	[1]
A $\star$-labeling ℓ of $\mathfrak{S}$, such that $\ell(\mathfrak{S}) \not\models \varphi(\star)$.	
Tree Semantics	[1]
A *new* structure $\mathfrak{S}' \cong_{cnt} \mathfrak{S}$ together with a $\star$-labeling ℓ of $\mathfrak{S}'$, such that $\ell(\mathfrak{S}') \not\models \varphi(\star)$.	
Bisimulation Semantics	[18]
A *new* structure $\mathfrak{S}' \cong \mathfrak{S}$ together with a $\star$-labeling ℓ of $\mathfrak{S}'$, such that $\ell(\mathfrak{S}') \not\models \varphi(\star)$.	

2. In the **structure semantics**, $\forall x$ ranges over all labelings of $\mathfrak{S}$, i.e., $\mathfrak{S} \models \forall x.\varphi(x)$ iff for all $\star$-labelings ℓ it holds that $\ell(\mathfrak{S}) \models \varphi(\star)$.
3. In the **tree semantics**, $\forall x$ ranges over all labelings of structures counting-bisimilar to $\mathfrak{S}$, i.e., $\mathfrak{S} \models \forall x.\varphi(x)$ iff for all $\mathfrak{S}' \cong_{cnt} \mathfrak{S}$ and $\star$-labelings ℓ of $\mathfrak{S}'$ it holds that $\ell(\mathfrak{S}') \models \varphi(\star)$.
4. In the **bisimulation semantics**, $\forall x$ ranges over all labelings of structures bisimilar to $\mathfrak{S}$, i.e., $\mathfrak{S} \models \forall x.\varphi(x)$ iff for all $\mathfrak{S}' \cong \mathfrak{S}$ and $\star$-labelings ℓ of $\mathfrak{S}'$ it holds that $\ell(\mathfrak{S}') \models \varphi(\star)$.

Importantly, all four unicausal semantics coincide when $\varphi(x)$ is monotonic in x; in this case, $\forall x.\varphi(x)$ is equivalent to $\varphi(\mathsf{false})$ which can be easily model checked [22,23]. Thus, the differences between the unicausal semantics appear only when $\varphi(x)$ is multipolar with respect to x.

When comparing the different notions of unicausal vacuity, it is natural to consider the evidence that the model checker can provide in case of non-vacuity, cf. item (iii) in the first output scheme of Section 1. In the literature, this evidence was referred to as *interesting witness* [5,22,23]. Table 1 summarizes the evidence we obtain for the different unicausal semantics above.

Examples illustrating the differences between these semantics are shown in Table 2. The specification there demonstrate that even on the small structures of Figure 1, the unicausal semantics differ tremendously.

Remark 1. The structure quantifier guarantees $\forall x.\varphi(x) \models \varphi(\psi)$ only when ψ is a state formula. In case of LTL, this means that ψ is either a propositional subformula or the whole specification.

Remark 2. The quantifiers used for unicausal vacuity detection have been studied independently of vacuity. The extension of a modal logic by the *formula quantifier* remains a modal logic, because an infinitary temporal formula cannot distinguish bisimilar models [2]. The *structure quantifier* can be used to distinguish bisimulation-equivalent models, and thus, the resulting logic is not a modal logic. (For example, the formula $(\forall x.x) \vee (\forall x.\neg x)$ holds true only on a structure with a single state.) The computational

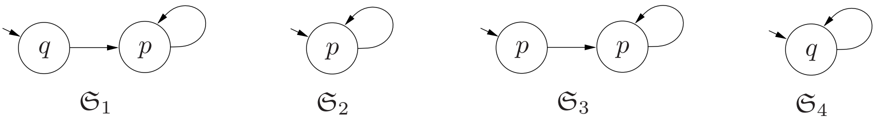


Fig. 1. Examples of Kripke structures

Table 2. Vacuity of $\varphi_1 = \mathbf{A}(p\,\mathbf{U}\,\mathbf{AG}(q \rightarrow \neg p))$, $\varphi_2 = \mathbf{AX}\,p \vee \mathbf{AX}\,\neg p$, $\varphi_3 = \mathbf{AG}\,p \vee \mathbf{AG}\,\neg p$, and $\varphi_4 = \mathbf{AG}(p \rightarrow \mathbf{AX}\neg p)$ with respect to p under different vacuity semantics. The structures are given in Figure 1 and Figure 2. Note that the non-vacuity witness $\mathfrak{S}_4$ for formula semantics is the only witness taken from Figure 1.

	Formula	**Structure**	**Tree**	**Bisimulation**
$\mathfrak{S}_1 \models \varphi_1(p)$	vacuous	vacuous	vacuous	vacuous
$\mathfrak{S}_2 \models \varphi_2(p)$	vacuous	vacuous	vacuous	$\mathfrak{S}'_0 \not\models \varphi_2(\star)$
$\mathfrak{S}_2 \models \varphi_3(p)$	vacuous	vacuous	$\mathfrak{S}'_3 \not\models \varphi_3(\star)$	$\mathfrak{S}'_3 \not\models \varphi_3(\star)$
$\mathfrak{S}_3 \models \varphi_3(p)$	vacuous	$\mathfrak{S}'_3 \not\models \varphi_3(\star)$	$\mathfrak{S}'_3 \not\models \varphi_3(\star)$	$\mathfrak{S}'_3 \not\models \varphi_3(\star)$
$\mathfrak{S}_4 \models \varphi_4(p)$	$\mathfrak{S}_4 \not\models \varphi_4(q)$	$\mathfrak{S}'_4 \not\models \varphi_4(\star)$	$\mathfrak{S}'_4 \not\models \varphi_4(\star)$	$\mathfrak{S}'_4 \not\models \varphi_4(\star)$

price to pay for the loss of modality is the undecidability of the quantified logic [16]. In combination with CTL, the *tree quantifier* is able to count the number of successors of a state [16], and thus able to break bisimulation-equivalence. While not a modal logic, the resulting logic is quite close to modal logic and retains decidability. The *bisimulation quantifier* has been rediscovered in the literature many times, and in different contexts, by the names of "bisimulation quantifier" [14], "Pitts quantifier" [34], "amorphous semantics" [16,18], and others. It is the natural quantifier to be used in the context of modal logic. Since it does not break bisimulation classes, it yields a conservative extension of a temporal logic. Uniform interpolation of the μ-calculus has been proved by elimination of bisimulation quantifiers [14].

Remark 3. Recent research has also considered vacuity detection for extensions of LTL by regular expressions [6,8]. This approach can also be viewed as an instance of unicausal semantics; for the sake of simplicity, however, we restrict the current paper to temporal logics.

2.1 Ramifications of Unicausal Vacuity

In this section, we discuss several problems and anomalies which arise from unicausal vacuity notions.

#1 Explanatory Power of Non-Vacuity Assertions

What confidence does the user gain in the model checking result when the model checker asserts non-vacuity? Recall Table 1 for the different notions of vacuity from this dual point of view:

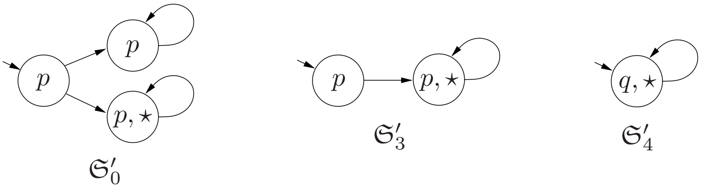


Fig. 2. Non-vacuity witnesses

- In *formula semantics*, the user knows that changing the specification $\varphi(\psi)$ into $\varphi(\theta)$ affects the truth value of the specification. Indeed, this explains the relevance of subformula ψ to the specification.
- In *structure semantics, tree semantics, and bisimulation semantics*, however, the evidence for non-vacuity is extremely weak: We change the specification $\varphi(\psi)$ into $\varphi(\star)$, where $\star$ is a new propositional symbol. Then we argue that the system $\mathfrak{S}$ (or a bisimilar system $\mathfrak{S}'$) can be labeled with $\star$ in such a way that $\varphi(\star)$ becomes false. (Thus, our non-vacuity argument is tantamount to formula semantics on a modified system with a new "imaginary" variable $\star$.) It is not clear how the introduction of $\star$ – which does not carry a meaning in the system $\mathfrak{S}$ – can give information about the relevance of subformula ψ to the user. Table 2 and its accompanying Figures 1 and 2 clearly indicate this problem.

  We see only one sound interpretation of these semantics: Suppose we know that our system $\mathfrak{S}$ is a coarse abstraction of the real system, i.e., our model $\mathfrak{S}$ is hiding many variables. Then the non-vacuity assertion states that it is consistent to assume the existence of a hidden variable $\star$ in the system which, if it were revealed, would give proof of non-vacuity in terms of formula semantics. The three semantics differ in the role of the imaginary variable $\star$: In structure semantics, variable $\star$ is uniquely defined on each state of the abstract system, while in tree semantics and bisimulation semantics, variable $\star$ depends on the execution history. A first exploration of the relationship between vacuity and abstraction was started in [18].

We conclude that only formula semantics gives confidence in the non-vacuity assertion. The other semantics provide tangible non-vacuity evidence only in very specific circumstances.

#2 Explanatory Power of Vacuity Assertions

What conclusions can the user draw from an assertion of vacuity? As temporal logic does not have quantifier elimination (cf. Section 3), it is in many cases impossible to write the vacuity ground $\forall x.\varphi(x)$ in plain temporal logic. The unicausal semantics, however, have the useful feature that each assertion of vacuity actually points out one or several subformulas which cause vacuous satisfaction. Comparing the different unicausal semantics, we obtain the following picture:

- In formula semantics, the vacuity assertion is relatively easy to understand: it says that no syntactic change in the subformulas of interest causes the specification to fail.

- In the other semantics, we are facing a problem dual to #1: The vacuity assertion expresses the fact that no hidden imaginary variable $\star$ can make $\varphi(\star)$ false. This criterion is stronger than the intuitive notion of vacuity – i.e., it will detect vacuity only in few cases.

We conclude that formula semantics again yields the most natural notion of unicausal vacuity, while the other three semantics report vacuity too rarely. This is dual to our observation in #1 that the semantics give weak evidence of non-vacuity.

#3 Expressive Power of Subformula Quantification

Vacuity detection by quantification over subformula occurrences entails a number of limitations discussed below.

#3.1 Pnueli's Observation

Pnueli [26] pointed out that a satisfied specification $\mathbf{AG}\,\mathbf{AF}\,p$ may be considered vacuous, when the model checker observes that the stronger formula $\mathbf{AG}\,p$ is true on the system. None of the unicausal vacuity notions is able to detect this notion of vacuity.

#3.2 Syntactically Unrelated Observations

Following Pnueli's example, there are arguable cases of vacuity where no syntactic relationship exists between the specification and the observation. For example, for a specification $\mathbf{EF}\,p$, the model checker may observe that in fact $\mathbf{AX}\,p$ holds. It is clear that this form of vacuity cannot be detected by unicausal semantics.

#3.3 Specifications with Single Propositions

The issues raised in #3.1 and #3.2 share the syntactic property that they contain a single occurrence of a propositional variable p. Consequently, the universal quantifier eliminates the truth-functional dependence on the propositional variable, and the quantified specification is either a tautology or a contradiction. For example, in Pnueli's example, quantification yields the formulas $\mathbf{AG}\,\mathsf{false}$ and $\mathbf{AG}\,\mathbf{AF}\,\mathsf{false}$ both of which are equivalent to false.[1] This proves that the observations (vacuity grounds) such as $\mathbf{AG}\,p$ from #3.1 and $\mathbf{AX}\,p$ from #3.2 are impossible in unicausal semantics.

#3.4 Vacuity by Disjunction

In the syntactic view of unicausal vacuity, certain specifications are always vacuous. In particular, if a disjunctive specification $\varphi \vee \psi$ holds true in a structure, then either $\varphi \vee \mathsf{false}$ or $\mathsf{false} \vee \psi$ holds true, and thus, the specification is inevitably vacuous.

#3.5 Stability Under Logical Equivalence

As explained in #3.4 above, $\varphi \vee \varphi$ is vacuous by construction, and thus, every specification is equivalent to a vacuous specification. A more interesting example is given by $\mathbf{EF}\,p$ which is equivalent to $p \vee \mathbf{EX}\,\mathbf{EF}\,p$. In the second formulation, the specification is always vacuous.

---

[1] Recall that in the unipolar case, universal quantification over a variable x is tantamount to setting it false.

Getting back to Pnueli's problem in #3.1, we even see that a reformulation of the specification $\mathbf{AG}\,\mathbf{AF}\,p$ into the equivalent specification $\mathbf{AG}(p \vee \mathbf{AF}\,p)$ enables us to quantify out the subformula $\mathbf{AF}\,p$, yielding a formula $\forall x.\mathbf{AG}(p \vee x)$ which is equivalent to $\mathbf{AG}(p \vee \mathsf{false})$, and thus to $\mathbf{AG}\,p$. Similarly, the problem raised in #3.3 can be solved using the specification $(\mathbf{EF}\,p) \vee (\mathbf{AX}\,p)$ instead of the (logically equivalent) specification $\mathbf{EF}\,p$.

#4 Causality

Our final concern (and indeed the original motivation for this research) is a fundamental issue raised by unicausal semantics. Given a specification $\varphi(\psi)$ and occurrences of a subformula ψ, each unicausal semantics associates the vacuity of φ with respect to ψ with a uniquely defined formula $\forall x.\varphi(x)$. Not only does such a construction revert the natural order between cause ($\mathfrak{S} \models \forall x.\varphi(x)$) and effect ($\mathfrak{S} \models \varphi(\psi)$), it is also independent of the system $\mathfrak{S}$. Thus, unicausal semantics represents a fairly simple instance of logical abduction.

3 Interpolation-Based Vacuity Detection

200. *Really "The proposition is either true or false" only means that it must be possible to decide for or against it. But this does not say what the ground for such a decision is like.* [35]

Recall from Section 1 that we define a vacuity ground as a simple formula φ_{Fact} such that

$$\mathfrak{S} \models \varphi_{\mathsf{Fact}} \qquad \text{and} \qquad \varphi_{\mathsf{Fact}} \models \varphi_{\mathsf{Spec}}. \tag{1}$$

Vacuity grounds serve as feedback for the verification engineer which helps him/her to decide whether the specification φ_{Spec} is vacuously satisfied. For example, in the cases #3.1, #3.2, and #3.4 above, natural candidates for vacuity grounds are $\mathbf{AG}\,p$, $\mathbf{AX}\,p$, and φ, respectively. In general, a model checker may output multiple vacuity grounds for a single specification.

When the vacuity grounds are chosen among modal temporal formulas, definition (1) is equivalent to

$$\chi_{\mathfrak{S}} \models \varphi_{\mathsf{Fact}} \qquad \text{and} \qquad \varphi_{\mathsf{Fact}} \models \varphi_{\mathsf{Spec}}. \tag{2}$$

where $\chi_{\mathfrak{S}}$ is the temporal formula which characterizes $\mathfrak{S}$ up to bisimulation equivalence. Thus, the vacuity ground φ_{Fact} is an *interpolant* between the system description $\chi_{\mathfrak{S}}$ and the specification φ_{Spec}. Consequently, vacuity analysis can be viewed as the process of finding *simple* interpolants between the system and the specification. Note that, technically, φ_{Spec} is usually itself a Craig interpolant, but not useful in vacuity detection. Thus, we need different notions of simplicity than the restriction to common variables.

3.1 Unicausal Vacuity Grounds and Interpolation

The unicausal vacuity grounds of Section 2 represent a specific *construction principle* for vacuity grounds using universal quantification, i.e., we have

$$\mathfrak{S} \models \forall x.\varphi(x) \qquad \text{and} \qquad \forall x.\varphi(x) \models \varphi(\psi) \tag{3}$$

with the special case

$$\mathfrak{S} \models \varphi(\mathsf{false}) \qquad \text{and} \qquad \varphi(\mathsf{false}) \models \varphi(\psi) \tag{4}$$

when $\varphi(\psi)$ is unipolar. Since the implication $\forall x.\varphi(x) \models \varphi(\psi)$ is a consequence of the construction, a model checking result $\mathfrak{S} \models \forall x.\varphi(x)$ indeed says that $\forall x.\varphi(x)$ is *one* vacuity ground. Consequently, with the exception of the special case mentioned in Remark 1, unicausal vacuity semantics indeed has a natural embedding into our framework.

The construction principle for unicausal vacuity grounds is itself closely related to interpolation. One of the main logical motivations for the introduction of propositional temporal quantifiers are proofs of Craig interpolation. In particular, uniform Craig interpolation of the μ-calculus was shown by quantifier elimination of bisimulation quantifiers [14]: Given a formula α and β where $\alpha \models \beta$, an interpolant is obtained by the quantified formula $\forall \mathbf{x}.\beta$, where $\mathbf{x}$ is the tuple of variables occurring only in β. By construction, it holds that

$$\alpha \models \forall \mathbf{x}.\beta \qquad \text{and} \qquad \forall \mathbf{x}.\beta \models \beta \tag{5}$$

whence it is sufficient to show that $\forall \mathbf{x}.\beta$ is equivalent to a quantifier-free formula to prove interpolation for the μ-calculus. Since this construction depends only on β and $\mathbf{x}$, $\forall \mathbf{x}.\beta$ is called a post-interpolant or right interpolant. (Existential quantification of α naturally yields pre-interpolants or left interpolants.)

CTL and LTL are well known not to have interpolation [24] because CTL and LTL do not admit elimination of bisimulation quantifiers. The quantified extensions of CTL and LTL, however, do have uniform interpolation, and the post-interpolants are naturally obtained by universal quantification analogously to (5). *Consequently, we conclude that unicausal vacuity grounds are generalizations of post-interpolants: they are the weakest formulas which imply* φ_{Spec} *without mentioning certain variables or subformulas that occur in* φ_{Spec}.

Let us note that our adversarial discussion of unicausal semantics in Section 2 does not inhibit the use of $\forall x.\varphi(x)$ *as* vacuity ground in the sense described here. The discussion of Section 2 only shows that no unicausal semantics by itself can adequately solve the problem of vacuity detection.

3.2 Computation of Vacuity Grounds

The discussion of Section 3.1 shows that the unicausal semantics yield a natural class of vacuity grounds. In the important unipolar case (cf. condition (4)), the vacuity grounds $\varphi(\mathsf{false})$ satisfy all important criteria: they are simpler than the specification, they are easy to model check, and they represent tangible feedback for the verification engineer.

When the quantifier in $\forall x.\varphi(x)$ cannot be eliminated, the situation is more complicated. As discussed in Section 2 (#2), the semantics of quantified temporal formulas has only limited explanatory value concerning vacuity. Moreover, the examples in Table 2 demonstrate that the verification engineer may obtain different vacuity feedback from

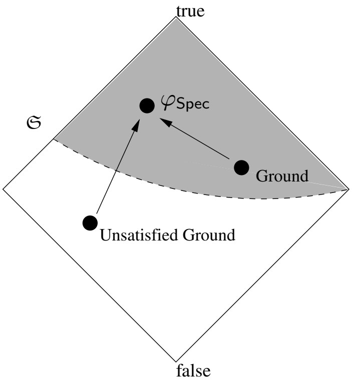


Fig. 3. In the lattice of temporal properties, system $\mathfrak{S}$ partitions the properties into satisfied properties (shaded dark) and unsatisfied ones. Vacuity grounds are located lower in the lattice order than the specification, but inside the shaded area of satisfied properties. The closer a vacuity ground is to the white area, the stronger is the intuitive strength of the vacuity assertion. The figure shows one satisfied vacuity ground and one unsatisfied ground.

different unicausal semantics, i.e., the vacuity ground for one semantics may hold true, while it is false for the other semantics. To appreciate this situation, the engineer has to understand the subtleties of the different semantics.

As argued in Section 2, it is important to obtain vacuity grounds different from the unicausal grounds. The search space for these vacuity grounds is illustrated in Figure 3. Not surprisingly, finding small vacuity grounds has the same complexity as the respective decision problem for validity: Let VAC-CTL be the decision problem if for a given structure $\mathfrak{S}$, a CTL specification φ_{Spec}, and an integer $k < |\varphi_{\mathsf{Spec}}|$, there exists a vacuity ground φ_{Fact} such that $|\varphi_{\mathsf{Fact}}| \leq k$ and $\varphi_{\mathsf{Spec}} \not\equiv \varphi_{\mathsf{Fact}}$. VAC-LTL is defined analogously for LTL. Then the following theorem is not hard to show:

Theorem 1. VAC-CTL *is* EXPTIME-*complete and* VAC-LTL *is* PSPACE-*complete.*

Thus, the complexity is not worse than checking $\varphi_{\mathsf{Fact}} \models \varphi_{\mathsf{Spec}}$. Nevertheless, it is a reasonable strategy to focus on methods which systematically enumerate perceptionally simple candidates for vacuity grounds; these methods may be based both on heuristics and formal considerations. As in the unicausal semantics, candidates φ_{Fact} will typically be chosen in such a way that $\varphi_{\mathsf{Fact}} \models \varphi_{\mathsf{Spec}}$ follows by construction, and $\mathfrak{S} \models \varphi_{\mathsf{Fact}}$ is determined by the model checker, cf. condition (1).

To obtain candidate formulas without expensive validity checks, one can systematically compute the closure of the specification under two syntactic operations, namely by rewriting and strengthening:

(i) Replace subformulas of the specification by equivalent subformulas typically involving disjunction, e.g., $\mathbf{EF}\,p$ by $p \vee \mathbf{EX}\,\mathbf{EF}\,p$, or $\mathbf{AF}\,p$ by $p \vee \mathbf{AX}\,\mathbf{AF}\,p$, etc.[2]

---

[2] Recent work by the authors [28,29,31] has characterized the distributivity over conjunction of temporal operators which yields also an analogous characterizations for disjunction. Such characterizations can be used to support rewriting of specifications.

(ii) Replace subformulas by stronger non-equivalent subformulas, e.g., replace whole subformulas by false as in unicausal semantics, $\mathbf{AF}\, p$ by p as in Pnueli's example, $\mathbf{EF}\, p$ by $\mathbf{AX}\, p$, $\mathbf{AX}\, p \vee \mathbf{AX}\, \neg p$ by $\mathbf{AG}\, p$, etc.

While this heuristic enumeration of antecedents naturally samples the space of possible vacuity grounds for φ_{Spec}, each candidate has to be model checked separately, similar as in unicausal semantics.

An alternative systematic approach for finding antecedents by strengthening subformulas in the unipolar case was presented in a predecessor paper [30] which introduced *parameterized vacuity*, a new approach to vacuity using temporal logic query solving. A temporal logic query solver [9] is a variant of a model checker which on input of a formula $\varphi(x)$ and a model $\mathfrak{S}$ finds the strongest formulas ψ such that $\mathfrak{S} \models \varphi(\psi)$. Such a formula ψ is called a *solution* of $\varphi(x)$ in $\mathfrak{S}$. To use temporal logic queries for computing vacuity grounds, we are interested in solutions ψ such that $\varphi(\psi)$ is a vacuity ground. In this way we are able to reduce vacuity detection to temporal logic query solving [30]. Our approach was motivated by the failure of unicausal semantics to handle Pnueli's problem. Recall that unicausal semantics cannot find vacuity ground $\mathbf{AG}\, p$ for $\mathbf{AF}\, \mathbf{AG}\, p$, cf. Section 2 (#3.1). Instead of reducing $\varphi(\psi)$ to a unicausal ground $\varphi(\mathsf{false})$, we use a temporal logic query solver to find a simple formula θ which implies ψ. Then, by monotonicity, it follows that $\varphi(\theta) \models \varphi(\psi)$. Thus, we obtain a vacuity ground $\varphi(\theta)$ which explains $\varphi(\psi)$ and solves Pnueli's problem. Several algorithms for solving temporal logic queries have been proposed in the literature; in particular, symbolic algorithms [9,28,32], automata-theoretic algorithms [7], and algorithms based on multi-valued model checking [10,20,21].

4 Conclusion

We have argued that vacuity of temporal specifications cannot be adequately captured by formal criteria. As vacuity expresses the inadequacy of a specification, it needs to be addressed by the verification engineer. We have therefore proposed a new approach to vacuity where the model checker itself does not decide on vacuity, but computes an interpolant – called a vacuity ground – which expresses a simple reason that renders the specification true. Candidate formulas for the interpolants can be obtained from existing notions of unicausal vacuity, from heuristics, and from temporal logic query solving. Equipped with the feedback vacuity grounds, the verification engineer can decide if the specification is vacuously satisfied.

The current paper has focused on the logical nature of vacuous satisfaction rather than on practical vacuity detection algorithms. We believe that future work should address the systematic computation of vacuity grounds, because vacuity is in the eye of the beholder.

Acknowledgments. The authors are grateful to Arie Gurfinkel, Kedar Namjoshi, and Richard Zach for discussions on vacuity.

References

1. Armoni, R., Fix, L., Flaisher, A., Grumberg, O., Piterman, N., Tiemeyer, A., Vardi, M.Y.: Enhanced vacuity detection in linear temporal logic. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 368–380. Springer, Heidelberg (2003)
2. Barwise, J., van Benthem, J.: Interpolation, preservation, and pebble games. Journal of Symbolic Logic 64(2), 881–903 (1999)
3. Beatty, D.L., Bryant, R.E.: Formally verifying a microprocessor using a simulation methodology. In: DAC 1994. Proc. 31st Annual ACM IEEE Design Automation Conference, pp. 596–602. ACM Press, New York (1994)
4. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in ACTL formulas. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 279–290. Springer, Heidelberg (1997)
5. Beer, I., Ben-David, S., Eisner, C., Rodeh, Y.: Efficient detection of vacuity in temporal model checking. Formal Methods in System Design (FMSD) 18(2), 141–163 (2001)
6. Ben-David, S., Fisman, D., Ruah, S.: Temporal antecedent failure: Refining vacuity. In: Caires, L., Vasconcelos, V.T. (eds.) CONCUR 2007. LNCS, vol. 4703, pp. 492–506. Springer, Heidelberg (2007)
7. Bruns, G., Godefroid, P.: Temporal logic query checking. In: LICS 2001. Proc. 16th Annual IEEE Symposium on Logic in Computer Science, pp. 409–417. IEEE Computer Society Press, Los Alamitos (2001)
8. Bustan, D., Flaisher, A., Grumberg, O., Kupferman, O., Vardi, M.Y.: Regular vacuity. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 191–206. Springer, Heidelberg (2005)
9. Chan, W.: Temporal-logic queries. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 450–463. Springer, Heidelberg (2000)
10. Chechik, M., Gurfinkel, A.: TLQSolver: A temporal logic query checker. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 210–214. Springer, Heidelberg (2003)
11. Chockler, H., Strichman, O.: Easier and more informative vacuity checks. In: MEMOCODE 2007, pp. 189–198. IEEE Computer Society Press, Los Alamitos (2007)
12. Clarke, E.M., Jha, S., Lu, Y., Veith, H.: Tree-like counterexamples in model checking. In: LICS 2002. Proc. 17th Annual IEEE Symposium on Logic in Computer Science, pp. 19–29. IEEE Computer Society Press, Los Alamitos (2002)
13. Clarke, E.M., Veith, H.: Counterexamples revisited: Principles, algorithms, applications. In: Dershowitz, N. (ed.) Verification: Theory and Practice. LNCS, vol. 2772, pp. 208–224. Springer, Heidelberg (2004)
14. D'Agostino, G., Hollenberg, M.: Logical questions concerning the μ-calculus: Interpolation, Lyndon and Loś-Tarski. Journal of Symbolic Logic 65(1), 310–332 (2000)
15. Dong, Y., Sarna-Starosta, B., Ramakrishnan, C., Smolka, S.A.: Vacuity checking in the modal mu-calculus. In: Kirchner, H., Ringeissen, C. (eds.) AMAST 2002. LNCS, vol. 2422, Springer, Heidelberg (2002)
16. French, T.: Decidability of quantified propositional branching time logics. In: Stumptner, M., Corbett, D.R., Brooks, M. (eds.) AI 2001: Advances in Artificial Intelligence. LNCS (LNAI), vol. 2256, pp. 165–176. Springer, Heidelberg (2001)
17. Groce, A., Kroening, D.: Making the most of BMC counterexamples. Electronic Notes in Theoretical Computer Science (ENTCS) 119(2), 67–81 (2005)
18. Gurfinkel, A., Chechik, M.: Extending extended vacuity. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 306–321. Springer, Heidelberg (2004)
19. Gurfinkel, A., Chechik, M.: How vacuous is vacuous? In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 451–466. Springer, Heidelberg (2004)

20. Gurfinkel, A., Chechik, M., Devereux, B.: Temporal logic query checking: A tool for model exploration. IEEE Transactions on Software Engineering (TSE) 29(10), 898–914 (2003)
21. Gurfinkel, A., Devereux, B., Chechik, M.: Model exploration with temporal logic query checking. In: Proc. 10th International Symposium on the Foundations of Software Engineering (FSE-10), pp. 139–148. ACM Press, New York (2002)
22. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. In: Pierre, L., Kropf, T. (eds.) CHARME 1999. LNCS, vol. 1703, pp. 82–96. Springer, Heidelberg (1999)
23. Kupferman, O., Vardi, M.Y.: Vacuity detection in temporal model checking. International Journal on Software Tools for Technology Transfer (STTT) 4(2), 224–233 (2003)
24. Maksimova, L.: Absence of interpolation and of Beth's property in temporal logics with "the next" operation. Siberian Mathematical Journal 32(6), 109–113 (1991)
25. Namjoshi, K.S.: An efficiently checkable, proof-based formulation of vacuity in model checking. In: Alur, R., Peled, D.A. (eds.) CAV 2004. LNCS, vol. 3114, pp. 57–69. Springer, Heidelberg (2004)
26. Pnueli, A.: 9th International Conference on Computer-Aided Verification. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, Springer, Heidelberg (1997) (cited from [5])
27. Purandare, M., Somenzi, F.: Vacuum cleaning CTL formulae. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 485–499. Springer, Heidelberg (2002)
28. Samer, M.: Reasoning about Specifications in Model Checking. PhD thesis, Vienna University of Technology (2004)
29. Samer, M., Veith, H.: Validity of CTL queries revisited. In: Baaz, M., Makowsky, J.A. (eds.) CSL 2003. LNCS, vol. 2803, pp. 470–483. Springer, Heidelberg (2003)
30. Samer, M., Veith, H.: Parameterized vacuity. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 322–336. Springer, Heidelberg (2004)
31. Samer, M., Veith, H.: A syntactic characterization of distributive LTL queries. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 1099–1110. Springer, Heidelberg (2004)
32. Samer, M., Veith, H.: Deterministic CTL query solving. In: TIME 2005, pp. 156–165. IEEE Computer Society Press, Los Alamitos (2005)
33. Simmonds, J., Davies, J., Gurfinkel, A., Chechik, M.: Exploiting resolution proofs to speed up LTL vacuity detection for BMC. In: FMCAD 2007. Proc. 7th International Conference on Formal Methods in Computer-Aided Design, IEEE Computer Society Press, Los Alamitos (to appear, 2007)
34. Visser, A.: Bisimulations, model descriptions and propositional quantifiers. Logic Group Preprint Series, Nbr. 161, Dept. Philosophy, Utrecht University (1996)
35. Wittgenstein, L.: On Certainty. In: Anscombe, G.E.M., von Wright, G.H. (eds.) Harper and Row (1968)

Whatever Happened to Deductive Question Answering?

Richard Waldinger

Artificial Intelligence Center, SRI International

Deductive question answering, the extraction of answers to questions from machine-discovered proofs, is the poor cousin of program synthesis. It involves much of the same technology—theorem proving and answer extraction—but the bar is lower. Instead of constructing a general program to meet a given specification for any input—the program synthesis problem—we need only construct answers for specific inputs; question answering is a special case of program synthesis. Since the input is known, there is less emphasis on case analysis (to construct conditional programs) and mathematical induction (to construct looping constructs), those bugbears of theorem proving that are central to general program synthesis. Program synthesis as a byproduct of automatic theorem proving has been a largely dormant field in recent years, while those seeking to apply theorem proving have been scurrying to find smaller problems, including question answering.

Deductive question answering had its roots in intuitionistic and constructive logical inference systems, which were motivated by philosophical rather than computational goals. The idea obtained computational force in McCarthy's 1958 Advice Taker, which proposed developing systems that inferred conclusions from declarative assertions in formal logic. The Advice Taker, which was never implemented, anticipated deductive question answering, program synthesis, and planning.

Slagle's Deducom obtained answers from proofs using a machine-oriented inference rule, Robinson's resolution principle; knowledge was encoded in a knowledge base of logical axioms (the [illegible]), the question was treated as a conjecture, a theorem prover attempted to prove that the conjecture followed from the axioms of the theory, and an answer to the question was extracted from the proof. The answer-extraction method was based on keeping track of how existentially quantified variables in the conjecture were instantiated in the course of the proof.

The QA3 program of Green, Yates, and Rafael integrated answer extraction with theorem proving via the [illegible]. Chang, Lee, Manna, and Waldinger introduced improved methods for conditional and looping answer construction. Answer extraction became a standard feature of automated resolution theorem provers, such as McCune's Otter and Stickel's SNARK, and was also the basis for logic programming systems, in which a special-purpose theorem prover served as an interpreter for programs encoded as axioms.

Deductive databases allow question answering from large databases but use logic programming rather than a general theorem prover to perform inference.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 15–16, 2007.
© Springer-Verlag Berlin Heidelberg 2007

The Amphion system (of Lowry et al.), for answering questions posed by NASA planetary astronomers, computed an answer by extracting from a SNARK proof a straight-line program composed of procedures from a subroutine library; because the program contained no conditionals and no loops, it was possible for Amphion to construct programs that were dozens of instructions long, completely automatically. Software composed by Amphion has been used for the planning of photography in the Cassini mission to Saturn.

While traditional question-answering systems stored all their knowledge as axioms in a formal language, this proves impractical when answers depend on large, constantly changing external data sources; a procedural-attachment mechanism allows external data and software sources to be consulted by a theorem prover while the proof is underway. As a consequence, relatively little information needs to be encoded in the subject domain theory; it can be acquired if and when needed. While external sources may not adhere to any standard representational conventions, procedural attachment allows the theorem prover to invoke software sources that translate data in the form produced by one source into that required by another. Procedural attachment is particularly applicable to Semantic Web applications, in which some of the external sources are Web sites, whose capabilities can be advertised by axioms in the subject domain theory.

SRI employed a natural-language front end and a theorem-proving central nervous system (SNARK) equipped with procedural attachment to answer questions posed by an intelligence analyst (QUARK) or an Earth systems scientist (GeoLogica). While non-computer-scientists found the natural language input to be more congenial than logic, it turned out to be difficult to restrict questions to be within the system's domain of expertise.

In this talk we will describe recent efforts (BioDeducta) for deductive question answering in molecular biology, done in collaboration with computational biologist Jeff Shrager. Questions expressed in logical form are treated as conjectures and proved by SNARK from a biological subject-domain theory; access to multiple biological data and software resources is provided by procedural attachment. We illustrate this with the discovery of the gene responsible for light adaptation in cyanobacteria, which are water bacteria capable of photosynthesis.

A proposed query-elicitation mechanism allows a biological researcher to construct a logical query without realizing it, by choosing among larger and larger English-language alternatives for fragments of the question. A projected explanation mechanism constructs from the proof a coherent English explanation and justification for the answer.

While the annotation language OWL has achieved some currency as a representation vehicle for subject domain knowledge, we argue that it and proposed Semantic Web rule languages built around it are inadequate to express queries and reasoning for Semantic Web question answering. Lacking are such central features as full quantification and equality reasoning. It is also impossible to express a closed-world assumption, which states that a particular source provides exhaustive information on a given topic.

Decidable Fragments of Many-Sorted Logic

Aharon Abadi, Alexander Rabinovich, and Mooly Sagiv

School of Computer Science, Tel-Aviv University, Israel
{aharon,rabinoa,msagiv}@post.tau.ac.il

Abstract. We investigate the possibility of developing a decidable logic which allows expressing a large variety of real world specifications. The idea is to define a decidable subset of many-sorted (typed) first- order logic. The motivation is that types simplify the complexity of mixed quantifiers when they quantify over different types. We noticed that many real world verification problems can be formalized by quantifying over different types in such a way that the relations between types remain simple.

Our main result is a decidable fragment of many-sorted first-order logic that captures many real world specifications.

1 Introduction

Systems with unbounded resources such as dynamically allocated objects and threads are heavily used in data structure implementations, web servers, and other areas. This paper develops new methods for proving properties of such systems. Our method is based on two principles: (i) formalizing the system and the required properties in many-sorted first-order logic and (ii) developing mechanisms for proving validity of formulas in that logic over finite models (which is actually harder than validity over arbitrary models).

This paper was inspired by the Alloy Analyzer— a tool for analyzing models written in Alloy, a simple structural modeling language based on first-order logic [10,11]. The Alloy Analyzer is similar to a bounded model checker [8], which means that every reported error is real, but Alloy can miss errors (i.e., produce false positives). Indeed, the Alloy tool performs an under-approximation of the set of reachable states, and is designed for falsifying rather than verifying properties.

Main Results. This paper investigates the applicability of first-order tools to reason about Alloy specifications. It is motivated by our initial experience with employing off-the-shelf resolution-based first-order provers to prove properties of formulas in many-sorted first-order logic.

The main results in this paper are decidable fragments of many-sorted first-order logic. Our methods can generate finite counter-examples and finite models satisfying a given specification which is hard for resolution-based theorem prover. The rest of this subsection elaborates on these results.

Motivation: Employing Ordered Resolution-Based Theorem Provers
Ordered Resolution-Based First-Order Systems such as `SPASS` [16] and `Vampire` [14] have been shown to be quite successful in proving first-order theorems. Ordering dramatically improves the performance of a prover and in some cases can even guarantee decidability.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 17–31, 2007.
© Springer-Verlag Berlin Heidelberg 2007

As a motivating experience reported in [3], we converted Alloy specifications into formulas in first-order logic with transitive closure. We then conservatively modeled transitive closure via sound first-order axioms similar to the ones in [12]. The result is that every theorem proved by SPASS about the Alloy specification is valid over finite models, but the first-order theorem prover may fail due to: (i) timeout in the inference rules, (ii) infinite models which violate the specification, and (iii) models that violate the specifications and the transitive closure requirement.

Encouragingly, SPASS was able to prove 8 out of the 12 Alloy examples tried without any changes or user intervention. Our initial study indicates that in many of the examples SPASS failed due to the use of transitive closure and the fact that SPASS considers infinite models that violate the specifications. It is interesting to note that SPASS was significantly faster than Alloy when the scope exceeded 7 elements in each type.

Adding Types. Motivated by our success with SPASS we investigated the possibility of developing a decidable logic which allows to express many of the Alloy examples. The idea is to define a decidable subset of first-order logic. Since Alloy specifications include different types, natural specifications use many-sorted first-order logic.

The problem of classifying fragments of first-order logic with respect to the decidability and complexity of the satisfiability problem has long been a major topic in the study of classical logic. In [7] the complete classification of fragments with decidable validity problem and fragments with finite model property according to quantifier prefixes and vocabulary is provided. However, this classification deals only with one-sorted logics, and usually does not apply to specifications of practical problems, many of which are many-sorted.

For example, finite model property fails for the formulas with the quantifier prefix $\forall\forall\exists$ and equality. Sorts can reduce the complexity of this prefix class. For example consider the formula: $\forall x, y : A \ \exists z : B \ \psi(x, y, z)$ where ψ is a quantifier-free formula with equality and without functions symbols. Each model M of the formula contains a sub-model M' that satisfies the formula and has only two elements. Indeed, let M be a model of the formula; we can pick two arbitrary elements $a_1 \epsilon A^M, b_1 \epsilon B^M$ such that $M \models \psi(a_1, a_1, b_1)$ and define M' to be M restricted to the universe $\{a_1, b_1\}$. Hence, many-sorted sentences with quantifier prefix $\forall x : A \forall y : A \exists z : B$ have the finite-model property. Usually, like in the above example, the inclusion of sorts simplifies the verification task.

Our Contribution. The main technical contribution of this paper is identification of a fragment of many-sorted logic which is (1) decidable (2) useful — can formalize many of the Alloy examples that do not contain transitive closure and (3) has a finite counter-model property which guarantees that a formula has a counter-model iff it has a finite counter-model (equivalently formula is valid iff it is valid over the finite models).

Our second contribution is an attempt to classify decidable prefix classes of many-sorted logic. We show that a naive extension of one-sorted prefix classes to a many-sorted case inherits neither decidability nor finite model property.

The rest of this paper is organized as follows. In Section 2, we describe three fragments of many-sorted logic and formalize some Alloy examples by formulas in these fragments. In Section 3 we prove that our fragments are decidable for validity over finite models. In Section 4, we investigate ways of generalizing decidable fragments from first-order logic to many-sorted logic.

The reader is referred to [3] for proofs, more examples of formalizing interesting properties using decidable logic, extensions for transitive closure, and a report on our experience with SPASS.

2 Three Fragments of Many-Sorted First-Order Logic

Safety properties of programs/systems can be usually formalized by universal sentences. The task of the verification that a program P satisfies a property θ can be reduced to the validity problem for sentences of the form $\psi \Rightarrow \theta$, where sentence ψ formulate the behavior of P.

In this section we introduce three fragments St_0, St_1 and St_2 of many-sorted logic for description of the behavior of programs and systems. The validity (and validity over the finite models) problems for formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_i$ and θ is universal are decidable. This allows us to prove that a given program/system satisfy a property expressed as a universal formula.

St_0 is a natural fragment of the universal formulas which has the following *finite model property*: if $\psi \in St_0$, then it has a model iff it has a finite model.

St_0 has an even stronger *satisfiability with finite extension* property which we introduce in Section 3. This property implies that the validity problem over finite models for the sentences of the form $\psi \Rightarrow \theta$ where $\psi \in St_0$ and θ is universal, is decidable. In Section 2.3 we formalize birthday book example in St_0.

Motivated by examples from Alloy we introduce in Section 2.1, a more expressive (though less natural) set of formulas St_1. The St_1 formulas also have satisfiability with finite extension property, and therefore might be suitable for automatic verification of safety properties. The behavior of many specifications from [1] can be formalized by St_1. We also describe the Railway safety example which cannot be formalized in St_1. Our attempts to formalize the Railway safety example led us to a fragment St_2 which is defined in Section 2.4. This fragment has the satisfiability with finite extension property. All except one specifications from [1] which do not use transitive closure can be formalized by formulas of the form $\psi \Rightarrow \theta$, where $\psi \in St_2$ and θ are universal.

2.1 St_0 Class

In this subsection we will describe a simple class of formulas denoted as St_0.

Definition 1 (Stratified Vocabulary). *A vocabulary Σ for many-sorted logic is* stratified *if there is a function* [illegible] *from sorts (types) into* [illegible] *such that for every function symbol $f : A_1 \times \ldots \times A_m \to B$*, [illegible]$(B) <$ [illegible](A_i) *for all $i = 1, \ldots, m$.*

It is clear that for a finite stratified vocabulary Σ and a finite set V of variables there are only finitely many terms over Σ with the variables in V.

St_0 Syntax. The formulas in St_0 are universal formulas, over a stratified vocabulary.

It is easy to show that St_0 has the finite model property, due to the finiteness of Herbrand model over St_0 vocabulary. We will extend this class to the class St_1.

2.2 St_1 Class

St_1 is an extension of St_0 with a restricted use of new atomic formula $x \in Im[f]$, where f is a function symbol. The formula $x \in Im[f]$ is a shorthand for $\exists y_1 : A_1 \ldots \exists y_n : A_n \; (x = f(y_1, \ldots, y_n))$.

This is formalized below.

St_1 Vocabulary. Contains predicates, function symbols, equality symbol and atomic formulas $x \in Im[f]$ where f is a function symbol.

St_1 Syntax. The formulas in St_1 are universal formulas, over a stratified vocabulary and for every function $f : A_1 \times \ldots \times A_n \rightarrow B$ that participates in a subformula $x \epsilon Im[f]$ f is the only function with the range B.

The semantics is as in many-sorted logic. For the new atomic formula the semantics is as for the formula $\exists y_1 : A_1 \ldots \exists y_n : A_n \; (x = f(y_1, \ldots, y_n))$.

In section 3 we will prove that St_1 has satisfiability with finite extension property which generalize finite model property.

2.3 Examples

Most of our examples come from Alloy [10,11][1]. The vast majority of Alloy examples include transitive closure, and thus cannot be formalized in our logic. We examined eight Alloy specifications without transitive closure and seven of them fit into our logic. This is illustrated by the birthday book example. The second example is a Railway Safety specification. This example cannot be formalized by formulas in St_1. However, it fits in St_2 which is an extension of St_1, and will be described in Section 2.4.

Birthday Book. Table 1 is used to model a simple Birthday book program[2]. A birthday book has two fields: $known$, a set of names (of persons whose birthdays are known), and $date$, set of triples (birthday book, person, the birthday date of that person). The operation $getDate$ gets the birthday date for a given birthday book and person. The operation $AddBirthday$ adds an association between a name and a date. The assertion $Assert$ checks that if you add an entry and then look it up, you get back what you just entered.

The specification assertion has the form $\psi \Rightarrow \theta$ where $\psi \in St_0$ and θ is universal.

The specification contains only one function $getDate: \; BirthdayBook \times Person \rightarrow Date$. We can define *level* as follows: *level(BirthdayBook)* $= 1$, *level(Person)* $= 1$ and *level(Date)* $= 0$.

Railway Safety Example. A policy for controlling the motion of trains in a railway system is analyzed. Gates are placed on track segments to prevent trains from colliding. We need a criterion to determine when gates should be closed. In [4] a different

---

[1] For details, see [1].

[2] The example originates in [15] and the translation to Alloy is given as an example in the Alloy distribution found at http://alloy.mit.edu

Table 1. Constants, Facts, and Formulas used in the Birthday Book example

Types	$Person, Date, BirthdayBook$
Relations	$known \subseteq BirthdayBook \times Person$ $date \subseteq BirthdayBook \times Person \times Date$
Functions	$getDate : BirthdayBook \times Person \to Date$
constants	$b1, b2 : BirthdayBook$ $d1, d2 : Date$ $p1 : Person$
facts	$\forall b : BirthdayBook\ \forall p : Person\ \forall d : Date\ date(b, p, d) \Rightarrow known(b, p)$ $\forall b : BirthdayBook\ \forall p : Person\ known(b, p) \Rightarrow date(b, p, getDate(b, p))$ $\forall b' : BirthdayBook\ \forall p' : Person\ \forall d', d'' : Date$ $\quad date(b', p', d') \wedge date(b', p', d'') \Rightarrow d' = d''$
Formulas	AddBirthday : $(bb, bb' : BirthdayBook, p : Person, d : Date)$ $\neg known(bb, p) \wedge \forall p' : Person\ \forall d' : Date\ date(bb', p', d') \Leftrightarrow$ $(p' = p \wedge d' = d) \vee date(bb, p', d')$
Assert	$Facts \wedge AddBirthday(b1, b2, p1, d1) \wedge date(b2, p1, d2) \Rightarrow d1 = d2$

Railroad crossing problem is formalized, where the time is treated as continuous time, while we use a discrete time. The Alloy formalization in [1,2] differs slightly from our formalization, however both of them represent the same specification. In our formalization the type *Movers* and the relation *moving* were added to represent sets of moving trains. In addition some of the relations and the functions have suffix *_current* or *_next* to represent interpretation at the current and the next period. For example instead of $P(t) \Rightarrow P(t+1)$ we write *P_current* $\Rightarrow$ *P_next*. Here $P(t) \Rightarrow P(t+1)$ means that if P holds at time t then P holds at time $t+1$ and *P_current* $\Rightarrow$ *P_next* means that if P holds at *current* time then P holds at *next* time.

Formulas Description

- $safe\_current$ and $safe\_next$ operations express that for any pair of distinct trains t_1 and t_2, the segment occupied by t_1 does not overlap with the segment occupied by t_2.
- $moveOk$ describes in which gate conditions it is legal for a set of trains to move.
- $trainMove$ is a physical constraint: a driver may not choose to cross from one segment into another segment which is not connected to it. The constraint has two parts. The first ensures that every train that moves ends up in the $next$ time on a segment that is a successor of the segment it was in the previous $current$ time. The second ensures that the trains that do not move stay on the same segments.
- $GatePolicy$ describes the safety mechanism, enforced as a policy on a gate state. It comprises two constraints. The first is concerned with trains and gates: it ensures that the segments that are predecessors of those segments that are occupied by trains should have closed gates. In other words, a gate should be down when there is a train ahead. This is an unnecessarily stringent policy, since it does

Table 2. Types, relations, functions, constants and facts used in the train example

Types	$\mathit{Train}, \mathit{Segment}, \mathit{GateState}, \mathit{Movers}$
Relations	$\mathit{next} \subseteq \mathit{Segment} \times \mathit{Segment}$ $\mathit{Overlaps} \subseteq \mathit{Segment} \times \mathit{Segment}$ $\mathit{on\_current} \subseteq \mathit{Train} \times \mathit{Segment}$ $\mathit{on\_next} \subseteq \mathit{Train} \times \mathit{Segment}$ $\mathit{Occupied\_current} \subseteq \mathit{Segment}$ $\mathit{Occupied\_next} \subseteq \mathit{Segment}$ $\mathit{moving} \subseteq \mathit{Movers} \times \mathit{Train}$ $\mathit{closed} \subseteq \mathit{GateState} \times \mathit{Segment}$
Functions	$\mathit{getSegment\_current} : \mathit{Train} \rightarrow \mathit{Segment}$ $\mathit{getSegment\_next} : \mathit{Train} \rightarrow \mathit{Segment}$
Constants	$g : \mathit{GateState}\ m : \mathit{Movers}$
Facts	–At any moment every train is on some segment $\forall t : \mathit{Train}\ \ \mathit{on\_current}(t, \mathit{getSegment\_current}(t))$ $\forall t : \mathit{Train}\ \ \mathit{on\_next}(t, \mathit{getSegment\_next}(t))$ –At any moment train is at most on one segment $\forall t : \mathit{Train}\ \ \forall s_1, s_2 : \mathit{Segment}$ $(\mathit{on\_current}(t, s_1) \wedge \mathit{on\_current}(t, s_2)) \Rightarrow s_1 = s_2$ $\forall t : \mathit{Train}\ \ \forall s_1, s_2 : \mathit{Segment}$ $(\mathit{on\_next}(t, s_1) \wedge on\_next(t, s_2)) \Rightarrow s_1 = s_2$ –Occupied gives the set of segments occupied by trains $\forall s : \mathit{Segment}\ \ \mathit{Occupied\_current}(s) \Rightarrow s \in \mathit{Im}[\mathit{getSegment\_current}]$ $\forall s : \mathit{Segment}\ \ \mathit{Occupied\_next}(s) \Rightarrow s \in \mathit{Im}[\mathit{getSegment\_next}]$ $\forall t : \mathit{Train}\ \ \forall s : \mathit{Segment}\ \ \mathit{on\_current}(t, s) \Rightarrow \mathit{Occupied\_current}(s)$ $\forall t : \mathit{Train}\ \ \forall s : \mathit{Segment}\ \ \mathit{on\_next}(t, s) \Rightarrow \mathit{Occupied\_next}(s)$ –Overlaps is symmetric and reflexive $\forall s_1, s_2 : \mathit{Segment}\ \ \mathit{Overlaps}(s_1, s_2) \Leftrightarrow \mathit{Overlaps}(s_2, s_1)$ $\forall s : \mathit{Segment}\ \ \mathit{Overlaps}(s, s)$

not permit a train to move to any successor of a segment when one successor is occupied. The second constraint is concerned with gates alone: it ensures that between any pair of segments that have an overlapping successor, at most one gate can not be closed.

The *Assert* implies that if a move is permitted according to the rules of *MoveOK*, and if the trains move according to the physical constraints of *TrainMove*, and if the safety mechanism described by *GatePolicy* is enforced, then a transition from a safe state will result in a state that is also safe. In other words, safety is preserved.

Tables 2 and 3 contains the specification of the train example.

The specification contains functions $getSegment\_current : Train \rightarrow Segment$ and $getSegment\_next : Train \rightarrow Segment$ where $getSegment\_current$ participates in formula $x \epsilon Im[getSegment\_current]$ in contrast to our requirements from [illegible]$_1$ formulas.

Table 3. Formulas and assert used in the train example

Formulas	safe_current : $\forall t_1, t_2 : \mathit{Train}\ \ \forall s_1, s_2 : \mathit{Segment}$ $(t_1 \neq t_2 \wedge \mathit{on\_current}(t_1, s_1) \wedge \mathit{on\_current}(t_2, s_2)) \Rightarrow$ $\neg \mathit{Overlaps}(s_1, s_2)$ safe_next: $\forall t_1, t_1 : \mathit{Train}\ \ \forall s_1, s_2 : \mathit{Segment}$ $(t_1 \neq t_2 \wedge \mathit{on\_next}(t_1, s_1) \wedge \mathit{on\_next}(t_2, s_2)) \Rightarrow$ $\neg Overlaps(s_1, s_2)$ $\mathit{moveOk}(g : \mathit{GateState}\ , m : \mathit{Movers})$: $\forall s : \mathit{Segment}\ \ \forall t : \mathit{Train}$ $(moving(m, t) \wedge \mathit{on\_current}(t, s)) \Rightarrow$ $\neg \mathit{closed}(g, s)$ $\mathit{trainMove}(m : \mathit{Movers})$ $\forall\ \ t : \mathit{Train} \forall s_1, s_2 : \mathit{Segment}$ $(moving(m, t) \wedge \mathit{on\_next}(t, s_2) \wedge on\_current(t, s_1)) \Rightarrow$ $\mathit{next}(s_1, s_2)$ $\wedge$ $\forall t : \mathit{Train}\ \ \forall s : \mathit{Segment}$ $\neg \mathit{moving}(m, t) \Rightarrow (\mathit{on\_next}(t, s) \Leftrightarrow \mathit{on\_current}(t, s))$ $\mathit{gatePolicy}(g : \mathit{GateState})$ $\forall s_1, s_2, s_3 : \mathit{Segment}$ $next(s_1, s_2) \wedge \mathit{Occupied\_current}(s_3) \wedge overlaps(s_2, s_3)) \Rightarrow$ $closed(g, s_1)$ $\wedge$ $\forall s_1, s_2, s_3, s_4 : \mathit{Segment}$ $(s_1 \neq s_2 \wedge next(s_1, s_3) \wedge next(s_2, s_4) \wedge overlaps(s_3, s_4)) \Rightarrow$ $(closed(g, s_1) \vee closed(g, s_2))$
Assert	$(\mathit{Facts} \wedge \mathit{safe\_current} \wedge \mathit{moveOk(g,m)} \wedge \mathit{trainMove(m)} \wedge \mathit{GatePolicy}(g)) \Rightarrow$ $\mathit{safe\_next}$

2.4 St_2 Class

St_2 Vocabulary. Contains predicates, function symbols, equality symbol and atomic formulas $x \in Im[f]$ where f is a function symbol.

St_2 Syntax. The formulas in St_2 are universal formulas over a stratified vocabulary, and for every function $f : A_1 \times \ldots \times A_k \rightarrow B$ that participates in a subformula $x \epsilon Im[f]$ the following condition holds:

For every function symbol $g : \bar{A}_1 \times \ldots \times \bar{A}_{\bar{k}} \rightarrow B$:

(*) $\forall a_1 : A_1, \ldots, \forall a_k : A_k\ \forall \bar{a}_1 : \bar{A}_1, \ldots, \forall \bar{a}_{\bar{k}} : \bar{A}_{\bar{k}}\ f(a_1, \ldots, a_k) = g(\bar{a}_2, \ldots, \bar{a}_{\bar{k}}) \Rightarrow$
$k = \bar{k} \wedge a_1 = \bar{a}_2 \wedge \ldots \wedge a_k = \bar{a}_{\bar{k}}$.

Notice that (*) is a semantical requirement. When we say that a Str_2 formula ψ is "satisfiable", we mean that it is satisfiable in a structure which fulfills this semantical requirement (*) .

In many cases formalized by us the requirement (*) above immediately follows from the intended interpretation of functions. In the railway safety example some work needs to be done to derive this requirement from the specification.

First we can notice that the specification contains functions $getSegment\_current : Train \rightarrow Segment$ and $getSegment\_next : Train \rightarrow Segment$. We can define *level* as follows: *level(Train)* $= 1,$ *level(Segment)* $= 0,$ *level(GateState)* $= 0$ and *level(Movers)* $= 0$.

It remains to prove that the semantic requirement holds. In the Train specification there are *getSegment_current*, *getSegment_next* functions such that $x \in Im[$*getSegment_current*$]$

participates in the formula. Let *M* be model such that $M \models$ *Assert_Train*. It suffices to show that $\forall t_1, t_2$: *Train* $(t_1 \neq t_2) \Rightarrow$ *getSegment_current*$(t_1) \neq$ *getSegment_next*(t_2). Let $t_1 \neq t_2$ and suppose that *getSegment_current*$(t_1) = s$. From the Train $Facts$ immediately follows that *Occupied_current*(s). Hence from *gatePolicy* follows that all previous *Segments* of s have a closed gate. Thus according to *moveOk* no train comes to s at next time. But $M \models$ *safe_current* so $s \neq$ *getSegment_current*(t_2). From this and from the fact that no train comes to s at next time follows that $s \neq$ *getSegment_next*(t_2).

3 Decidability of Validity Problem

Let $\mathcal{F}_1$ and $\mathcal{F}_2$ be sets of formulas. We denote by $\mathcal{F}_1 \Rightarrow \mathcal{F}_2$ the set $\{\psi \Rightarrow \varphi : \psi \in \mathcal{F}_1$ and $\varphi \in \mathcal{F}_2\}$. The set of universal sentences will be denoted by UN. The main results of this section is stated in the following theorem.

Theorem 2. *The validity problem for* [illegible]$_2 \Rightarrow UN$ *is decidable.*

We also prove that every sentence in [illegible]$_2 \Rightarrow UN$ is valid iff it holds over the class of finite models.

The section is organized as follows. First, we introduce basic definitions. Next, following Beauquier and Slissenko in [5,6] we provide sufficient semantical conditions for decidability of validity problem. Unfortunately, these semantical conditions are undecidable. However, we show that the formulas in [illegible]$_2 \Rightarrow UN$ satisfy these semantical conditions.

3.1 Basic Definitions

Definition 3 (Partial Model). *Let L be a many-sorted first-order language. A partial Model M' of L consists of the following ingredients:*

- *For every sort s a non-empty set D'_s, called the domain of M'.*
- *For every predicate symbol p^i_s of L with argument types $s_1, \ldots, s_n$ an assignment of an n-place relation $(p^i_s)^{M'}$ in $D'_{s_1}, \ldots, D'_{s_n}$.*
- *For every function symbol f^i_s of L with type $f^i_s : s_1 \times s_2 \times \ldots s_n \rightarrow s$ an assignment of a partial n-place operation $(f^i_s)^{M'}$ in $D'_{s_1} \times \ldots \times D'_{s_n} \rightarrow D'_s$.*
- *For every individual constant c^i_s of L an assignment of an element $(c^i_s)^{M'}$ of D'_s.*

We say that a partial model is finite if every D'_s is finite.

A partial model M' is a model if every function $(f_s^i)^{M'} : D'_{s_1} \times \ldots \times D'_{s_n} \to D'_s$ is total.

The following definition strengthens the notion of finite model property.

Definition 4 (Satisfiability with Finite Extension). *A formula ψ is satisfiable with a finite extension iff for every finite partial model M': if M' can be extended to a model M of ψ, then M' can be extended to a finite model $\bar{M}$ of ψ.*

The satisfiability with finite extension definition was inspired by (but is quite different from) the definition of *C-satisfiable* with augmentation for complexity (k, n) in [5,6].

Definition 5 (k-Refutability). *A formula ψ is k-refutable iff for every counter-model M of ψ there exists a finite partial model M' such that:*

- *For every sort s : $|D'_s| \leq k$*
- *M is an extension of M'*
- *any extension of M' to a model is a counter-model of ψ.*

We say that a formula is finitely refutable if it is k-refutable for some k$\in Nat$.

Example 6 (k-Refutability). Recall the formula *safe_current* of Railway Safety system:

safe_current :

$$\begin{aligned}&\forall t_1, t_1 : \textit{Train}\ \forall s_1, s_2 : \textit{Segment}\\ &\quad(t_1 \neq t_2 \wedge \textit{on\_current}(t_1, s_1) \wedge \textit{on\_current}(t_2, s_2))\\ &\quad\Rightarrow \neg \textit{Overlaps}(s_1, s_2)\end{aligned}$$

The constraint ensures that at *current* moment for any pair of distinct trains t_1 and t_2, the segment that t_1 occupies is not a member of the set of segments that overlap with the segment t_2 occupies. Let us show that *safe_current* is 2-refutable. Suppose that *safe_current* has a counter model M then there are: $t_1, t_2 : \textit{Train}^M$, $s_1, s_2 : \textit{Segment}^M$ such that $M \models \neg(\textit{on\_current}(t_1, s_2) \wedge \textit{on\_current}(t_2, s_2) \wedge t_1 \neq t_2 \Rightarrow \neg \textit{Overlaps}(s_1, s_2))$. Take M' sub model of M with the domains $\textit{Train}^{M'} = \{t_1, t_2\}$, $\textit{Segment}^{M'} = \{s_1, s_2\}$. For any extension of M' to model $\bar{M}$ it still holds that $\bar{M} \models \neg(\textit{on\_current}(t_1, s_2) \wedge \textit{on\_current}(t_2, s_2) \wedge t_1 \neq t_2 \Rightarrow \neg \textit{Overlaps}(s_1, s_2))$, so $\bar{M}$ is a counter model of *safe_current*.

From the above example we can learn that if M is a counter-model for a k-refutable formula, then M contains k elements in the domain that cause a contradiction. If we take the partial model obtained by the restriction of M to these elements, then any extension of it still contains these elements and therefore still is a counter-model.

In the rest of this section we will prove the decidability of formulas of the form $\theta \Rightarrow \vartheta$ where θ is *satisfiable with finite extension* and ϑ is *k-refutable* for some k. In addition we will prove that:

- Every formula in [illegible]$_2$ is *satisfiable with finite extension*.
- A formula is equivalent to a formula from UN iff the formula is *k-refutable* for some k.

This will complete the proof of decidability of formulas of the form [illegible]$_2 \to UN$.

3.2 Sufficient Semantical Conditions for Decidability

The next lemma is a consequence of the definitions 4 and 5.

Lemma 7 (Finite Counter-Model Property). *Let ψ be a formula of the form $\theta \Rightarrow \varphi$, where θ is satisfiable with finite extension and φ is finitely refutable. Then $\neg\psi$ has the finite model property.*

Notice that the lemma does not give a bound to the size of the model.

Theorem 8 (Sufficient Conditions for Decidability). *Let $\mathcal{F}_{fin-ref}$ be a set of sentences in many-sorted first-order logic which are finitely refutable and let $\mathcal{F}_{sat-fin-ext}$ be a set of sentences in many-sorted first-order logic which are satisfiable with finite extension. Then the validity problem for $\mathcal{F}_{sat-fin-ext} \Rightarrow \mathcal{F}_{fin-ref}$ is decidable. Moreover, if $\psi \in \mathcal{F}_{sat-fin-ext} \Rightarrow \mathcal{F}_{fin-ref}$, then ψ is valid iff it is valid over the finite models.*

Proof: The validity problem for many-sorted first-order logic is recursively enumerable. By lemma 7 if a sentence in this class is not valid then it has a finite counter-model. Hence, in order to check whether a sentence φ in this class is valid we can start (1) to enumerate proofs looking for a proof of φ and (2) to enumerate all finite models looking for a counter-model for φ. Either (1) or (2) will succeed. If (1) succeeds then φ is valid, if (2) succeeds φ it is not valid. □

Since lemma 7 does not provide a bound of the size of the model, we cannot provide a concrete complexity bound on the algorithm in theorem 8.

Theorem 8 provides semantical conditions on a class of formulas which ensure decidability of validity problem for this class. Unfortunately, these semantical conditions are undecidable.

Theorem 9. *The following semantical properties of sentences are undecidable:*

1. **Input:** *A formula ψ.*
   Question *: Is ψ finitely refutable ?*
2. *For every $k \in Nat$:*
   Input: *A formula ψ.*
   Question: *is ψ k-refutable?*
3. **Input:** *A formula ψ.*
   Question: *Is ψ* satisfiable with finite extension*?*

In the next two subsections we describe syntactical conditions which ensure

(1) finitely refutable property.
(2) satisfiability with finite extension.

3.3 Syntactical Conditions for Decidability

The proof of the following lemma uses the preservation theorem from first-order logic, which says that a sentence ψ is equivalent to universal formula iff any submodel of a model of ψ is a model of ψ. The preservation theorem is valid also for the many-sorted first-order logics.

Lemma 10 (Syntactical Conditions for Finite Refutability). *A formula ψ is k-refutable for some k iff ψ is equivalent to a universal formula.*

Usually safety properties are easily formalized by universal formulas. Hence, the class $\mathcal{F}_{sat-fin-ext} \Rightarrow UN$ is appropriate for verification of safety properties and has decidable validity problem.

The next theorem is our main technical theorem.

Theorem 11. *Let ψ be a formula in* [illegible]$_2$ *then ψ is satisfiable with finite extension.*

Proof (Sketch)
Assume that a formula $\psi \in$ [illegible]$_2$ is satisfiable in M and that M' is a finite partial sub-model of M.

First, we extend M' to a finite partial sub-model M'' of M such that $Im[f]$ has a "correct" interpretation. Assume that the level of types in Σ are in the set $\{0, \ldots, m\}$.

Let $M_0 = M'$ and for $i = 0, \ldots, m$ we define D_i N_i and M_{i+1} as follows. Let D_i be the set of elements in M_i of the types at level i such that $b \in D_i$ iff $M \models b \in Im[f]$, however, there is no tuple $\bar{a} \in M_i$ with $M \models f(\bar{a}) = b$.

Now for every $b \in D_i$ choose $\bar{a} \in M$ such that $M \models f(\bar{a}) = b$. Observe that each element in $\bar{a}$ has type at level $> i$. Let N_i be the set of all chosen elements (for all elements in D_i and all function symbols in Σ). Let M_{i+1} be the partial sub-model of M over $Dom(M_i) \cup N_i$.

It is not difficult to show that M_{i+1} is a finite partial submodel of M and for every $b \in Dom(M_i)$ if B is the type of b and the level of B is at most i, then there is $\bar{a} \in M_{i+1}$ such that $M \models f(\bar{a}) = b$ iff there is $\bar{a'} \in M$ such that $M \models f(\bar{a'}) = b$.

In particular, for every $b \in Dom(M_{m+1})$ if $M \models b \in Im[f]$, then there is a tuple $\bar{a} \in Dom(M_{m+1})$ such that $M \models f(\bar{a}) = b$.

Next, let M'' be defined as M_{m+1} and let Ass be the set of assignments to the variables with values in $Dom(M'')$ and let $\bar{D}$ be the set of values (in M) of all terms over Σ under these assignments. The set $\bar{D}$ is finite, because our vocabulary is stratified and M'' is finite. Let $\bar{M}$ be the partial submodel of M over the domain $\bar{D}$. From the definition of $\bar{M}$ follows that $\bar{M}$ is a submodel of M. Moreover, it is not difficult to show using the semantic requirement (*) , that the interpretations of $Im[f]$ in M and in $\bar{M}$ agree, i.e. for every $b \in Dom(\bar{M})$, $M \models b \in Im[f]$ iff $\bar{M} \models b \in Im[f]$. $\square$

Finally, Theorem 2 is an immediate consequence of Theorem 8, Lemma 10 and Theorem 11.

4 Some Fragments of Many-Sorted Logic

In the previous section we introduced decidable fragments of many-sorted logic. In this section, we consider classes from first-order logic which have the finite-model property. We try to find a way to extend these classes to many-sorted logic.

We use the notation from [7]. According to [7] the following classes have the finite model property:

- $[\exists^*\forall^*, all]_=$ (Ramsey 1930) the class of all sentences with quantifier prefix $\exists^*\forall^*$ over arbitrary relational vocabulary with equality.

- $[\exists^*\forall\exists^*, all]_=$ (Ackermann 1928) the class of all sentences with quantifier prefix $\exists^*\forall\exists^*$ over arbitrary relational vocabulary with equality.
- $[\exists^*, all, all]_=$ (Gurevich 1976) the class of all sentences with quantifier prefix $\exists^*$ over arbitrary vocabulary with equality.
- $[\exists^*\forall, all, (1)]_=$ (Grädel 1996) the class of all sentences with quantifier prefix $\exists^*\forall$ over vocabulary that contain unary function and arbitrary predicate symbols with equality.
- FO^2 (Mortimer 1975) [13] the class of all sentences of relational vocabulary that contain two variables and equality.

Below we describe a generic natural way to generalize a class of first-order formulas to many-sorted logic. Unfortunately, finite model property and decidability are not preserved under this generalization.

Let $Q_1 \dots Q_m$ be a quantifier prefix in many-sorted logic. Its projection on a type A is obtained by erasing all quantifiers over the variables of types distinct from A. One can hope that if for every type A the projection of the quantifier prefix on A is in a decidable class of one sorted logic, then this prefix is in a decidable class of many-sorted logic. However, we show that neither decidability nor finite model property for a prefix of many-sorted logic is inherited from the corresponding properties of projections.

When we take a projection of a formula to a type, in addition to removing the quantifiers over other types we should also modify the quantifier free part of the formula. Here is a definition:

Definition 12 (Projection of a Formula Onto Type A). *Let ψ be a formula of many-sorted logic in the prenex normal form. Its projection on type A is denoted by $\bar{\psi}^A$ and is obtained as follows:*

1. *For each type T different from A:*

   (a) Eliminate all quantifiers of type T.

   (b) Replace every term of type T by constant C^T.

2. *Let $R(t_1, \dots t_k)$ be an atomic sub-formula which contains new constants C^{T_j} ($1 \leqslant j \leqslant m$) at positions $i_1, i_2, \dots i_m$.*
   Introduce a new predicate name $P_{i_1,i_2,\dots,i_m}$ with an arity $k - m$
   and replace $R(t_1, \dots t_k)$
   by $P_{i_1,i_2,\dots,i_m}(t_1, \dots t_{i_1-1}, t_{i_1+1}, \dots t_{i_2-1}, t_{i_2+1} \dots t_{i_m-1}, t_{i_m+1} \dots t_k)$.

3. *Let $f(t_1, \dots t_k)$ be a term which contains new constants C^{T_j} ($1 \leqslant j \leqslant m$) at positions $i_1, i_2, \dots i_m$.*
   Introduce a new function name $f_{i_1,i_2,\dots,i_m}$ with an arity $k - m$
   and replace $f(t_1, \dots t_k)$
   by $f_{i_1,i_2,\dots,i_m}(t_1, \dots t_{i_1-1}, t_{i_1+1}, \dots t_{i_2-1}, t_{i_2+1} \dots t_{i_m-1}, t_{i_m+1} \dots t_k)$.

For a formula ψ its projection on A is the formula $\bar{\psi}^A$ with one type; hence it can be considered as the first-order logic formula.

Definition 13 (Naive Extension)
A set of many-sorted first-order formulas D^{ext} is a naive extension of a set of first-order formulas D if for every $\psi \in D^{ext}$ and for every type A holds that $\bar{\psi}^A \in D$.

Examples

1. Let ψ be $\forall x_1 : A\, \forall x_2 : B\, \exists y_1 : A\, \forall y_2 : B\; p(x_1, y_1, x_2) \vee q(y_1, y_2)$.
   Let us look at its projections on A and B. After first two steps we obtain the formulas $\forall x_1 : A\;\; \exists y_1 : A\, p(x_1, y_1, c^B) \vee q(y_1, c^B)$ and $\forall x_2 : B\, \forall y_2 : B\, p(c^A, c^A, x_2) \vee q(c^A, y_2)$. After replacing predicates we obtain : $\forall x_1 : A\;\; \exists y_1 : A\; p_3(x_1, y_1) \vee q_2(y_1)$ and $\forall x_2 : B\, \forall y_2 : B\; p_{1,2}(x_2) \vee q_1(y_2)$. Both formulas are in FO^2. Hence, ψ is in FO^2_{ext}.
2. Let ψ be $\forall x_1 : A\, \forall x_2 : B\, \exists y_1 : A\, \exists y_2 : B\; p(x_1, y_1, x_2) \vee p(x_1, x_1, y_2) \vee q(y_1, x_1)$. Its projections on A and B are $\forall x_1 : A\, \exists y_1 : A\; p_3(x_1, y_1) \vee p_3(x_1, x_1) \vee q(y_1, x_1)$ and $\forall x_2 : A\; \exists y_2 : A\; p_{1,2}(x_2) \vee p_{1,2}(y_2) \vee q_{12}$. Since, the projections are in Ackermann class, ψ is in the extension of Ackermann class.

Note that the extension of the Ramsey class to many-sorted logic is a fragment of [illegible]$_0$ and has the finite model property and thus is decidable. It is easy to prove that the naive extension of Gurevich class is decidable. The next two theorems state that the naive extensions of Ackermann, Grädel and Mortimer classes do not have the finite model property and, even more disappointing, are undecidable.

Theorem 14 (Finite Model Property Fails). *Each of the following fragments has a formula which is satisfiable only in infinite structures: $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$ and $[\exists^*\forall\exists^*, all]^{ext}_{=}$.*

Proof: see [3].

Theorem 15 (Undecidability). *The satisfiability problem is undecidable for each of the following fragments: $[\exists^*\forall, all, (1)]^{ext}_{=}$, $[FO^2]^{ext}$ and $[\exists^*\forall\exists^*, all]^{ext}_{=}$.*

Our proof of the above theorem (see [3]) provides formalization of two register machine and is similar to the proofs in [7].

It is well known that $[\forall\ \forall\ \exists]_=$ and $[\forall\ \exists\forall]_=$ are undecidable classes for one-sorted first-order logic (see [9]). The following theorem says that for many-sorted first-order logic the only undecidable three quantifier prefix classes are these two one-sorted.

Although we did not found any practical use for this result we think that it has some theoretical interest.

Theorem 16. *The satisfiability problem is decidable for sentences of the form $Q_1 Q_2 Q_3 \psi$, where ψ is a quantifier free many-sorted formula with equality without functions symbols and $Q_1 Q_2 Q_3$ is a quantifier prefix not of the form $[\forall x_1 : A \forall x_2 : A \exists x_3 : A]$ or $[\forall x_1 : A \exists x_2 : A \forall x_3 : A]$ for some sort A.*

Proof: see [3] .

5 Conclusion

In this paper we initiated a systematic study of fragments of many-sorted logic, which are decidable/have the finite model property and have a potential for practical use. To our knowledge, the idea of looking at this problem in a systematic way has not been explored previously (despite the well-known complete classification in the one-sorted case, presented in the book by Boerger, Graedel and Gurevich [7]).

We presented a number of decidable fragments of many-sorted first-order logic. The first one, [illegible]$_0$, is based on a stratified vocabulary. The stratification property guarantees that only a finite number of terms can be built with a given finite set of variables. As a result, the Herbrand universe is finite and the small model property holds. Moreover, a stronger property of satisfiability with finite extension holds.

Subsequently, we extended the class [illegible]$_0$ to class [illegible]$_1$ and then to [illegible]$_2$, and proved that these classes also have the satisfiability with finite extension property (and therefore, the finite model property). The added expressive power in [illegible]$_2$ is the ability to test whether an element is in the image of a function. Even though this particular extension may seem less natural from a syntactic viewpoint, it is very useful in many formalizations.

We provided semantical sufficient conditions for decidability. As a consequence, we obtained that for the sentences of the form $\psi \Rightarrow \varphi$, where $\psi \in$ [illegible]$_2$ and φ is universal, the validity problem is decidable. In order to illustrate the usefulness of the fragment, we formalized in it many examples from [1] - the Alloy finite model finder.

Finally, we looked at classes corresponding to decidable classes (or classes with the finite-model property) of first-order logic. We observed that just requiring the decidability of projections of the quantifier prefix onto each type individually is not a sufficient condition for the decidability (respectively, the finite-model property) in general. Future work is needed in order to carry out a complete classification for the many-sorted logic.

In [3] we extended our results to a logic which allows a restricted use of the transitive closure. We succeeded in formalizing some of Alloy specifications by formulas of this logic; however, the vast majority of Alloy examples that contain the transitive closure are not covered by this fragment. Future work is needed to evaluate its usefulness and to find its decidable extensions.

Acknowledgments

We are grateful to the anonymous referees for their suggestions and remarks. We also thank Tal Lev-Ami and Greta Yorsh for their comments.

References

1. The alloy analyzer home page: `http://alloy.mit.edu`
2. `http://alloy.mit.edu/case-studies.php`
3. Abadi, A.: Decidable fragments of many-sorted logic. Master's thesis, Tel-Aviv University (2007)
4. Beauquier, D., Slissenko, A.: Verification of timed algorithms: Gurevich abstract state machines versus first order timed logic. In: Proc. of ASM 2000 International Workshop (March 2000)

5. Beauquier, D., Slissenko, A.: Decidable verification for reducible timed automata specified in a first order logic with time. Theoretical Computer Science 275, 347–388 (2002)
6. Beauquier, D., Slissenko, A.: A first order logic for specification of timed algorithms: Basic properties and a decidable class. Annals of Pure and Applied Logic 113, 13–52 (2002)
7. Borger, E., Gradel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Heidelberg (1997)
8. Clarke, E.M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design 19(1), 7–34 (2001)
9. Goldfarb, W.D.: The unsolvability of the godel class with identity. The Journal of Symbolic Logic 49(4), 1237–1252 (1984)
10. Jackson, D.: Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol. 11(2), 256–290 (2002)
11. Jackson, D.: Micromodels of software:lightweight modelling and analysis with alloy. Technical report, MIT Lab for Computer Science (2002)
12. Lev-Ami, T., Immerman, N., Reps, T.W., Sagiv, M., Srivastava, S., Yorsh, G.: Simulating reachability using first-order logic with applications to verification of linked data structures. In: CADE, pp. 99–115 (2005)
13. Mortimer, M.: On languages with two variables. Zeitschr. f. math. Logik u. Grundlagen d. Math., 135–140 (1975)
14. Riazanov, A., Voronkov, A.: The design and implementation of vampire. AI Communications 15(2-3), 91–110 (2002)
15. Spivey, J.M.: The Z notation: a reference manual. Prentice-Hall, Englewood Cliffs (1992)
16. Weidenbach, C., Gaede, B., Rock, G.: Spass & flotter version 0.42. In: CADE-13. Proceedings of the 13th International Conference on Automated Deduction, pp. 141–145. Springer, Heidelberg (1996)

One-Pass Tableaux for Computation Tree Logic

Pietro Abate[1], Rajeev Goré[1], and Florian Widmann[1,2]

[1] The Australian National University
Canberra ACT 0200, Australia
[2] Logic and Computation Programme
Canberra Research Laboratory
NICTA Australia
{Pietro.Abate,Rajeev.Gore,Florian.Widmann}@anu.edu.au

Abstract. We give the first single-pass ("on the fly") tableau decision procedure for computational tree logic (CTL). Our method extends Schwendimann's single-pass decision procedure for propositional linear temporal logic (PLTL) but the extension is non-trivial because of the interactions between the branching inherent in CTL-models, which is missing in PLTL-models, and the "or" branching inherent in tableau search. Our method extends to many other fix-point logics like propositional dynamic logic (PDL) and the logic of common knowledge (LCK).

The decision problem for CTL is known to be EXPTIME-complete, but our procedure requires 2EXPTIME in the worst case. A similar phenomenon occurs in extremely efficient practical single-pass tableau algorithms for very expressive description logics with EXPTIME-complete decision problems because the 2EXPTIME worst-case behaviour rarely arises. Our method is amenable to the numerous optimisation methods invented for these description logics and has been implemented in the Tableau Work Bench (`twb.rsise.anu.edu.au`) without these extensive optimisations. Its one-pass nature also makes it amenable to parallel proof-search on multiple processors.

1 Introduction and Motivation

Propositional fix-point logics like propositional linear temporal logic (PLTL), computation tree logic (CTL) and full computation tree logic (CTL*) are useful for digital circuit verification [10] and reasoning about programs [18]. The usual route is to use model-checking to ensure that a given model satisfies certain desirable properties since this can be done in time linear in the size of the model. Model-checking cannot answer the general question of whether every model for a given set of formulae Γ satisfies a certain property φ: that is model-checking cannot be used to perform automated deduction in the chosen fix-point logic. The decision problems for fix-point logics are typically at least PSPACE-complete (PLTL), and are often EXPTIME-complete (CTL) or even 2EXPTIME-complete (CTL*). These logics are all fragments of monadic second order logic whose decision problem has non-elementary complexity when restricted to (infinite) tree-models, meaning that its complexity is a tower of

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 32–46, 2007.
© Springer-Verlag Berlin Heidelberg 2007

exponentials of a height determined by the size of the initial formula. Consequently, automated theorem provers tailored for specific fix-point logics are of importance in computer science.

The main methods for automating deduction in logics like PLTL and CTL are optimal tableau-based methods [24,6], optimal automata-based methods [22] and optimal resolution-based methods [7,4]. The inverse method has also been applied to simpler modal logics like K [23], but it is possible to view this approach as an automata-based method [2]. We are trying to obtain further details of a new method for PLTL which appears to avoid an explicit loop-check [8].

Most existing automated theorem provers for the PSPACE-complete fix-point logic PLTL are tableau-based [16,9,17,20], but resolution provers for PLTL have also been developed recently [13]. It is easy to construct examples where the (goal-directed) tableau-based provers out-perform the resolution provers, and vice versa [13], so both methods remain of interest. Theorem provers based on the always optimal automata-based methods which do not actually build the required automata [21] are still in their infancy because good optimisation techniques have not been developed to date.

For CTL, however, we know of no efficient implemented automated theorem provers, even though tableau-based [6,19], resolution-based [4] and automata-based [22] deduction methods for CTL are also known.

The simplest non-technical explanation is that proof-search in many modal logics requires some form of “loop check” to guarantee termination, but fix-point logics require a further test to distinguish a “good loop” that represents a path in a model from a “bad loop” that represents an infinite branch with no hope of ever giving a model. The harder the decision problem, the greater the difficulty of separating good loops from bad loops.

Most tableau-based methods for fix-point logics solve this problem using a two-pass procedure [24,5,6]. The first pass applies the tableau rules to construct a finite rooted cyclic graph. The second pass prunes nodes that are unsatisfiable because they contain contradictions like $\{p, \neg p\}$, and also remove nodes which give rise to “bad loops”. The main practical disadvantage of such two-pass methods is that the cyclic graph built in the first pass has a size which is [illegible] exponential in the size of the initial formula. So the very act of building this graph immediately causes EXPTIME behaviour even in the average case.

One-pass tableau methods avoid this bottle-neck by building a rooted cyclic [illegible] (where all cyclic edges loop back to ancestors) one branch at a time, using backtracking. The experience from one-pass tableaux for very expressive description logics [12] of similar worst-case complexity shows that their average case behaviour is often much better since the given formulae may not contain the full complexity inherent in the decision problem, particularly if the formula arises from real-world applications. Of course, there is no free lunch, since in the worst case, these one-pass methods may have significantly worse behaviour than the known optimal behaviour: 2EXPTIME than EXPTIME in the case of CTL for example. Moreover, the method for separating “good loops” from “bad loops” becomes significantly more complicated since it cannot utilise the global view offered by a graph built during a previous pass. Ideally, we want to evaluate each branch on its own during construction, or during backtracking, using only information which is "local" to this branch since this allows us to explore these branches in parallel using multiple processors.

Implemented one-pass [17,20] and two-pass [16] tableau provers already exist for PLTL. A comparison between them [13] shows that the median running time for Janssen's highly optimised two-pass prover for PLTL is greater than the median running time for Schwendimann's not-so-optimised one-pass prover for PLTL [20] [illegible], indicating that the two-pass prover spends most of its time in the first pass building the cyclic graph. There is also a one-pass "tableau" method for propositional dynamic logic (PDL) [3] which constructs a rooted cyclic tree and uses a finite collection of automata, pre-computed from the initial formula, to distinguish "good loops" from "bad loops", but the expressive powers of PDL and CTL are orthogonal: each can express properties which cannot be expressed in the other. But we know of no one-pass tableau method for CTL or its extensions.

For many applications, the ability to exhibit a (counter) model for a formula φ is just as important as the ability to decide whether φ is a theorem. In digital circuit verification, for example, if the circuit does not obey a desired property expressed by a formula φ, then it is vital to exhibit a (CTL) counter-model which falsifies φ so that the circuit can be modified.

Finally, the current Gentzen-style proof-theory of fix-point logics [14,15] requires either infinitary rules, or worst-case finitary branching rules, or "cyclic proofs" with sequents built from formula occurrences or "focussed formulae".

We present a one-pass tableau method for automating deduction in CTL which has the following properties:

Ease of implementation: although our tableau rules are cumbersome to describe and difficult to prove sound and complete, they are extremely easy to implement since they build a rooted cyclic tree as usual, and the only new operations they require are set intersection, set membership, and the operations of min/max on integers;

Ease of optimisation: our method can be optimised using techniques which have proved successful for (one-pass) tableaux for description logics [11];

Ease of generating counter-models and proofs: the soundness proof of our systematic tableau procedure for testing CTL-satisfiability immediately gives an effective procedure for turning an "open" tableau into a CTL-model;

Ease of generating proofs: our tableau calculus can be trivially turned into a cut-free Gentzen-style calculus with "cyclic proofs" where sequents are built from sets of formulae rather than multisets of formula [illegible] or occurrences of "focused formulae". Moreover, unlike existing cut-free Gentzen-style calculi for fix-point logics [14,15] we can give an optimal, rather than worst-case, bound for the finitary version of the omega rule for CTL;

Potential for parallelisation: our current rules build the branches independently, but combine their results during backtracking, so it is possible to implement our procedure on a bank of parallel processors.

Working Prototype: we have implemented a (non-parallelised) prototype of our basic tableau method using the Tableau Work Bench (`twb.rsise.anu.edu.au`) which allows users to test arbitrary CTL formulae over the web.

Our work is one step toward the holy grail of an efficient tableau-based automated theorem prover for the full computational tree logic CTL* [19].

2 Syntax, Semantics and Hintikka Structures

Definition 1. [illegible] AP [illegible] $\{p_0, p_1, p_2, \dots\}$ [illegible] Fml [illegible] CTL [illegible]

- [illegible] AP $\subset$ Fml
- [illegible] φ [illegible] Fml [illegible] $\neg\varphi$ $EX\varphi$ [illegible] $AX\varphi$
- [illegible] φ [illegible] ψ [illegible] Fml [illegible] $\varphi \wedge \psi$ $\varphi \vee \psi$ $E(\varphi U \psi)$ $A(\varphi U \psi)$ $A(\varphi B \psi)$ [illegible] $E(\varphi B \psi)$

[illegible] $EX\varphi$ $AX\varphi$ $E(\varphi U \psi)$ [illegible] $A(\varphi U \psi)$ [illegible] EX AX EU [illegible] AU [illegible] FmlEU [illegible] FmlAU [illegible] EU [illegible] AU [illegible]

Implication, equivalence, and $\top$ are not part of the core language but can be defined as $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$, and $\top := p_0 \vee \neg p_0$.

Definition 2. [illegible] transition frame [illegible] (W, R) [illegible] W [illegible] R [illegible] W i.e. $\forall w \in W. \exists v \in W. w R v$

Definition 3. [illegible] (W, R) [illegible] transition sequence σ [illegible] (W, R) [illegible] $\sigma_0, \sigma_1, \sigma_2, \dots$ [illegible] W [illegible] $\sigma_i R \sigma_{i+1}$ [illegible] $i \in \mathbb{N}$ [illegible] $w \in W$ [illegible] w-sequence σ [illegible] (W, R) [illegible] (W, R) [illegible] $\sigma_0 = w$ [illegible] $w \in W$ [illegible] $\mathcal{B}(w)$ [illegible] w [illegible] (W, R) [illegible] (W, R) [illegible]

Definition 4. [illegible] model $M = (W, R, L)$ [illegible] (W, R) [illegible] $L : W \rightarrow 2^{\mathrm{AP}}$ [illegible] $w \in W$ [illegible] $L(w)$ [illegible] w

Definition 5. [illegible] $M = (W, R, L)$ [illegible] satisfaction relation $\Vdash$ [illegible]

$M, w \Vdash p$ iff $p \in L(w)$, [illegible] $p \in \mathrm{AP}$
$M, w \Vdash \neg\psi$ iff $M, w \nVdash \psi$
$M, w \Vdash \varphi \wedge \psi$ iff $M, w \Vdash \varphi \;\&\; M, w \Vdash \psi$
$M, w \Vdash \varphi \vee \psi$ iff $M, w \Vdash \varphi$ [illegible] $M, w \Vdash \psi$
$M, w \Vdash EX\varphi$ iff $\exists v \in W. w R v \;\&\; M, v \Vdash \varphi$
$M, w \Vdash AX\varphi$ iff $\forall v \in W. w R v \;\Rightarrow\; M, v \Vdash \varphi$
$M, w \Vdash E(\varphi U \psi)$ iff $\exists \sigma \in \mathcal{B}(w). \exists i \in \mathbb{N}. [M, \sigma_i \Vdash \psi \;\&\; \forall j < i. M, \sigma_j \Vdash \varphi]$
$M, w \Vdash A(\varphi U \psi)$ iff $\forall \sigma \in \mathcal{B}(w). \exists i \in \mathbb{N}. [M, \sigma_i \Vdash \psi \;\&\; \forall j < i. M, \sigma_j \Vdash \varphi]$
$M, w \Vdash E(\varphi B \psi)$ iff $\exists \sigma \in \mathcal{B}(w). \forall i \in \mathbb{N}. [M, \sigma_i \Vdash \psi \;\Rightarrow\; \exists j < i. M, \sigma_j \Vdash \varphi]$
$M, w \Vdash A(\varphi B \psi)$ iff $\forall \sigma \in \mathcal{B}(w). \forall i \in \mathbb{N}. [M, \sigma_i \Vdash \psi \;\Rightarrow\; \exists j < i. M, \sigma_j \Vdash \varphi]$.

Definition 6. [illegible] $\varphi \in \mathrm{Fml}$ [illegible] satisfiable [illegible] $M = (W, R, L)$ [illegible] $w \in W$ [illegible] $M, w \Vdash \varphi$. [illegible] $\varphi \in \mathrm{Fml}$ [illegible] valid [illegible] $\neg\varphi$ [illegible]

Definition 7. [illegible] $\varphi \in \mathrm{Fml}$ [illegible] negation normal form [illegible] $\neg$ [illegible] $\varphi \in \mathrm{Fml}$ [illegible] $\mathrm{nnf}(\varphi)$ [illegible] e.g. [illegible] $\varphi \leftrightarrow \mathrm{nnf}(\varphi)$ [illegible] $\sim\varphi := \mathrm{nnf}(\neg\varphi)$

Note that $E(\varphi B \psi) \leftrightarrow \neg A(\neg\varphi U \psi)$ and $A(\varphi B \psi) \leftrightarrow \neg E(\neg\varphi U \psi)$ are valid.

Table 1. Smullyan's $\alpha-$ and $\beta-$notation to classify formulae

α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$E(\varphi B \psi)$	$\sim\psi$	$\varphi \vee EXE(\varphi B \psi)$
$A(\varphi B \psi)$	$\sim\psi$	$\varphi \vee AXA(\varphi B \psi)$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$E(\varphi U \psi)$	ψ	$\varphi \wedge EXE(\varphi U \psi)$
$A(\varphi U \psi)$	ψ	$\varphi \wedge AXA(\varphi U \psi)$

Proposition 8. [illegible] $\alpha \leftrightarrow \alpha_1 \wedge \alpha_2$ [illegible] $\beta \leftrightarrow \beta_1 \vee \beta_2$ [illegible]

Note that some of these equivalences require the fact that the binary relation of every model is total.

Definition 9. [illegible] $\phi \in \mathrm{Fml}$ [illegible] closure $\mathrm{cl}(\phi)$ [illegible] ϕ [illegible]

[illegible] ϕ [illegible] ϕ [illegible] $\mathrm{cl}(\phi)$.
[illegible] $E(\varphi B \psi)$ [illegible] $\mathrm{cl}(\phi)$ [illegible] $EXE(\varphi B \psi)$ [illegible] $\varphi \vee EXE(\varphi B \psi)$.
[illegible] $A(\varphi B \psi)$ [illegible] $\mathrm{cl}(\phi)$ [illegible] $AXA(\varphi B \psi)$ [illegible] $\varphi \vee AXA(\varphi B \psi)$.
[illegible] $E(\varphi U \psi)$ [illegible] $\mathrm{cl}(\phi)$ [illegible] $EXE(\varphi U \psi)$ [illegible] $\varphi \wedge EXE(\varphi U \psi)$.
[illegible] $A(\varphi U \psi)$ [illegible] $\mathrm{cl}(\phi)$ [illegible] $AXA(\varphi U \psi)$ [illegible] $\varphi \wedge AXA(\varphi U \psi)$

[illegible] extended closure $\mathrm{ecl}(\phi)$ [illegible] ϕ [illegible] $\mathrm{ecl}(\phi) := \mathrm{cl}(\phi) \cup \{\sim\varphi : \varphi \in \mathrm{cl}(\phi)\}$

Definition 10. [illegible] structure (W, R, L) [[illegible] $\varphi \in \mathrm{Fml}$] [illegible] (W, R) [illegible] $L : W \to 2^{\mathrm{Fml}}$ [illegible] $w \in W$ [illegible] $L(w)$ [illegible] [[illegible] $\varphi \in L(v)$ [illegible] $v \in W$]

Definition 11. [illegible] pre-Hintikka structure $H = (W, R, L)$ [[illegible] $\varphi \in \mathrm{Fml}$] [illegible] [[illegible] φ] [illegible] $w \in W$ [illegible] α [illegible] β [illegible]

$$
\begin{aligned}
&\mathrm{H1} : \neg p \in L(w) \ \ (p \in \mathrm{AP}) \ \Rightarrow \ p \notin L(w);\\
&\mathrm{H2} : \alpha \in L(w) \ \Rightarrow \ \alpha_1 \in L(w) \ \& \ \alpha_2 \in L(w);\\
&\mathrm{H3} : \beta \in L(w) \ \Rightarrow \ \beta_1 \in L(w) \ \text{[illegible]} \ \beta_2 \in L(w);\\
&\mathrm{H4} : EX\varphi \in L(w) \ \Rightarrow \ \exists v \in W.\, w\,R\,v \ \& \ \varphi \in L(v);\\
&\mathrm{H5} : AX\varphi \in L(w) \ \Rightarrow \ \forall v \in W.\, w\,R\,v \ \Rightarrow \ \varphi \in L(v).
\end{aligned}
$$

[illegible] Hintikka structure $H = (W, R, L)$ [[illegible] $\varphi \in \text{Fml}$] [illegible] [[illegible] φ] [illegible]

$$\begin{aligned}
&\text{H6} : E(\varphi\, U\, \psi) \in L(w) \ \Rightarrow\ \exists\sigma \in \mathcal{B}(w).\, \exists i \in \mathbb{N}.\, [\psi \in L(\sigma_i)\ \&\ \forall j < i.\, \varphi \in L(\sigma_j)];\\
&\text{H7} : A(\varphi\, U\, \psi) \in L(w) \ \Rightarrow\ \forall\sigma \in \mathcal{B}(w).\, \exists i \in \mathbb{N}.\, \psi \in L(\sigma_i).
\end{aligned}$$

Although $H3$ captures the fix-point semantics of $E(\varphi\, U\, \psi)$ and $A(\varphi\, U\, \psi)$ by "locally unwinding" the fix-point, it does not guarantee a [illegible] fix-point which requires that ψ has to be true eventually. We therefore additionally need $H6$ and $H7$ which act "globally". Note that $H2$ is enough to capture the correct behaviour of $E(\varphi\, B\, \psi)$ and $A(\varphi\, B\, \psi)$ as they have a [illegible] fix-point semantics.

Proposition 12. [illegible] $\varphi \in \text{Fml}$ [illegible] ff [illegible] φ

3 A One-Pass Tableau Algorithm for *CTL*

A [illegible] is a systematic search for a model of a formulae ϕ. Its data structures are (upside-down) single-rooted finite trees – called [illegible] – where each node is labelled with a set of formulae that is derived from the formula set of its parent according to some given rules (unless it is the root, of course). The algorithm starts with a single node that is labelled with the singleton set $\{\phi\}$ and incrementally expands the tableau by applying the rules mentioned before to its leaves. The result of the tableau algorithm is a tableau where no more rules can be applied. Such tableaux are called [illegible]. On any branch of the tableau, a node t is an [illegible] of a node s iff t lies above s on the unique path from the root down to s.

An expanded tableau can be associated with a pre-Hintikka structure H for ϕ, and ϕ is satisfiable if and only if H is a Hintikka structure for ϕ. To be able to determine whether H is a Hintikka structure, the algorithm stores additional information with each node of the tableau using [illegible] and [illegible] [20]. A history is a mechanism for collecting extra information during proof search and passing it from parents to children. A variable is a mechanism to propagate information from children to parents.

In the following, we restrict ourselves to the tableau algorithm for CTL.

Definition 13. [illegible] tableau node x [illegible] $(\Gamma :: \text{HCr} :: \text{mrk}, \text{uev})$ [illegible]

Γ [illegible]
HCr [illegible] x
mrk [illegible]
uev [illegible] $\mathbb{N}_{>0}$

The list HCr is the only history since its value in a node is determined by the parent node, whereas mrk and uev are variables since their values in a node are determined by the children. In the following we call tableau nodes just nodes when the meaning is clear.

Informally, the value of mrk at node x is **true** if x is "closed". Since repeated nodes cause "cycles" or "loops", a node that is not "closed" is not necessarily "open" as in traditional tableaux. That is, although we have enough information to detect that further expansion of the node will cause an infinite branch, we may not yet have enough information to determine the status of the node. Informally, if a node x lies on such a "loop" in the tableau, and an "eventuality" EU- or AU-formula φ appears on this loop but remains unfulfilled, then uev of x is defined for φ by setting $\mathrm{uev}(\varphi) = n$, where n is the height of the highest ancestor of x which is part of the loop.

We postpone the definition of a rule for a moment and proceed with the definition of a tableau.

Definition 14. [illegible] tableau [illegible] $\Gamma \subseteq \mathrm{Fml}$ [illegible] HCr [illegible] $(\Gamma :: \mathrm{HCr} :: \mathrm{mrk}, \mathrm{uev})$ [illegible] x [illegible] x i.e. [illegible] expanded [illegible]

Note that mrk and uev in the definition are not given but are part of the result as they are determined by the children of the root.

Definition 15. [illegible] $\mathrm{uev}_\perp : \mathrm{Fml} \rightharpoonup \mathbb{N}_{>0}$ [illegible] i.e. $\mathrm{uev}_\perp(\psi) = \perp$ [illegible] $\psi \in \mathrm{Fml}$

[illegible] In the following, we use Λ to denote a set containing only propositional variables or their negations ([illegible] $\varphi \in \Lambda \Rightarrow \exists p \in \mathrm{AP}. \varphi = p$ or $\varphi = \neg p$). To focus on the "important" parts of the rule, we use "$\cdots$" for the "unimportant" parts which are passed from node to node unchanged ([illegible] $(\Gamma :: \cdots :: \cdots)$).

3.1 The Rules

[illegible]

$$(id) \quad \frac{(\Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{} \quad \{p, \neg p\} \subseteq \Gamma \text{ for some } p \in \mathrm{AP}$$

with mrk := **true** and uev := $\mathrm{uev}_\perp$. The intuition is that the node is "closed" so we pass this information up to the parent by putting mrk to **true**, and putting uev as undefined for all formulae.

[illegible] α [illegible]

$$(\wedge) \quad \frac{(\varphi \wedge \psi \,;\, \Gamma :: \cdots :: \cdots)}{(\varphi \,;\, \psi \,;\, \Gamma :: \cdots :: \cdots)} \qquad (D) \quad \frac{(AX\Delta \,;\, \Lambda :: \cdots :: \cdots)}{(EX(p_0 \vee \neg p_0) \,;\, AX\Delta \,;\, \Lambda :: \cdots :: \cdots)}$$

$$(EB) \quad \frac{(E(\varphi\, B\, \psi) \,;\, \Gamma :: \cdots :: \cdots)}{(\sim\psi \,;\, \varphi \vee EXE(\varphi\, B\, \psi) \,;\, \Gamma :: \cdots :: \cdots)}$$

$$(AB) \quad \frac{(A(\varphi\, B\, \psi) \,;\, \Gamma :: \cdots :: \cdots)}{(\sim\psi \,;\, \varphi \vee AXA(\varphi\, B\, \psi) \,;\, \Gamma :: \cdots :: \cdots)}$$

The $\wedge$-rule is standard and the D-rule captures the fact that the binary relation of a model is total by ensuring that every potential dead-end contains at least one EX-formula. The EB- and AB-rules capture the fix-point nature of the corresponding formulae according to Prop. 8. These rules do not modify the histories or variables at all.

[illegible] β [illegible]

$$(\vee)\ \frac{(\varphi \vee \psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\varphi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

$$(EU)\ \frac{(E(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\varphi\ ;\ EXE(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

$$(AU)\ \frac{(A(\varphi\, U\, \psi))\ ;\ \Gamma :: \cdots :: \mathrm{mrk}, \mathrm{uev})}{(\psi\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_1, \mathrm{uev}_1) \mid (\varphi\ ;\ AXA(\varphi\, U\, \psi)\ ;\ \Gamma :: \cdots :: \mathrm{mrk}_2, \mathrm{uev}_2)}$$

with:

$$\mathrm{mrk} := \mathrm{mrk}_1 \ \&\ \mathrm{mrk}_2$$

$$\mathrm{excl}_\phi(f)(\chi) := \begin{array}{ll} \bot & \text{if } \chi = \phi \\ f(\chi) & \text{otherwise} \end{array}$$

$$\mathrm{uev}'_1 := \begin{array}{ll} \mathrm{uev}_1 & \text{for the } \vee\text{-rule} \\ \mathrm{excl}_{E(\varphi\, U\, \psi)}(\mathrm{uev}_1) & \text{for the } EU\text{-rule} \\ \mathrm{excl}_{A(\varphi\, U\, \psi)}(\mathrm{uev}_1) & \text{for the } AU\text{-rule} \end{array}$$

$$\min\nolimits_\bot(f, g)(\chi) := \begin{array}{ll} \bot & \text{if } f(\chi) = \bot \text{ or } g(\chi) = \bot \\ \min(f(\chi), g(\chi)) & \text{otherwise} \end{array}$$

$$\mathrm{uev} := \begin{array}{ll} \mathrm{uev}_\bot & \text{if } \mathrm{mrk}_1 \ \&\ \mathrm{mrk}_2 \\ \mathrm{uev}'_1 & \text{if } \mathrm{mrk}_2 \ \&\ \text{not } \mathrm{mrk}_1 \\ \mathrm{uev}_2 & \text{if } \mathrm{mrk}_1 \ \&\ \text{not } \mathrm{mrk}_2 \\ \min\nolimits_\bot(\mathrm{uev}'_1, \mathrm{uev}_2) & \text{otherwise} \end{array}$$

The $\vee$-rule is standard except for the computation of uev. The EU- and AU-rules capture the fix-point nature of the EU- and AU-formulae, respectively, according to Prop. 8. The intuitions of the definitions of the histories and variables are:

mrk: the value of the variable mrk is **true** if the node is "closed", so the definition of mrk just captures the "universal" nature of these rules whereby the parent node is closed if both children are closed.

excl: the definition of $\mathrm{excl}_\phi(f)(\psi)$ just ensures that $\mathrm{excl}_\phi(f)(\phi)$ is undefined.

uev'_1: the definition of uev'_1 ensures that its value is undefined for the principal formulae of the EU- and AU-rules.

$\min_\bot$: the definition of $\min_\bot$ ensures that we take the minimum of $f(\chi)$ and $g(\chi)$ only when both functions are defined for χ.

uev: if both children are "closed" then the parent is also closed via mrk so we ensure that uev is undefined in this case. If only the right child is closed, we take uev_1', which is just uev_1 modified to ensure that it is undefined for the principal *EU*- or *AU*-formula. Similarly if only the left child is closed. Finally, if both children are unmarked, we define uev for all formulae that are defined in the uev of both children but map them to the minimum of their values in the children, and undefine the value for the principal formula.

[illegible]

$$(EX)\ \frac{\begin{array}{l} EX\varphi_1\,;\ldots;EX\varphi_n\ ;\ EX\varphi_{n+1}\,;\ldots;EX\varphi_{n+m}\ ;\ AX\Delta\ ;\ \Lambda \\ ::\mathrm{HCr}::\mathrm{mrk},\mathrm{uev}\end{array}}{\begin{array}{l}\varphi_1\ ;\ \Delta \\ ::\mathrm{HCr}_1::\mathrm{mrk}_1,\mathrm{uev}_1\end{array}\ \Big|\ \ldots\ \Big|\ \begin{array}{l}\varphi_n\ ;\ \Delta \\ ::\mathrm{HCr}_n::\mathrm{mrk}_n,\mathrm{uev}_n\end{array}}$$

where:

(1) $\{p, \neg p\} \not\subseteq \Lambda$
(2) $n + m \geq 1$
(3) $\forall i \in \{1, \ldots, n\}.\ \forall j \in \{1, \ldots, \mathrm{len}(\mathrm{HCr})\}.\ \{\varphi_i\} \cup \Delta \neq \mathrm{HCr}[j]$
(4) $\forall k \in \{n+1, \ldots, n+m\}.\ \exists j \in \{1, \ldots, \mathrm{len}(\mathrm{HCr})\}.\ \{\varphi_k\} \cup \Delta = \mathrm{HCr}[j]$

with:

$$\mathrm{HCr}_i := \mathrm{HCr}\ @\ [\{\varphi_i\} \cup \Delta] \text{ for } i = 1, \ldots, n$$

$$\begin{array}{rl}\mathrm{mrk} := & \mathop{}_{i=1}^{n} \mathrm{mrk}_i \text{ or} \\ & \exists i \in \{1, \ldots, n\}.\ \exists \psi \in \{\varphi_i\} \cup \Delta.\ \bot \neq \mathrm{uev}_i(\psi) > \mathrm{len}(\mathrm{HCr})\end{array}$$

$$\begin{array}{rl}\mathrm{uev}_k(\cdot) := & j \in \{1, \ldots, len(\mathrm{HCr})\} \text{ such that } \{\varphi_k\} \cup \Delta = \mathrm{HCr}[j] \\ & \text{for } k = n+1, \ldots, n+m\end{array}$$

$$\mathrm{uev}(\psi) := \begin{array}{ll} \mathrm{uev}_j(\psi) & \text{if } \psi \in \mathrm{FmlEU}\ \&\ \psi = \varphi_j \quad (j \in \{1, \ldots, n+m\}) \\ l & \text{if } \psi \in \mathrm{FmlAU} \cap \Delta\ \& \\ & \quad l = \max\{\mathrm{uev}_j(\psi) \neq \bot \mid j \in \{1, \ldots, n+m\}\} \\ \bot & \text{otherwise} \end{array}$$

$$(\text{where } \max(\emptyset) := \bot)$$

Some intuitions are in order:

(1) The *EX*-rule is applicable if the parent node contains no α- or β-formulae and Λ, which contains propositional variables and their negations only, contains no contradictions.
(2) Both n and m can be zero, but not together.
(3) If $n > 0$, then each $EX\varphi_i$ for $1 \leq i \leq n$ is not "blocked" by an ancestor, and has a child containing $\varphi_i; \Delta$, thereby generating the required *EX*-successor;
(4) If $m > 0$, then each $EX\varphi_k$ for $n+1 \leq k \leq n+m$ is "blocked" from creating its child $\varphi_k; \Delta$ because some ancestor does the job;

HCr_i: is just the HCr of the parent but with an extra entry to extend the "history" of nodes on the path from the root down to the ith child.

mrk: captures the "existential" nature of this rule whereby the parent is marked if some child is closed or if some child contains a formula whose uev is defined and "loops" lower than the parent. Moreover, if n is zero, then mrk is set to **false** to indicate that this branch is not "closed".

uev$_k$: for $n+1 \leq k \leq n+m$ the kth child is blocked by a proxy child higher in the branch. For every such k we set uev$_k$ to be the [illegible] function which maps every formula to the level of this proxy child. Note that this is just a temporary function used to define uev as explained next.

uev(ψ): for an EU-formula $\psi = E(\psi_1 \, U \, \psi_2)$ such that there is a principal formula $EX\varphi_i$ with $\varphi_i = \psi$, we take uev of ψ from the child if $EX\psi$ is "unblocked", or set it to be the [illegible] of the proxy child higher in the branch if it is "blocked". For an AU-formula $\psi = A(\psi_1 \, U \, \psi_2) \in \Delta$, we put uev to be the maximum of the defined values from the real children and the levels of the proxy children. For all other formulae, we put uev to be undefined. The intuition is that a defined uev(ψ) tells us that there is a "loop" which starts at the parent and eventually "loops" up to some blocking node higher up on the current branch. The actual value of uev(ψ) tells us the level of the proxy because we cannot distinguish whether this "loop" is "good" or "bad" until we backtrack up to that level.

Note that the EX-rule and the *id*-rule are mutually exclusive since their side-conditions cannot be simultaneously true.

3.2 Fullpaths, Virtual Successors and Termination of Proof Search

Definition 17. [illegible] $G = (W, R)$ [illegible] e.g. [illegible] R [illegible] [full]path π [illegible] G [illegible] [[illegible]] [illegible] $x_0, x_1, x_2, \ldots$ [illegible] W [illegible] $x_i \, R \, x_{i+1}$ [illegible] x_i [illegible] π [illegible] $x \in W$ [illegible] x-[full]path π [illegible] G [illegible] [[illegible]] [illegible] G [illegible] $x_0 = x$

Definition 18. [illegible] $x = (\varGamma :: \text{HCr} :: \text{mrk}, \text{uev})$ [illegible] φ [illegible] $\varDelta$ [illegible] $\varphi \in x$ $[\varDelta \subseteq x]$ [illegible] $\varphi \in \varGamma$ $[\varDelta \subseteq \varGamma]$ [illegible] HCr mrk [illegible] uev [illegible] x [illegible] HCr$_x$ mrk$_x$ [illegible] uev$_x$ [illegible] x [illegible] marked [illegible] mrk$_x$ [illegible] **true**

Definition 19. [illegible] x [illegible] EX [illegible] T i.e. [illegible] EX [illegible] x [illegible] x [illegible] state [illegible] x [illegible] pre-states [illegible] EX [illegible] EX [illegible] $EX\varphi_i \in x$ [illegible] blocked [illegible] $n+1 \leq i \leq n+m$ [illegible] $EX\varphi_i \in x$ [illegible] y [illegible] T [illegible] x [illegible] $\{\varphi_i\} \cup \varDelta$ [illegible] y [illegible] y [illegible] virtual successor [illegible] $EX\varphi_i$ [illegible] $EX\varphi_i$ [illegible] x [illegible] successor [illegible] $EX\varphi_i$ [illegible] i^{th} [illegible] EX [illegible]

Note that a state is just another term for an EX-node, whereas a pre-state can be any type of node (it may even be a state).

Proposition 20 (Termination). [illegible] $\phi \in \mathrm{Fml}$ [illegible] T [illegible] $(\{\phi\} :: \cdots :: \cdots)$ [illegible]

[illegible] It is obvious that T is a tree. Although it is not as trivial as it might seem to be at first sight, it is not too hard to show that every node in T can contain only formulae of the extended closure $\mathrm{ecl}(\phi \wedge EX(p_0 \vee \neg p_0))$. It is obvious that $\mathrm{ecl}(\phi \wedge EX(p_0 \vee \neg p_0))$ is finite. Hence, there are only a finite number of different sets that can be assigned to the nodes, in particular the pre-states. As the EX-rule guarantees that all pre-states on a path possess different sets of formulae, there can only be a finite number of pre-states on a path. Furthermore, from any pre-state, there are only a finite number of consecutive nodes on a path until we reach a state. As every state in a path is followed by a pre-state and there are only a finite number of pre-states, all paths in T must be finite. This, the obvious fact that every node in T has finite degree, and König's lemma complete the proof. □

3.3 Soundness and Completeness

Let $\phi \in \mathrm{Fml}$ be a formula in negation normal form and T an expanded tableau with root $r = (\{\phi\} :: [] :: \mathrm{mrk}, \mathrm{uev})$: that is, the initial formula set is $\{\phi\}$ and the initial HCr is the empty list.

Theorem 21. [illegible] r [illegible] ϕ

Theorem 22. [illegible] $x = (\Gamma :: \cdots :: \cdots)$ [illegible] T [illegible] $_{\varphi \in \Gamma} \varphi$ [illegible] r [illegible] ϕ [illegible]

Detailed proofs can be found in the extended version of this paper (`users.rsise.anu.edu.au/~florian`).

3.4 A Fully Worked Example

As an example, consider the formula $E(p_1 \, U \, p_2) \wedge \neg E(\top \, U \, p_2)$ which is obviously not satisfiable. Converting the formula into negation normal form gives us $E(p_1 \, U \, p_2) \wedge A(\bot \, B \, p_2)$. Hence, any expanded tableau with root $E(p_1 \, U \, p_2) \wedge A(\bot \, B \, p_2)$ should be marked.

Figure 1 and Fig. 2 show such a tableau where the root node is node (1) in Fig. 1 and where Fig. 2 shows the sub-tableau rooted at node (5). Each node is classified as a ρ-node if rule ρ is applied to that node in the tableau. The unlabelled edges go from states to pre-states. Dotted frames indicate that the sub-tableaux at these nodes are not shown because they are very similar to sub-tableaux of other nodes: that is node (6a) behaves the same way as node (3a). Dots "$\cdots$" indicate that the corresponding values are not important because they are not needed to calculate the value of any other history or variable. The partial function UEV maps the formula $E(p_1 \, U \, p_2)$ to 1 and is undefined otherwise as explained below. The history HCR is defined as $HCR := [\{E(p_1 \, U \, p_2), A(\bot \, B \, p_2)\}]$.

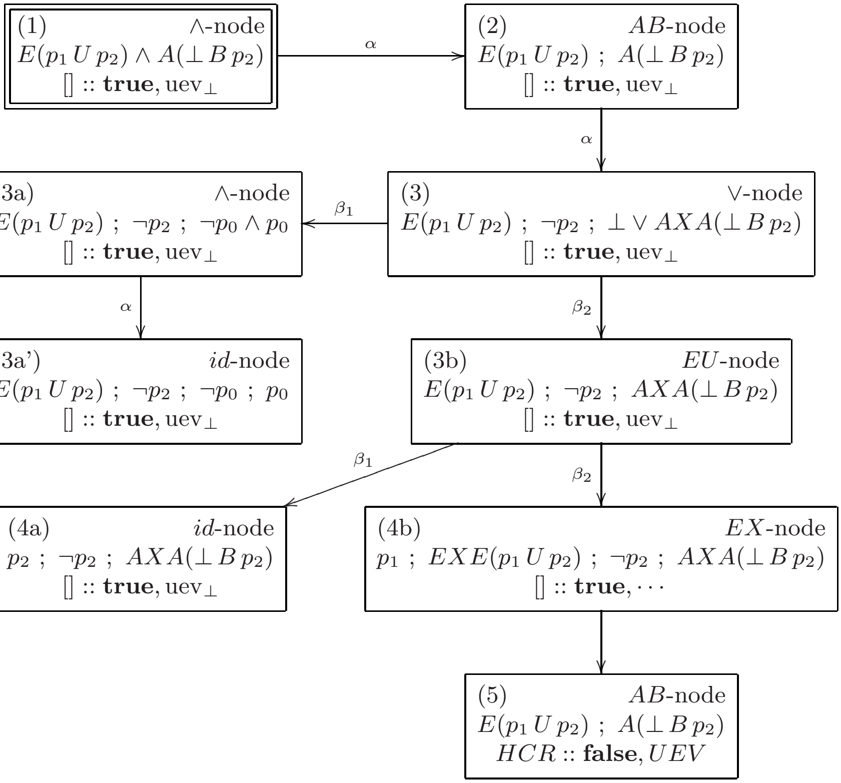

(1) ∧-node
$E(p_1 \, U \, p_2) \wedge A(\bot \, B \, p_2)$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
α
(2) AB-node
$E(p_1 \, U \, p_2) \; ; \; A(\bot \, B \, p_2)$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
α
(3a) ∧-node
$E(p_1 \, U \, p_2) \; ; \; \neg p_2 \; ; \; \neg p_0 \wedge p_0$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
β_1
(3) ∨-node
$E(p_1 \, U \, p_2) \; ; \; \neg p_2 \; ; \; \bot \vee AXA(\bot \, B \, p_2)$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
α
β_2
(3a') id-node
$E(p_1 \, U \, p_2) \; ; \; \neg p_2 \; ; \; \neg p_0 \; ; \; p_0$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
(3b) EU-node
$E(p_1 \, U \, p_2) \; ; \; \neg p_2 \; ; \; AXA(\bot \, B \, p_2)$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
β_1
β_2
(4a) id-node
$p_2 \; ; \; \neg p_2 \; ; \; AXA(\bot \, B \, p_2)$
$[] :: \textbf{true}, \mathrm{uev}_\bot$
(4b) EX-node
$p_1 \; ; \; EXE(p_1 \, U \, p_2) \; ; \; \neg p_2 \; ; \; AXA(\bot \, B \, p_2)$
$[] :: \textbf{true}, \cdots$
(5) AB-node
$E(p_1 \, U \, p_2) \; ; \; A(\bot \, B \, p_2)$
$HCR :: \textbf{false}, UEV$

Fig. 1. An example: a tableau for $E(p_1 \, U \, p_2) \wedge A(\bot \, B \, p_2)$

The marking of the nodes (1) to (4a) in Fig. 1 with **true** is straightforward. Note that $\bot$ is just an abbreviation for $\neg p_0 \wedge p_0$ to save some space and make things easier for the reader; the tableau procedure as described in this paper does not know about the symbol $\bot$. It is, however, not a problem to adapt the rules so that the tableau procedure can handle $\top$ and $\bot$ directly. For node (5), our procedure constructs the tableau shown in Fig. 2. The leaf (7b) is an EX-node, but it is "blocked" from creating the desired successor containing $\{E(p_1 \, U \, p_2), A(\bot \, B \, p_2)\}$ because there is a $j \in \mathbb{N}$ such that $\mathrm{HCr}_{7\mathrm{b}}[j] = HCR[j] = \{E(p_1 \, U \, p_2), A(\bot \, B \, p_2)\}$: namely $j = 1$. Thus the EX-rule computes $UEV(E(p_1 \, U \, p_2)) = 1$ as stated above and also puts $\mathrm{mrk}_{7\mathrm{b}} := \textbf{false}$. As the nodes (7a) and (6a) are marked, the function UEV is passed on to the nodes (6b), (6), and (5) according to the corresponding β- and α-rules.

The crux of our procedure happens at node (4b) which is an EX-node with $\mathrm{HCr}_{4\mathrm{b}} = []$ and hence $len(\mathrm{HCr}_{4\mathrm{b}}) = 0$. The EX-rule therefore finds a child node (5) and a formula $E(p_1 \, U \, p_2)$ in it such that $1 = UEV(E(p_1 \, U \, p_2)) = \mathrm{uev}_5(E(p_1 \, U \, p_2)) > len(\mathrm{HCr}_{4\mathrm{b}}) = 0$. That is, node (4b) "sees" a child (5) that "loops lower", meaning that node (5) is the root of an "isolated" subtree which

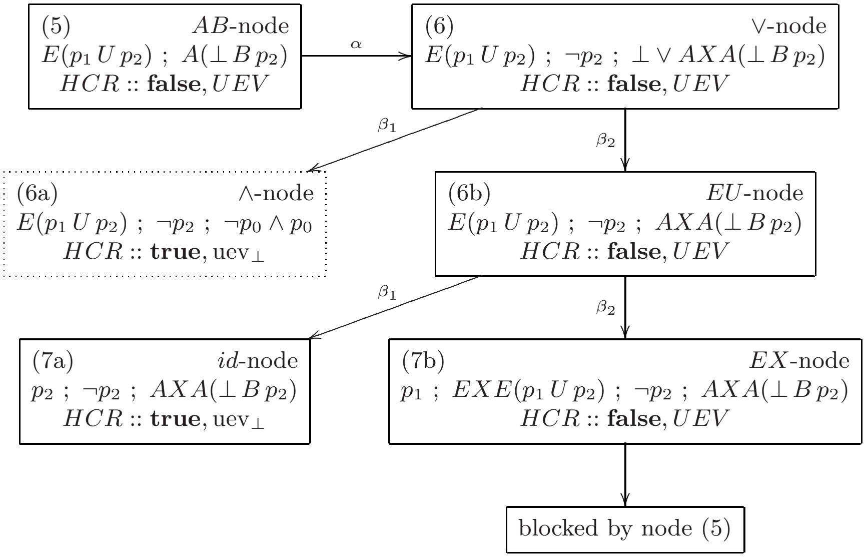

(5) AB-node
$E(p_1\,U\,p_2)$; $A(\bot\,B\,p_2)$
HCR :: **false**, UEV
α
(6) $\vee$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\bot \vee AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
β_1
β_2
(6a) $\wedge$-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $\neg p_0 \wedge p_0$
HCR :: **true**, $\mathrm{uev}_\bot$
(6b) EU-node
$E(p_1\,U\,p_2)$; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
β_1
β_2
(7a) id-node
p_2 ; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **true**, $\mathrm{uev}_\bot$
(7b) EX-node
p_1 ; $EXE(p_1\,U\,p_2)$; $\neg p_2$; $AXA(\bot\,B\,p_2)$
HCR :: **false**, UEV
blocked by node (5)

Fig. 2. An example: a tableau for $E(p_1\,U\,p_2) \wedge A(\bot\,B\,p_2)$ (continued)

does not fulfil its eventuality $E(p_1\,U\,p_2)$. Thus the EX-rule sets $\mathrm{mrk}_{4\mathrm{b}} = \mathbf{true}$, marking (4b) as "closed". The propagation of **true** to the root is then just via simple β- and α-rule applications.

3.5 The One-Pass Algorithm and Its Complexity

Most tableau-based algorithms apply the rules in a particular order: namely, apply all the α- and β-rules until none are applicable, and then apply the EX-rule once. Of course, no more rules are applied if the *id*-rule is applicable to close the branch. We have designed the rules so that they naturally capture this strategy, thereby giving a non-deterministic algorithm for constructing/traversing the tableau by just applying any one of the rules that are applicable. By fixing an arbitrary rule order and an arbitrary formula order, we can safely determinise this algorithm.

The use of histories and variables gives rise to an algorithm that constructs and traverses a tableau (deterministically) at the same time. On its way down the tableau, it constructs the set of formulae and the histories of a node by using information from the parent node; and on its way up, it synthesises the variables of a node according to the values of the variables of its children. Both steps are described by the rule that is applied to the node.

As soon as the algorithm has left a node on its way up, there is no need to keep the node in memory, it can safely be reclaimed as all important information has been passed up by the variables. Hence, the algorithm requires just one pass. Moreover, at any time, it only has to keep the current branch of the tableau in memory. The final result of the decision procedure can be obtained by looking at the variable mrk of the root which is the last node that has its variables set.

Of course, it is not always necessary to build the entire tableau. If, for example, the first child of an EX-rule is marked, the algorithm can mark the parent without having to look at the other children. (It is easy to see that if a node x is marked then the value of uev_x is irrelevant and does not need to be calculated.) Dually, if the left child of a β-node is unmarked and has $\mathrm{uev}_\perp$ then there is no need to explore the right child since we can safely say that the parent is unmarked and has $\mathrm{uev}_\perp$. Other optimisations are possible and some of them are incorporated in our implementation in the Tableau Work Bench (`twb.rsise.anu.edu.au/twbdemo`), a generic tableau engine designed for rapid prototyping of (propositional) tableau calculi [1]. The high-level code of the prover for CTL is also visible there using the special input language designed for the TWB.

Theorem 23. [illegible] CTL [illegible]

A detailed proof can be found in the extended version of this paper (`users.rsise.anu.edu.au/~florian`).

References

1. Abate, P.: The Tableau Workbench: a framework for building automated tableau-based theorem provers. PhD thesis, The Australian National University (2006)
2. Baader, F., Tobies, S.: The inverse method implements the automata approach for modal satisfiability. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 92–106. Springer, Heidelberg (2001)
3. Baader, F.: Augmenting concept languages by transitive closure of roles: an alternative to terminological cycles. Technical Report, DFKI (1990)
4. Basukoski, A., Bolotov, A.: A clausal resolution method for branching time logic ECTL+. Annals of Mathematics and Artificial Intelligence 46(3), 235–263 (2006)
5. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. In: Proceedings of Principles of Programming Languages (1981)
6. Emerson, E.A., Halpern, J.Y.: Decision procedures and expressiveness in the temporal logic of branching time. J. of Computer and System Sci. 30, 1–24 (1985)
7. Fisher, M., Dixon, C., Peim, M.: Clausal temporal resolution. ACM Transactions on Computational Logic (2001)
8. Gaintzarain, J., Hermo, M., Lucio, P., Navarro, M., Orejas, F.: A Cut-Free and Invariant-Free Sequent Calculus for PLTL. In: 6th EACSL Annual Conference on Computer Science and Logic (to appear, 2007)
9. Gough, G.: Decision procedures for temporal logics. Master's thesis, Dept. of Computer Science, University of Manchester, England (1984)
10. Grumberg, O., Clarke, E.M., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)
11. Horrocks, I., Patel-Schneider, P.F.: Optimising description logic subsumption. Journal of Logic and Computation 9(3), 267–293 (1999)

12. Horrocks, I., Sattler, U.: A tableaux decision procedure for SHOIQ. In: Proc. of the 19th Int. Joint Conf. on Artificial Intelligence, pp. 448–453 (2005)
13. Hustadt, U., Konev, B.: TRP++: A temporal resolution prover. In: Collegium Logicum, Kurt Gödel Society, pp. 65–79 (2004)
14. Jäger, G., Alberucci, L.: About cut elimination for logics of common knowledge. Ann. Pure Appl. Logic 133(1-3), 73–99 (2005)
15. Jäger, G., Kretz, M., Studer, T.: Cut-free common knowledge. Journal of Applied Logic (to appear)
16. Janssen, G.: Logics for Digital Circuit Verification: Theory, Algorithms, and Applications. PhD thesis, Eindhoven University of Technology, The Netherlands (1999)
17. Kesten, Y., Manna, Z., McGuire, H., Pnueli, A.: A decision algorithm for full propositional temporal logic. In: Courcoubetis, C. (ed.) CAV 1993. LNCS, vol. 697, pp. 97–109. Springer, Heidelberg (1993)
18. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer, Heidelberg (1992)
19. Reynolds, M.: Towards a CTL* tableau. In: Ramanujam, R., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 384–395. Springer, Heidelberg (2005)
20. Schwendimann, S.: A new one-pass tableau calculus for PLTL. In: de Swart, H. (ed.) TABLEAUX 1998. LNCS (LNAI), vol. 1397, pp. 277–291. Springer, Heidelberg (1998)
21. Vardi, M.Y., Pan, G., Sattler, U.: BDD-based decision procedures for K. Journal of Applied Non-Classical Logics 16(1-2), 169–208 (2006)
22. Vardi, M.Y., Wolper, P.: Automata-theoretic techniques for modal logics of programs. Journal of Computer Systems and Science (1986)
23. Voronkov, A.: How to optimize proof-search in modal logics: new methods of proving redundancy criteria for sequent calculi. ACM Trans. on Comp. Logic 2(2) (2001)
24. Wolper, P.: Temporal logic can be more expressive. Inf. and Cont. 56, 72–99 (1983)

Extending a Resolution Prover for Inequalities on Elementary Functions

Behzad Akbarpour and Lawrence C. Paulson

Computer Laboratory, University of Cambridge, England
{ba265,lcp}@cl.cam.ac.uk

Abstract. Experiments show that many inequalities involving exponentials and logarithms can be proved automatically by combining a resolution theorem prover with a decision procedure for the theory of real closed fields (RCF). The method should be applicable to any functions for which polynomial upper and lower bounds are known. Most bounds only hold for specific argument ranges, but resolution can automatically perform the necessary case analyses. The system consists of a superposition prover (Metis) combined with John Harrison's RCF solver and a small amount of code to simplify literals with respect to the RCF theory.

1 Introduction

Despite the enormous progress that has been made in the development of decision procedures, many important problems are not decidable. In previous work [1], we have sketched ideas for solving inequalities over the elementary functions, such as exponentials, logarithms, sines and cosines. Our approach involves reducing them to inequalities in the theory of real closed fields (RCF), which is decidable. We argued that merely replacing occurrences of elementary functions by polynomial upper or lower bounds sufficed in many cases. However, we offered no implementation. In the present paper, we demonstrate that the method can be implemented by combining an RCF decision procedure with a resolution theorem prover.

The alternative approach would be to build a bespoke theorem prover that called theory-specific code. Examples include Analytica [8] and Weierstrass [6], both of which have achieved impressive results. However, building a new system from scratch will require more effort than building on existing technology. Moreover, the outcome might well be worse. For example, despite the well-known limitations of the sequent calculus, both Analytica and Weierstrass rely on it for logical reasoning. Also, it is difficult for other researchers to learn from and build upon a bespoke system. In contrast, Verifun [10] introduced the idea of combining existing SAT solvers with existing decision procedures; other researchers grasped the general concept and now SMT (SAT Modulo Theories) has become a well-known system architecture.

Our work is related to SPASS+T [17], which combines the resolution theorem prover SPASS with a number of SMT tools. However, there are some differences

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 47–61, 2007.
© Springer-Verlag Berlin Heidelberg 2007

between the two approaches. SPASS+T extends the resolution's test for unsatisfiability by allowing the SMT solver to declare the clauses inconsistent, and its objective is to improve the handling of quantification in SMT problems. We augment the resolution calculus to simplify clauses with respect to a theory, and our objective is to solve problems in this theory.

Our work can therefore be seen to address two separate questions.

- Can inequalities over the elementary functions be solved effectively?
- Can a modified resolution calculus serve as the basis for reasoning in a highly specialized theory?

At present we only have a small body of evidence, but the answer to both questions appears to be [illegible]. The combination of resolution with a decision procedure for RCF can prove many theorems where the necessary case analyses and other reasoning steps are found automatically. An advantage of this approach is that further knowledge about the problem domain can be added declaratively (as axioms) rather than procedurally (as code). We achieve a principled integration of two technologies by using one (RCF) in the simplification phase of the other (resolution).

We eventually intend to output proofs where at least the main steps are justified. Claims would then not have to be taken on trust, and such a system could be integrated with an interactive prover such as Isabelle [16]. The tools we have combined are both designed for precisely such an integration [13,15].

[illegible]. We begin (§2) by presenting the background for this work, including specific upper and lower bounds for the logarithm and exponential functions. We then describe our methods (§3): which axioms we used and how we modified the automatic prover. We work through a particular example (§4), indicating how our combined resolution/RCF solver proves it. We present a table of results (§5) and finally give brief conclusions (§6).

2 Background

The initial stimulus for our project was Avigad's formalization, using Isabelle, of the Prime Number Theorem [2]. This theorem concerns the logarithm function, and Avigad found that many obvious properties of logarithms were tedious and time-consuming to prove. We expect that proofs involving other so-called elementary functions, such as exponentials, sines and cosines, would be equally difficult. Avigad, along with Friedman, made an extensive investigation [3] into new ideas for combining decision procedures over the reals. Their approach provides insights into how mathematicians think. They outline the leading decision procedures and point out how easily they perform needless work. They present the example of proving

$$\frac{1+x^2}{(2+y)^{17}} < \frac{1+y^2}{(2+x)^{10}}$$

from the assumption $0 < x < y$: the argument is obvious by monotonicity, while mechanical procedures are likely to expand out the exponents. To preclude this

$$\frac{x-1}{x} \le \ln x \le \frac{3x^2-4x+1}{2x^2} \qquad \left(\frac{1}{2} \le x \le 1\right)$$
$$\frac{-x^2+4x-3}{2} \le \ln x \le x-1 \qquad (1 \le x \le 2)$$
$$\frac{-x^2+8x-8}{8} \le \ln x \le \frac{x}{2} \qquad (2 \le x \le 4)$$

Fig. 1. Upper and Lower Bounds for Logarithms

$$\frac{x^3+3x^2+6x+6}{6} \le \exp x \le \frac{x^2+2x+2}{2} \qquad (-1 \le x \le 0)$$
$$\frac{2}{x^2-2x+2} \le \exp x \le \frac{6}{-x^3+3x^2-6x+6} \qquad (0 \le x \le 1)$$

Fig. 2. Upper and Lower Bounds for Exponentials

possibility, Avigad and Friedman have formalized theories of the real numbers in which the distributivity axioms are restricted to multiplication by constants. As computer scientists, we do not see how this sort of theory could lead to practical tools or be applied to the particular problem of logarithms. We prefer to use existing technology, augmented with search and proof heuristics to this problem domain. We have no interest in completeness—these problems tend to be undecidable anyway—and do not require the generated proofs to be elegant.

Our previous paper [1] presented families of upper and lower bounds for the exponential and logarithm functions. These families, indexed by natural numbers, converge to their target functions. The examples described below use some members of these families which are fairly loose bounds, but adequate for many problems.

Figure 1 presents the upper and lower bounds for logarithms used in this paper. Note that each of these bounds constrains the argument to a closed interval. This particular family of bounds is only useful for finite intervals, and proofs involving unbounded intervals must refer to other properties of logarithms, such as monotonicity. Figure 2 presents upper and lower bounds for exponentials, which are again constrained to narrow intervals. Such constraints are necessary: obviously there exists no polynomial upper bound for $\exp x$ for unbounded x, and a bound like $\ln x \le x-1$ is probably too loose to be useful for large x. Tighter constraints on the argument allow tighter bounds, but at the cost of requiring case analysis on x, which complicates proofs.

Our approach to proving inequalities is to replace occurrences of functions such as ln by suitable bounds, and then to prove the resulting algebraic inequality. Our previous paper walked through a proof of

$$-\frac{1}{2} \le x \le 3 \Longrightarrow \ln(1+x) \le x. \tag{1}$$

One of the cases reduced to the following problem:

$$\frac{x}{1+x} + \frac{1}{2}\left(\frac{-x}{1+x}\right)^2 \leq x$$

As our paper shows, this problem is still non-trivial, but fortunately it belongs to a decidable theory. We have relied on two readable histories of this subject [9,15]. Tarski proved the decidability of the theory of Real Closed Fields (RCF) in the 1930s: quantifiers can be eliminated from any inequality over the reals involving only the operations of addition, subtraction and multiplication. It is inefficient: the most sophisticated decision procedure, [illegible] (CAD), can be doubly exponential. We use a simpler procedure, implemented by McLaughlin and Harrison [15], who in their turn credit Hörmander [12] and others. For the RCF problems that we generate, the decision procedure usually returns quickly: as Table 1 shows, most inequalities are proved in less than one second, and each proof involves a dozen or more RCF calls.

Although quantifier elimination is hyper-exponential, the critical parameters are the degrees of the polynomials and the number of variables in the formula. The length of the formula appears to be unimportant. At present, all of our problems involve one variable, but the simplest way to eliminate quotients and roots involves introducing new variables. We do encounter situations where RCF does not return.

Our idea of replacing function occurrences by upper or lower bounds involves numerous complications. In particular, most bounds are only valid for limited argument ranges, so proofs typically require case splits to cover the full range of possible inputs. For example, three separate upper bounds are required to prove equation (1). Another criticism is that bounds alone cannot prove the trivial theorem

$$0 < x \leq y \Longrightarrow \ln x \leq \ln y,$$

which follows by the monotonicity of the function ln. Special properties such as monotonicity must somehow be built into the algorithm. Search will be necessary, since some proof attempts will fail. If functions are nested, the approach has to be applied recursively. We could have written code to perform all of these tasks, but it seems natural to see whether we can add an RCF solver to an existing theorem prover instead.

For the automatic theorem prover, we chose Metis [13], developed by Joe Hurd. It is a clean, straightforward implementation of the superposition calculus [4]. Metis, though not well known, is ideal at this stage in our research. It is natural to start with a simple prover, especially considering that the RCF decision procedure is more likely to cause difficulties.

3 Method

Most resolution theorem provers implement some variant of the inference loop described by McCune and Wos [14]. There are two sets of clauses, [illegible] and [illegible]. The Active set enjoys the invariant that every one of its elements has been resolved with every other, while the Passive set consists of clauses waiting to be processed. At each iteration, these steps take place:

- An element of the Passive set (called the [illegible] clause) is selected and moved to the Active set.
- The given clause is resolved with every member of the Active set.
- Newly inferred clauses are first simplified, for example by rewriting, then added to the Passive set. (They can also simplify the Active and Passive sets by subsumption, an important point but not relevant to this paper.)

Resolution uses purely syntactical unification: no theory unification is involved. Our integration involves modifying the simplification phase to take account of the RCF theory. Algebraic terms are simplified and put into a canonical form. Literals are deleted if the RCF solver finds them to be inconsistent with algebraic facts present in the clauses. Both simplifications are essential. The canonical form eliminates a huge amount of redundant representations, for example the $n!$ permutations of the terms of $x_1+\cdots+x_n$. Literal deletion generates the empty clause if a new clause is inconsistent with existing algebraic facts, and more generally it eliminates much redundancy from clauses.

To summarize, we propose the following combination method:

1. Negate the problem and Skolemize it, finally converting the result into conjunctive normal form (CNF) represented by a list of conjecture clauses.
2. Combine the conjecture clauses with a set of axioms and make a problem file in TPTP format, for input to the resolution prover.
3. Apply the resolution procedure to the clauses. Simplify new clauses as described below before adding them to the Passive set.
4. If a contradiction is reached, we have refuted the negated formula.

3.1 Polynomial Simplification

All terms built up using constants, negation, addition, subtraction, and multiplication can be considered as multivariate polynomials. Following Grégoire and Mahboubi [11], we have chosen a canonical form for them: Horner normal form, also called the [illegible] representation.[1] An alternative is the [illegible] representation, a flat sum with an ordering of the monomials; however, our approach is often adopted and is easy to implement.

Any univariate polynomial can be rewritten in recursive form as

$$p(x) = a_n x^n + \cdots + a_1 x + a_0 = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1} + x a_n))$$

We can consider a multivariate polynomial as a polynomial in one variable whose coefficients are themselves a canonical polynomial in the remaining variables. We maintain a list with the innermost variable at the head, and this will determine the arrangement of variables in the canonical form. We adopt a [illegible] representation: zero terms are omitted.

---

[1] A representation is called *canonical* if two different representations always correspond to two different objects.

For example, if variables from the inside out are x, y and z, then we represent the polynomial $3xy^2 + 2x^2yz + zx + 3yz$ as

$$[y(z3)] + x([z1 + y(y3)] + x[y(z2)]),$$

where the terms in square brackets are considered as coefficients. Note that we have added numeric literals to Metis: the constant named 3, for example, denotes that number, and $3 + 2$ simplifies to 5.

We define arithmetic operations on canonical polynomials, subject to a fixed variable ordering. For addition, our task is to add $c + xp$ and $d + yq$. If x and y are different, one or other is recursively added to the constant coefficient of the other. Otherwise, we just compute $(c + xp) + (d + xq) = (c + d) + x(p + q)$, returning simply $c + d$ if $p + q = 0$. For negation, we recursively negate the coefficients, while subtraction is an easy combination of addition and negation.

We can base a recursive definition of polynomial multiplication on the following equation, solving the simpler sub-problems $p \times d$ and $p \times q$ recursively:

$$p \times (d + yq) = (p \times d) + (0 + y(p \times q))$$

However, for $0 + y(p \times q)$ to be in canonical form we need y to be the topmost variable overall, with p having no variables strictly earlier in the list. Hence, we first check which polynomial has the earlier topmost variable and exchange the operands if necessary. Powers p^n (for fixed n) are just repeated multiplication.

Any algebraic term can now be translated into canonical form by transforming constants and variables, then recursively applying the appropriate canonical form operations. We simplify a formula of the form $X \leq Y$ by converting $X - Y$ to canonical form, finally generating an equivalent form $X' \leq Y'$ with X' and Y' both canonical polynomials with their coefficients all positive. We simplify $1 + x \leq 4$ to $x \leq 3$, for example. Any fixed format can harm completeness, but note that the literal deletion strategy described below is indifferent to the particular representation of a formula.

3.2 Literal Deletion

We can distinguish a literal L in a clause by writing the clause as $A \vee L \vee B$, where A and B are disjunctions of zero or more literals. Now if $\neg A \wedge \neg B \wedge L$ is inconsistent, then the clause is equivalent to $A \vee B$; therefore, L can be deleted. We take the formula $\neg A \wedge \neg B \wedge L$ to be inconsistent if the RCF solver reduces it to **false**.

For example, consider the clause $x \leq 3 \vee x = 3$. (Such redundancies arise frequently.) We focus on the first literal and note that $x \neq 3 \wedge x \leq 3$ is consistent, so that literal is preserved. We focus on the second literal and note that $\neg(x \leq 3) \wedge x = 3$ is inconsistent, so that literal is deleted: we simplify the clause to $x \leq 3$.

This example illustrates a subtle point concerning variables and quantifiers. In the clause $x \leq 3 \vee x = 3$, the symbol x is a constant, typically a Skolem constant originating in a existentially quantified variable in the negated conjecture. Before calling RCF, we again convert x to an existentially quantified variable. Here is a precise justification of this practice. In the semantics of first-order logic [5], a *structure* for a first-order language L comprises a non-empty domain D and interprets the constant, function and relation symbols of L as corresponding elements, functions and relations over D. The clause $x \leq 3 \vee x = 3$ is satisfied by any structure that gives its symbols their usual meanings over the real numbers and also maps x to 3; it is, however, falsified if the structure instead maps x to 4. To prove the inconsistency of say $\neg(x \leq 3) \wedge x = 3$, we give the RCF solver a closed formula, $\exists x\, [\neg(x \leq 3) \wedge x = 3]$; if this formula is equivalent to **false**, then there exists no interpretation of x satisfying the original formula. More generally, we can prove the inconsistency of a ground formula by existentially quantifying over its uninterpreted constants before calling RCF.

We strengthen this test by taking account of all algebraic facts known to the prover. (At present, we consider only algebraic facts that are ground.) Whenever the automatic prover asserts new clauses, we extract those that concern only addition, subtraction and multiplication. We thus accumulate a list of algebraic clauses that we include in the formula that is delivered to the RCF solver. For example, consider proving $-\frac{1}{2} \leq u \leq 3 \Longrightarrow \ln(u+1) \leq u$. Upon termination, it has produced the following list of algebraic clauses:

```
F, ~(1 <= u), ~(0 <= u) \/ ~(u <= 1), 0 <= 1 + u * 2, u <= 3
```

This list is built from right to left. It begins with $u \leq 3$ from the problem statement, while $0 \leq 1 + u \times 2$ is soon derived from $-\frac{1}{2} \leq u$. The other clauses arise during the proof, which terminates with **false**.

To summarize, literal deletion works as follows, where C denotes the conjunction of all ground algebraic clauses known to the prover.

1. Identify a candidate literal L in a clause by writing the clause as $A \vee L \vee B$. Note that L must be algebraic.
2. Form the existential closure of the formula $\neg A \wedge \neg B \wedge C \wedge L$, retaining only the algebraic literals of A and B.
3. If RCF quantifier elimination reduces this formula to **false**, then delete L from the clause.

The formulas given to RCF contain no universal quantifiers, because at present they are constructed from ground clauses. As mentioned above, every uninterpreted constant contained in the formula becomes an existentially quantified variable.

3.3 Axioms

We have sought to find a general set of axioms describing the real numbers, and in particular their ordering. We combine these general axioms with upper and lower bounds for the functions of interest. The resulting axiom set replaces inequalities concerning those functions by algebraic inequalities. These, in turn, will be simplified by the RCF solver.

We include axioms that relate division to multiplication, since RCF solvers do not accept division.

$$\begin{aligned}
X \le Y \times Z &\implies X/Z \le Y \vee Z \le 0 \\
X \le Y/Z &\implies X \times Z \le Y \vee Z \le 0 \\
X \times Z \le Y &\implies X \le Y/Z \vee Z \le 0 \\
X/Z \le Y &\implies X \le Y \times Z \vee Z \le 0
\end{aligned}$$

We include similar axioms for equality.

$$\begin{aligned}
Y/Z = X &\implies Z = 0 \vee Y = X \times Z \\
X = Y/Z &\implies Z = 0 \vee X \times Z = Y
\end{aligned}$$

These axioms simplify inequalities between quotients.

$$\begin{aligned}
X/Y \le W/Z &\implies Y \le 0 \vee Z \le 0 \vee X \times Z \le Y \times W \\
Y \times W \le X \times Z &\implies W/Z \le X/Y \vee Y \le 0 \vee Z \le 0
\end{aligned}$$

Our use of a canonical polynomial representation eliminates the need for the usual axioms for addition and multiplication, such as commutative laws.

We have experimented with axioms defining the standard properties of a linear ordering:

$$X \le X \tag{2}$$

$$X \le Y \vee Y \le X \tag{3}$$

$$X \le Y \wedge Y \le X \implies X = Y \tag{4}$$

$$X \le Y \wedge Y \le Z \implies X \le Z \tag{5}$$

Note that inequalities are formalized using $\le$. We formalize $<$ by the equivalence $X < Y \iff \neg(Y \le X)$. This eliminates the need to have, for example, four versions of transitivity.

$$\begin{gathered}
\neg(X < Y) \vee \neg(Y \le X) \\
(X < Y) \vee (Y \le X)
\end{gathered}$$

However, the experiments reported below do not use axioms (2)–(5). Transitivity, in particular, blows up the search space. When transitivity is omitted, the lower and upper bound axioms must be modified in the obvious way; for example, the axiom $\phi \implies \ln X \le e$ must become $\phi \wedge e \le Y \implies \ln X \le Y$. We intend to do more work to find the best treatment of ordering properties.

These axioms are rather general. We can influence the way they are applied by means of [illegible], which influence the selection of literals in ordered resolution. Giving high weights (500 000) to the functions ln and exp encourages the prover to eliminate them. We also give division a high weight (50), encouraging its replacement by multiplication. It is obvious that occurrences of certain functions must be discouraged, but the effect of adding weights was more powerful than we expected.

4 Worked Example

In order to see how this approach works, let us follow its proof of the formula

$$\forall X\,\bigl[-1/2 \leq X \wedge X \leq 3 \Longrightarrow \ln(1+X) \leq X\bigr].$$

Below, for the sake of readability, we write

$$L_1 \wedge \ldots \wedge L_n \implies \textbf{false}$$

rather than

$$\neg L_1 \vee \ldots \vee \neg L_n.$$

We also use standard mathematical notation rather than Horner canonical form. The input file appears as Appendix A.

After negation and Skolemization, our problem consists of three conjecture clauses:

$$-1/2 \leq u$$
$$u \leq 3$$
$$\neg(\ln(1+u) \leq u)$$

The first conjecture clause resolves with one of the divisibility axioms and yields

$$-1 \leq u \times 2 \vee 2 \leq 0,$$

which simplifies to $0 \leq 1 + u \times 2$.

The high weight of ln will ensure that literals containing it are selected. Therefore, the negative literal above will combine with complementary literals in the axiom clauses specifying upper bounds of $\ln x$: that is, those shown in Fig. 1. One of these (combined with transitivity as described in §3.3) is

$$1 \leq X \wedge X \leq 2 \wedge X - 1 \leq Y \Longrightarrow \ln X \leq Y.$$

Resolution of the third conjecture clause with this axiom yields

$$1 \leq 1+u \wedge 1+u \leq 2 \wedge 1+u-1 \leq u \implies \textbf{false} \tag{6}$$

Performing the obvious simplifications, we get

$$0 \leq u \wedge u \leq 1 \implies \textbf{false}.$$

We have now deduced $u < 0 \vee u > 1$.

Another upper bound axiom (combined with transitivity) is

$$2 \leq X \wedge X \leq 4 \wedge \frac{X}{2} \leq Y \Longrightarrow \ln X \leq Y$$

when resolution with the third conjecture clause yields

$$2 \leq 1+u \wedge 1+u \leq 4 \wedge \frac{1+u}{2} \leq u \implies \textbf{false}.$$

The simplified clause (RCF deletes $u \leq 3$) is

$$1 \leq u \wedge \frac{1+u}{2} \leq u \implies \textbf{false}. \tag{7}$$

Resolution of this with the appropriate division axiom produces

$$1 \leq u \wedge 1 + u \leq u \times 2 \implies 2 \leq 0,$$

which simplifies to

$$1 \leq u \implies \textbf{false},$$

so we have deduced $u < 1$. Indeed (since $u < 0 \vee u > 1$) we have $u < 0$, and RCF will notice this.

Resolution of the third conjecture clause with the third upper bound axiom and the division axiom yields

$$1/2 \leq 1 + u \wedge 1 + u \leq 1 \wedge 3(1+u)^2 - 4(1+u) + 1 \leq 2u(1+u)^2 \implies 2u^2 \leq 0.$$

The obvious simplifications yield

$$-1/2 \leq u \wedge u \leq 0 \wedge 0 \leq u^2 + u^3 \implies 2u^2 \leq 0,$$

but given the facts $u < 0$ and $0 \leq 1 + u \times 2$, RCF further simplifies it to

$$\textbf{false}.$$

As mentioned earlier, this proof generates the following series of algebraic clauses, from right to left.

```
F, ~(1 <= u), ~(0 <= u) \/ ~(u <= 1), 0 <= 1 + u * 2, u <= 3
```

5 Results and Discussion

We have run only a few dozen examples, but the results are promising (Table 1). We are aware of no other system that can solve such problems, so we present the table merely to give an impression of what can be solved, rather than as a basis for comparison. Most of these problems are proved in under three seconds on a 3GHz Pentium D.

We clearly need to broaden our range of problems. Our examples all take the same form: that a basic inequality holds over a specific interval. The potential strength of a combination of resolution and RCF is that we might be able to solve such problems when they occur indirectly buried in some more complicated goals, perhaps resulting from the unification of other variables.

Limitations of our approach cause it to fail on some problems. Our bounds shown in Figs. 1 and 2 are sometimes too loose. For example, to prove $0 \leq x \leq 1/2 \implies -3x/2 \leq \ln(1-x)$, we have to use a tighter logarithmic lower bound:

$$\frac{11x^3 - 18x^2 + 9x - 2}{6x^3} \leq \ln x \qquad \left(\frac{1}{2} \leq x \leq 1\right)$$

Table 1. Problems and Runtimes

problem	seconds
$1/2 \le x \le 4 \Longrightarrow \ln x \le x - 1$	0.608
$1 \le x \le 4 \Longrightarrow \ln x \le x^2 - x$	0.060
$1/2 \le x \le 3/2 \Longrightarrow \ln x \le 2x^2 - 3x + 1$	0.643
$1/2 \le x \le 1 \Longrightarrow \ln x \le (3 - 3x)/2$	0.779
$-1/2 \le x \le 3 \Longrightarrow \ln(1 + x) \le x$	0.527
$0 \le x \le 3 \Longrightarrow \ln(1 + x) \le x + x^2$	0.060
$-1/2 \le x \le 1/2 \Longrightarrow \ln(1 + x) \le x + 2x^2$	2.500
$-1/2 \le x \le 0 \Longrightarrow \ln(1 + x) \le (-3x)/2$	2.457
$-3 \le x \le 1/2 \Longrightarrow \ln(1 - x) \le -x$	1.929
$-3 \le x \le 0 \Longrightarrow \ln(1 - x) \le x^2 - x$	0.219
$-1/2 \le x \le 1/2 \Longrightarrow \ln(1 - x) \le 2x^2 - x$	4.256
$0 \le x \le 1/2 \Longrightarrow \ln(1 - x) \le (3x)/2$	3.303
$1/2 \le x \le 4 \Longrightarrow (x - 1)/x \le \ln x$	0.272
$1 \le x \le 4 \Longrightarrow -x^2 + 3x - 2 \le \ln x$	0.305
$1/2 \le x \le 3/2 \Longrightarrow -2x^2 + 5x - 3 \le \ln x$	0.258
$-1/2 \le x \le 3 \Longrightarrow x/(1 + x) \le \ln(1 + x)$	2.592
$0 \le x \le 3 \Longrightarrow x - x^2 \le \ln(1 + x)$	0.723
$-1/2 \le x \le 1/2 \Longrightarrow x - 2x^2 \le \ln(1 + x)$	1.643
$-3 \le x \le 1/2 \Longrightarrow -x/(1 - x) \le \ln(1 - x)$	0.836
$-3 \le x \le 0 \Longrightarrow -x - x^2 \le \ln(1 - x)$	0.779
$-1/2 \le x \le 1/2 \Longrightarrow -x - 2x^2 \le \ln(1 - x)$	1.133
$-1 \le x \le 0 \Longrightarrow \exp x \le (2 + x)/2$	0.307
$-1 \le x \le 0 \Longrightarrow \exp x \le (4 + x)/4$	0.363
$0 \le x \le 1 \Longrightarrow \exp x \le 1 + x + x^2$	0.781
$-1 \le x < 1 \Longrightarrow \exp x \le 1/(1 - x)$	0.800
$0 \le x \le 1 \Longrightarrow \exp(-x) \le (2 - x)/2$	0.319
$-1 < x \le 1 \Longrightarrow \exp(-x) \le 1/(1 + x)$	0.614
$-1 \le x \le 1 \Longrightarrow 1 + x \le \exp x$	0.611
$0 \le x \le 1 \Longrightarrow (4 + x)/4 \le \exp x$	0.926
$-1 \le x \le 0 \Longrightarrow (4 + 7x)/4 \le \exp x$	0.613
$-1 \le x \le 1 \Longrightarrow 1 - x \le \exp(-x)$	0.993

Similarly, proving $0 \le x \le 1 \implies \exp x \le (4 + 7x)/4$ requires using a tighter exponential upper bound:

$$\exp x \le 120/(-x^5 + 5x^4 - 20x^3 + 60x^2 - 120x + 120) \qquad \Big(0 \le x \le 1\Big)$$

Some problems cannot be solved because the RCF decision procedure runs forever. One example is $0 \le x \le 1 \implies \exp(x - x^2) \le 1 + x$, which calls RCF on the following existentially quantified formula:

```
exists u. 0 <= u /\ u <= 1 /\
          ~ (u * u * u * u * (1 + u * u * 2) <=
             u * u * (3 + u * (2 + u * u * (3 + u * u))))
```

Introducing new variables allows some problems to be solved, but increases the danger that the decision procedure will loop. The problem $-1 < x \implies \exp(x/(1+x)) \leq 1+x$ is easily proved if we modify it, replacing the quotient by an extra variable y such that $(1+x)y = x$. As a second example, the univariate problem $-1/2 \leq x \leq 0 \implies x/\sqrt{1+x} \leq \ln(1+x)$ is first converted to a problem with two variables to avoid the square root: $-1/2 \leq x \leq 0 \land 0 \leq y \land y^2 = 1+x \implies x/y \leq \ln(1+x)$. Attempting the new problem will generate the following formula, which RCF cannot handle:

```
exists u v. 0 <= 1 + u * 2 /\ u <= 0 /\ 0 <= v /\ 1 + u = v * v /\
            ~ (u * ((2 + v * 2) + (u * ((4 + v * 3)+ u * 2))) <= 0)
```

These examples are too hard for a simple algorithm like Cohen-Hörmander. Eliminating two quantifiers from formulas involving nonlinear polynomials is not trivial. The first example is more marginal, but the doubly exponential complexity of Hörmander's algorithm seems to become noticeable when the degree of the polynomial exceeds 5. These examples suggest that we use a more powerful procedure, such as QEPCAD-B [7].

6 Conclusions

Inequalities concerning the elementary functions, such as exp and ln, can be proved by a simple combination of a resolution theorem prover and an RCF decision procedure. The architecture is simple and principled: it merely involves modifying resolution's simplification phase to take account of the RCF theory.

The axiom system requires further development and testing. It could contain a greater variety of upper and lower bounds. For example, the very loose bound $\ln x \leq x - 1$ might be useful when x can be arbitrarily large. Use of bounds such as $\ln x \leq 2(\sqrt{x} - 1)$ requires a means of eliminating the square root operator, through a translation such as $\exists y\, [y^2 = x \to \cdots]$. Most of the bounds are infinite families of axioms, so we must develop a preprocessing phase that inserts required instances of these axioms into the problem automatically. The general ordering axioms, such as transitivity, greatly expand the search space; we need to explore other methods of handling ordering properties.

Also, we need to consider a wider range of problems, including difficult features such as nested applications of elementary functions or sums and products of them. We need to consider equalities as well as inequalities. Introducing new variables to eliminate roots and quotients can cause the RCF procedure to run forever, so we intend to try using QEPCAD-B [7] instead.

[illegible] The research was funded by the EPSRC grant EP/C013409/1, [illegible]. John Harrison contributed his code for the RCF decision procedure, which Amine Chaieb helped to port from Ocaml to Standard ML. Joe Hurd offered much help with his Metis prover. Christoph Benzmüller and the referees commented on this paper.

References

1. Akbarpour, B., Paulson, L.C.: Towards automatic proofs of inequalities involving elementary functions. In: Cook, B., Sebastiani, R. (eds.) PDPAR: Pragmatics of Decision Procedures in Automated Reasoning, pp. 27–37 (2006)
2. Avigad, J., Donnelly, K., Gray, D., Raff, P.: A formally verified proof of the prime number theorem. ACM Transactions on Computational Logic (in press)
3. Avigad, J., Friedman, H.: Combining decision procedures for the reals. Logical Methods in Computer Science 2(4) (2006)
4. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, ch. 2, pp. 19–99. Elsevier Science, Amsterdam (2001)
5. Barwise, J.: An introduction to first-order logic. In: Barwise, J. (ed.) Handbook of Mathematical Logic, North-Holland, pp. 5–46 (1977)
6. Beeson, M.: Automatic generation of a proof of the irrationality of e. JSC 32(4), 333–349 (2001)
7. Brown, C.W.: QEPCAD B: a program for computing with semi-algebraic sets using CADs. SIGSAM Bulletin 37(4), 97–108 (2003)
8. Clarke, E., Zhao, X.: Analytica: A theorem prover for Mathematica. Mathematica Journal 3(1), 56–71 (1993)
9. Dolzmann, A., Sturm, T., Weispfenning, V.: Real quantifier elimination in practice. Technical Report MIP-9720, Universität Passau, D-94030, Germany (1997)
10. Flanagan, C., Joshi, R., Ou, X., Saxe, J.B.: Theorem proving using lazy proof explication. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 355–367. Springer, Heidelberg (2003)
11. Grégoire, B., Mahboubi, A.: Proving equalities in a commutative ring done right in coq. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 98–113. Springer, Heidelberg (2005)
12. Hörmander, L.: The Analysis of Linear Partial Differential Operators II: Differential Operators with Constant Coefficient. Springer, Heidelberg (2006) First published in 1983; cited by Mclaughlin and Harrison [15]
13. Hurd, J.: Metis first order prover (2007), `http://gilith.com/software/metis/`
14. McCune, W., Wos, L.: Otter: The CADE-13 competition incarnations. Journal of Automated Reasoning 18(2), 211–220 (1997)
15. McLaughlin, S., Harrison, J.: A proof-producing decision procedure for real arithmetic. In: Nieuwenhuis, R. (ed.) CADE-20. LNCS (LNAI), vol. 3632, pp. 295–314. Springer, Heidelberg (2005)
16. Nipkow, T., Paulson, L.C., Wenzel, M. (eds.): Isabelle/HOL. LNCS, vol. 2283. Springer, Heidelberg (2002)
17. Prevosto, V., Waldmann, U.: SPASS+T. In: Sutcliffe, G., Schmidt, R., Schulz, S.: (eds.) FLoC 2006 Workshop on Empirically Successful Computerized Reasoning, vol. 192 of CEUR Workshop Proceedings, pp. 18–33 (2006)

A Input File for the Sample Problem

```
cnf(leq_left_divide_mul,axiom,
    ( ~ less_equal(X,multiply(Y,Z))
    | less_equal(divide(X,Z),Y)
    | less_equal(Z,0) )).

cnf(leq_left_mul_divide,axiom,
    ( ~ less_equal(X,divide(Y,Z))
    | less_equal(multiply(X,Z),Y)
    | less_equal(Z,0) )).

cnf(leq_right_divide_mul,axiom,
    ( ~ less_equal(multiply(X,Z),Y)
    | less_equal(X,divide(Y,Z))
    | less_equal(Z,0) )).

cnf(leq_right_mul_divide,axiom,
    ( ~ less_equal(divide(X,Z),Y)
    | less_equal(X,multiply(Y,Z))
    | less_equal(Z,0) )).

cnf(eq_left_divide_mul,axiom,
    ( ~ equal(divide(Y,Z),X)
    | equal(Z,0)
    | equal(Y,multiply(X,Z)) )).

cnf(eq_right_divide_mul,axiom,
    ( ~ equal(X,divide(Y,Z))
    | equal(Z,0)
    | equal(multiply(X,Z),Y) )).

cnf(leq_double_divide_mul,axiom,
    ( ~ less_equal(divide(X,Y),divide(W,Z))
    | less_equal(Y,0)
    | less_equal(Z,0)
    | less_equal(multiply(X,Z),multiply(Y,W)) )).

cnf(leq_double_mul_divide,axiom,
    ( ~ less_equal(multiply(Y,W),multiply(X,Z))
    | less_equal(divide(W,Z),divide(X,Y))
    | less_equal(Y,0)
    | less_equal(Z,0) )).

cnf(log_upper_bound_case_1,axiom,
    ( ~ less_equal(divide(1,2),X)
    | ~ less_equal(X,1)
    | ~ less_equal(divide(add(multiply(3,power(X,2)),
                   add(neg(multiply(4,X)),1)),multiply(2,power(X,2))),Y)
    | less_equal(ln(X),Y) )).
```

```
cnf(log_upper_bound_case_2,axiom,
    ( ~ less_equal(1,X)
    | ~ less_equal(X,2)
    | ~ less_equal(subtract(X,1),Y)
    | less_equal(ln(X),Y) )).

cnf(log_upper_bound_case_3,axiom,
    ( ~ less_equal(2,X)
    | ~ less_equal(X,4)
    | ~ less_equal(divide(X,2),Y)
    | less_equal(ln(X),Y) )).

cnf(log_upper_bound_problem_5_1,negated_conjecture,
    (less_equal(divide(neg(1),2),u) )).

cnf(log_upper_bound_problem_5_2,negated_conjecture,
    (less_equal(u,3) )).

cnf(log_upper_bound_problem_5_3,negated_conjecture,
    (~ less_equal(ln(add(1,u)),u) )).
```

Model Checking the First-Order Fragment of Higher-Order Fixpoint Logic

Roland Axelsson[1] and Martin Lange[2]

[1] Institut für Informatik, Ludwig-Maximilians-Universität München, Germany
[2] Department of Computer Science, University of Aarhus, Denmark

Abstract. We present a model checking algorithm for HFL1, the first-order fragment of Higher-Order Fixpoint Logic. This logic is capable of expressing many interesting properties which are not regular and, hence, not expressible in the modal μ-calculus. The algorithm avoids best-case exponential behaviour by localising the computation of functions and can be implemented symbolically using BDDs.

We show how insight into the behaviour of this procedure, when run on a fixed formula, can be used to obtain specialised algorithms for particular problems. This yields, for example, the competitive antichain algorithm for NFA universality but also a new algorithm for a string matching problem.

1 Introduction

Properties (of words or trees) that can be expressed in the modal μ-calculus are at most regular, i.e. they can also be defined by a finite automaton [4]. The expressive power of temporal logics like LTL, CTL etc. is even strictly below that of full ω-regularity. Nevertheless, there are many natural examples of interesting properties that are not regular in the language theoretic sense, for example: detection of buffer underflows, unlimited counting, repetition of sequences of actions, etc.

Non-regular properties have recently attracted more and more attention, and the need for algorithmic handling of such properties is about to become accepted. This is for example manifested in CaRet, a specification formalism for linear time properties [1]. It can express some, but not all context-free languages.

Here we use the (branching temporal) fixpoint logic HFL1, the first-order fragment of [illegible] (HFL) [9]. HFL achieves high expressive power by combining the modal μ-calculus with a simply typed λ-calculus. We show that the first-order fragment which is obtained by restricting the λ-calculus part to first-order functions, is already capable of expressing interesting program properties like the ones mentioned above. Moreover, it turns out that many other problems – not necessarily from program verification only – can be reduced to the model checking problem for HFL1, e.g. the universality problem for non-deterministic finite automata (NFA-UNIV), the evaluation problem for quantified Boolean formulas (QBF), the satisfiability problem for modal logic

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 62–76, 2007.
© Springer-Verlag Berlin Heidelberg 2007

(K-SAT), etc. Commonly, these problems can be [illegible] in HFL1 in the sense that there is a [illegible] formula whose models are exactly the (encodings of) positive instances to such problems. Note that HFL1's model checking problem is EXPTIME-complete [2]. This comparably high computational complexity is necessarily the price to pay for the increased expressive power.

It is straight-forward to extend a symbolic model checking algorithm for $\mathcal{L}_\mu$ to HFL1 by doing fixpoint iterations in function space lattices. This, however, yields a [illegible] running time that is exponential in the size of the underlying model. Sect. 3 presents a model checking algorithm for HFL1 that localises the computation of fixpoints in function spaces and, hence, shows exponential behaviour in the [illegible] only.

Our algorithm exploits the fact that in order to model check an HFL1 property, one usually does not need to know the value of a function on all elements of its domain. Instead it computes these functions in a demand-driven fashion. Due to fixpoint operators the value of a function may be defined recursively, i.e. it may depend on the value of the same function on a different argument. By incorporating into fixpoint iteration the collecting of such function arguments we approximate the total functions in the HFL1 semantics by partial ones that agree with them on those arguments that cause the demand.

In static program analysis this technique is known as [illegible] [5].[1] It resembles [illegible] – avoiding computations that are unnecessary for the verification task. On the other hand, the algorithm is [illegible] in the sense that it computes in one go all states of a transition system that satisfy the given formula.

The algorithm works on sets of states using only standard operations. Thus, it can be implemented fully symbolically, i.e. using BDDs to represent state sets and transitions.

Below we first present HFL1 and give a few examples of expressible properties that are of interest in program verification. This is followed by the model checking algorithm and the proof of its soundness and completeness. The rest of the paper focuses on the "model checking is more than program verification" aspect mentioned above. We exemplarily pick out NFA-UNIV and show how it can be expressed in HFL1. We then use the Tabakov/Vardi model of random NFAs [8] to measure the gain of local fixpoint computations in comparison to the naïve extension of the modal μ-calculus model checker which would do fixpoint iterations in the function space lattice. We then show how to take advantage of the fact that NFA-UNIV can be reduced to the model checking problem for HFL1 on a [illegible]. This induces a special instance of our model checking algorithm which can be optimised w.r.t. that fixed formula. It turns out that this instance coincides with the competitive algorithm for NFA-UNIV by Henzinger et al., based on antichains [10]. We conclude by discussing further extensions of the model checking algorithm and further special instances for other problems like the ones mentioned above.

---

[1] Note that this has nothing to do with lazy evaluation in functional programming, let alone non-strict evaluation of higher-order functions!

2 The First-Order Fragment of HFL

Let Σ be a finite set of action names, $\mathcal{P}$ be an at most countably infinite set of propositions and $\mathcal{V}$ a countably infinite set of variable names. Formulas of HFL1 in positive normal form[2] are given by the following grammar.

$$\begin{array}{rcl} \varphi & := & q \mid \neg q \mid X \mid \neg X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \varphi\ \varphi \mid \lambda X.\varphi \mid \mu X^{\tau}.\varphi \mid \nu X^{\tau}.\varphi \\ \tau & := & \mathrm{Pr} \mid \mathrm{Pr} \to \tau \end{array}$$

where $a \in \Sigma$, $q \in \mathcal{P}$, and $X \in \mathcal{V}$. We require each fixpoint variable X to only occur positively in its binding formula $\sigma X.\varphi$, for some $\sigma \in \{\mu, \nu\}$, and never to be bound more than once. Hence, each variable comes with a unique fixpoint type μ or ν.

The τ are called HFL1-types. Note that each one is of the form $\tau_k = \mathrm{Pr} \to \ldots \to \mathrm{Pr} \to \mathrm{Pr}$ with k function arrows in it. The type τ_0 is used to model predicates, and τ_k for $k \geq 1$ models k-ary predicate transformers. A predicate is the same as a 0-ary predicate transformer.

Type annotations are used in order to avoid polymorphic effects. With these annotations, each formula obtains a unique type. We assume formulas to be well-typed using standard typing rules from the simply typed λ-calculus. For example, q, $\langle a \rangle \varphi$, $\varphi_1 \vee \varphi_2$ all have type Pr. In $\varphi\ \psi$, formula ψ must have type Pr, and φ must have type τ_k for some $k \geq 1$. The type of $\varphi = \lambda X.\psi$ can be inferred assuming that X has type Pr: if ψ has type τ_k then φ has type τ_{k+1}, etc.

Let $\mathcal{T} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \Sigma\}, L)$ be a transition system with state set $\mathcal{S}$, binary transition relations $\xrightarrow{a}$ for every $a \in \Sigma$, and a labeling function $L : \mathcal{P} \to 2^{\mathcal{S}}$ that assigns to every atomic proposition the set of states in which it is true.

We write $\mathcal{D}(\mathcal{S})$ for the domain of k-ary predicate transformers, $k \geq 0$. Formally, $\mathcal{D}(\mathcal{S})$ is the least fixpoint of the domain equation $\mathcal{X} = 2^{\mathcal{S}} + (2^{\mathcal{S}} \to \mathcal{X})$. Note that $\mathcal{D}(\mathcal{S}) \simeq \bigcup_{k \in \mathbb{N}} \mathcal{D}^k(\mathcal{S})$ where $\mathcal{D}^k(\mathcal{S}) = 2^{\mathcal{S}} \to \ldots \to 2^{\mathcal{S}} \to 2^{\mathcal{S}}$ with k arrows in it. Each $\mathcal{D}^k(\mathcal{S})$ forms a complete lattice with pointwise inclusion ordering $\sqsubseteq$ and meets $\sqcap$ and joins $\sqcup$. In the case of $k = 0$, these operations simply boil down to $\subseteq$, $\cap$ and $\cup$.

Let $\rho : \mathcal{V} \to \mathcal{D}(\mathcal{S})$ be an environment mapping variables to predicate transformers. The semantics of HFL1 is explained as follows. Note that a subformula of the form $\neg X$ can only be of type Pr, and X must be λ-bound in this case.

$$\begin{array}{rcl} [\![q]\!]^{\mathcal{T}}_{\rho} & := & L(q) \\ [\![\neg q]\!]^{\mathcal{T}}_{\rho} & := & \mathcal{S} \setminus L(q) \\ [\![X]\!]^{\mathcal{T}}_{\rho} & := & \rho(X) \\ [\![\neg X]\!]^{\mathcal{T}}_{\rho} & := & \mathcal{S} \setminus \rho(X) \\ [\![\varphi \vee \psi]\!]^{\mathcal{T}}_{\rho} & := & [\![\varphi]\!]^{\mathcal{T}}_{\rho} \cup [\![\psi]\!]^{\mathcal{T}}_{\rho} \\ [\![\varphi \wedge \psi]\!]^{\mathcal{T}}_{\rho} & := & [\![\varphi]\!]^{\mathcal{T}}_{\rho} \cap [\![\psi]\!]^{\mathcal{T}}_{\rho} \end{array}$$

---

[2] Positive normal form does not impede the expressive power but simplifies the presentation of the semantics.

$$
\begin{array}{rcl}
[\![\langle a\rangle\varphi]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \exists t \in [\![\varphi]\!]^{\mathcal{T}}_{\rho} \text{ s.t. } s \xrightarrow{a} t\} \\
[\![[a]\varphi]\!]^{\mathcal{T}}_{\rho} & := & \{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \text{ implies } t \in [\![\varphi]\!]^{\mathcal{T}}_{\rho}\} \\
[\![\varphi\ \psi]\!]^{\mathcal{T}}_{\rho} & := & [\![\varphi]\!]^{\mathcal{T}}_{\rho}([\![\psi]\!]^{\mathcal{T}}_{\rho}) \\
[\![\lambda X.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \lambda T.[\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto T]} \text{ for } T \in 2^{\mathcal{S}} \\
[\![\mu X^{\tau_k}.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \bigsqcap \{f \in \mathcal{D}^k \mid [\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto f]} \sqsubseteq f\} \\
[\![\nu X^{\tau_k}.\varphi]\!]^{\mathcal{T}}_{\rho} & := & \bigsqcup \{f \in \mathcal{D}^k \mid f \sqsubseteq [\![\varphi]\!]^{\mathcal{T}}_{\rho[X\mapsto f]}\}
\end{array}
$$

The modal μ-calculus is easily seen to be a fragment of HFL1 and is in fact obtained by disallowing λ-abstraction and function application which implies that all subformulas are predicates only. HFL1 also subsumes FLC [9] which is basically obtained by restricting all types to τ_0 and τ_1 only.

[illegible] The following formula is true in a model iff it is bisimilar to a balanced tree. Note that bisimulation-invariance is an inherent property of HFL1, because it can be embedded into infinitary modal logic which in turn is incapable of distinguishing bisimilar models. It is therefore not possible to state that the model indeed [illegible] a balanced tree.

$$\varphi_{bal} \ := \ \big(\nu X^{\tau_2}.\lambda Z.\lambda Y.(\neg Z \vee \neg Y) \wedge (X\ \langle -\rangle Z\ \langle -\rangle Y)\big)\ \langle -\rangle \mathtt{tt}\ [-]\mathtt{ff}$$

This is best understood by unfolding the fixpoint formula to an infinite conjunction. It then simply says: for all $k \in \mathbb{N}$ it is not the case that both $\langle -\rangle^{k+1}\mathtt{tt}$ and $\langle -\rangle^{k}[-]\mathtt{ff}$ hold, i.e. if there is a path of length at least $k+1$ then there is no maximal path of length k only.

This property can be used to show for example that all runs of a non-deterministic program terminate after the same number of steps. It is also closely related to Emerson's [illegible] [3].

[illegible] HFL1 can easily express the absence of underflows in unbounded buffers – due to unboundedness clearly not a regular property.

$$\varphi_{buf} \ := \ \mu X^{\tau_1}.(\lambda Z.\langle \mathtt{out}\rangle Z \vee \langle \mathtt{in}\rangle(X\ (X\ Z)))\ \mathtt{tt}$$

This formula is best understood by comparing it to the CFG $X \to \mathtt{out} \mid \mathtt{in}\,X\,X$. It generates the language of all words $w = u\mathtt{out}$ s.t. $|u|_{\mathtt{in}} = |u|_{\mathtt{out}}$ and for all prefixes v of w we have: $|v|_{\mathtt{in}} \geq |v|_{\mathtt{out}}$. These are exactly the prefixes of buffer runs which are violating due to an underflow.

Of course, for buffers of fixed capacity n this property can easily be expressed in the modal μ-calculus. Being fixed is not only sufficient but also necessary for definability in the μ-calculus. Hence, there are two distinct advantages that logics like HFL1 have over the μ-calculus.

- HFL1 enables [illegible] verification when the exact size of the underlying finite model is unknown.
- Properties like the ones above can be formalised in HFL1 using a [illegible] formula, whereas expressing them in the μ-calculus requires a family of formulas that grow at least linearly in the size of the models.

3 Model Checking HFL1

A naïve extension of the standard global model checker for the modal μ-calculus would represent the semantics of a subformula of type τ_k as a table of the following form. Let $\mathcal{S} = \{s_0, s_1, \dots, s_{n-1}\}$ be the finite state space of the underlying transition system.

[illegible] 1	$\emptyset$	$\{s_0\}$	$\{s_1\}$	$\dots$	$\{s_0, s_1\}$	$\dots$	$\mathcal{S}$	$\emptyset$	$\{s_0\}$	$\dots$
[illegible] 2	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$	$\emptyset$	$\dots$	$\emptyset$	$\{s_0\}$	$\{s_0\}$	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
[illegible] k	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$	$\dots$	$\emptyset$	$\emptyset$	$\emptyset$	$\dots$
[illegible]	T_0	T_1	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

It simply lists the value of that function for every possible argument from its finite domain. Note that this has uncurried the function silently. Fixpoint iteration can be done on these objects in the same way. For a least fixpoint such a table is initialised with $T_i = \emptyset$ for all i. Successive values of this function approximating the least fixpoint can be found by iterating the corresponding functional on these values and updating the table. Note that the width of the table is 2^{kn}. Hence, a fixpoint iteration might take $O(n \cdot 2^{kn})$ many steps due to monotonicity. This leads, for any $\varphi \in$ HFL1, to an exponential model checking algorithm for this fixed φ.

However, this procedure can be localised. It is never necessary to know the entire value of a function. Functions only occur as applicators, i.e. only their value on certain arguments are needed for the model checking problem. Only in the worst case and when embedded in a fixpoint iteration, the value of a function on all possible arguments might be required. This leads to the following idea of a localised model checker for HFL1, depicted in Fig. 1.

Each variable X of type τ_k in the input formula φ_0 is associated with a [illegible] function of type $\mathcal{D}^k(\mathcal{S}) \to 2^{\mathcal{S}}$ represented for example by a table as shown above. Since they are allowed to be partial, not necessarily all columns are present. An argument to such a function is written as a list $[T_1, \dots, T_k] \in (2^{\mathcal{S}})^k$, or abbreviated as $\boldsymbol{T}$ for instance. Note that the empty list $[]$ for $k = 0$ is possible. We write [illegible] (f) for the set of arguments on which f is defined; $f = g$ if f and g have the same domain and agree on that; $f\{\boldsymbol{T} \mapsto U\}$ for the update of f either overwriting a value or extending the domain; and $\{\boldsymbol{T} \mapsto U\}$ for the partial function that contains only this single binding.

Algorithm [illegible] takes an HFL1 formula of type τ_k and a list of length k of subsets of the underlying state space $\mathcal{S}$. It returns the semantics of φ applied to these arguments w.r.t. an environment ρ that is given by the global variable [illegible] which maps each HFL1-variable to a partial function. Note that the semantics of HFL1 is defined using [illegible] functions though.

Algorithm [illegible] simply computes the semantics of a formula recursively according to the definition of HFL1. If the semantics of a fixpoint formula is needed – i.e. the value of the corresponding function on a particular element of its domain – then it performs a fixpoint iteration in the corresponding function space. However, it only computes needed values, hence, localises this fixpoint iteration. It starts with

```
global $env : \mathcal{V} \to \mathcal{D}(\mathcal{S})$

$MC(\varphi, [T_1, ..., T_k]) =$
  case $\varphi$ of $q$                    : $L(q)$
            $\neg q$                 : $\mathcal{S} \setminus L(q)$
            $\psi_1 \vee \psi_2$     : $MC(\psi_1, []) \cup MC(\psi_2, [])$
            $\psi_1 \wedge \psi_2$   : $MC(\psi_1, []) \cap MC(\psi_2, [])$
            $\langle a \rangle \psi$ : $\{s \in \mathcal{S} \mid \exists t \in MC(\psi, []) \text{ s.t. } s \xrightarrow{a} t\}$
            $[a]\psi$                : $\{s \in \mathcal{S} \mid \forall t \in \mathcal{S} : s \xrightarrow{a} t \Rightarrow t \in MC(\psi, [])\}$
            $X$                      : if $env(X)([T_1, ..., T_k]) =$ undef
                                       then let $T :=$ if $fp(X) = \mu$ then $\emptyset$ else $\mathcal{S}$
                                                 $env(X) := env(X)\{[T_1, \ldots, T_k] \mapsto T\}$
                                       return $env(X)([T_1, \ldots, T_k])$
            $\neg X$                 : $\mathcal{S} \setminus env(X)([])$
            $\lambda X.\psi$         : $env(X) := \{[] \mapsto T_1\}$
                                       return $MC(\psi, [T_2, ..., T_k])$
            $\psi_1\ \psi_2$         : $MC(\psi_1, [MC(\psi_2, []), T_1, ..., T_k])$
            $\sigma X.\psi$          : if $\sigma = \mu$ then $T := \emptyset$ else $T := \mathcal{S}$
                                       $env(X) := \{[T_1, \ldots, T_k] \mapsto T\}$
                                       repeat
                                         $f := env(X)$
                                         for all $[T'_1, ..., T'_k] \in Dom(env(X))$
                                           $env(X) := env(X)\{[T'_1, ..., T'_k] \mapsto MC(\psi, [T'_1, ..., T'_k])\}$
                                       until $f = env(X)$
                                       return $env(X)([T_1, \ldots, T_k])$
```

Fig. 1. A symbolic model checking algorithm for HFL1

the function that maps the given argument to the initial iteration value – $\emptyset$ or $\mathcal{S}$. If a fixpoint variable is reached during this iteration then the value of the semantics may be required on a different argument. In this case, the algorithm adds the new argument to the domain of this function and includes this in further iterations. This is why a global variable representing the environment is necessary.

We write $_{\rho}(\varphi, [T_1, \ldots, T_k])$ for the result of the call to with arguments φ and $[T_1, \ldots, T_k]$ when, at the beginning, the global variable resembles the environment ρ in the following way: for all $X \in \mathcal{V}$ of type τ_k and all $\boldsymbol{T} \in (2^{\mathcal{S}})^k$ we have

- if X is λ-bound then $(X) = \{[] \mapsto \rho(X)\}$,
- if X is μ-bound then $\rho(X)(\boldsymbol{T}) \neq \emptyset$ implies $(X)(\boldsymbol{T}) = \rho(X)(\boldsymbol{T})$,
- if X is ν-bound then $\rho(X)(\boldsymbol{T}) \neq \mathcal{S}$ implies $(X)(\boldsymbol{T}) = \rho(X)(\boldsymbol{T})$.

To illustrate how the algorithm works, consider the formula

$$\varphi_0 \ := \ \big(\mu X^{\tau_1}.\lambda Z.Z \vee \bigvee_{a \in \Sigma} X\ [a]Z\big)\ \neg q$$

and the transition system shown on the right side in Fig. 2. Intuitively, φ asserts that there is a sequence of actions s.t. all paths under that sequence lead to

X	$\{3\}$	$\{2,3\}$	$\{1,2,3\}$
0	$\emptyset$		
1	$\{3\}$	$\emptyset$	
2	$\{3\}$	$\{2,3\}$	$\emptyset$
3	$\{2,3\}$	$\{2,3\}$	$\{1,2,3\}$
4	$\{2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
5	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
6	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$

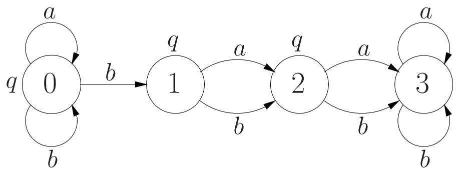


Fig. 2. Algorithm *MC* running on a simple example

a state not satisfying q. States $1, 2, 3$ satisfy this property, state 0 does not. However, the meaning of this formula is irrelevant for the understanding of how it is evaluated by algorithm MC.

The table on the left of Fig. 2 shows the successive calculation of the semantics of the fixpoint formula. Although only two rows need to be stored in each iteration step – the current one and the last one for comparison – we depict all stages in this example for the reader to be be able to follow this step-by-step.

At the beginning, the formula $\neg q$ is evaluated to $\{3\}$. This forms the initial argument in the table. It is to be read as follows: time proceeds line by line from left to right. Each row below the arguments contains a snapshot of the current state at the end of an iteration over the current domain. Note that in general fixpoint approximants cannot easily be read off the table since different columns may be at different stages of approximation. As computation proceeds, arguments are added to the list.

Row 6 then represents a partial function that agrees with the total function that is the semantics of the corresponding fixpoint formula. The return value is the one in the first column – the value of the fixpoint function applied to the original argument.

Theorem 1. [illegible] $\mathcal{T}$ [illegible] ρ [illegible] $\varphi \in$ [illegible] Pr [illegible] $_{\rho}(\varphi, []) = [\![\varphi]\!]^{\mathcal{T}}_{\rho}$

[illegible] By induction on the structure of the formula φ. However, it should be clear that the statement is too weak as an inductive invariant because of subformulas of types other than Pr. Instead, we prove the stronger statement

$$\forall\varphi, \forall\rho, \forall[T_1, \ldots, T_k] : \text{[illegible]}_{\rho}(\varphi, [T_1, \ldots, T_k]) = [\![\varphi]\!]^{\mathcal{T}}_{\rho}([T_1, \ldots, T_k]) \qquad (1)$$

where φ is a (not necessarily closed) formula of type τ_k, ρ is an environment that maps any free variable in φ to a function which in turn is defined on all arguments, and $T_i \subseteq \mathcal{S}$ are subsets of states of the underlying transition system $\mathcal{T}$. We also identify elements of type τ_0, i.e. such subsets, with functions from the singleton domain. Writing $[\![\varphi]\!]^{\mathcal{T}}_{\rho}([])$ rather than $[\![\varphi]\!]^{\mathcal{T}}_{\rho}$ simply spares us some case distinctions in the notations.

[illegible] Claim (1) is immediately seen to be true for the cases of $\varphi = q$ or $\varphi = \neg q$ for some $q \in \mathcal{P}$. It also follows directly from the hypothesis in the cases $\varphi = \psi_1 \vee \psi_2$, $\varphi = \psi_1 \wedge \psi_2$, $\varphi = \langle a \rangle \psi$ and $\varphi = [a]\psi$. Note that in all these cases, φ must have type τ_0, and so have ψ, ψ_1, ψ_2. Hence, we must have $k = 0$ and the argument list $[T_1, \ldots, T_k]$ must in fact be empty.

The statement is also easily seen to be true for the cases of $\varphi = X$ or $\varphi = \neg X$. Note that by assumption, the call of [illegible]$_\rho(X, [T_1, \ldots, T_k])$ returns $\rho(X)$. Also remember that negated variables must be λ-bound and therefore of type τ_0.

[illegible] Now consider the case $\varphi = \lambda X.\psi$. Note that φ cannot be of type τ_0, i.e. we have $k \geq 1$. Then [illegible]$_\rho(\varphi, [T_1, \ldots, T_k]) =$ [illegible]$_{\rho'}(\psi, [T_2, \ldots, T_k])$ with $\rho' := \rho\{X \mapsto T_1\}$. By hypothesis, this is equal to $[\![\psi]\!]^{\mathcal{T}}_{\rho'}([T_2, \ldots, T_k])$ which, by β-reduction, is also the same as $[\![\lambda X.\psi]\!]^{\mathcal{T}}_{\rho}([T_1, \ldots, T_k])$.

The case of $\varphi = \psi_1\ \psi_2$ is proved analogously. Again, note that here ψ_2 must be of type τ_0.

The only cases posing some difficulties are those of $\varphi = \sigma X.\psi$ for $\sigma \in \{\mu, \nu\}$. Here it is helpful to prove soundness (direction "$\subseteq$" in (1)) and completeness (direction "$\supseteq$") separately. However, the soundness proof for the μ-case is entirely analogous to the completeness proof of the ν-case and vice-versa. Thus, we only present soundness and completeness of the μ-case here.

[illegible] μ [illegible] Consider the call [illegible]$_\rho(\mu X.\psi, [T_1, \ldots, T_k])$ and the following statement.

$$\forall [T'_1, \ldots, T'_k] \in \text{[illegible]}(\text{[illegible]}(X)) : \text{[illegible]}(X)([T'_1, \ldots, T'_k]) \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}([T'_1, \ldots, T'_k]) \tag{2}$$

where [illegible] is assumed to resemble the environment ρ in the way described above. This is in fact an invariant of the `repeat`-loop in Algorithm [illegible]. It trivially holds before the loop because [illegible]$(\rho(X)) = \{[T_1, \ldots, T_k]\}$ only, and [illegible](X) maps this tuple to the empty set.

Furthermore, if statement (2) holds at the beginning of one iteration of the `repeat`-loop then it also holds after this iteration. This is simply a consequence of monotonicity, the hypothesis, and the fact that $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}$ is a fixpoint of ψ w.r.t. $\sqsubseteq$: if we have [illegible]$(X)([T'_1, \ldots, T'_k]) \subseteq [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}([T'_1, \ldots, T'_k])$ for all such tuples then, by monotonicity and the definition of the pointwise inclusion ordering, we also have $[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto env(X)\}} \sqsubseteq [\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto [\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}\}}$. But the latter is equal to $[\![\mu X.\psi]\!]^{\mathcal{T}}_{\rho}$, and the former is, by hypothesis, the content of [illegible](X) on all members of [illegible]([illegible](X)) at the end of this `repeat`-loop iteration.

This implicitly shows that – on finite transition systems – the loop eventually terminates. Since the domain of [illegible](X) at most grows in each iteration, we have $[T_1, \ldots, T_k] \in$ [illegible]([illegible](X)) at termination point, and the soundness part of (1) immediately follows from the fact that (2) holds at this point.

[illegible] μ [illegible] We will prove this part using fixpoint induction. For any set $D \subseteq (2^{\mathcal{S}})^k$ of k-tuples of subsets of the underlying state set $\mathcal{S}$, and two functions $f, g \in \mathcal{D}^k(\mathcal{S})$ we write $f \sqsubseteq_D g$ iff for all $[T'_1, \dots, T'_k] \in D$: $f([T'_1, \dots, T'_k]) \subseteq g([T'_1, \dots, T'_k])$.

Now consider again the call [illegible]$_\rho(\mu X.\psi, [T_1, \dots, T_k])$. Let $D :=$ [illegible]$(env(X))$ upon termination of the `repeat`-loop. An immediate consequence of the induction hypothesis for ψ is the following:

$$[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto f\}} \sqsubseteq_D env(X) \tag{3}$$

for any function f that agrees with $env(X)$ on all arguments in D. This is because the `repeat`-loop is iterated on the whole of D until stability is reached, i.e. until [illegible]$_{\rho\{X \mapsto env(X)\}}(\psi, [T'_1, \dots, T'_k]) = env(X)([T'_1, \dots, T'_k])$ holds for all $[T'_1, \dots, T'_k] \in D$.

We now extend the function $env(X)$ to a function $env^\top(X)$ in the following way.

$$env^\top(X)([T'_1, \dots, T'_k]) := \begin{cases} env(X)([T'_1, \dots, T'_k]) & \text{, if } [T'_1, \dots, T'_k] \in D \\ \mathcal{S} & \text{, o.w.} \end{cases}$$

Now note that we have

$$[\![\psi]\!]^{\mathcal{T}}_{\rho\{X \mapsto env^\top(X)\}} \sqsubseteq env^\top(X)$$

i.e. the function on the right subsumes the one on the left on all arguments from $\mathcal{D}^k(\mathcal{S})$. For arguments in D this is stated in (3) above. For all other arguments this is trivially true by the construction of $env^\top(X)$. But then $env^\top(X)$ is a prefixpoint of ψ and, hence, we have $[\![\mu X.\psi]\!]^{\mathcal{T}}_\rho \sqsubseteq env^\top(X)$. In particular, inclusion holds for all argument tuples in D. Since the domain of $env(X)$ at most grows in each iteration of the `repeat`-loop, we have $[T_1, \dots, T_k] \in D$ and therefore $[\![\mu X.\psi]\!]^{\mathcal{T}}_\rho([T_1, \dots, T_k]) \subseteq$ [illegible]$_\rho(\mu X.\psi, [T_1, \dots, T_k])$ which finishes the proof. □

4 NFA-UNIV as a Model Checking Problem

As stated in the introduction, HFL1 is powerful enough to enable the encoding of various interesting problems as model checking instances. We exemplarily pick the universality problem for non-deterministic finite automata (NFA) to demonstrate how encoding a problem as a model checking instance can lead to an efficient solution.

In the following, an NFA is always of the form $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is the state set, Σ the alphabet, δ the transition relation, q_0 the starting state and F the set of final states. Recall the model in Ex. 3. If the proposition q is interpreted as a flag for being a final state then the whole model can easily be viewed as an NFA. In this context the formula

$$\varphi_0 := \left(\mu X^{\tau_1}.\lambda Z.Z \vee \bigvee_{a \in \Sigma} X\ [a]Z\right) \neg q$$

translates to “there is a word w, s.t. all states reachable under w are non-final”. NFA-UNIV is solved by checking whether or not the starting state satisfies this formula.

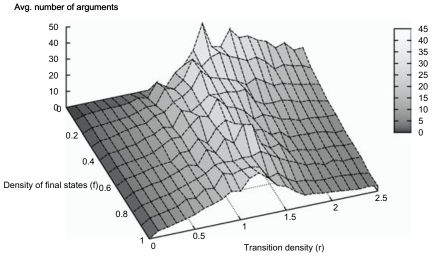


Fig. 3. Number of arguments in function table ($n = 10$)

4.1 Local Fixpoint Computation in Practice

We now give empirical evidence of the benefits of local fixpoint computations and demonstrate that the necessity to compute larger fragments of the complete domain rarely occurs. Algorithm ● has been implemented as a prototype in OCaml and run on the Tabakov/Vardi random model for NFAs in order to guarantee a wide spectrum of test cases. Two parameters s and t determine the number of randomly chosen final states and transitions in an NFA for a given number of states n. The ratios $f := \frac{s}{n}$ and $r := \frac{t}{n}$ are called final state density and transition density respectively. For further details see [8]. To perform the universality tests, we fix $n = 10$ and generate 20 random NFAs for each of 250 pairs (r, f) with $0 \leq r \leq 2.5$ and $0 \leq f \leq 1$.

The average number of arguments needed in the fixpoint computation by algorithm ● in dependence of (r, f) is depicted in Fig. 3. Note that the number of possible arguments $|2^{\mathcal{S}}|$ is 1024 in this case. Fig. 3 shows that in all cases the algorithm is far away from exhaustive fixpoint calculation on the full argument set $2^{\mathcal{S}}$. Even for the most difficult instances which in our tests are $f = 0.1$ and r between 1.4 and 1.6, the number of needed arguments never gets anywhere near that. The average number of arguments distributed over all 5000 tests is just 13.2 and the highest number of arguments ever measured during the tests is 109.

It is reasonable to assume that the approach of guiding the fixpoint iterations locally through neededness analysis also proves to be successful in other cases (on different formulas) unless the underlying models have been constructed pathologically to enforce an exponential behaviour.

4.2 Optimising Algorithm MC w.r.t. a Fixed Formula

There are still several standard performance enhancements available, e.g. acceleration of the fixpoint computation by exploiting monotonicity, in order to optimise this algorithm.

However, we need to observe that algorithm ● will be used on fixed formulas in most cases. In many verification tasks the property to be checked is fixed while the models change. This holds especially for non-regular properties since non-regularity often eliminates dependence on model sizes, etc. It is therefore much more beneficial to regard [illegible] as a [illegible] for specialised cases rather than a general algorithm for all kinds of verification purposes. Model checking a fixed formula bears a higher potential for algorithm optimisations which possibly cannot be achieved for varying formulas.

Consider the algorithm's behaviour on the formula of Ex. 3 as depicted in the table there. If we follow the succession of the fixpoint iteration closely, a simple pattern can be observed: the iterated function $\lambda Y.Y \vee \bigvee_{a \in \Sigma} X\ [a]Y$ takes an argument (initially the set $[\![\neg q]\!]^{\mathcal{A}}$) and returns its union with the set of its recursive $[a]$-predecessors for all $a \in \Sigma$. But this set is exactly the union of the elements of [illegible] (X), each of them the result of a single $[a]Y$ computation step. So the return value does not provide any additional information if the set of needed arguments is known. Furthermore, since only a union operation is performed, it suffices to keep track of $\subseteq$-maximal sets of arguments. This insight immediately leads to an optimisation by discarding all redundant information. It is obviously not necessary to protocol all these values in the fixpoint iterations – when in the end all we want to know is whether or not the initial automaton state is included in the union over all arguments. It suffices to iterate this schema until no more arguments enter the table, and then to form their unions. This, however, means that, by monotonicity of the $[a]$-operators, one can always discard the larger of two arguments that are comparable w.r.t. $\subseteq$ which leads to the idea of storing $Dom(X)$ as an [illegible].

An antichain over an NFA $\mathcal{A}$ is a set $\mathcal{C}$ of pairwise incomparable (w.r.t. set inclusion) sets of states of $\mathcal{A}$. These antichains form a complete lattice when equipped with the following order.

$$\mathcal{C} \sqsubseteq \mathcal{C}' \quad \text{iff} \quad \forall C \in \mathcal{C}\ \exists C' \in \mathcal{C}' \text{ s.t. } C \subseteq C'$$

This naturally induces a notion of supremum $\mathcal{C} \sqcup \mathcal{C}'$ as the smallest antichain (w.r.t. $\sqsubseteq$) which contains both $\mathcal{C}$ and $\mathcal{C}'$.

The basic principle of the optimization is to populate an antichain with sets of states which uphold the possibility of generating a word that is not included in the language of the automaton. This can be achieved by loosely speaking applying the modal $[a]$-operator (for all $a \in \Sigma$) to its elements and minimizing the resulting set to an antichain. More formally, define the following monotone operation on antichains:

$$[illegible]\ (\mathcal{C}) \ := \ \lceil \{S \subseteq Q \mid \exists T \in \mathcal{C}\ \exists a \in \Sigma \text{ s.t. } S = [\![[a]X]\!]^{\mathcal{A}}_{\{X \mapsto T\}}\} \rceil$$

where the $\lceil \cdot \rceil$ operator discards all sets which are subsumed by another set in this set of sets – i.e. it makes an antichain of the expression on the right-hand side.

Henzinger et al. show how to characterise NFA-UNIV using least fixpoints in antichain lattices.

Proposition 1. [10, Thm. 2] [illegible] $\mathcal{A}$ [illegible] Σ [illegible] Q [illegible] q_0 [illegible] F [illegible] $L(\mathcal{A}) \neq \Sigma^*$ iff $\{\{q_0\}\} \sqsubseteq \bigsqcap\{\mathcal{C} \mid$ [illegible] $(\mathcal{C}) \sqcup \{Q \setminus F\} \sqsubseteq \mathcal{C}\}$

Of course, the least fixpoint can be computed by a straight-forward fixpoint iteration: Define $\mathcal{C}^0 := \{\emptyset\}$ and $\mathcal{C}^i :=$ [illegible] $(\mathcal{C}^{i-1}) \sqcup \{Q \setminus F\}$. The following table compares in parallel two runs of [illegible] and the antichain method on Ex. 3:

X	$\{3\}$	$\{2,3\}$	$\{1,2,3\}$
0	$\emptyset$		
1	$\{3\}$	$\emptyset$	
2	$\{3\}$	$\{2,3\}$	$\emptyset$
3	$\{2,3\}$	$\{2,3\}$	$\{1,2,3\}$
4	$\{2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
5	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$
6	$\{1,2,3\}$	$\{1,2,3\}$	$\{1,2,3\}$

$\mathcal{C}^0 := \{\emptyset\}$

$\mathcal{C}^1 :=$ [illegible] $(\mathcal{C}^0) \sqcup \{Q \setminus F\} = \{\{3\}\}$

$\mathcal{C}^2 :=$ [illegible] $(\mathcal{C}^1) \sqcup \{Q \setminus F\} = \{\{2,3\}\}$

$\mathcal{C}^3 :=$ [illegible] $(\mathcal{C}^2) \sqcup \{Q \setminus F\} = \{\{1,2,3\}\}$

$\mathcal{C}^4 :=$ [illegible] $(\mathcal{C}^3) \sqcup \{Q \setminus F\} = \{\{1,2,3\}\}$

The cost reduction of the antichain method is established by the fact that it simply computes $\lceil$ [illegible] $(X)\rceil$, i.e. the antichain of the currently present arguments. One can show that $\lceil$ [illegible] $(X^i)\rceil = \mathcal{C}^{i+1}$, where [illegible] (X^i) is the currently needed domain of the ith fixpoint approximation w.r.t. a given argument and a partial evaluation according to [illegible].

It turns out that the result of this optimisation is exactly the method devised by Henzinger et al. in [10]. Their tool shows a very good performance on the universality test for NFAs and does apparently outperform the classical powerset construction by several orders of magnitude.

5 Conclusion and Outlook

We have presented a model checking algorithm for HFL1. This extends the scope of properties which can automatically be checked on finite state systems way beyond that of regular ones whilst keeping the complexity at most singly exponential in the size of the system. In order to avoid exponential best-case behaviour we suggest to localise computations of fixpoints in function spaces.

This algorithm fully supports symbolic model checking using a BDD library. A prototypical implementation has been created which shows the gain of local fixpoint computations in this setting. This approach is most successful on instances with fixed formulas. This allows to optimise algorithm [illegible] further w.r.t. that particular formula as seen with the NFA-UNIV example where a rigorous optimisation of algorithm [illegible] yields the competitive antichain algorithm of Henzinger et al.

5.1 Other Hard Decision Problems as Model Checking Instances

By not just restricting the term “model checking” to a method used in automatic program verification but understanding it as a general logic problem we can obtain BDD-based algorithms for various other problems as well. Note that NFA-UNIV is PSPACE-complete, and it is therefore reasonable to try to encode the standard PSPACE-complete problem QBF as an HFL1 model checking problem.

It is well-known that every quantified Boolean formula can be put into prenex CNF normal form $Q_1 x_1 . \ldots Q_n x_n . \bigwedge_i \bigvee_{j_i} l_{i,j_i}$ with the $Q_k \in \{\exists, \forall\}$, and the l_{i,j_i}

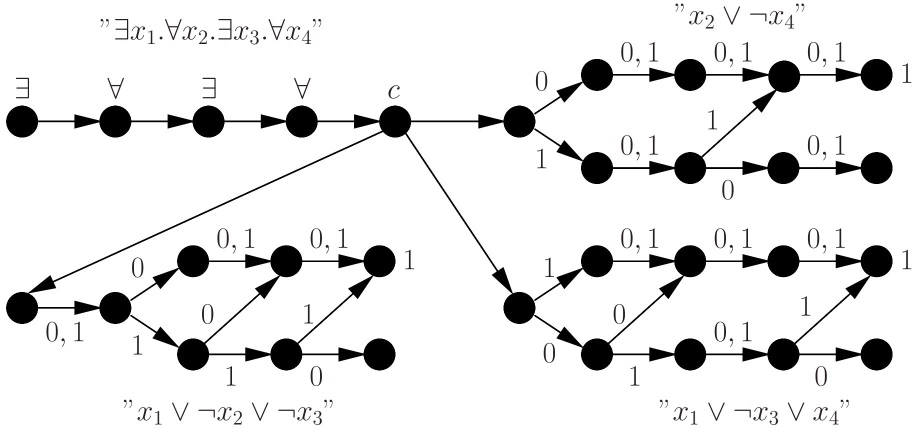


Fig. 4. A transition system representation of a QBF formula

literals over the variables $x_1, \ldots, x_n$. The problem QBF is to decide whether or not such a formula evaluates to 1 under the usual interpretation of the Boolean operators and the quantifiers over the domain $\{0, 1\}$.

With each QBF formula $\varPhi$ we associate a loop-free transition system $\mathcal{T}_\varPhi$ which is exemplarily shown in Fig. 4 for $\varPhi = \exists x_1.\forall x_2.\exists x_3.\forall x_4.(x_2 \vee \neg x_4) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$. It uses atomic propositions $\exists$, $\forall$ to mark the type of quantification over a variable, c to indicate the branching into the different clauses, and 1 to mark the value of a clause under an assignment valuation given by a path through each clause's component. Its actions are 0 and 1 for representing variable values, and an anonymous one for branching into different clauses and for separating the quantifiers in the prefix.

Evaluation to 1 of $\varPhi$ can now be expressed in HFL1 as follows.

$$\begin{aligned}\varphi_{QBF} \ :=\ \Big(\mu X^{\tau_1}.\lambda Z.\big(c \to [-]Z\big) \wedge \big(\exists \ \to\ \langle -\rangle(X\ \langle 0\rangle Z) \vee \langle -\rangle(X\ \langle 1\rangle Z)\big) \wedge \\ \big(\forall \ \to\ \langle -\rangle(X\ \langle 0\rangle Z) \wedge \langle -\rangle(X\ \langle 1\rangle Z)\big)\Big)\ 1\end{aligned}$$

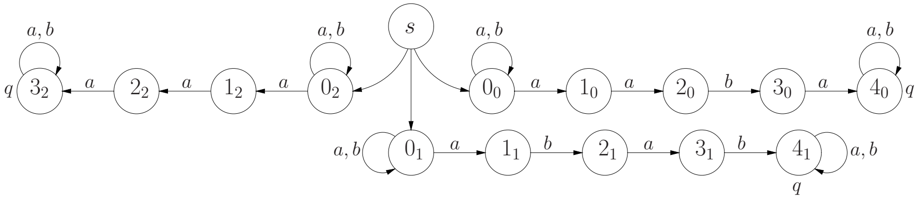
Again, φ_{QBF} does not depend on the underlying QBF formula $\varPhi$. It is therefore possible to obtain a (BDD-based – if desired) QBF solver by analysing the behaviour of algorithm • on φ_{QBF} and specialised transition systems $\mathcal{T}_\varPhi$. For example, it is not hard to see that the fixpoint iteration always terminates after a number of steps given by the length of the quantifier prefix. It can therefore be made explicit through a `for`-loop. Furthermore, antichains can also be used to replace the arguments of the function table. Preliminary results show that this is far from yielding a competitive QBF solver. However, it may be interesting to investigate combinations of this bottom-up approach with existing solvers that mostly work top-down.

Algorithms for other problems can also be obtained by instantiating the template • with a particular formula: satisfiability of modal logic where it remains to compare the result to the BDD-based algorithm by Pan et al. [7]; various problems from automata-theory like emptiness of alternating finite automata; etc. Due to space restrictions we will elaborate on details of those elsewhere.

5.2 Encoding Optimisation Problems

Some optimisation problems that require more than a yes/no answer can also be dealt with using an extension of algorithm • that keeps track of parts of the solution to be computed. We sketch a new algorithm for the Shortest Common Supersequence problem (SCS): given a set $\{w_1, \dots, w_n\}$ of finite words of some alphabet Σ, find a shortest $v \in \Sigma^*$ that contains all w_i as subwords. The algorithm is obtained from the template • using an antichain optimisation as in the case of NFA-UNIV.

The first step consists of building a transition system $\mathcal{T}$, here depicted for the words $\{aaba, abab, aaa\}$.



Next, consider the HFL1 formula $\varphi_{SCS} := \big(\mu X^{\tau_1}.\lambda Z.[-]Z \vee \bigvee_{a \in \Sigma} X\ \langle a \rangle Z\big)\ q$. Each state in $\mathcal{T}$ satisfies φ_{SCS} which only reflects the fact that for every finite set of words there is a word containing all of them. However, suppose the arguments in the table for the fixpoint iteration in this formula are annotated in the following way: the initial argument receives the annotation ϵ, and if an argument Z with annotation w causes another argument to be created in the table through the recursive call of $X\ \langle a \rangle Z$ then the new argument receives the annotation aw.

Now note the apparent similarity of this formula with the one from Ex. 3 expressing NFA-UNIV. In both cases the subformulas $X\ \psi(Z)$ only occur under a disjunction. Hence, the argument row of the function table can again be optimised into an antichain, and the evaluation of the formula can be regarded as a fixpoint iteration in an antichain lattice. It terminates when the topmost state of $\mathcal{T}$ occurs in an element of the current antichain, and that element's annotation is the solution to the SCS problem.

The computation of the solution $aaabab$ using annotated antichains is found as follows. Let $I := \{4_0, 4_1, 3_2\}$. For a set S we write S_I^w to abbreviate $(S \cup I)^w$ where the superscript simply denotes the word annotation of this set.

$$
\begin{aligned}
\mathcal{C}_0 &:= \{I^\epsilon\} \\
\mathcal{C}_1 &:= \{\{2_2, 3_0\}_I^a, \{3_1\}_I^b\} \\
\mathcal{C}_2 &:= \{\{2_2, 2_1, 3_0\}_I^{ab}, \{2_2, 1_2, 3_0\}_I^{aa}, \{3_1, 2_0\}_I^{ba}\} \\
\mathcal{C}_3 &:= \{\{2_2, 2_1, 3_0, 1_0\}_I^{aba}, \{2_2, 1_2, 0_2, 3_0\}_I^{aaa}, \{3_1, 1_1, 2_0\}_I^{bab}\} \\
\mathcal{C}_4 &:= \{\{2_2, 2_1, 0_1, 3_0, 1_0\}_I^{abab}, \{2_2, 1_2, 0_2, 3_0\}_I^{aaaa}, \{2_2, 1_2, 3_0, 0_0\}_I^{aaba}, \\
&\qquad \{0_2, 3_1, 2_0\}_I^{baaa}, \{3_1, 1_1, 2_0\}_I^{baba}\}
\end{aligned}
$$

$$\begin{aligned}\mathcal{C}_5 &:= \{\{\ldots\}_I^{ababa}, \{\ldots\}_I^{abaaa}, \{\ldots\}_I^{aaaaa}, \{2_2, 1_2, 0_1, 3_0, 0_0\}_I^{aabab},\\ &\qquad \{\ldots\}_I^{baaba}, \{\ldots\}_I^{baaaa}, \{\ldots\}_I^{babab}\}\\ \mathcal{C}_6 &:= \{\ldots, \{2_2, 1_2, 0_2, 0_0, 0_1\}_I^{aaabab}, \ldots\}\end{aligned}$$

Finally, since a set containing $\{0_0, 0_1, 0_2\}$ has been found, s is included in the next iteration, and the solution is the annotation of this witnessing set.

We will compare this new algorithm for SCS to existing ones and investigate its use in bio-informatics for example elsewhere.

References

1. Alur, R., Etessami, K., Madhusudan, P.: A temporal logic of nested calls and returns. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 467–481. Springer, Heidelberg (2004)
2. Axelsson, R., Lange, M., Somla, R.: The complexity of model checking higher-order fixpoint logic. In: Logical Methods in Computer Science (accepted for publication, 2007)
3. Emerson, E.A.: Uniform inevitability is tree automaton ineffable. Information Processing Letters 24(2), 77–79 (1987)
4. Emerson, E.A., Jutla, C.S.: Tree automata, μ-calculus and determinacy. In: Proc. 32nd Symp. on Foundations of Computer Science, San Juan, Puerto Rico, pp. 368–377. IEEE Computer Society Press, Los Alamitos (1991)
5. Jørgensen, N.: Finding fixpoints in finite function spaces using neededness analysis and chaotic iteration. In: LeCharlier, B. (ed.) SAS 1994. LNCS, vol. 864, pp. 329–345. Springer, Heidelberg (1994)
6. Müller-Olm, M.: A modal fixpoint logic with chop. In: Meinel, C., Tison, S. (eds.) STACS 99. LNCS, vol. 1563, pp. 510–520. Springer, Heidelberg (1999)
7. Pan, G., Sattler, U., Vardi, M.Y.: BDD-based decision procedures for the modal logic K. Journal of Applied Non-Classical Logics 16(1-2), 169–208 (2006)
8. Tabakov, D., Vardi, M.Y.: Experimental evaluation of classical automata constructions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 396–411. Springer, Heidelberg (2005)
9. Viswanathan, M., Viswanathan, R.: A higher order modal fixed point logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 512–528. Springer, Heidelberg (2004)
10. De Wulf, M., Doyen, L., Henzinger, T.A., Raskin, J.-F.: Antichains: A new algorithm for checking universality of finite automata. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 17–30. Springer, Heidelberg (2006)

Monadic Fragments of Gödel Logics: Decidability and Undecidability Results

Matthias Baaz, Agata Ciabattoni[*], and Christian G. Fermüller[**]

Technische Universität Wien, Vienna, Austria

Abstract. The monadic fragments of first-order Gödel logics are investigated. It is shown that all finite-valued monadic Gödel logics are decidable; whereas, with the possible exception of one ($\mathbf{G}_\uparrow$), all infinite-valued monadic Gödel logics are undecidable. For the missing case $\mathbf{G}_\uparrow$ the decidability of an important sub-case, that is well motivated also from an application oriented point of view, is proven. A tight bound for the cardinality of finite models that have to be checked to guarantee validity is extracted from the proof. Moreover, monadic $\mathbf{G}_\uparrow$, like all other infinite-valued logics, is shown to be undecidable if the projection operator $\triangle$ is added, while all finite-valued monadic Gödel logics remain decidable with $\triangle$.

1 Introduction

Many-valued logics have various applications in computer science (see, e.g., [10]). They are particularly useful for modeling reasoning with graded notions and vague information. In the latter context, the family of (finite- and infinite-valued) [illegible] appears as a prominent example. These are the only many-valued logics that are completely specified by the [illegible] of the underlying set of truth values. This fact characterizes Gödel logics as logics of comparative truth and renders them an important case of so-called [illegible] (see [11]).

Propositional finite-valued Gödel logics were introduced by Gödel [9] to show that intuitionistic logic does not have a characteristic finite matrix. They were generalized by Dummett [7] to an infinite set of truth values. First-order Gödel logic based on the closed unit interval $[0, 1]$ as set of truth values was introduced and axiomatized by Takeuti and Titani in [15] and called "intuitionistic fuzzy logic", there. In a more general view, the truth values for Gödel logics can be taken from any $V \subseteq [0, 1]$, that contains 0 and 1, and is closed under infima and suprema. (Gödel logic coincides with classical logic for $V = \{0, 1\}$.) In contrast to the propositional case, where there is only one infinite-valued Gödel logic with respect to validity, different sets V of truth values determine different [illegible] Gödel logics $\mathbf{G}_V$, in general. As shown in [4], $\mathbf{G}_V$ is recursively axiomatizable only when V is either finite or is order isomorphic to $[0, 1]$ or to $\{0\} \cup [\frac{1}{2}, 1]$.

---

[*] Supported by Austrian science foundation (FWF), project MAT P18731.

[**] Supported by Austrian science foundation (FWF), project MAT P18563-N12.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 77–91, 2007.
© Springer-Verlag Berlin Heidelberg 2007

We investigate [illegible], i.e. first-order $\mathbf{G}_V$ in which all predicate letters are unary (monadic). Many-valued monadic predicates can be interpreted as fuzzy sets and therefore many-valued monadic logics suffice to formalize the central concept of a [illegible], like: "IF $A(x)$ and $B(x)$ THEN $C(x)$", where the predicates A, B, and C are fuzzy, i.e., they apply to x possibly only to some degree.

We show that all finite-valued monadic Gödel logics are decidable, while for infinite sets V of truth values all monadic Gödel logics are undecidable, with the possible exception of monadic $\mathbf{G}_\uparrow$, where $V = \{1 - 1/n : n \geq 1\} \cup \{1\}$. The missing case, $\mathbf{G}_\uparrow$, is interesting, since it coincides with the intersection of all monadic finite-valued Gödel logics. Its decidability status remains open. However, we prove the decidability of an important sub-case, that we call the [illegible] of $\mathbf{G}_\uparrow$.

The untangled fragment of a logic consists of those (monadic) formulas in which each subformula contains at most one free variable. To appreciate the usefulness of this fragment, notice that its classical counterpart was used in [12] to formalize the knowledge base of the medical expert system CADIAG-1, represented as (classical) IF-THEN rules. This formalization made it possible to prove the decidability of the consistency checking problem in CADIAG-1 and led to a simple algorithm to actually carry out such checks.

Our decision procedure for the untangled fragment of $\mathbf{G}_\uparrow$ also provides a tight bound for the cardinality of finite models that have to be checked to guarantee validity. This bound implies a considerable gain in efficiency for the corresponding fragments of finite-valued Gödel logics (including classical logic). An elegant axiomatization for the untangled fragment of $\mathbf{G}_\uparrow$ can also be extracted from the decision procedure, contrasting the fact that $\mathbf{G}_\uparrow$ is not recursively axiomatizable [3,4].

We also investigate monadic Gödel logics extended with the projection operator $\triangle$, see [1]. This operator maps $\triangle P$ to the distinguished truth value 1 if the value of P equals 1, and to 0 otherwise, and thus allows to recover classical reasoning inside Gödel logics. The addition of $\triangle$ does not affect the decidability of the finite-valued logics, however [illegible] infinite-valued monadic Gödel logics, including $\mathbf{G}_\uparrow$, turn out to be undecidable in presence of $\triangle$, even when restricted to their prenex fragments.

2 Basic Facts About Gödel Logics

Kurt Gödel [9] has introduced the following truth functions for conjunction, disjunction, and implication:

$$\|A \wedge B\|_\mathcal{I} = \min(\|A\|_\mathcal{I}, \|B\|_\mathcal{I}), \qquad \|A \vee B\|_\mathcal{I} = \max(\|A\|_\mathcal{I}, \|B\|_\mathcal{I}),$$

$$\|A \to B\|_\mathcal{I} = \begin{cases} 1 & \text{if } \|A\|_\mathcal{I} \leq \|B\|_\mathcal{I} \\ \|B\|_\mathcal{I} & \text{otherwise.} \end{cases}$$

Formulas are evaluated over some set V of [illegible], where $\{0,1\} \subseteq V \subseteq [0,1]$. The propositional constant $\bot$ is semantically fixed by $\|\bot\|_\mathcal{I} = 0$. $\neg A$ abbreviates $A \to \bot$ and $A \leftrightarrow B$ abbreviates $(A \to B) \wedge (B \to A)$; therefore

$$\|\neg A\|_\mathcal{I} = \begin{cases} 1 & \text{if } \|A\|_\mathcal{I} = 0 \\ 0 & \text{otherwise} \end{cases} \quad \|A \leftrightarrow B\|_\mathcal{I} = \begin{cases} 1 & \text{if } \|A\|_\mathcal{I} = \|B\|_\mathcal{I} \\ \min(\|A\|_\mathcal{I}, \|B\|_\mathcal{I}) & \text{otherwise.} \end{cases}$$

Obviously, $\|.\|_\mathcal{I}$ extends every interpretation $\mathcal{I}$, that maps propositional variables into V, uniquely to arbitrary propositional formulas. $\mathcal{I}$ [illegible] a formula F and is called a [illegible] of F if $\|F\|_\mathcal{I} = 1$; F is [illegible] if all interpretations are models. We identify a [illegible] with its set of valid formulas.

Different choices of V in general induce different logics. The truth functions, above, imply that only the respective [illegible], but not the particular arithmetic values of the truth values are relevant for validity or satisfiability. If $|V| = n$ $(n \geq 2)$ the set of valid formulas is called the n-valued Gödel logic. Obviously, two-valued Gödel logic is classical logic. At the propositional level there is only one infinite-valued Gödel logic $\mathbf{G}_\infty$, which is also the intersection of all finite-valued Gödel logics. Dummett [7] has shown that $\mathbf{G}_\infty$ can be axiomatized by adding the [illegible]

$$(A \to B) \vee (B \to A) \tag{1}$$

to any Hilbert-style system for intuitionistic logic. Therefore $\mathbf{G}_\infty$ is sometimes also called Gödel-Dummett logic or Dummett's **LC**. More recently $\mathbf{G}_\infty$ emerged as one of the main formalizations of [illegible] (see, e.g., [11]). In this context it is very useful to enrich the logics by adding the unary operator $\triangle$ with the following meaning [1]:

$$\|\triangle A\|_\mathcal{I} = \begin{cases} 1 & \text{if } \|A\|_\mathcal{I} = 1 \\ 0 & \text{otherwise.} \end{cases}$$

The situation for infinite sets of truth values gets more interesting at the first-order level. We introduce predicates and quantifiers as follows. Instead of being propositional variables, atomic formulas are now of the form $P(t_1, \ldots, t_n)$, where P is a predicate symbol and $t_1, \ldots, t_n$ are terms, where a term, here, is either an (object) variable or a constant symbol. An interpretation $\mathcal{I}$ consists of a non-empty [illegible] D and a [illegible] $v_\mathcal{I}$ that maps constant symbols and object variables to elements of D. Moreover, $v_\mathcal{I}$ maps every n-ary predicate symbol P to a function from D^n into V. The truth value of an atomic formula $P(t_1, \ldots, t_n)$ is thus defined as

$$\|P(t_1, \ldots, t_n)\|_\mathcal{I} = v_\mathcal{I}(P)(v_\mathcal{I}(t_1), \ldots, v_\mathcal{I}(t_n)).$$

To fix the meaning of quantifiers we define the [illegible] of a formula A with respect to a free variable x in an interpretation $\mathcal{I}$ as $\mathrm{distr}_\mathcal{I}(A(x)) = \{\|A(x)\|_{\mathcal{I}'} \mid \mathcal{I}' \sim_x \mathcal{I}\}$, where $\mathcal{I}' \sim_x \mathcal{I}$ means that $\mathcal{I}'$ is exactly as $\mathcal{I}$ with the possible exception of the domain element assigned to x. The quantifiers correspond to the infimum and supremum, respectively, in the following sense:

$$\|(\forall x)A(x)\|_{\mathcal{I}} = \inf \mathrm{distr}_{\mathcal{I}}(A(x)) \qquad \|(\exists x)A(x)\|_{\mathcal{I}} = \sup \mathrm{distr}_{\mathcal{I}}(A(x)).$$

Note that the above definition of an interpretation as a pair $(D, v_{\mathcal{I}})$ covers also classical logic. However, to enhance clarity, we will use (in Sect. 4, below) $\bot$ and $\top$ instead of 0 and 1, respectively, for the classical truth values.

In the following we investigate (fragments of) first-order Gödel logics, with and without the operator $\triangle$. Every truth value set V, $\{0,1\} \subseteq V \subseteq [0,1]$, that is closed under suprema and infima induces a first-order logic $\mathbf{G}_V$ over the language without $\triangle$ and a logic $\mathbf{G}_V^{\triangle}$ if $\triangle$ is present. [illegible] is $\mathbf{G}_{[0,1]}$; i.e., the logic over the full real unit interval as truth value set, see, e.g., [11,15]. We use $\mathbf{G}_n$ to denote the n-valued first-order Gödel logic for $n \geq 2$. $\mathbf{G}_{\uparrow}$ results from taking $V = \{1\} \cup \{1 - \frac{1}{k} \mid k \geq 1\}$ (or any other order isomorphic truth value set); $\mathbf{G}_{\downarrow}$ arises from $V = \{0\} \cup \{\frac{1}{k} \mid k \geq 1\}$.

Like in intuitionistic logic, also in Gödel logics (with or without $\triangle$) quantifiers cannot be shifted arbitrarily. In other words, arbitrary formulas are not equivalent to prenex formulas, in general. However, we have the following (stated in [4] without proof):

Proposition 1. [illegible] x [illegible] B [illegible] Q [illegible] $\exists$ [illegible] $\forall$ [illegible]

$$(\mathsf{Q}x)(A \wedge B) \leftrightarrow ((\mathsf{Q}x)A \wedge B) \tag{2}$$

$$(\mathsf{Q}x)(A \vee B) \leftrightarrow ((\mathsf{Q}x)A \vee B) \tag{3}$$

$$(\exists x)(A \rightarrow B) \rightarrow ((\forall x)A \rightarrow B) \tag{4}$$

$$(\exists x)(B \rightarrow A) \rightarrow (B \rightarrow (\exists x)A) \tag{5}$$

$$(\forall x)(A \rightarrow B) \leftrightarrow ((\exists x)A \rightarrow B) \tag{6}$$

$$(\forall x)(B \rightarrow A) \leftrightarrow (B \rightarrow (\forall x)A) \tag{7}$$

[illegible] Given the truth functions for quantifiers, presented above, it suffices to note that, for all sets of reals A and all reals b the following statements hold.

- Corresponding to (2): $\inf\{\min(a,b) \mid a \in A\} = \min(\inf A, b)$ and $\sup\{\min(a,b) \mid a \in A\} = \min(\sup A, b)$.
- Corresponding to (3): $\inf\{\max(a,b) \mid a \in A\} = \max(\inf A, b)$ and $\sup\{\max(a,b) \mid a \in A\} = \max(\sup A, b)$.
- Corresponding to (4): If $a \leq b$ for some $a \in A$, then $\inf A \leq b$.
- Corresponding to (5): If $b \leq a$ for some $a \in A$, then $b \leq \sup A$.
- Corresponding to (6): $a \leq b$ for all $a \in A$ iff $\sup A \leq b$.
- Corresponding to (7): $b \leq a$ for all $a \in A$ iff $b \leq \inf A$.

(In fact almost all of these schemes are already intuitionistically valid.) $\square$

Note that the schemes that are dual to (4) and (5) are not valid in general (but are valid in $\mathbf{G}_\uparrow$ and $\mathbf{G}_n$, $n \geq 2$; see Proposition 3). Counterexamples are readily obtained for standard Gödel logic $\mathbf{G}_{[0,1]}$.

To emphasize that different sets of valid formulas result from different V, in general, consider the following formula schemes:

$$(\exists x)(A(x) \to (\forall x)A(x)) \tag{8}$$

$$(\exists x)((\exists y)A(y) \to A(x)) \tag{9}$$

Any instance of (8) is satisfied in an interpretation $\mathcal{I}$ if and only if the infimum of $\mathrm{distr}_{\mathcal{I}}(A(x))$ is a minimum, i.e., an element of $\mathrm{distr}_{\mathcal{I}}(A(x))$. Therefore (8) is valid in $\mathbf{G}_\uparrow$ and in any $\mathbf{G}_n$, but not, e.g., in $\mathbf{G}_{[0,1]}$ or in $\mathbf{G}_\downarrow$. Similarly (9) expresses that every supremum of a distribution is a maximum, with the possible exception of the value 1. Therefore (9) is valid in $\mathbf{G}_\downarrow$, in $\mathbf{G}_\uparrow$, and in all $\mathbf{G}_n$ for $n \geq 2$, but not, e.g., in $\mathbf{G}_{[0,1]}$. In fact there are infinitely many different infinite-valued first-order Gödel logics, according to [4]. The conjecture that there are just countable many different Gödel logics has recently been settled in [5]. $\mathbf{G}_{[0,1]}$ and $\mathbf{G}^{\triangle}_{[0,1]}$ are well known to be recursively axiomatizable, see, e.g., [11]. In contrast, $\mathbf{G}_\uparrow$ and $\mathbf{G}_\downarrow$ are not recursively axiomatizable, see [3,4].

The fact that $\mathbf{G}_\uparrow = \bigcap_{n \geq 2} \mathbf{G}_n$ also holds at the first-order level, see [4]. However, this is no longer the case if we add the projection operator $\triangle$. In the enriched language, the intersection of all finite-valued Gödel logics is not a Gödel logic:

Proposition 2. $\mathbf{G}^{\triangle}_V \neq \bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ *for any* V.

Proof. Since $\mathbf{G}^{\triangle}_n$ is a proper subset of $\mathbf{G}^{\triangle}_m$ whenever $n > m$, $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ cannot coincide with any finite-valued Gödel logic. To show that $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ also cannot be an infinite-valued Gödel logic, consider the formula

$$\triangle(\exists x)A(x) \to (\exists x)\triangle A(x). \tag{10}$$

It is valid in all finite-valued Gödel logics and therefore also in $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$. But not every interpretation $\mathcal{I}$ for $\mathbf{G}^{\triangle}_\uparrow$ satisfies all instances of (10). Take, e.g., the positive integers as domain of $\mathcal{I}$ and let $v_{\mathcal{I}}(P)(n) = 1 - \frac{1}{n}$ for some predicate symbol P. We obtain $\|(\exists x)P(x)\|_{\mathcal{I}} = \|\triangle(\exists x)P(x)\|_{\mathcal{I}} = 1$ and $\|\triangle P(x)\|_{\mathcal{I}} = \|(\exists x)\triangle P(x)\|_{\mathcal{I}} = 0$. Consequently, $\mathbf{G}^{\triangle}_\uparrow \neq \bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$. On the other hand, $\mathbf{G}_\uparrow = \bigcap_{n \geq 2} \mathbf{G}_n \subset \bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ and therefore all instances of schemes (8) and (9) are in $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$. As noted above, this implies that in all interpretations of $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ every infimum of a distribution is a minimum and every supremum of a distribution is either 1 or a maximum. In other words: if $\bigcap_{n \geq 2} \mathbf{G}^{\triangle}_n$ were identical with some $\mathbf{G}_V$, then its set V of truth values could not contain any accumulation point except 1. But all infinite subsets of $[0,1]$ containing 0 and 1, that satisfy this property are order isomorphic to $\{1\} \cup \{1 - \frac{1}{k} \mid k \geq 1\}$, which is the case that we have excluded above. $\square$

Remark. Note that we only had to refer to a unary predicate symbol in the above proof. I.e., Proposition 2 holds already for the monadic fragments.

3 Decidability of All Finite-Valued Monadic Gödel Logics

From now on, we will restrict our attention to [illegible] Gödel logics, i.e., all predicate symbols are unary. $\mathbf{G}_2$ is classical logic and therefore, as is well known, monadic $\mathbf{G}_2$ is decidable, whereas already a single binary predicate symbol leads to undecidability. It is straightforward to generalize this classic result to all finite-valued logics.

Theorem 1. [illegible] $\mathbf{G}_n^{\triangle}$ [illegible] $n \geq 2$

[illegible] Let A be any monadic formula that is not valid in $\mathbf{G}_n^{\triangle}$. Hence, there exists an interpretation $\mathcal{I}$ based on the set of truth values $V = \{\frac{j}{n-1} \mid 0 \leq j \leq n-1\}$ such that $\|A\|_{\mathcal{I}} < 1$. Let $\{P_1, \ldots, P_k\}$ be the set of different predicate symbols occurring in A. $\mathcal{I}$ induces the following equivalence relation $\equiv_{\mathcal{I}}$ on the domain D of $\mathcal{I}$:

$$c \equiv_{\mathcal{I}} d \Longleftrightarrow_{df} v_{\mathcal{I}}(P_i)(c) = v_{\mathcal{I}}(P_i)(d) \text{ for all } i \in \{1, \ldots, k\}.$$

Note that $c \equiv_{\mathcal{I}} d$ expresses that the domain elements c and d are indistinguishable with respect to the interpretation $\mathcal{I}$. Let $[c]_{\mathcal{I}}$ denote the equivalence class of the element $c \in D$, induced by $\equiv_{\mathcal{I}}$. We define a new interpretation $\mathcal{I}'$ with domain $D' = \{[c]_{\mathcal{I}} \mid c \in D\}$. D' is finite, since according to the definition of $\equiv_{\mathcal{I}}$ there can be at most n^k elements that are pairwise inequivalent. Let $v_{\mathcal{I}'}(P_i)([c]_{\mathcal{I}}) = v_{\mathcal{I}}(P_i)(c)$ for $i \in \{1, \ldots, k\}$. It is straightforward to check that $\mathcal{I}'$ is well-defined and that $\|A\|_{\mathcal{I}'} = \|A\|_{\mathcal{I}}$. This means that A is valid in $\mathbf{G}_n^{\triangle}$ iff it is satisfied in all interpretations with domain $\{1, \ldots, n^k\}$. Since there are at most $n^{k \cdot n^k}$ different such interpretations, and since evaluation of formulas over finite domains is computable, we have proved the decidability of $\mathbf{G}_n^{\triangle}$. □

[illegible] Clearly, the 'filtration argument' of the above proof applies to the monadic fragments of arbitrary finite-valued logics, not just of Gödel logics. (In fact, the proof is probably 'folklore'. To render the paper self contained, and since there seems to be no appropriate reference in the literature, we decided to include it here.)

[illegible] It is well known that the bound n^k for the cardinality of relevant model domains is optimal in the case $n = 2$, i.e., for classical logic. Better bounds might be achievable in general; however all such bounds seem to depend on the number of truth values n and are exponential in the number of different predicate symbols k. We show in Sect. 5 that much better bounds can be achieved for an interesting, non-trivial sub-case of the monadic fragments.

4 Undecidability of Infinite-Valued Gödel Logics

We prove the undecidability of each Gödel logic $\mathbf{G}_V$, where the set V of truth values contains infinitely many values below some value that is distinct from 1. Our proof adapts and generalizes the undecidability proof sketched in [8] for monadic '**LC** with constant domains', which coincides with monadic $\mathbf{G}_{[0,1]}$. With the notable exception of $\mathbf{G}_{\uparrow}$, all infinite-valued Gödel logics satisfy the above condition on V, see Corollary 1.

We will also consider infinite-valued Gödel logics extended with the projection operator $\triangle$. Monadic prenex $\mathbf{G}^{\triangle}_{[0,1]}$ was shown to be undecidable in [2]. This result is generalized below, where we show that in fact for [illegible] infinite V, monadic $\mathbf{G}^{\triangle}_{V}$ is undecidable, even when restricted to prenex formulas.

Theorem 2. [illegible] $\mathbf{G}_V$ [illegible] V [illegible] $\exists p \in V, p < 1$ [illegible] $V_p = \{y \in V \mid y \leq p\}$ [illegible] $\mathbf{G}_V$ [illegible]

[illegible] The classical theory **CE** of two equivalence relations $\equiv_1$ and $\equiv_2$ was shown to be undecidable in [14]. Let $\mathbf{G}_V$ be any Gödel logic, where V satisfies the condition: $\exists p \in V, p < 1$, such that $V_p = \{y \in V \mid y \leq p\}$ is infinite. We faithfully interpret **CE** in the monadic fragment of $\mathbf{G}_V$. The idea is to translate formulas of the form $x \equiv_i y$ into formulas $P_i(x) \leftrightarrow P_i(y)$, $i = 1, 2$, of the monadic fragment of $\mathbf{G}_V$, where P_1 and P_2 are different unary predicate symbols. Without loss of generality, we can assume formulas in **CE** to be in prenex normal form. Let S be the following formula of this kind:

$$\mathsf{Q}^* \bigwedge (\bigwedge_j x_j \equiv y_j \rightarrow \bigvee_k u_k \equiv v_k),$$

where each occurrence of $\equiv$ is either $\equiv_1$ or $\equiv_2$, and where Q^* is a string $(\mathsf{Q}_1 \mathsf{z}_1) \ldots (\mathsf{Q}_n \mathsf{z}_n)$ of n quantifier occurrences. I.e., for all $i = 1, \ldots, n$, $\mathsf{Q}_i \in \{\forall, \exists\}$, and z_i denotes some variable. Let $S^\sharp$ be the following monadic formula:

$$\mathsf{Q}^* \bigwedge (\bigwedge_j (P(x_j) \leftrightarrow P(y_j)) \rightarrow [(\bigvee_k P(u_k) \leftrightarrow P(v_k)) \vee (\exists x) P_1(x) \vee (\exists x) P_2(x)]),$$

where P is P_1 or P_2, according to whether $\equiv$ is $\equiv_1$ or $\equiv_2$. We show that S is valid in **CE** if and only if $S^\sharp$ is valid in $\mathbf{G}_V$.

Let $\mathcal{M} = (D, v_{\mathcal{M}})$ be an interpretation of **CE**. By the Löwenheim-Skolem theorem we can assume D to be countable without loss of generality. We define a corresponding interpretation $\mathcal{I}(\mathcal{M}) = (D, v_{\mathcal{I}(\mathcal{M})})$ of $\mathbf{G}_{\uparrow}$ as follows. We set $v_{\mathcal{I}(\mathcal{M})}(z) = v_{\mathcal{M}}(z)$ for all variables z. (It suffices to work in a language without constant symbols.) Let us use $\equiv_i^{\mathcal{M}}$ to denote the equivalence relation $v_{\mathcal{M}}(\equiv_i)$. Note that $\equiv_i^{\mathcal{M}}$ induces a partition of the domain D into equivalence classes $E_i^c = \{d \mid d \equiv_i^{\mathcal{M}} c\}$, where $c \in D$ ($i \in \{1, 2\}$). Since $V_p = \{y \in V \mid y \leq p\}$ is infinite and D is countable, we can take some subset $W = \{w_0, w_1, \ldots\}$ of V_p as the set of (unique) indices in an enumeration $E_i^{w_0}, E_i^{w_1}, \ldots$ without repetitions of all such equivalence classes. (This enumeration is assumed to be the same for all interpretations that only differ in their variable assignments.) Referring to this enumeration of equivalence classes, we can define $v_{\mathcal{I}(\mathcal{M})}$ by

$$v_{\mathcal{I}(\mathcal{M})}(P_i)(d) = w_k \text{ if and only if } d \in E_i^{w_k},$$

where $e, d \in D$ and $i = 1, 2$.

Moreover, for each interpretation $\mathcal{I} = (D, v_{\mathcal{I}})$ of $\mathbf{G}_V$ we define the interpretation $\mathcal{M}(\mathcal{I}) = (D, v_{\mathcal{M}(\mathcal{I})})$ of **CE** by

$$v_{\mathcal{M}(\mathcal{I})}(\equiv_i)(d, e) = \top \text{ if and only if } v_{\mathcal{I}}(P_i)(d) = v_{\mathcal{I}}(P_i)(e).$$

for all $d, e \in D$ and $i = 1, 2$.

We prove the following claims about $\mathcal{I}(\mathcal{M})$ and $\mathcal{M}(\mathcal{I})$ by induction on the number n of quantifier occurrences in S and $S^\sharp$.

($\Rightarrow$) For every interpretation $\mathcal{M} = (D, v_{\mathcal{M}})$ of **CE**, where $\|S\|_{\mathcal{M}} = \bot$, we have $\|S^\sharp\|_{\mathcal{I}(\mathcal{M})} \leq p$.

($\Leftarrow$) For every interpretation $\mathcal{I}$ of $\mathbf{G}_V$, where $\|S^\sharp\|_{\mathcal{I}} < 1$, we have $\|S\|_{\mathcal{M}(\mathcal{I})} = \bot$.

[illegible] $n = 0$ (i.e., there are no quantifiers).
($\Rightarrow$) Let $\|S\|_{\mathcal{M}} = \bot$ for some interpretation $\mathcal{M} = (D, v_{\mathcal{M}})$ of **CE**. By definition of $\mathcal{I}(\mathcal{M})$, we have $\|P_i(x) \leftrightarrow P_i(y)\|_{\mathcal{I}(\mathcal{M})} = 1$ if and only if $\|x \equiv_i y\|_{\mathcal{M}} = \top$. The exhibited conjunct of S is evaluated to $\bot$ in $\mathcal{M}$ if and only if $\| \bigwedge_j x_j \equiv y_j\|_{\mathcal{M}} = \top$ and $\| \bigvee_k u_k \equiv v_k\|_{\mathcal{M}} = \bot$. This, in turn, implies $\| \bigwedge_j P(x_j) \leftrightarrow P(y_j)\|_{\mathcal{I}(\mathcal{M})} = 1$ and $\| \bigvee_k P(u_k) \leftrightarrow P(v_k)\|_{\mathcal{I}(\mathcal{M})} = \max_k \min(\|P(u_k)\|_{\mathcal{I}(\mathcal{M})}, \|P(v_k)\|_{\mathcal{I}(\mathcal{M})}) \leq p$. Since $\|(\exists x)P_1(x) \vee (\exists x)P_2(x)\|_{\mathcal{I}(\mathcal{M})} \leq \sup(V_p) = p$, we obtain $\|S^\sharp\|_{\mathcal{I}(\mathcal{M})} \leq p$.

($\Leftarrow$) Let $\mathcal{I}$ be an interpretation of $\mathbf{G}_V$, such that $\|S^\sharp\|_{\mathcal{I}} < 1$. Then, for some conjunct of $S^\sharp$ (which without loss of generality we identify with the exhibited one) we have

$$\|(\bigvee_k P(u_k) \leftrightarrow P(v_k)) \vee (\exists x)P_1(x) \vee (\exists x)P_2(x)\|_{\mathcal{I}} < \|\bigwedge_j P(x_j) \leftrightarrow P(y_j)\|_{\mathcal{I}}.$$

This implies $\|\bigwedge_j P(x_j) \leftrightarrow P(y_j)\|_{\mathcal{I}} = 1$, since $\|\bigwedge_j P(x_j) \leftrightarrow P(y_j)\|_{\mathcal{I}}$ is either 1 or not greater than $\sup\{v_{\mathcal{I}}(P_i)(d) \mid d \in D, i = 1, 2\} = \|(\exists x)P_1(x) \vee (\exists x)P_2(x)\|_{\mathcal{I}}$. By the definition of $\mathcal{M}(\mathcal{I})$ we have for all variables z, z': $\|z \equiv_i z'\|_{\mathcal{M}(\mathcal{I})} = \top$ if and only if $\|P_i(z) \leftrightarrow P_i(z')\|_{\mathcal{I}} = 1$ $(i = 1, 2)$. Therefore $\| \bigwedge_j x_j \equiv y_j\|_{\mathcal{M}(\mathcal{I})} = \top$ and $\| \bigvee_k u_k \equiv v_k\|_{\mathcal{M}(\mathcal{I})} = \bot$. Hence $\|S\|_{\mathcal{M}(\mathcal{I})} = \bot$.

[illegible] Assuming that the claims hold for S and $S^\sharp$, we have to show that they also hold for $S_1 = (\mathsf{Q}x)S$ and $S_1^\sharp = (\mathsf{Q}x)S^\sharp$, where $\mathsf{Q} \in \{\exists, \forall\}$ and x denotes any variable.

Let S_1 be $(\exists x)S$. ($\Rightarrow$) If $\|(\exists x)S\|_{\mathcal{M}} = \bot$ then $\|S\|_{\mathcal{M}[^d/_x]} = \bot$ for all $d \in D$, where $\mathcal{M}[^d/_x]$ denotes an interpretation that is like $\mathcal{M}$, except for assigning the domain element d to the variable x. By the induction hypothesis we have $\|S^\sharp\|_{\mathcal{I}(\mathcal{M}[^d/x])} \leq p$, where $\mathcal{I}(\mathcal{M}[^d/x])$ is the interpretation of $\mathbf{G}_V$ corresponding to $\mathcal{M}[^d/_x]$. By definition, the interpretations $\mathcal{I}(\mathcal{M}[^d/x])$ are identical for all $d \in D$, except for the element assigned to x, since we required the underlying enumeration of equivalence classes to be the same for all $\mathcal{I}(\mathcal{M}[^d/x])$. We thus obtain $\sup_{d \in D}(\|S^\sharp\|_{\mathcal{I}(\mathcal{M}[^d/x])}) = \|(\exists x)S^\sharp\|_{\mathcal{I}(\mathcal{M})} = \|S_1^\sharp\|_{\mathcal{I}(\mathcal{M})} \leq p$, as required.

Similarly for ($\Leftarrow$): If $\|S_1^\sharp\|_{\mathcal{I}} = \|(\exists x)S^\sharp\|_{\mathcal{I}} < 1$ then $\|S^\sharp\|_{\mathcal{I}[^d/x]} < 1$ for all $d \in D$, where $\mathcal{I}[^d/x]$ denotes an interpretation for $\mathbf{G}_V$ that is like $\mathcal{I}$, except for assigning the domain element d to the variable x. By the induction hypothesis we have $\|S\|_{\mathcal{M}(\mathcal{I}[^d/x])} = \bot$, where the interpretations $\mathcal{M}(\mathcal{I}[^d/x])$ are identical for all $d \in D$, except for the element assigned to x. We thus obtain $\|(\exists x)S\|_{\mathcal{M}(\mathcal{I})} = \bot$ as required.

The case $S_1 = (\forall x)S$ is analogous. □

Corollary 1. [illegible] $\mathbf{G}_\uparrow$ [illegible]

[illegible] Let V be any infinite set of reals, such that $\{0,1\} \subseteq V \subseteq [0,1]$. Suppose V does not satisfy the condition of Theorem 2. Then V contains only finitely many different elements below any given $p < 1$ for $p \in V$. It is not difficult to see that all such V are order isomorphic to $\{1\} \cup \{1 - \frac{1}{n} \mid n \geq 1\}$; i.e., to the set of truth values of $\mathbf{G}_\uparrow$. □

Theorem 2 can be strengthened as follows, if we augment the language of our logics by the projection operator $\triangle$.

Theorem 3. [illegible] $\mathbf{G}_V^\triangle$ [illegible] V [illegible]

[illegible] Similarly to the proof of Theorem 2, above, we translate classical formulas of the form $x \equiv_i y$ into formulas $\triangle(P_i(x) \leftrightarrow P_i(y))$ $(i = 1, 2)$. More exactly, let S be a formula of **CE**, like in the proof of Theorem 2. Let the corresponding formula $S^\sharp_\triangle$, to be interpreted in $\mathbf{G}_V^\triangle$, be

$$\mathsf{Q}^* \bigwedge (\bigwedge_j \triangle(P(x_j) \leftrightarrow P_i(y_j)) \rightarrow \bigvee_k \triangle(P(u_k) \leftrightarrow P_i(v_k)).$$

The proof that S is valid in **CE** if and only if $S^\sharp_\triangle$ is valid in $\mathbf{G}_V^\triangle$ is analogous to that of the corresponding claim in Theorem 2. However, in defining $\mathcal{I}(\mathcal{M})$ we may now take [illegible] subset of (the infinite set) V as the set of indices in the underlying enumeration of equivalence classes $E_i^{w_k}$. The reason for this is that, in any interpretation $\mathcal{I}$, $\|\triangle(P(x) \leftrightarrow P(y))\|_\mathcal{I} = 0$ if $v_\mathcal{I}(P)(v_\mathcal{I}(x)) \neq v_\mathcal{I}(P)(v_\mathcal{I}(y))$, and $\|\triangle(P(x) \leftrightarrow P(y))\|_\mathcal{I} = 1$ otherwise. Hence $S^\sharp_\triangle$ itself behaves like a classical formula, i.e., it always evaluates either to 0 or to 1 Consequently, it suffices that V is infinite to be able to encode different equivalence classes by different truth values in the required way.

Finally note that, in contrast to Theorem 2, $S^\sharp_\triangle$ is a prenex formula. □

5 Efficient Decidability of Untangled $\mathbf{G}_\uparrow$ and $\mathbf{G}_n$

As mentioned in the introduction, application oriented investigations draw our attention to monadic formulas that exhibit a restricted form of overlap between scopes of different quantifier occurrences. We propose to view quantifier scopes as being [illegible] in general, but [illegible] in the following case.

Definition 1. [illegible] F [illegible] untangled [illegible] F [illegible]

[illegible] $(\exists y)((\forall x)P(x) \to Q(y))$ and $(\forall y)((\exists z)((\forall x)P(x) \vee Q(z)) \to P(y))$ are untangled, but $(\exists y)(\forall x)(P(x) \to Q(y))$ and $(\exists x)(\exists y)(P(x) \wedge P(y))$ are not untangled.

The monadic fragment of classical logic was used in [12] to formalize the knowledge base of the medical expert system CADIAG-1, represented as (classical) IF-THEN rules. This formalization made it possible to prove the decidability of the consistency checking problem in CADIAG-1 and led to a simple algorithm to actually carry out such checks. An inspection of this application reveals that in fact only the untangled fragment of classical logic is needed for this purpose. In a many-valued context unary predicates are interpreted as [illegible]. This allows to formalize [illegible] IF-THEN rules in the untangled fragments of many-valued logics (including Gödel logics). Therefore (efficient) decision procedures for these fragments are of particular interest for fuzzy expert systems.

Remember from Proposition 1 that most quantifier shift laws are valid in all Gödel logics. For the decidability proof, below, we have to apply also the two remaining quantifier shift laws, that are not valid, e.g., in $\mathbf{G}_{[0,1]}$, but are valid in $\mathbf{G}_\uparrow$ and in $\mathbf{G}_n$.

Proposition 3. [illegible] x [illegible] B [illegible] $\mathbf{G}_\uparrow$ [illegible] $\mathbf{G}_n$ [illegible] $n \geq 2$[illegible]

$$((\forall x)A(x) \to B) \to (\exists x)(A(x) \to B) \tag{11}$$

$$(B \to (\exists x)A(x)) \to (\exists x)(B \to A(x)) \tag{12}$$

[illegible] The underlying truth value set of $\mathbf{G}_\uparrow$ is $V = \{1\} \cup \{1 - \frac{1}{k} \mid k \geq 1\}$. Therefore, for every formula $A(x)$ and every interpretation $\mathcal{I}$ of $\mathbf{G}_\uparrow$ there exists an element e in the domain of $\mathcal{I}$ such that $\|A(x)\|_{\mathcal{I}[e/x]} = \inf \mathrm{distr}_{\mathcal{I}}(A(x))$, where $\mathcal{I}[e/x]$ is like $\mathcal{I}$ except (possibly) for assigning e to the variable x. Similarly, for every formula $A(x)$ and every interpretation $\mathcal{I}$ either $\|(\exists x)A(x)\|_{\mathcal{I}} = 1$ or there exists an element e in the domain of $\mathcal{I}$ such that $\|A(x)\|_{\mathcal{I}[e/x]} = \sup \mathrm{distr}_{\mathcal{I}}(A(x))$. The validity of (11) and (12) follows directly from these observations. ☐

Definition 2. [illegible] contexts [illegible] $\neg$ [illegible] $\leftrightarrow$ [illegible]

– [illegible] $[\cdot]$ [illegible]
– [illegible] C [illegible] F [illegible] $(C \vee F)$ $(F \vee C)$ $(C \wedge F)$ $(F \wedge C)$ $(F \to C)$ $(\forall x)C$ [illegible] $(\exists x)C$ [illegible] $(C \to F)$ [illegible]
– [illegible] C [illegible] F [illegible] $(C \vee F)$ $(F \vee C)$ $(C \wedge F)$ $(F \wedge C)$ $(F \to C)$ $(\forall x)C$ [illegible] $(\exists x)C$ [illegible] $(C \to F)$ [illegible]

[illegible] $[\cdot]$ [illegible] C [illegible] A [illegible] $C[A]$ [illegible] $C[A]^+$ [illegible] occurrence [illegible] A [illegible] $C[A]$ [illegible] positive [illegible] C [illegible] $C[A]^-$ [illegible] negative occurrence [illegible] A [illegible] $C[A]$

[illegible] $C[(\exists x)A]^+$ [illegible] $C[(\forall x)A]^-$ [illegible] weak [illegible] $C[(\exists x)A]^-$ [illegible] $C[(\forall x)A]^+$ [illegible] strong

Proposition 4. [illegible] $\triangle$ [illegible]

- [illegible] $A \to B$ [illegible] $C[A]^+ \to C[B]^+$ [illegible]
- [illegible] $A \to B$ [illegible] $C[B]^- \to C[A]^-$ [illegible]

[illegible] By induction on the complexity of C.

The base case, where C is the empty context (and therefore positive) trivially holds.

We spell out two of the twelve different propositional cases of the induction step. The validity of $A \to B$ implies $\|A\|_{\mathcal{I}} \leq \|B\|_{\mathcal{I}}$ for every interpretation $\mathcal{I}$. By the induction hypothesis we have $\|C[A]^+\|_{\mathcal{I}} \leq \|C[B]^+\|_{\mathcal{I}}$. Thus, $\|F \wedge C[A]^+\|_{\mathcal{I}} = \min(\|F\|_{\mathcal{I}}, \|C[A]^+\|_{\mathcal{I}}) \leq \min(\|F\|_{\mathcal{I}}, \|C[B]^+\|_{\mathcal{I}}) = \|F \wedge C[B]^+\|_{\mathcal{I}}$. I.e., $(F \wedge C[A]^+) \to (F \wedge C[B]^+)$ is valid.

For a context of the form $C[\cdot]^+ \to F$ we have to show that $(C[B]^+ \to F) \to (C[A]^+ \to F)$ is valid if $\|A\|_{\mathcal{I}} \leq \|B\|_{\mathcal{I}}$ for all $\mathcal{I}$. (Remember that the occurrence of A is [illegible] in $C^+[A] \to F$.) We distinguish two cases.

(a) $\|C[B]^+\|_{\mathcal{I}} \leq \|F\|_{\mathcal{I}}$: By the induction hypothesis $\|C[A]^+\|_{\mathcal{I}} \leq \|C[B]^+\|_{\mathcal{I}}$. Therefore also $\|C[A]^+\|_{\mathcal{I}} \leq \|F\|_{\mathcal{I}}$ and consequently $\|C[A]^+ \to F\|_{\mathcal{I}} = 1$, which implies that $\|(C[B]^+ \to F) \to (C[A]^+ \to F)\|_{\mathcal{I}} = 1$.

(b) $\|C[B]^+\|_{\mathcal{I}} > \|F\|_{\mathcal{I}}$: this implies $\|C[B]^+ \to F\|_{\mathcal{I}} = \|F\|_{\mathcal{I}}$. If $\|C[A]^+\|_{\mathcal{I}} \leq \|F\|_{\mathcal{I}}$ then $\|C[A]^+ \to F\|_{\mathcal{I}} = 1$. Otherwise $\|C[A]^+\|_{\mathcal{I}} > \|F\|_{\mathcal{I}}$ and consequently also $\|C[A]^+ \to F\|_{\mathcal{I}} = \|F\|_{\mathcal{I}}$. In both cases $\|C[B]^+ \to F\|_{\mathcal{I}} \leq \|C[A]^+ \to F\|_{\mathcal{I}}$ and therefore, again, $\|(C[B]^+ \to F) \to (C[A]^+ \to F)\|_{\mathcal{I}} = 1$.

All other propositional cases are similar. The quantifier cases are straightforward, too. We just present the case for $(\forall x)C[\cdot]^-$. (The other cases are similar.) Assume that $A \to B$ is valid, i.e., $\|A\|_{\mathcal{I}} \leq \|B\|_{\mathcal{I}}$ for all interpretations $\mathcal{I}$. By the induction hypothesis $\|C[B]^-\|_{\mathcal{I}} \leq \|C[A]^-\|_{\mathcal{I}}$ for all $\mathcal{I}$. But this implies that $\inf \mathrm{distr}_{\mathcal{I}}(C[B(x)]^-) \leq \inf \mathrm{distr}_{\mathcal{I}}(C[A(x)]^-)$ and therefore also $\|(\forall x)C[B]^-\|_{\mathcal{I}} \leq \|(\forall x)C[A]^-\|_{\mathcal{I}}$ for all $\mathcal{I}$; i.e., $(\forall x)C[B]^- \to (\forall x)C[A]^-$ is valid, too. ☐

Theorem 4. [illegible] $\mathbf{G}_\uparrow$ [illegible]

[illegible] We first prove that untangled formulas in $\mathbf{G}_\uparrow$ remain valid if all strong quantifier occurrences are replaced by new constant symbols. To this aim it suffices to show that

- $C[(\forall x)A(x)]^+$ is valid if and only if $C[A(d)]^+$ is valid, and
- $C[(\exists x)A(x)]^-$ is valid if and only if $C[A(d)]^-$ is valid,

where d is a constant that does not occur in $C[A]$.

The 'only if' part of these claims follows directly from Proposition 4 and the validity of $(\forall x)A(x) \to A(d)$ and of $A(d) \to (\exists x)A(x)$, respectively.

For the 'if' part note that the validity of $F(d)$ implies the validity of $(\forall x)F(x)$ if d does not occur in F: any interpretation $\mathcal{I}$, where $\|(\forall x)F(x)\|_{\mathcal{I}} < 1$, can be extended to include the new d in such a way that $\|(\forall x)F(x)\|_{\mathcal{I}} = \|F(d)\|_{\mathcal{I}}$. Therefore the claims follow, if the following schemes are valid:

- $(\forall x)C[A(x)]^+ \rightarrow C[(\forall x)A(x)]^+$ and
- $(\forall x)C[A(x)]^- \rightarrow C[(\exists x)A(x)]^-$,

where the only free occurrences of x in C are in $A(x)$ and $A(x)$ is not in the scope of any quantifier occurrence in $C[A(x)]$. The validity of these schemes is obtained by repeatedly applying the quantifier shift laws of Propositions 1 and 3 in combination with the context rules of Proposition 4. (This is possible only because the formulas are untangled; see Remark 4, below.)

So far, we have shown that every untangled formula F can be transformed (in linear time) into a formula F' that only contains weak quantifier occurrences, but is equivalent to F with respect to validity in $\mathbf{G}_\uparrow$ (and $\mathbf{G}_n$, see Remark 5, below). Let us call such a formula [illegible]. To obtain a decision procedure, we finally prove that every weak formula G is valid if and only if it is satisfied by all interpretations, where the size of the domain is bounded by the number of constant symbols occurring in G.

Let $\mathcal{I}$ be an arbitrary interpretation and let $d_1, \ldots, d_n$ be the different constant symbols occurring in a weak formula G. Let $\mathcal{I}'$ be the interpretation that is obtained from $\mathcal{I}$ by removing from the domain D of $\mathcal{I}$ all elements except those $e \in D$, where $v_{\mathcal{I}}(d_i) = e$ for some $i \in \{1, \ldots, n\}$. It remains to check that $\|G\|_{\mathcal{I}} < 1$ implies $\|G\|_{\mathcal{I}'} < 1$. In other words: if there is a counter model for G, then there is already one with a restricted domain, as indicated. To this aim, note that the quantifier shift laws of Propositions 3 and 1 entail that every weak G is equivalent to a formula of the form $(\exists x)G'(x)$, where $G'(x)$ is weak, too. Obviously, $\mathrm{distr}_{\mathcal{I}'}(G'(x)) \subseteq \mathrm{distr}_{\mathcal{I}}(G'(x))$. Since $Y \subseteq X$ implies $\sup X \geq \sup Y$, we obtain $\|G\|_{\mathcal{I}} = \sup \mathrm{distr}_{\mathcal{I}}(G'(x)) \geq \sup \mathrm{distr}_{\mathcal{I}'}(G'(x)) = \|G'\|_{\mathcal{I}'}$. By repeating this argument for all (weak) quantifier occurrences, we obtain $\|G\|_{\mathcal{I}'} \leq \|G\|_{\mathcal{I}}$, as required.

Finally, remember that, in Gödel logics, it only depends on the relative order, but not on the absolute values of assigned truth values different from 0 and 1, whether a given interpretation satisfies a formula. This implies that the number of different interpretations with finite domain is bounded by the size of the domain and the number of relevant (unary) predicates symbols. Hence we have shown that validity for untangled $\mathbf{G}_\uparrow$ is decidable. ☐

[illegible] Note that, in proving the validity of $(\forall x)C[A(x)]^+ \rightarrow C[(\forall x)A(x)]^+$ and of $(\forall x)C[A(x)]^- \rightarrow C[(\exists x)A(x)]^-$, we had to shift quantifiers in and out (depending on the type of context). But those shifts are only over closed subformulas. This is where the defining condition for untangled formulas is used. In contrast, quantifiers cannot be moved into the scope of other quantifiers, in general. Indeed, e.g., $(\exists y)(\forall x)(P(y) \wedge Q(x))$ entails $(\exists y)(P(y) \wedge Q(d))$, which in turn entails $(\forall x)(\exists y)(P(y) \wedge Q(x))$. But the latter formula does not entail $(\exists y)(\forall x)(P(y) \wedge Q(x))$. Moreover, note that Proposition 3 only holds for $\mathbf{G}_\uparrow$ and for $\mathbf{G}_n$, $n \geq 2$, but not for other Gödel logics.

The following statement can be directly extracted from the proof of Theorem 4.

Corollary 2. [illegible] F [illegible] $\mathbf{G}_\uparrow$ [illegible] $m + c$ [illegible] m [illegible]

[illegible] F [illegible] c [illegible] ff [illegible] F [illegible]

To see that the mentioned bound is tight consider a formula of the form

$$F_m = \bigwedge_{1 \leq i \leq m} (\exists x_i)(\diamond_i^1 P_1(x_i) \wedge \ldots \wedge \diamond_i^k P_k(x_i)),$$

where $\diamond_i^j$ is either $\neg$ or empty, and where the vectors $(\diamond_i^1, \ldots \diamond_i^k)$ and $(\diamond_j^1, \ldots \diamond_j^k)$ are different if $i \neq j$. Clearly, $\neg F_m$ is not valid, since F_m can be satisfied in an interpretation that assigns different elements to the m different variables. On the other hand, in any domain with less than m elements at least two conjuncts of the form $P_\ell(x_i)$ and $\neg P_\ell(x_j)$ cannot be satisfied simultaneously, which means that $\neg F_m$ is satisfied in all corresponding interpretations.

Note that our proof of Theorem 4 of implies that every untangled formula F can be translated into a propositional formula $\Pi(F)$, that is equivalent to F with respect to validity in $\mathbf{G}_\uparrow$. To this aim one replaces subformulas of F of the form $(\forall x)A(x)$ by $\bigwedge_{1 \leq i \leq m+c} A(d_i)$ and subformulas of the form $(\exists x)A(x)$ by $\bigvee_{1 \leq i \leq m+c} A(d_i)$, where each $d \in \{d_1, \ldots, d_{m+c}\}$ is either one of the c constant symbols that already occur in F or corresponds to one of the m strong quantifier occurrence in F.

It is well known that every propositional formula A is valid in $\mathbf{G}_\uparrow$ (which coincides with $\mathbf{G}_\infty$ in the propositional case) if and only if it is valid in $\mathbf{G}_{n+2}$, where n is the number of different propositional variables in A (see, e.g., [11]). Therefore one can reduce testing validity of untangled formulas in $\mathbf{G}_\uparrow$

Corollary 3. [illegible] F [illegible] $\mathbf{G}_\uparrow$ [illegible] $\Pi(F)$ [illegible] $\mathbf{G}_{(m+c)\cdot k+2}$ [illegible] m [illegible] c [illegible] ff [illegible] F [illegible] k [illegible] ff [illegible] F

Testing validity is well known to be in co-NP for all finite-valued logics. Clearly, for every untangled F, the parameters m, c, and k are all linearly bounded by the size of F. Therefore Corollary 3 implies that testing validity for untangled formulas in $\mathbf{G}_\uparrow$ is in co-NP, as well. This should be contrasted with the fact that testing validity for arbitrary monadic formulas is NEXPTIME-hard already for classical logic, see [6].

[illegible] Although we have stated Theorem 4 only for $\mathbf{G}_\uparrow$, it is clear from the proof that the untangled fragments of finite-valued logics $\mathbf{G}_n$ can be decided in the same manner. Of course, untangled $\mathbf{G}_n$ is only a subclass of monadic $\mathbf{G}_n$. However, the bounds mentioned in in Corollaries 2 and 3 do not depend on n or on the number of different predicate symbols. Therefore they are drastically better, in general, than the corresponding bounds for the unrestricted monadic fragments (cf. Remark 3 in Sect. 3).

6 Axiomatization of Untangled $\mathbf{G}_\uparrow$

The decidability proof of Sect. 5 for the untangled fragment of $\mathbf{G}_\uparrow$ referred to the semantics of $\mathbf{G}_\uparrow$ at several places. However, a close inspection of the proof shows that in fact all formula schemes and rules that have been used are valid in *all* Gödel logics, except for the quantifier shift laws (11) and (12) of Proposition 3.

This observation is significant, because in [3] it has been proved that $\mathbf{G}_\uparrow$ is not recursively enumerable. In other words: (full) $\mathbf{G}_\uparrow$ cannot be recursively axiomatized. In contrast to this fact, we obtain an elegant Hilbert style axiom system that is sound and complete for untangled formulas in $\mathbf{G}_\uparrow$ from the proof of Theorem 4.

We rely on a well known axiom system for $\mathbf{G}_{[0,1]}$ (see, e.g., [11]). Remember that $\mathbf{G}_{[0,1]}$ is the intersection of all Gödel logics. The $\mathbf{G}_\uparrow$-specific laws (11) and (12) can already be derived in intuitionistic logic from the schemes (8) and (9). This leads to the following system for $\mathbf{G}_\uparrow$:

Intuitionistic axioms and rules: (any choice)
Linearity axiom:
$(A \to B) \vee (B \to A)$
General quantifier axiom (valid in $\mathbf{G}_{[0,1]}$, and thus in all $\mathbf{G}_V$):
$(\forall x)(A(x) \vee B) \to ((\forall x)A \vee B)$, where x is not free in B
Specific quantifier axioms (valid only in $\mathbf{G}_\uparrow$):
$(\exists x)(A(x) \to (\forall x)A(x))$
$(\exists x)((\exists y)A(y) \to A(x))$

According to [3], this system (like any other recursively presented proof system) cannot be complete for full $\mathbf{G}_\uparrow$. Nevertheless it is complete for *untangled* $\mathbf{G}_\uparrow$, since all laws that have been used in the proof of Theorem 4 can be derived.

7 Conclusion

We have investigated the decision problem for monadic fragments of Gödel logics. In presence of the projection operator Δ the emerging picture is clear and simple: validity is decidable for all *finite-valued* monadic Gödel logics, but is undecidable for all *infinite-valued* monadic Gödel logics. (The latter even holds for prenex formulas.) Without Δ all, but possible one, infinite-valued monadic Gödel logics remain undecidable. (Obviously, the decidability result for finite-valued logics also carries over to the language without $\triangle$.) The missing case, $\mathbf{G}_\uparrow$, is an important and interesting logic, since it coincides with the intersection of all finite-valued logics. The (un)decidability of monadic $\mathbf{G}_\uparrow$ remains open. It is reminiscent of a long-standing open problem (see, e.g., [13]) that seems to be related: the (un)decidability of validity for monadic Łukasiewicz logic **Ł**.

Motivated by a potentially important application of many-valued logics, we have singled out a natural sub-case of monadic logic; namely, the set of *untangled* formulas. Validity in $\mathbf{G}_\uparrow$ for this fragment is shown to be decidable. In fact, efficient and tight bounds are readily extracted from our decidability proof.

These bounds point to a considerable more efficient decision procedure for untangled formulas also in the case of finite-valued logics (compared to the standard decision method for the unrestricted monadic fragments). Moreover, since all quantifier shifts are valid in **Ł**, we conjecture that this decidability result can be transferred to the untangled fragment of **Ł** (and to similar logics as well).

References

1. Baaz, M.: Infinite-valued Gödel logics with 0-1-projections and relativizations. In: Proceedings Gödel 1996. Kurt Gödel's Legacy. LNL, vol. 6, pp. 23–33. Springer, Heidelberg (1996)
2. Baaz, M., Ciabattoni, A., Fermüller, C.G.: Herbrand's Theorem for Prenex Gödel Logic and its Consequences for Theorem Proving. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 201–216. Springer, Heidelberg (2001)
3. Baaz, M., Leitsch, A., Zach, R.: Incompleteness of an infinite-valued first-order Gödel Logic and of some temporal logic of programs. In: Kleine Büning, H. (ed.) CSL 1995. LNCS, vol. 1092, pp. 1–15. Springer, Heidelberg (1996)
4. Baaz, M., Preining, N., Zach, R.: First-order Gödel logics. Annals of Pure and Applied Logic 147/1-2, 23–47 (2007)
5. Beckmann, A., Goldstern, G., Preining, N.: Continuous Fraïssé Conjecture. Sumitted, preprint at `http://arxiv.org/abs/math/0411117`
6. Börger, E., Grädel, E., Gurevich, Y.: The classical Decision Problem. Springer, Heidelberg (1997)
7. Dummett, M.: A propositional calculus with denumerable matrix. J. of Symbolic Logic 24, 97–106 (1959)
8. Gabbay, D.M.: Decidability of some intuitionistic predicate theories. J. of Symbolic Logic 37, 579–587 (1972)
9. Gödel, K.: Zum intuitionistischen Aussagenkalkül. Anz. Akad. Wiss. Wien 69, 65–66 (1932)
10. Gottwald, S.: A Treatise on Many-Valued Logics. Studies in Logic and Computation 9, Research Studies Press (2001)
11. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer Academic Publishers, Dordrecht (1998)
12. Moser, W., Adlassnig, K.-P.: Consistency checking of binary categorical relationships in a medical knowledge base. Artificial Inteligence in Medicine 7, 389–407 (1992)
13. Ragaz, M.: Die Unentscheidbarkeit der einstelligen unendlichwertigen Prädikatenlogik. Arch. math. Logik 23, 129–139 (1983)
14. Rogers, H.: Certain logical reduction and decision problems. Annals of Mathematics 64, 264–284 (1956)
15. Takeuti, G., Titani, T.: Intuitionistic fuzzy logic and intuitionistic fuzzy set theory. J. of Symbolic Logic 49, 851–866 (1984)

Least and Greatest Fixed Points in Linear Logic

David Baelde and Dale Miller

INRIA & LIX/École Polytechnique, Palaiseau, France
`david.baelde@ens-lyon.org`, `dale.miller@inria.fr`

Abstract. The first-order theory of MALL (multiplicative, additive linear logic) over only equalities is an interesting but weak logic since it cannot capture unbounded (infinite) behavior. Instead of accounting for unbounded behavior via the addition of the exponentials (! and ?), we add least and greatest fixed point operators. The resulting logic, which we call $\mu\text{MALL}^{=}$, satisfies two fundamental proof theoretic properties. In particular, $\mu\text{MALL}^{=}$ satisfies cut-elimination, which implies consistency, and has a complete focused proof system. This second result about focused proofs provides a strong normal form for cut-free proof structures that can be used, for example, to help automate proof search. We then consider applying these two results about $\mu\text{MALL}^{=}$ to derive a focused proof system for an intuitionistic logic extended with induction and co-induction. The traditional approach to encoding intuitionistic logic into linear logic relies heavily on using the exponentials, which unfortunately weaken the focusing discipline. We get a better focused proof system by observing that certain fixed points satisfy the structural rules of weakening and contraction (without using exponentials). The resulting focused proof system for intuitionistic logic is closely related to the one implemented in Bedwyr, a recent model checker based on logic programming. We discuss how our proof theory might be used to build a computational system that can partially automate induction and co-induction.

1 Introduction

In order to justify the design and implementation architecture of a computational logic system, foundational results concerning the normal forms of proofs are often used. One starts with the *cut-elimination theorem* since it usually guarantees other properties of the logic (*e.g.*, consistency) and that there is no need to automate the creation of *lemmas* during proof search. In many situations, the cut-elimination theorem implies that all formulas considered during the search for a proof are subformulas of the original, proposed theorem. This does not hold, in particular, when higher-order (relation) variables are used, which is the case in this paper where the rules for induction and co-induction use such higher-order variables. A second normal form theorem, usually related to *focused proofs* [And92] is also important to establish. Such "focusing" theorems provide normal forms that organize invertible and non-invertible inference rules into collections: such striping of the inference rules in a cut-free derivation can be used to understand which choices in building proofs might need to be reconsidered (via backtracking) and which do not. As we shall see, focusing yields useful structure in cut-free proofs, even when the subformula property does not hold.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 92–106, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Various computational systems have employed different focusing theorems: much of Prolog's design and implementations can be justified by the completeness of SLD-resolution [AvE82]; uniform proofs (goal-directed proofs) in intuitionistic and intuitionistic linear logics have been used to justify λProlog [MNPS91] and Lolli [HM94]; the classical linear logic programming languages LO [AP91] and Forum [Mil96] have used directly Andreoli's general focusing result [And92] for linear logic.

In this paper, we establish these two foundational proof-theoretic properties for the following logic. We first extend the multiplicative and additive fragment of linear logic (MALL) with equality and quantification (via $\forall$ and $\exists$) over simply typed λ-terms. Because of the bounded use of formulas during proof construction, provability in this logic, call it $MALL^=$, can be reduced to deciding unification problems (under a mixed prefix) which is decidable for the first-order fragment of $MALL^=$. An elegant and well known way to make this logic more expressive is to add the exponentials ! and ? and the rules of inference that allow for certain occurrences of formulas marked with these systems to be contracted and weakened [Gir87]. Such modal-like operators are not, however, without their problems. In particular, the exponentials are not canonical since there are different ways to formulate the rules for the promotion and structural rules for exponentials and some of these choices lead to different versions of logic (for example, elementary and light linear logics [Gir98] and soft linear logic [Laf04]). Even if we fix the inference rules for the exponentials, as in standard linear logic, the rules do not describe unique exponentials. If one gives a red tensor and a blue tensor the same inference rules, then one can prove that these two tensors are, in fact, equivalent. All of linear logic connectives except the exponentials yield similar theorems. It is certainly possible to consider a (partially ordered) collection of exponentials on top of MALL (see, for example, [DJS93]).

An alternative to strengthen MALL with exponentials is to extend it with fixed points. Early approaches to adding fixed points [Gir92, SH93] involved inference rules that could only unfold fixed point descriptions: as a consequence, such logics could not discriminate between a least and greatest fixed point. Stronger systems that allow induction [MM00] as well as co-induction [Tiu04, MT03] include inference rules using a higher-order variable that ranges over prefixed or postfixed points (invariants). Of course, approaches that use (co)induction are not without problems as well: various restrictions on fixed point expressions and on invariants may need to be considered. In any case, we shall explore this alternative to exponentials: in particular, we extend the logic $MALL^=$ to μMALL$^=$ by adding the two fixed points μ and ν.

Besides considering fixed points as alternatives to the exponentials, there are other reasons for examining μMALL$^=$. First, least and greatest fixed points are de Morgan duals of one another and, hence, the classical nature of linear logic should offer some economy and elegance in developing their proof theory, in contrast to intuitionistic logic. Second, since linear logic can be seen as the logic behind intuitionistic logic, it will be rather easy to develop a focusing proof system for intuitionistic logic and fixed points based on the structure of the one we develop for μMALL$^=$.

It is important to stress that we are using linear logic here as "the logic behind computational logic" and not, as it is more traditionally understood, as the logic of resource management (in the sense of multiset rewriting, database updates, Petri nets, etc).

Instead, we find the proof theory of linear logic an appropriate and powerful setting for exploring the structure of proofs in various intuitionistic logics (see [LM07] for another such use of linear logic).

In the next section, we define $\mu\text{MALL}^=$ and prove some of the most basic aspects of its proof theory, including the cut-elimination theorem. Section 3 presents a focused proof system that is complete for $\mu\text{MALL}^=$. In Section 4 we describe a few examples of (focused) derivations in $\mu\text{MALL}^=$. Section 5 shows how the proof theory of $\mu\text{MALL}^=$ can be applied to an intuitionistic logic extended with induction and co-induction, and to the intuitionistic logic of fixed point unfoldings that is the foundation of the recent computational system Bedwyr [BGM$^+$07].

2 Linear Logic Extended with Fixed Points

For clarity, we will use simply typed λ-calculus as our language of formulas. We assume that formulas are always in $\beta\eta$-long form. We make few restrictions on the language of terms in this work and choose simply typed λ-calculus for them as well: we assume that the reader understands the basics involving substitution, equality, and complete set of unifiers for such terms. In most of our examples variables will be of ground type, and thus the possibly infinite complete set of unifiers can be replaced by the most general unifier when there is one. Depending on one's interests, it is possible to choose weaker (*e.g.*, first-order) or more powerful (*e.g.*, dependently typed) terms.

In the following, terms are denoted by s, t; vectors of terms are denoted by $\boldsymbol{s}, \boldsymbol{t}$; formulas (objects of type o) are denoted by P, Q; eigenvariables are denoted by x, c. Finally, the syntactic variable B represents a formula abstracted over by a predicate and n terms $(\lambda p \lambda x_1 \ldots \lambda x_n . P p x_1 \ldots x_n)$. We have the following formula constructors:

$$\begin{aligned} P ::= {} & P \otimes P \mid P \oplus P \mid P \parr P \mid P \mathbin{\&} P \mid \mathbf{1} \mid \mathbf{0} \mid \bot \mid \top \\ \mid {} & \exists_\gamma x.Px \mid \forall_\gamma x.Px \mid s \overset{\gamma}{=} t \mid s \overset{\gamma}{\neq} t \mid \mu_{\gamma_1 \ldots \gamma_n} B\boldsymbol{t} \mid \nu_{\gamma_1 \ldots \gamma_n} B\boldsymbol{t} \end{aligned}$$

The syntactic variable γ ranges over all simple types that do not contain o. The quantifiers have type $(\gamma \to o) \to o$ and the equality and inequality have type $\gamma \to \gamma \to o$. The connectives μ and ν have type $(\tau \to \tau) \to \tau$ where τ is $\gamma_1 \to \cdots \to \gamma_n \to o$ for some arity $n \geq 0$. We shall almost always elide the references to γ, assuming that they can be determined from context when it is important to know their value. Formulas with top-level connective μ or ν are called fixed point expressions and can be arbitrarily nested. The first argument of a fixed point expression, denoted by B, is called its *body*.

Quantifiers and (in)equality are not new and play a small role in the proof theory results: they are, however, crucial for our example applications. The central feature here is the fixed point constructs. Finally, note that there are no atoms in the $\mu\text{MALL}^=$ grammar. We shall see in the following the advantages of using fixed points instead.

Definition 1. *We define the* negation $\overline{B}$ *of a body B, and extend the usual definition of the involutive* negation *as follows:*

$$\overline{B} \overset{def}{=} \lambda p.\lambda \boldsymbol{x}.(B(\lambda \boldsymbol{x}.(p\boldsymbol{x})^\perp)\boldsymbol{x})^\perp \qquad (s = t)^\perp \overset{def}{=} s \neq t \qquad (\mu B\boldsymbol{t})^\perp \overset{def}{=} \nu \overline{B}\boldsymbol{t}$$

MALL rules

$$\frac{}{\vdash \mathbf{1}} \qquad \frac{\vdash \Gamma, P \quad \vdash \Delta, Q}{\vdash \Gamma, \Delta, P \otimes Q} \qquad \frac{\vdash \Gamma, P, Q}{\vdash \Gamma, P \parr Q} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \bot}$$

$$\frac{}{\vdash \Delta, \top} \qquad \frac{\vdash \Gamma, P \quad \vdash \Gamma, Q}{\vdash \Gamma, P \,\&\, Q} \qquad \frac{\vdash \Gamma, P_i}{\vdash \Gamma, P_0 \oplus P_1}$$

First-order structure

$$\frac{\vdash \Gamma, Pt}{\vdash \Gamma, \exists x.Px} \qquad \frac{\vdash \Gamma, Pc}{\vdash \Gamma, \forall x.Px}\ c \text{ new}$$

$$\frac{}{\vdash t = t} \qquad \frac{\{\vdash \Gamma\theta : \theta \in csu(s \doteq t)\}}{\vdash \Gamma, s \neq t}$$

Fixed points (where S is closed, $\mathbf{x}$ is new)

$$\frac{\vdash \Gamma, B(\mu B)\mathbf{t}}{\vdash \Gamma, \mu B\mathbf{t}}\ \mu \qquad \frac{\vdash \Gamma, S\,\mathbf{t} \quad \vdash BS\,\mathbf{x}, (S\,\mathbf{x})^\perp}{\vdash \Gamma, \nu B\mathbf{t}}\ \nu \qquad \frac{}{\vdash \mu B\mathbf{t}, \nu\overline{B}\mathbf{t}}\ \mu\nu$$

Fig. 1. Inference rules for μMALL$^=$

A body B is said to be monotonic *when for any variables p and $\mathbf{t}$, the negation normal and λ-normal form of $Bp\mathbf{t}$ does not contain any negated instance of p.*

We shall assume that *all bodies are monotonic*. In other words, negation ($\bullet^\perp$ for formulas and $\overline{\bullet}$ for bodies) is not part of the syntax since negation normal form of formulas and bodies without atoms do not contain negations and since we forbid them explicitly in fixed point expressions. When we write negation in some inference rules, we shall be considering it as implicitly computing the negation normal form.

The monotonicity of a function is also a natural condition for the existence of fixed points in lattices or other models. The condition of monotonicity is used only syntactically here since we are not studying the semantics of μMALL$^=$.

We present the inference rules for μMALL$^=$ in Figure 1. The initial rule is restricted to fixed points. In the ν rule, which provides both induction and coinduction, S is called the (co)invariant, and has the same type as νB, of the form $\gamma_1 \to \cdots \to \gamma_n \to o$. The treatment of equality dates back to [Gir92, SH93]. In the inequality rule, *csu* stands for complete set of unifiers. This set has at most one element in the first-order case, but can be infinite in presence of higher-order term variables, which we do not exclude. In that case, the proofs are infinitely branching but still have a finite depth. They are handled easily in our proofs by means of transfinite inductions. Again, the use of higher-order terms, and even the presence of the equality connectives are not essential to this work. All the results presented below hold in the logic without equality, and they do not make much assumptions on the language of terms.

Proposition 1. *The following inference rules are derivable:*

$$\frac{}{\vdash P, P^\perp}\ init \qquad \frac{\vdash \Gamma, B(\nu B)\mathbf{t}}{\vdash \Gamma, \nu B\mathbf{t}}\ \nu R$$

These results are standard, cf. [Tiu04]. The proof of the second one relies on monotonicity and is obtained by applying the ν rule with $B(\nu B)$ as the co-invariant.

Definition 2. *We classify as* asynchronous *(resp.* synchronous*) the connectives $\parr$, $\bot$, $\&$, $\top$, $\forall$, $\neq$, ν (resp. $\otimes$, $\mathbf{1}$, $\oplus$, $\mathbf{0}$, $\exists$, $=$, μ). A formula is said to be asynchronous (resp. synchronous) when its top-level connective is asynchronous (resp. synchronous). A formula* *is said to be* fully asynchronous *(resp.* fully synchronous*) when all of its connectives are asynchronous (resp. synchronous). Finally, a body* $\lambda p \lambda \mathbf{x}.Bp\mathbf{x}$ *is said to be fully asynchronous (resp. fully synchronous) when the formula* $Bp\mathbf{x}$ *is fully asynchronous (resp. fully synchronous).*

Notice, for example, that $\lambda p \lambda \mathbf{x}.p\mathbf{x}$ is fully asynchronous and fully synchronous.

Proposition 2. *The following structural rules are admissible provided that* B *is fully asynchronous:*

$$\frac{\vdash \Gamma, \nu B\mathbf{t}, \nu B\mathbf{t}}{\vdash \Gamma, \nu B\mathbf{t}}\ \nu C \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \nu B\mathbf{t}}\ \nu W$$

Hence, the following structural rules hold for any fully asynchronous formula P*:*

$$\frac{\vdash \Gamma, P, P}{\vdash \Gamma, P}\ C \qquad \frac{\vdash \Gamma}{\vdash \Gamma, P}\ W$$

The proof of this proposition can be found in [BM07]. This property plays a central role in the focusing proof system presented in Section 3 and is crucial in Section 5 for our encoding of intuitionistic logic extended with least and greatest fixed points.

Example 1. Units can be represented by means of $=$ and $\neq$. Assuming that 2 and 3 are two distinct constants, then we have $2 = 2 \multimapboth \mathbf{1}$ and $2 = 3 \multimapboth \mathbf{0}$ (and hence $2 \neq 2 \multimapboth \bot$ and $2 \neq 3 \multimapboth \top$). Here, $P \multimapboth Q$ denotes $\vdash (P \multimap Q) \,\&\, (Q \multimap P)$ and $P \multimap Q$ denotes the formula $P^{\perp} \parr Q$.

Example 2. The μ (resp. ν) connective is meant to represent least (resp. greatest) fixed points. For example $\nu(\lambda p.p)$ is provable (take any provable formula as the co-invariant), while its dual $\mu(\lambda p.p)$ is not provable. More precisely: $\mu(\lambda p.p) \multimapboth \mathbf{0}$ and $\nu(\lambda p.p) \multimapboth \top$.

Example 3. The least fixed point, as expected, entails the greatest. The following is a proof of $\mu B\mathbf{t} \multimap \nu B\mathbf{t}$.

$$\dfrac{\dfrac{\dfrac{}{\vdash B(\mu B)\mathbf{x}, \overline{B}(\nu\overline{B})\mathbf{x}}\ init}{\vdash B(\mu B)\mathbf{x}, \nu\overline{B}\mathbf{x}}\ \nu R \qquad \dfrac{}{\vdash \mu B\mathbf{t}, \nu\overline{B}\mathbf{t}}\ \mu\nu}{\vdash \nu\overline{B}\mathbf{t}, \nu B\mathbf{t}}\ \nu \text{ on } \nu B\mathbf{t} \text{ with } S := \mu B$$

The greatest fixed point entails the least fixed point when the fixed points are noetherian, *i.e.*, all unfoldings of B and $\overline{B}$ terminate.

In this paper we are investigating how far one can go without the exponentials, getting the infinite behavior from the meaning of fixed points instead of modalities. If we were to add, however, the usual inference rules for exponentials, the resulting proof system would yield $\mu B\mathbf{t} \multimapboth\ !\mu B\mathbf{t}$ (and equivalently $?\nu\overline{B}\mathbf{t} \multimapboth \nu\overline{B}\mathbf{t}$) provided that B is fully synchronous. In the language of the Logic of Unity (LU) [Gir93], fully asynchronous (resp. fully synchronous) would be negative (resp. positive) or right-permeable (resp. left-permeable) formulas. Mixing synchronous and asynchronous connectives would yield a neutral formula.

We now outline the proof of cut-elimination. Although it is indirect and relies on cut-elimination for full second-order linear logic (LL2), this is still a syntactic proof of cut-elimination. It yields consistency of μMALL$^=$ as well as relative soundness and completeness with respect to LL2.

Theorem 1. *The logic μMALL$^=$ enjoys cut-elimination.*

Proof. Our proof consists in first translating μMALL$^=$ formulas and proofs into full second-order linear logic derivations, which are then normalized and focused, and finally translated back to cut-free μMALL$^=$ derivations. Formally speaking, the previous work on proof normalization for LL2 does not include equality, but all the previous work on equality has shown that it has little role to play in normalization.

We first define the translation from first-order to second-order. The translation commutes with the connectives of MALL$^=$ and the negation, and is defined as follows on the least fixed points:

$$\lceil \mu B\boldsymbol{x} \rceil = \forall S\ .\ !(\forall \boldsymbol{y}\ .\ \lceil B \rceil S\boldsymbol{y} \multimap S\boldsymbol{y}) \multimap S\boldsymbol{x}$$

The corresponding transformation of proofs is straightforward, relying on the monotonicity of bodies. We get a proof where all second-order instantiations are either of the form $\lceil I \rceil$ (from ν rules) or second-order eigenvariables (from $\mu\nu$ rules). Cut-elimination and focusing never change these instantiations.

It is possible to normalize the resulting LL2 derivations, and then apply Andreoli's result to yield even more structured normal forms. (We shall temporarily assume that the reader is familiar with the focusing proof system in [And92]. A description of this kind of system may otherwise be found in Section 3.) Doing so, we get exactly the derivations we want for transforming them back to μMALL$^=$. For example, focusing on an unfolding hypothesis translates immediately to the μ rule:

$$\dfrac{\dfrac{\vdash \Theta : \Gamma \Downarrow \lceil B_i \rceil S_i\boldsymbol{y} \qquad \dfrac{}{\vdash \Theta : S_i\boldsymbol{x} \Downarrow (S_i\boldsymbol{x})^\perp}}{\vdash \Theta : \Gamma, S_i\boldsymbol{x} \Downarrow \lceil B_i \rceil S_i\boldsymbol{x} \otimes (S_i\boldsymbol{x})^\perp}}{\vdash \Theta : \Gamma, S_i\boldsymbol{x} \Downarrow \exists \boldsymbol{y}.\lceil B_i \rceil S_i\boldsymbol{y} \otimes (S_i\boldsymbol{y})^\perp}$$

Similarly, focusing on the translation of a ν gives us either an instance of the ν rule:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash \Theta :\Uparrow \lceil BI\boldsymbol{y} \multimap I\boldsymbol{y} \rceil}{\vdash \Theta :\Uparrow \forall \boldsymbol{y}.\lceil BI\boldsymbol{y} \multimap I\boldsymbol{y} \rceil}}{\vdash \Theta :\Downarrow !\forall \boldsymbol{y}.\lceil BI\boldsymbol{y} \multimap I\boldsymbol{y} \rceil} \qquad \vdash \Theta : \Gamma \Downarrow \lceil I\boldsymbol{x} \rceil^\perp}{\vdash \Theta : \Gamma \Downarrow (!\forall \boldsymbol{y}.\lceil B \rceil \lceil I \rceil \boldsymbol{y} \multimap \lceil I \rceil \boldsymbol{y}) \otimes \lceil I\boldsymbol{x} \rceil^\perp}}{\vdash \Theta : \Gamma \Downarrow \exists S.!(\forall \boldsymbol{y}.\lceil B \rceil S\boldsymbol{y} \multimap S\boldsymbol{y}) \otimes (S\boldsymbol{x})^\perp}\ S := \lceil I \rceil$$

or an instance of $\mu\nu$ (the unfolding hypothesis for S is in Θ):

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Theta :\Downarrow !\forall \boldsymbol{y}.\lceil B \rceil S\boldsymbol{y} \multimap S\boldsymbol{y}} \qquad \dfrac{}{\vdash \Theta : S\boldsymbol{x} \Downarrow (S\boldsymbol{x})^\perp}}{\vdash \Theta : S\boldsymbol{x} \Downarrow (!\forall \boldsymbol{y}.\lceil B \rceil S\boldsymbol{y} \multimap S\boldsymbol{y}) \otimes (S\boldsymbol{x})^\perp}}{\vdash \Theta : S\boldsymbol{x} \Downarrow \exists S.!(\forall \boldsymbol{y}.\lceil B \rceil S\boldsymbol{y} \multimap S\boldsymbol{y}) \otimes (S\boldsymbol{x})^\perp}$$

For a more detailed proof, see [BM07]. □

As shown in the above proof, fixed points can be encoded by means of second-order quantification and exponentials. However, first-order MALL with exponentials and first-order MALL with fixed points are incomparable.

It has been observed [Gir92, SH93] that exponentials and non-monotonic definitions combine to yield inconsistency: for example, the definition $p \equiv p^\perp$ (that is, the fixed point $\mu\lambda p.p^\perp$) does not lead to an inconsistency, whereas the definition $p \equiv ?(p^\perp)$ (that is, $\mu\lambda p.\, ?(p^\perp)$) does. To reproduce the latter inconsistency in μMALL$^=$, one needs to be able to unfold the expression $\nu\lambda p.\, !(p^\perp)$. But this is not implied by Proposition 1 since its body is not monotonic. Thus, even in presence of exponentials, we currently do not have any example of non-monotonic definition that invalidates the consistency of μMALL$^=$.

3 Focused Proofs

As we have explained in the introduction, completeness of a focused proof system is a valuable property for a logic to possess. Focused proofs have applications in proof-search since it reduces the proof-search space by limiting the situations when backtracking is necessary. Focused proofs are also useful for justifying game theoretic semantics [MS05] and have been central to the design of Ludics [Gir01].

A good focused proof system for μMALL$^=$ is not a simple consequence of the translation of fixed points into LL2 that is used in the proof of Theorem 1: applying linear logic focusing to the result of that translation leads to a poorly structured system that is not consistent with our classification of connectives as asynchronous and synchronous. On the contrary, we present the proof system in Figure 2 as a good candidate for a focused proof system for μMALL$^=$. We use explicit annotations of the sequents in the style of Andreoli. In the synchronous phase sequents have the form $\vdash \Gamma \Downarrow P$. In the asynchronous phase they have the form $\vdash \Gamma \Uparrow \Delta$ where Γ and Δ are both multisets of formulas. In both sequents, Γ is a multiset of synchronous formulas and ν-expressions. The convention on Δ is a slight departure from Andreoli's original proof system where Δ is a list (which can be used to provide a fixed but arbitrary ordering of the asynchronous phase).

The rules for equality are not surprising. The main novelty here is the treatment of fixed points. Depending on the body, both μ and ν rules can be applied any number of times — but not with any co-invariant concerning ν. Notice for example that an instance of $\mu\nu$ can be η-expanded into a larger derivation, unfolding both fixed points to apply $\mu\nu$ on the recursive occurrences. As a result, each of the fixed point connectives has two rules in the focused system: one treats it as "an atom" and the other one as an expression with "internal structure."

In accord with Definition 2, μ is treated during the synchronous phase and ν during the asynchronous phase. (Alternatives to this choice are discussed later.) Roughly, what the focused system implies is that if a proof involving a ν-expression proceeds by co-induction on it, then this co-induction can be done at the beginning; otherwise that formula can be ignored in the whole derivation, except for the $\mu\nu$ rule. Focusing on a μ-expression yields two choices: unfolding or applying the initial rule for fixed points. If the body is fully synchronous, the focusing will never be lost. For example, if *nat* is the

Asynchronous phase

$$\frac{\vdash \Gamma \Uparrow P, Q, \Delta}{\vdash \Gamma \Uparrow P \parr Q, \Delta} \qquad \frac{\vdash \Gamma \Uparrow P, \Delta \quad \vdash \Gamma \Uparrow Q, \Delta}{\vdash \Gamma \Uparrow P \mathbin{\&} Q, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow \Delta}{\vdash \Gamma \Uparrow \bot, \Delta} \qquad \frac{}{\vdash \Gamma \Uparrow \top, \Delta} \qquad \frac{\{\vdash \Gamma\theta \Uparrow \Delta\theta : \theta \in csu(s \doteq t)\}}{\vdash \Gamma \Uparrow s \neq t, \Delta}$$

$$\frac{\vdash \Gamma \Uparrow Pc, \Delta}{\vdash \Gamma \Uparrow \forall x.Px, \Delta}\ c \text{ new}$$

$$\frac{\vdash \Gamma \Uparrow S\,\mathbf{t}, \Delta \quad \vdash \Uparrow BS\,\mathbf{x}, S\,\mathbf{x}^{\perp}}{\vdash \Gamma \Uparrow \nu B\mathbf{t}, \Delta}\ \mathbf{x} \text{ new} \qquad \frac{\vdash \Gamma, \nu B\mathbf{t} \Uparrow \Delta}{\vdash \Gamma \Uparrow \nu B\mathbf{t}, \Delta}$$

Synchronous phase

$$\frac{\vdash \Gamma \Downarrow P \quad \vdash \Gamma' \Downarrow Q}{\vdash \Gamma, \Gamma' \Downarrow P \otimes Q} \qquad \frac{\vdash \Gamma \Downarrow P_i}{\vdash \Gamma \Downarrow P_0 \oplus P_1}$$

$$\frac{}{\vdash \Downarrow \mathbf{1}} \qquad \frac{}{\vdash \Downarrow t = t}$$

$$\frac{\vdash \Gamma \Downarrow Pt}{\vdash \Gamma \Downarrow \exists x.Px}$$

$$\frac{\vdash \Gamma \Downarrow B(\mu B)\mathbf{x}}{\vdash \Gamma \Downarrow \mu B\mathbf{x}} \qquad \frac{}{\vdash \nu\overline{B}\mathbf{x} \Downarrow \mu B\mathbf{x}}$$

Switching (where P is synchronous, Q asynchronous)

$$\frac{\vdash \Gamma, P \Uparrow \Delta}{\vdash \Gamma \Uparrow P, \Delta} \qquad \frac{\vdash \Gamma \Downarrow P}{\vdash \Gamma, P \Uparrow} \qquad \frac{\vdash \Gamma \Uparrow Q}{\vdash \Gamma \Downarrow Q}$$

Fig. 2. A focused proof-system for μMALL$^{=}$

$$\dfrac{\dfrac{\dfrac{\Pi}{\vdash \Gamma, P, S\,\mathbf{t}} \quad \dfrac{\Pi_S}{\vdash BS\,\mathbf{x}, S\,\mathbf{x}^{\perp}}}{\vdash \Gamma, P, \nu B\mathbf{t}} \qquad \dfrac{\dfrac{\Pi'}{\vdash \Gamma, P', S'\mathbf{t}} \quad \dfrac{\Pi_{S'}}{\vdash BS'\mathbf{x}, S'\mathbf{x}^{\perp}}}{\vdash \Gamma, P', \nu B\mathbf{t}}}{\vdash \Gamma, P \mathbin{\&} P', \nu B\mathbf{t}}$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi}{\vdash \Gamma, P, S\,\mathbf{t}}}{\vdash \Gamma, P, S\,\mathbf{t} \oplus S'\mathbf{t}} \quad \dfrac{\dfrac{\Pi'}{\vdash \Gamma, P', S'\mathbf{t}}}{\vdash \Gamma, P', S\,\mathbf{t} \oplus S'\mathbf{t}}}{\vdash \Gamma, P \mathbin{\&} P', S\,\mathbf{t} \oplus S'\mathbf{t}} \qquad \dfrac{\dfrac{\phi_1(\Pi_S)}{\vdash B(S \oplus S')\mathbf{x}, (S\,\mathbf{x})^{\perp}} \quad \dfrac{\phi_2(\Pi_{S'})}{\vdash B(S \oplus S')\mathbf{x}, (S'\mathbf{x})^{\perp}}}{\vdash B(S \oplus S')\mathbf{x}, ((S \oplus S')\mathbf{x})^{\perp}}\ \&}{\vdash \Gamma, P \mathbin{\&} P', \nu B\mathbf{t}}$$

Fig. 3. The permutation of the & and the co-induction rules

(fully synchronous) expression $\mu(\lambda nat.\lambda x.\ x = 0 \oplus \exists y.x = s\ y \otimes nat\ y)$, then focusing puts a lot of structure on a proof of $\Gamma \Downarrow nat\ t$: either t is a ground term representing a natural number and Γ is empty, or $t = s^n x$ for some $n \geq 0$ and Γ is $\{(nat\ x)^{\perp}\}$.

Theorem 2. *The focused system is sound and complete with respect to μMALL$^{=}$.*

Proof. Soundness is trivial. We only give an outline of the completeness proof: see [BM07] for more details. The proof is by (transfinite) induction on $(h_\mu(\Pi), |\Pi|)$ where $h_\mu(\Pi)$ is the height of Π in terms of fixed point rules, and $|\Pi|$ is the size of the derivation's conclusion. We first prove two permutation lemmas which preserve this measure: one shows that if there is any asynchronous formula in the conclusion, the proof can be transformed such that this formula is active in the conclusion; the other shows that when there is no more asynchronous in the conclusion, it is possible to focus on a synchronous if it is *maximal*. Finally we prove that there is always a maximal formula in such a sequent. The notion of maximality is due to Alexis Saurin [MS07] and is crucial to make the proof clear and simple.

It is worth pointing out, however, that there is a non-trivial permutation of & and ν in the first of these lemmas. This permutation, which requires the ability to sum co-invariants (a consequence of the monotonicity assumption on fixed point expressions) is illustrated in Figure 3. □

4 Examples

We shall now give a few theorems in μMALL$^=$. Although we do not give their derivations here, we stress that all of these examples are proved naturally in the focused proof system. The reader will also note that although μMALL$^=$ is linear, these derivations are intuitive and their structure resemble that of proofs in intuitionistic logic.

We first define a few least fixed points expressing basic properties of natural numbers. We assume two constants z and s of respective types n and $n \to n$. Note that all these definitions are fully synchronous.

$$\begin{aligned}
nat &\stackrel{def}{=} \mu(\lambda nat \lambda x.\ x = z \oplus \exists y.\ x = s\ y \otimes nat\ y)\\
even &\stackrel{def}{=} \mu(\lambda even \lambda x.\ x = z \oplus \exists y.\ x = s\ (s\ y) \otimes even\ y)\\
plus &\stackrel{def}{=} \mu(\lambda plus \lambda a \lambda b \lambda c.\ a = z \otimes b = c\\
&\qquad\qquad \oplus \exists a' \exists c'. a = s\ a' \otimes c = s\ c' \otimes plus\ a'\ b\ c')\\
leq &\stackrel{def}{=} \mu(\lambda leq \lambda x \lambda y.\ x = y \oplus \exists y'.\ y = s\ y' \otimes leq\ x\ y')\\
half &\stackrel{def}{=} \mu(\lambda half \lambda x \lambda h.\ (x = z \oplus x = s\ z) \otimes h = z\\
&\qquad\qquad \oplus \exists x' \exists h'.\ x = s\ (s\ x') \otimes h = s\ h' \otimes half\ x'\ h')
\end{aligned}$$

The following statements are theorems, all of which can be proved by induction. The main insights required for proving these theorems involve deciding which fixed point expression should be introduced by induction: the proper invariant is not the difficult choice here since the context itself is adequate in these cases.

$$\begin{aligned}
&\vdash \forall x.\ nat\ x \multimap even\ x \oplus even\ (s\ x)\\
&\vdash \forall x.\ nat\ x \multimap \forall y \exists z.\ plus\ x\ y\ z\\
&\vdash \forall x.\ nat\ x \multimap plus\ x\ z\ x\\
&\vdash \forall x.\ nat\ x \multimap \forall y.\ nat\ y \multimap \forall z.\ plus\ x\ y\ z \multimap nat\ z
\end{aligned}$$

In the last theorem, the assumption $(nat\ x)^\perp$ is not needed and can be weakened, thanks to Proposition 2. In order to prove $(\forall x.\ nat\ x \multimap \exists h.\ half\ x\ h)$ one has to use a complete induction, *i.e.*, use the strengthened invariant $(\lambda x.\ nat\ x \otimes \forall y.\ leq\ y\ x \multimap \exists h.\ half\ y\ h)$.

A typical example of co-induction involves the simulation relation. Assume that $step : state \to label \to state \to o$ is an inductively defined relation encoding a labeled transition system. Simulation can be defined using the definition

$$sim \stackrel{def}{=} \nu(\lambda sim \lambda p \lambda q.\ \forall a \forall p'.\ step\ p\ a\ p' \multimap \exists q'.\ step\ q\ a\ q' \otimes sim\ p'\ q').$$

Reflexivity of simulation ($\forall p.\ sim\ p\ p$) is proved easily by co-induction with the co-invariant ($\lambda p \lambda q.\ p = q$). Instances of *step* are not subject to induction but are treated "as atoms". Proving transitivity, that is,

$$\forall p \forall q \forall r.\ sim\ p\ q \multimap sim\ q\ r \multimap sim\ p\ r$$

is done by co-induction on ($sim\ p\ r$) with the co-invariant ($\lambda p \lambda r.\ \exists q.\ sim\ p\ q \otimes sim\ q\ r$). The focus is first put on $(sim\ p\ q)^{\perp}$, then on $(sim\ q\ r)^{\perp}$. The fixed points ($sim\ p'\ q'$) and ($sim\ q'\ r'$) appearing later in the proof are treated "as atoms", as are all negative instances of *step*.

Except for the totality of *half*, all these theorems seem simple to prove using a limited number of heuristics. For example, one could first try to treat fixed points "as atoms", an approach that would likely fail quickly if inappropriate. Second, depending on the "rigid" structure of the arguments to a fixed point expression, one might choose to either unfold the fixed point or attempt to use the surrounding context to generate an invariant.

5 Translating Intuitionistic Logic

The examples in the previous section make it clear that despite its simplicity and linearity, μMALL$^{=}$ can be related to a more conventional logic. In particular we are interested in drawing some connections with an extension of intuitionistic logic with inductive and coinductive definitions. We will show that the focusing of μMALL$^{=}$ derivations yields a similar result in the intuitionistic setting. A general approach for making such a connection is to first encode intuitionistic logic in μMALL$^{=}$, focus the derivations of encodings, and translate them back to intuitionistic derivations. When doing so, it is interesting to minimize the use of exponentials in the encoding since these connectives weaken the focusing discipline. This is precisely what the extension of the asynchronous/synchronous classification allows. In the following, we show a simple first step to this program, in which we actually capture a non-trivial fragment of intuitionistic logic extended with fixed points even though μMALL$^{=}$ does not have exponentials at all.

We shall consider an intuitionistic logic in which there are no atomic formulas but were there are (positive) equalities and the two fixed point constructors μ and ν. Let μLJ$^{=}$ be the proof system that extends Gentzen's cut-free LJ [Gen69] with the following rules for equality and (co)inductive expressions.

$$\frac{\{(\Gamma \vdash G)\theta : \theta \in csu(s \doteq t)\}}{\Gamma, s = t \vdash G} = L \qquad \frac{}{\Gamma \vdash t = t} = R$$

$$\frac{BS\mathbf{x} \vdash S\mathbf{x} \quad \Gamma, S\mathbf{t} \vdash G}{\Gamma, \mu B\mathbf{t} \vdash G}\ \mu L \qquad \frac{}{\Gamma, \mu B\mathbf{t} \vdash \mu B\mathbf{t}}\ \mu_0 \qquad \frac{\Gamma \vdash B(\mu B)\mathbf{t}}{\Gamma \vdash \mu B\mathbf{t}}\ \mu R$$

$$\frac{\Gamma, B(\nu B)\mathbf{t} \vdash G}{\Gamma, \nu B\mathbf{t} \vdash G}\ \nu L \qquad \frac{}{\Gamma, \nu B\mathbf{t} \vdash \nu B\mathbf{t}}\ \nu_0 \qquad \frac{S\mathbf{x} \vdash BS\mathbf{x} \quad \Gamma \vdash S\mathbf{t}}{\Gamma \vdash \nu B\mathbf{t}}\ \nu R$$

We have observed (Prop. 2) that structural rules are admissible for fully asynchronous formulas of μMALL$^{=}$. This property will allow us to get a faithful encoding of a fragment of μLJ$^{=}$ in μMALL$^{=}$ despite the absence of exponentials. The encoding must be organized so that formulas appearing in the left-hand side of μLJ$^=$ sequents must be encoded as fully asynchronous μMALL$^=$ formulas. The only connectives allowed to appear negatively will thus be $\wedge$, $\vee$, $=$, μ and $\exists$. Moreover, the encoding must commute with negation, in order to translate the (co)induction rules correctly. This leaves no choice in the following design.

Definition 3. *We restrict formulas to two fragments described by the two syntactic variables $\mathcal{G}$ and $\mathcal{H}$:*

$$\begin{aligned}\mathcal{G} &::= \mathcal{G}\wedge\mathcal{G} \mid \mathcal{G}\vee\mathcal{G} \mid s=t \mid \mu(\lambda p\mathbf{x}.\mathcal{G}p\mathbf{x})\mathbf{t} \mid \exists x.\mathcal{G}x\\ &\quad\mid\ \forall x.\mathcal{G}x \mid \mathcal{H}\supset\mathcal{G} \mid \nu(\lambda p\mathbf{x}.\mathcal{G}p\mathbf{x})\mathbf{t}\\ \mathcal{H} &::= \mathcal{H}\wedge\mathcal{H} \mid \mathcal{H}\vee\mathcal{H} \mid s=t \mid \mu(\lambda p\mathbf{x}.\mathcal{H}p\mathbf{x})\mathbf{t} \mid \exists x.\mathcal{H}x\end{aligned}$$

Formulas in $\mathcal{H}$ and $\mathcal{G}$ are translated in μMALL$^=$ as follows:

$$\begin{aligned}[][P\wedge Q] &\stackrel{def}{=} [P]\otimes[Q]\\ [P\vee Q] &\stackrel{def}{=} [P]\oplus[Q]\\ [s=t] &\stackrel{def}{=} s=t\\ [\mu B\mathbf{t}] &\stackrel{def}{=} \mu[B]\mathbf{t}\\ [\exists x.Px] &\stackrel{def}{=} \exists x.[Px]\end{aligned}\qquad\begin{aligned}[][\forall x.Px] &\stackrel{def}{=} \forall x.[Px]\\ [\nu B\mathbf{t}] &\stackrel{def}{=} \nu[B]\mathbf{t}\\ [P\supset Q] &\stackrel{def}{=} [P]\multimap[Q]\\ [\lambda p\lambda\mathbf{x}.Bp x] &\stackrel{def}{=} \lambda p\lambda\mathbf{x}.[Bpx]\end{aligned}$$

Proposition 3. *For any $P\in\mathcal{G}$, P is provable in μLJ$^=$ if and only if $[P]$ is provable in μMALL$^=$, under the restrictions that (co)invariants $\lambda\mathbf{x}.S\,\mathbf{x}$ in μMALL$^=$ (resp. μLJ$^=$) are such that $S\,\mathbf{x}$ is in $[\mathcal{H}]$ (resp. $\mathcal{H}$).*

Proof. The proof transformations are simple and compositional. The induction rule is mapped to ν rule for $(\mu B\mathbf{t})^\perp$; the left unfolding for co-inductives to μ for $(\nu B\mathbf{t})^\perp$. In order to restore the additive behavior of some intuitionistic rules (*e.g.*, $\wedge R$) and translate the structural rules, we can contract and weaken our fully asynchronous formulas on the left of μLJ$^=$ sequents. □

Linear logic provides an appealing proof theoretic setting because of its emphasis on dualities and on its clear separation of concepts (additive/multiplicative, asynchronous/synchronous). Our experience is that μMALL$^=$ is a good place to study focusing in the presence of least and greatest fixed point operators. To get similar results for intuitionistic logic, one can either work from scratch entirely within, say, μLJ$^=$, or use an encoding into linear logic. Given a mapping from intuitionistic to linear logic, and a complete focused proof system for linear logic, one can often build a complete "focalized" proof-system for intuitionistic logic. The usual encoding of intuitionistic logic into linear logic involves exponentials, which can damage focusing structures (by causing both synchronous and asynchronous phases to end). Hence, a careful study of the polarity of linear connectives must be done (cf. [DJS93, LM07]) in order to minimize the role played by the exponentials in such encodings. Here, as a result of Proposition 3, it is possible to get a complete focused system for μLJ$^=$ on $\mathcal{G}$ (under the assumptions that (co)invariants are in $\mathcal{H}$) that inherits the strong structure of the linear focusing derivations.

Although $\mathcal{G}$ is not as expressive as full $\mu LJ^=$, it catches many interesting and useful problems. For example, any Horn-clause specification can be expressed in $\mathcal{H}$ as a least fixed point and theorems that state properties such as totality or functionality of predicates defined in this manner are in $\mathcal{G}$. Theorems that state more model-checking properties, for example, $\forall x.p(x) \supset q(x)$, where p and q are one-placed least fixed point expressions over $[H]$, are also in $\mathcal{G}$. Finally, the theorems about natural numbers presented in Section 4 are within $[\mathcal{G}]$ although two of the derivations (for the totality of *half* and that the sum of natural numbers is a natural number) do not satisfy the restriction on co-invariants.

The logic $\mu LJ^=$ is closely related to LINC [Tiu04]. The main difference is the absence of the ∇ quantifier in our system: we suspect that ∇ can be added to $\mu MALL^=$ in the same relatively orthogonal fashion that LINC added it to LJ. The resulting extension to $\mu MALL^=$ (and $\mu LJ^=$) should allow natural ways to reason about specifications involving variable bindings, in the manner illustrated in [BGM⁺07, Tiu04, Tiu05]. Another difference is that fixed points in LINC have to satisfy a stratification condition, which is strictly stronger than monotonicity; co-invariants also have to satisfy a technical restriction related to stratification. While our system, derived from linear logic, does not share such restrictions, neither difference is relevant when we restrict our attention to formulas in $\mathcal{G}$.

Interestingly, the fragment $\mathcal{G}$ has already been identified in LINC [TNM05], and the Bedwyr system [BGM⁺07] implements a proof-search strategy for it that is complete under the assumption that all fixed points are noetherian (and hence that least and greatest fixed points coincide and that (co)induction can be restricted to unfolding). This strategy coincides with the focused system for $\mu LJ^=$ restricted to noetherian fixed points: there is no need for any explicit contraction and you can always eagerly eliminate left-hand side (asynchronous) connectives before working on the goal (right-hand side); moreover there is no need for the initial rule $\mu\nu$.

6 Discussion About the Focusing System

The design of the above focused proof system for $\mu MALL^=$ is rather satisfactory. For example, its treatment of μ as synchronous and ν as asynchronous is consistent with a similar treatment of these operators via game semantics given in [MS05, Sti96]. Focusing is also natural and helpful when trying to prove theorems in $\mu MALL^=$, such as the examples proposed in Section 4. Finally, as we have seen in Section 5, this focused proof system yields another one for an intuitionistic logic similarly extended with fixed points, and accounts for the proof search strategy underlying the implemented prover Bedwyr [BGM⁺07]. It is worth noting, however, two unusual aspects of focused proofs in $\mu MALL^=$.

6.1 A Choice Inside Asynchronous Rules

As we noted, there are two rules for each of the fixed point connectives. Having a choice of rules in the asynchronous phase is, at first, rather surprising since it is during this phase of proof construction that we expect to see invertible rules and no choices. One way to look at this is that, in fact, the ν-connective should be *annotated* or divided into an infinite number of different connectives. In particular, consider replacing the ν constructor with both ν_ϵ (with the same types and arity as ν) and ν_S (where S is an annotated formula abstraction of the appropriate type). Now consider the proof system that results from replacing the three rules involving ν in Figure 2 by the rules

$$\frac{\vdash \Gamma \Uparrow S\,\mathbf{t}, \Delta \quad \vdash \Uparrow BS\,\mathbf{x}, S\,\mathbf{x}^\perp}{\vdash \Gamma \Uparrow \nu_S B\mathbf{t}, \Delta}\ \mathbf{x}\text{ new} \qquad \frac{\vdash \Gamma, \nu_\epsilon B\mathbf{x} \Uparrow \Delta}{\vdash \Gamma \Uparrow \nu_\epsilon B\mathbf{x}, \Delta} \qquad \frac{}{\vdash \nu_\epsilon \overline{B}\mathbf{x} \Downarrow \mu B\mathbf{x}}$$

Notice that using such annotated formulas, there is no longer any choice in the asynchronous phase. Furthermore, if in the expression $\nu_S B$ it is really the case that S is a co-invariant, *i.e.*, $(BS\,\mathbf{x}, S\,\mathbf{x}^\perp)$ is provable, then the first inference rule is invertible.

From a focused proof of F, it is possible to extract an annotation of F that is provable in the disambiguated focused system. This extraction requires the non-trivial composition of co-invariants in a manner similar to that used for the permutation of ν and &. Such annotations might be useful for the partial automation of proof search involving induction and co-induction. For example, ν connectives could be labeled with partial information about what to do with the connective in the asynchronous phase: unfold, freeze (*i.e.*, treat as atomic), use the sequent as the invariant, etc. Such hints might be enough to mechanize a large amount of simple but tedious proofs by (co)induction. Notice that since we have annotated ν but not μ, we should not think that ν's with annotations are logical connectives: instead, such annotations hint at the structure of a particular proof involving that annotated expression.

6.2 Are the Polarities of μ and ν Forced?

While the classification of μ as synchronous and ν as asynchronous is rather satisfying and is backed by several other observations, that choice does not seem to be forced from the focusing point of view alone. Maybe μ can be handled in the asynchronous phase, instead? After all the μ rule is invertible. Consider replacing the fixed point rules in the focused proof system in Figure 2 with the following four inference rules:

$$\frac{\vdash \Gamma \Uparrow B(\mu B)\mathbf{t}, \Delta}{\vdash \Gamma \Uparrow \mu B\mathbf{t}, \Delta} \qquad \frac{\vdash \Gamma, \mu B\mathbf{t} \Uparrow \Delta}{\vdash \Gamma \Uparrow \mu B\mathbf{t}, \Delta} \qquad \frac{\vdash \Gamma \Downarrow S\,\mathbf{t} \quad \vdash \Uparrow BS\,\mathbf{x}, (S\,\mathbf{x})^\perp}{\vdash \Gamma \Downarrow \nu B\mathbf{t}} \qquad \frac{}{\vdash \mu \overline{B}\mathbf{t} \Downarrow \nu B\mathbf{t}}$$

We conjecture that the resulting proof system is complete for μMALL$^=$. The non-trivial step in such a proof would involve the permuting of the inference rules for μ and &. The invertibility of μ allows it, but we have not proved the termination of the whole transformation.

To go one step further, one wonders if arbitrary assignment of "bias" to expressions such as $(\mu B\mathbf{t})$ and $(\nu B\mathbf{t})$ can be made in a fashion similar to the way literals are given fixed but arbitrary "bias" in Andreoli's original focused proof system [And92]. Thus, maybe some μ expressions can be synchronous while others are asynchronous.

7 Conclusion and Future Work

μMALL$^=$ is an elegant logic supporting reasoning on inductive and co-inductive specifications. We have shown that it has two important proof-theoretic properties: namely, cut-elimination and the completeness of focused proofs. The design and completeness of a focused proof system is the major contribution of this paper. We have also shown that $\mu\text{MALL}^=$ is expressive and formally connected it to a fragment of intuitionistic logic extended with fixed points, a step that brings $\mu\text{MALL}^=$ closer to applications. Finally, we have identified an implemented system that attempts to find focused proofs within the noetherian part of this logic.

There are a number of interesting open questions to consider next. At the proof theory level, we would like to understand better whether or not dropping the monotonicity requirement leads to inconsistency or not and to what extent we can provide alternative assignment of polarities (synchronous/asynchronous) to fixed points. We can also consider adding exponentials and atomic formulas to $\mu\text{MALL}^=$ so that all of $\mu\text{LJ}^=$ could be encoded (in which case, a precise connection to the focused proof systems of [LM07] should be explored). Such an extension to $\mu\text{MALL}^=$ could also be used to generalize the uses of induction in the linear logic programming setting of [PM05]. At the system designing and implementation level, our focused proof system should help in designing a logic engine that attempts to prove formulas involving induction and co-induction. Our hope is that the focused proof system would help in understanding the strengths and limitations of various heuristics for generating invariants and co-invariants.

Acknowledgments. We thank Alexis Saurin for helpful discussions and the anonymous reviewers of a previous draft of this paper for their comments, which helped us to reorganize this paper. This work has been supported in part by INRIA through the "Equipes Associées" Slimmer and by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2005-015905 MOBIUS project.

References

[And92] Andreoli, J.-M.: Logic programming with focusing proofs in linear logic. J. of Logic and Computation 2(3), 297–347 (1992)

[AP91] Andreoli, J.M., Pareschi, R.: Linear objects: Logical processes with built-in inheritance. New Generation Computing 9(3-4), 445–473 (1991)

[AvE82] Apt, K.R., van Emden, M.H.: Contributions to the theory of logic programming. J. of the ACM 29(3), 841–862 (1982)

[BGM$^+$07] Baelde, D., Gacek, A., Miller, D., Nadathur, G., Tiu, A.: The Bedwyr system for model checking over syntactic expressions. In: Pfenning, F. (ed.) 21th Conference on Automated Deduction. LNCS (LNAI), vol. 4603, pp. 391–397. Springer, Heidelberg (2007)

[BM07] Baelde, D., Miller, D.: Least and greatest fixed points in linear logic: extended version. Technical report, available from the first author's web page (April 2007)

[DJS93] Danos, V., Joinet, J.-B., Schellinx, H.: The structure of exponentials: Uncovering the dynamics of linear logic proofs. In: Mundici, D., Gottlob, G., Leitsch, A. (eds.) KGC 1993. LNCS, vol. 713, pp. 159–171. Springer, Heidelberg (1993)

[Gen69] Gentzen, G.: Investigations into logical deductions. In: Szabo, M.E. (ed.) The Collected Papers of Gerhard Gentzen, North-Holland, Amsterdam, pp. 68–131 (1969)

[Gir87] Girard, J.-Y.: Linear logic. Theoretical Computer Science 50, 1–102 (1987)

[Gir92] Girard, J.-Y.: A fixpoint theorem in linear logic. An email posting to the mailing list linear@cs.stanford.edu. (February 1992)

[Gir93] Girard, J.-Y.: On the unity of logic. Annals of Pure and Applied Logic 59, 201–217 (1993)

[Gir98] Girard, J.-Y.: Light linear logic. Information and Computation 143 (1998)

[Gir01] Girard, J.-Y.: Locus solum. Mathematical Structures in Computer Science 11(3), 301–506 (2001)

[HM94] Hodas, J., Miller, D.: Logic programming in a fragment of intuitionistic linear logic. Information and Computation 110(2), 327–365 (1994)

[Laf04] Lafont, Y.: Soft linear logic and polynomial time. Theoretical Computer Science 318(1-2), 163–180 (2004)

[LM07] Liang, C., Miller, D.: Focusing and polarization in intuitionistic logic. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 451–465. Springer, Heidelberg (2007)

[Mil96] Miller, D.: Forum: A multiple-conclusion specification logic. Theoretical Computer Science 165(1), 201–232 (1996)

[MM00] McDowell, R., Miller, D.: Cut-elimination for a logic with definitions and induction. Theoretical Computer Science 232, 91–119 (2000)

[MNPS91] Miller, D., Nadathur, G., Pfenning, F., Scedrov, A.: Uniform proofs as a foundation for logic programming. Annals of Pure and Applied Logic 51, 125–157 (1991)

[MS05] Miller, D., Saurin, A.: A game semantics for proof search: Preliminary results. In: Proceedings of the Mathematical Foundations of Programming Semantics (MFPS) (2005)

[MS07] Miller, D., Saurin, A.: From proofs to focused proofs: a modular proof of focalization in linear logic. In: Duparc, J., Henzinger, T.A. (eds.) CSL 2007. LNCS, vol. 4646, pp. 405–419. Springer, Heidelberg (2007)

[MT03] Momigliano, A., Tiu, A.: Induction and co-induction in sequent calculus. In: Berardi, S., Coppo, M., Damiani, F. (eds.) TYPES 2003. LNCS, vol. 3085, pp. 293–308. Springer, Heidelberg (2004)

[PM05] Pimentel, E., Miller, D.: On the specification of sequent systems. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 352–366. Springer, Heidelberg (2005)

[SH93] Schroeder-Heister, P.: Rules of definitional reflection. In: Vardi, M. (ed.) Eighth Annual Symposium on Logic in Computer Science, pp. 222–232. IEEE Computer Society Press, Los Alamitos (1993)

[Sti96] Stirling, C.: Games for bisimulation and model checking. Notes for Mathfit Workshop on Finite Model Theory, University of Wales, Swansea (July 1996)

[Tiu04] Tiu, A.: A Logical Framework for Reasoning about Logical Specifications. PhD thesis, Pennsylvania State University (May 2004)

[Tiu05] Tiu, A.: Model checking for π-calculus using proof search. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 36–50. Springer, Heidelberg (2005)

[TNM05] Tiu, A., Nadathur, G., Miller, D.: Mixing finite success and finite failure in an automated prover. In: Proceedings of ESHOL 2005: Empirically Successful Automated Reasoning in Higher-Order Logics, pp. 79–98 (December 2005)

The Semantics of Consistency and Trust in Peer Data Exchange Systems

Leopoldo Bertossi[1] and Loreto Bravo[2]

[1] Carleton University, School of Computer Science, Ottawa, Canada
`bertossi@scs.carleton.ca`
[2] University of Edinburgh, School of Informatics, Edinburgh, UK
`lbravo@inf.ed.ac.uk`

Abstract. We propose and investigate a semantics for *peer data exchange systems* (or peer data management systems) where different peers are pairwise related to each other by means of data exchange constraints and trust relationships. These two elements plus the data at the peers' sites and the local integrity constraints for a peer are made compatible via the proposed semantics by determining a set of *solution instances*, which are the intended virtual instances for the peer. The semantically correct answers from a peer to a query, called its *peer consistent answers*, are defined as those answers that are invariant under all its different solution instances. We show that solution instances can be specified as the models of logic programs with a stable model semantics.

1 Introduction

A peer data exchange system (PDES) consists of a finite set of peers $\{\mathtt{P}_1, \ldots \mathtt{P}_n\}$, each of them with a local database instance. Peers may be pairwise related by means of logical sentences, called [illegible] (DECs), which are expressed in terms of the participating schemas and are expected to be satisfied by the combined data. Furthermore, a peer P may trust its data the same as or less than other peers' data, i.e. there may be trust relationships between pairs of peers. We may also have integrity constraints (ICs) that are local to each peer.

The DECs could be seen as ICs on a global database obtained by conceptually putting together all the peers' schemas and data. Most likely, these DECs will not be satisfied in this global database, but virtually enforcing their satisfaction at query time has the effect of shipping (sub)queries and data between peers. Actually, in such a PDES, a query $\mathcal{Q}$ is posed to a peer P, who, in order to answer it, may need to consider both its own data and the data stored at other peers' sites that are related to P by DECs. Keeping P's DECs satisfied at query time may imply getting data from other peers to complement P's data, but also not using part of its own data.

The decision by a peer P on what other data to consider does not depend only on its DECs, but also on the [illegible] that P has with other peers. For example, if peer P trusts peer Q's data more than its own, P will accommodate its data to Q's in order to keep the DECs between them satisfied.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 107–122, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In order for P to return meaningful answers, its local semantic constraints have to be taken into account. The consistency of its instance should be preserved when it is virtually updated due to the interaction with other peers. In consequence, still at query time, P may also need to "virtually repair" its data.

Our semantics makes all these elements compatible, by defining a set of virtual global instances called [illegible] (or simply solutions) for a peer. In consequence, the "data" for a peer, the one in its solution instances, depends upon its instance, the local instances of the related peers, the satisfaction of the DECs, and the satisfaction by P of its local ICs. After that, having a precise definition of the intended solution instances for a peer P, the [illegible] (PCAs) from peer P to query $\mathcal{Q}$ are defined as those answers that can be retrieved from [illegible] possible solution for P.

The definition of solution for P may suggest that P may physically change other peers' data, but this is not the case. The notion of solution is used as an auxiliary notion to characterize the correct answers from P's point of view. Ideally, P should be able to obtain its peer consistent answers just by querying the already available local instances. This resembles the approach to [illegible] (CQA) in databases, where consistent answers to a query posed to a database, which is possibly inconsistent wrt to a given set of ICs, are defined as those answers that can be retrieved from every minimally repaired version of the original instance. Methods have been developed for computing these answers without having to physically repair the given instance [1, 9, 10].

Our work goes in the direction of semantic approaches to peer-to-peer data exchange [27, 29, 28, 14, 21, 7, 15, 23]. In [7] trust relationships were introduced for the first time in this scenario and the notions of solution instance for a peer and of peer consistent answer to a query were introduced. The case of a peer and its immediate neighbors was considered and investigated. However, the situation where a peer is related by logical transitivity via DECs to other, non neighboring peers was not modelled. Actually, in [7] it was only indicated that giving a semantics to solution instances for a peer that consider the transitive relationships to other peers could be done by using logic programs with stable model semantics. We fully develop this idea here. First, we provide a general model-theoretic definition of solutions for a peer, including the transitive case, and next, we specify the solutions as the models of disjunctive logic programs with stable model semantics [24], aka. [illegible] [25, 3]. An extended version of this paper can be found in [8].

Logic programs can capture the different ways the system stabilizes after satisfying the DECs, the trust relationships, and the local ICs. Disjunctive programs allow for the specification of alternative virtual updates on data sources under certain conditions. We propose appropriate logic programs and then we establish that there is a one-to-one correspondence between the set of solution instances for a peer and the set of stable models of the program. Logic programs provide an expressive language for representing and specifying the alternative solutions for a peer, and become executable specifications for computing peer consistent answers. This approach has been exploited in CQA, where database repairs are specified as stable models of a program [2, 26, 4, 17].

2 A Semantics for PDESs

We consider peers that have mutually disjoint relational schemas; but all of them share a possibly infinite database domain $\mathcal{U}$. Peers are denoted by `A`, `B`, `P`, `Q`, ...

Definition 1. A [illegible] $\mathfrak{P}$ consists of: (a) A finite set $\mathcal{P}$ of peers, with each peer `P` owning a relational database schema $\mathcal{R}(\mathtt{P})$, and a database instance $D(\mathtt{P})$ conforming to schema $\mathcal{R}(\mathtt{P})$. The schemas determine FO languages, e.g. $\mathcal{L}(\mathtt{P})$, $\mathcal{L}(\mathtt{P},\mathtt{Q})$. (b) For each peer `P`, collections $\Sigma(\mathtt{P},\mathtt{Q})$ of sentences of $\mathcal{L}(\mathtt{P},\mathtt{Q})$, which contain the DECs between `P` and a peer `Q`. Here, $\Sigma(\mathtt{P}) := \bigcup_{\mathtt{Q}} \Sigma(\mathtt{P},\mathtt{Q})$ and $\Sigma := \bigcup_{\mathtt{P}\in\mathcal{P}} \Sigma(\mathtt{P})$. (c) For each peer `P`, a set of $\mathcal{L}(\mathtt{P})$-sentences [illegible](`P`) that are ICs on $\mathcal{R}(\mathtt{P})$. Here, $IC = \bigcup_{\mathtt{P}\in\mathcal{P}} IC(\mathtt{P})$. (d) A relation [illegible] $\subseteq \mathcal{P} \times \{$[illegible]$\} \times \mathcal{P}$, with exactly one triple of the form $\langle \mathtt{P}, \cdot, \mathtt{Q}\rangle$ for each non empty $\Sigma(\mathtt{P},\mathtt{Q})$. □

The intended semantics of (`A`, [illegible], `B`) $\in$ [illegible] is that peer `A` trusts itself less than `B`; while (`A`, [illegible], `B`) $\in$ [illegible] indicates that `A` trusts itself the same as `B`.

Definition 2. (a) A [illegible] (UDEC) between peers `P1`, `P2` is a first-order (FO) formula of form:

$$\forall \bar{x}\Big(\bigwedge_{i=1}^{n} R_i(\bar{x}_i) \;\longrightarrow\; \big(\bigvee_{j=1}^{m} Q_j(\bar{y}_j) \vee \varphi\big)\Big); \tag{1}$$

where the R_i, Q_j are relations in $\mathcal{R}(\mathtt{P1}) \cup \mathcal{R}(\mathtt{P2})$, φ is a formula containing built-in atoms[1] only, and $\bar{x}_i, \bar{y}_j \subseteq \bar{x}$. (b) A [illegible] (RDEC) between peers `P1`, `P2` is an $\mathcal{L}(\mathtt{P1},\mathtt{P2})$-sentence of the form:

$$\forall \bar{x}(R(\bar{x}) \longrightarrow \exists \bar{y}\; Q(\bar{x}', \bar{y})); \tag{2}$$

where $R, Q \in \mathcal{R}(\mathtt{P1}) \cup \mathcal{R}(\mathtt{P2})$, and $\bar{x}' \subseteq \bar{x}$. □

Notice that the sets of DECs $\Sigma(\mathtt{P1},\mathtt{P2})$ and $\Sigma(\mathtt{P2},\mathtt{P1})$ can be different. When exchanging or repairing data, the existential quantifier in RDECs will be interpreted as a null value. By having one database atom in the consequent, we avoid existential joins that, when filled with nulls, do not have a clear semantics (cf. [12] for a discussion and a FO semantics of nulls in SQL databases).[2]

[illegible] Consider a PDES $\mathfrak{P}$ with four peers and $\mathcal{R}(\mathtt{P1}) = \{R^1(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P2}) = \{R^2(\cdot,\cdot), S^2(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P3}) = \{R^3(\cdot,\cdot)\}$, $\mathcal{R}(\mathtt{P4}) = \{R^4(\cdot,\cdot,\cdot)\}$, and DECs:

$\Sigma(\mathtt{P1},\mathtt{P2}) = \{\forall xy(R^2(x,y) \rightarrow R^1(x,y))\}$,
$\Sigma(\mathtt{P2},\mathtt{P3}) = \{\forall xy(R^2(x,y) \wedge R^3(x,y) \rightarrow \textbf{false})\}$,
$\Sigma(\mathtt{P4},\mathtt{P2}) = \{\forall xyz(R^2(x,y) \wedge S^2(y,z) \rightarrow R^4(x,y,z))\}$,
$\Sigma(\mathtt{P4},\mathtt{P3}) = \{\forall xy(R^3(x,y) \rightarrow \exists z R^4(x,y,z))\}$.

---

[1] For example, $x \neq 3$, $y = z$ and $z > 3$.

[2] Our framework can be easily adapted to cases where, instead of *null*, one uses arbitrary elements of the database domain [13] or labelled nulls [30].

Here **false** in the second DEC is a built-in atom that is false in every instance, so the DEC specifies that relations R^2 and R^3 are disjoint. The DECs in $\Sigma(\texttt{P1},\texttt{P2})$, $\Sigma(\texttt{P2},\texttt{P3})$ and $\Sigma(\texttt{P4},\texttt{P2})$ are UDECs and the one in $\Sigma(\texttt{P4},\texttt{P3})$ is a RDEC. Finally, we could have [illegible] = {(P1, [illegible], P2), (P2, [illegible], P3), (P4, [illegible], P2), (P4, [illegible], P3)}. Peer P1 trusts P2 more than itself, and it will import all the data from R^2 into its table R^1. On the other hand, peer P2 trusts peer P3 as much as itself, and its DEC states that it is not possible to have the same tuple in both R^3 and R^2. □

Local ICs [illegible](P) are also of the form in Definition 2, but with all database predicates in $\mathcal{R}(\texttt{P})$. So, we can identify [illegible](P) with $\Sigma(\texttt{P},\texttt{P})$. All the common ICs found in database practice can be accommodated into these syntactic classes. In particular, using the atom **false**, denial constraints are of the form (1). In Example 1, we could have [illegible](P2) $= \{\forall x \forall y(R^2(x,y) \wedge S^2(x,y) \rightarrow \textbf{false})\}$.

Each peer P is responsible for maintaining its material instance consistent with respect to its ICs [illegible](P), independently from other peers; and we may assume that this is the case. However, our semantics works the same if we allow peers with locally inconsistent data. According to our semantics presented below, unsatisfied local ICs are considered when solution instances are specified, which is done by using a repair semantics and techniques developed for CQA [10, 6].

When a peer P is posed a query, it may have to submit queries to other peers according to its DECs, using data in other peers' relations appearing in them. Data brought from other peers will possibly cause virtual updates on P's data, which may create virtual violations of P's local ICs, which has to be considered.

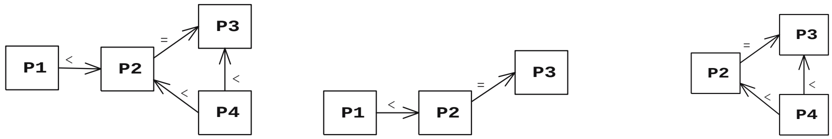
From the perspective of a peer P, its own database may be inconsistent with respect to the data owned by another peer Q it trusts more or the same, and the DECs in $\Sigma(\texttt{P},\texttt{Q})$. When P queries its own database, the answers from P should be consistent with $\Sigma(\texttt{P},\texttt{Q})$ and its own ICs [illegible](P). In principle, P, which is not allowed to change other peers' data, could try to physically repair its database in order to satisfy $\Sigma(\texttt{P}) \cup$ [illegible](P). This is not a realistic approach. Rather, P should solve its semantic conflicts at query time. This leads to a set of virtual instances, the minimal repairs of P's local database, where P's DECs and [illegible]s are satisfied, while respecting P's trust relationships to other peers. The answers returned by P to the user are those that are true of all these instances.

The solution instances of a peer will be determined not only by its relationships with its neighbors, but also by the neighbors of its neighbors, etc.

Definition 3. (a) The [illegible] $\mathcal{G}(\mathfrak{P})$ of a PDES $\mathfrak{P}$ contains a vertex for each peer $\texttt{P} \in \mathcal{P}$ and a directed edge from Pi to Pj if $\Sigma(\texttt{Pi},\texttt{Pj})$ is non empty. An edge from Pi to Pj is labelled with "<" when (Pi, [illegible], Pj) ∈ [illegible], and with "=" when (Pi, [illegible], Pj) ∈ [illegible].[3] (b) Peer P' is [illegible] from P if there is a path in $\mathcal{G}(\mathfrak{P})$ from P to P' or if P'=P. Peer P' is a [illegible] of P if there is an edge from P to P' in $\mathcal{G}(\mathfrak{P})$, or if P' = P. With $\mathcal{AC}(\texttt{P})$ and $\mathcal{N}(\texttt{P})$ we denote the sets of peers that are accessible from P and the neighbors of P, respectively. For $\texttt{P} \in \mathcal{P}$, $\mathcal{G}(\texttt{P})$ is the restriction of $\mathcal{G}(\mathfrak{P})$ to $\mathcal{AC}(\texttt{P})$. □

---

[3] In case a peer P trusts itself more than another peer, the information of the latter is irrelevant to P.



(a) $\mathcal{G}(\mathfrak{P})$ (b) $\mathcal{G}(\texttt{P1})$ (c) $\mathcal{G}(\texttt{P4})$

Fig. 1. Graphs for Example 2

[illegible] (Example 1 continued) In this system, $\mathcal{AC}(\texttt{P1}) = \{\texttt{P1},\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{AC}(\texttt{P3}) = \{\texttt{P3}\}$, $\mathcal{AC}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$, $\mathcal{N}(\texttt{P1}) = \{\texttt{P1},\texttt{P2}\}$, $\mathcal{N}(\texttt{P2}) = \{\texttt{P2},\texttt{P3}\}$, $\mathcal{N}(\texttt{P3}) = \{\texttt{P3}\}$, and $\mathcal{N}(\texttt{P4}) = \{\texttt{P2},\texttt{P3},\texttt{P4}\}$. □

The data distributed across different peers has to be appropriately gathered to build solution instances for a peer, and different semantics may emerge as candidates, depending on the granularity of the data sent between peers. We develop one of them,[4] according to which, the data that a peer `P` receives from a neighbor `Q` to build its own solutions is the [illegible] for `Q`. After `P` collects this data, only `P`'s DECs and ICs are considered. This is a recursive definition since the solutions for the neighbors have to be determined, under the same semantics. Base cases of the recursion are peers with no relevant DECs. In consequence, this semantics requires an acyclic accessibility graph.

In [27] problematic cases involving cyclic dependencies through DECs are identified, which implicitly involve a cyclic accessibility graph. For example, we may have a PDES $\mathfrak{P}$ with $\mathcal{P} = \{\texttt{P1},\texttt{P2},\texttt{P3}\}$, with relations $R^1(\cdot), R^2(\cdot), R^3(\cdot)$, resp., and DECs $\Sigma(\texttt{P1}) = \{\forall x(R^2(x) \rightarrow R^1(x))\}$, $\Sigma(\texttt{P2}) = \{\forall x(R^3(x) \rightarrow R^2(x))\}$, $\Sigma(\texttt{P3}) = \{\forall x(R^1(x) \rightarrow R^3(x))\}$, each of them satisfied only by importing data into the peer who owns the DEC. The implicit trust relation $\{(\texttt{P1},$ [illegible] $,\texttt{P2}), (\texttt{P2},$ [illegible] $,\texttt{P3}), (\texttt{P3},$ [illegible] $,\texttt{P1})\}$ makes $\mathcal{AC}(\mathfrak{P})$ cyclic. In [27] it is assumed that no cycles of this kind appear. In the following, we will also assume that $\mathcal{G}(\mathfrak{P})$ is acyclic, and then for each particular peer `P`, $\mathcal{G}(\texttt{P})$ is acyclic.

Null values will be used to satisfy referential DECs and local referential ICs; and the repair semantics based on introduction of null values, presented and developed in [12, 11] for single relational databases and RICs, can be adapted here, but now taking into account the trust relationships. Data sources at the peers' sites may contain null values that, as those used to satisfy referential constraints, will have a semantics that corresponds to the way nulls are handled by DBMSs that follow the SQL standard. In particular, there is only one constant, [illegible], that is used as the null value.[5] We use a semantics for IC satisfaction in the presence of nulls that generalizes the one implemented in DBMSs, and coincides with the first-order notion of formula satisfaction in databases without nulls [11].

---

[4] In [11] also two other alternative semantics are fully developed and compared, in particular establishing some conditions under which they coincide or differ. The other semantics assume that more detailed information, such as mappings and trust relationships, can be sent between peers.

[5] This null is obviously different from the multiple labelled null values that are considered in data exchange for satisfying existential quantifiers (cf. [30] for a survey).

A formal development of the notion of constraint satisfaction, denoted $\models \psi$, with D possibly containing *null* and ψ a constraint, can be found in [12, 11] (cf. [8, appendix B] for a review). What matters most for the rest of this paper is that the satisfaction of a constraint ψ depends upon the presence of *null* in attributes of database relations which are also *relevant attributes* for ψ.

Solutions for a peer should stay close to its original physical instance while satisfying the DECs and local ICs. We do not want to import or give up more data than strictly required to satisfy the constraints. To formalize this idea, we first need to compare tuples that may contain *null*. A constant c provides *less or equal information* than a constant d, denoted $c \sqsubseteq d$, iff c is *null* or $c = d$ [31]. A tuple $\bar{t}_1 = (c_1, \ldots, c_n)$ provides less or equal information than $\bar{t}_2 = (d_1, \ldots, d_n)$, denoted $\bar{t}_1 \sqsubseteq \bar{t}_2$, iff $c_i \sqsubseteq d_i$ for every $i = 1, \ldots, n$. Finally, $\bar{t}_1 \sqsubset \bar{t}_2$ means $\bar{t}_1 \sqsubseteq \bar{t}_2$ and $\bar{t}_1 \neq \bar{t}_2$. In the following, a database instance is identified with a finite set of ground database atoms; and $\Delta(\cdot, \cdot)$ denotes the symmetric difference of sets.

Definition 4. Let D, D', D'' be database instances for the same schema. It holds that $D' \leq_D D''$ iff for every $P(\bar{a}) \in \Delta(D, D')$, there exists $P(\bar{a}')$, such that: (a) $P(\bar{a}') \in \Delta(D, D'')$; (b) $\bar{a} \sqsubseteq \bar{a}'$; and (c) if $\bar{a} \sqsubset \bar{a}'$, then $P(\bar{a}') \notin \Delta(D, D')$. Finally, $D'' <_D D'$ means $D'' \leq_D D'$ but not $D' \leq_D D''$. □

If $D' \leq_D D''$, we say that D' is closer to D than D''. Condition (c) in Def. 4 ensures that a database that adds to D a tuple with *null* is closer to D than other that adds other constant. This condition will later ensure that the satisfaction of RDECs is enforced by using *null*.

For an instance D of a schema $\mathcal{S}$, and $\mathcal{S}'$ a subschema of $\mathcal{S}$, $D|\mathcal{S}'$ denotes the restriction of D to $\mathcal{S}'$. Thus, if R is a predicate in $\mathcal{S}$ and D is an instance for $\mathcal{S}$, $D|\{R\}$ denotes the extension of R in D. If $\mathcal{R}(\texttt{P}) \subseteq \mathcal{S}$, $D|\texttt{P}$ is the restriction of D to $\mathcal{R}(\texttt{P})$. A neighborhood solution for $\texttt{P}$ and a database for its whole neighborhood is a closest database that satisfies $\texttt{P}$'s DECs, ICs, and trust relationships.

Definition 5. Given a peer $\texttt{P}$ in a PDES $\mathfrak{P}$ and instances D, D' on schema $\bigcup_{\texttt{Q} \in \mathcal{N}(\texttt{P})} \mathcal{R}(\texttt{Q})$, D' is a *neighborhood solution for* P *on* D if : (a) $D' \models \bigcup_{\texttt{Q} \in \mathcal{N}(\texttt{P})} \Sigma(\texttt{P}, \texttt{Q}) \cup IC(\texttt{P})$. (b) $D'|\{R\} = D|\{R\}$ for every predicate $R \in \mathcal{R}(\texttt{Q})$ with $(\texttt{P}, less, \texttt{Q}) \in trust$. (c) There is no instance D'' that satisfies (a) and (b), and such that $D'' <_D D'$. □

We do not require in (a) $IC(\texttt{Q})$ to be satisfied, because $\texttt{Q}$ will move data to $\texttt{P}$'s site, where inconsistencies will be solved locally, according to Definition 6, where $S(\texttt{P})$ denotes the set of solutions for peer $\texttt{P}$.

Definition 6. Given a peer $\texttt{P}$ in a PDES $\mathfrak{P}$ with local instance $D(\texttt{P})$, an instance D over $\mathcal{R}(\texttt{P})$ is a *solution* for $\texttt{P}$ if: (a) $D = D(\texttt{P})$ and $\Sigma(\texttt{P}) = \emptyset$; or (b) $\Sigma(\texttt{P}) \neq \emptyset$, $D = \overline{D}|\texttt{P}$ where $\overline{D}$ is a neighborhood solution for $\texttt{P}$ and the database instance $D(\texttt{P}) \cup \bigcup_{\texttt{Q} \in (\mathcal{N}(\texttt{P}) \smallsetminus \{\texttt{P}\})} \bigcap_{I \in S(\texttt{Q})} I$ over schema $\bigcup_{\texttt{Q} \in \mathcal{N}(\texttt{P})} \mathcal{R}(\texttt{Q})$. □

Intuitively, before constructing $\texttt{P}$'s solutions, $\texttt{P}$ has its local instance $D(\texttt{P})$ and each of its neighbors has as local instance the intersection of its own solutions. This produces a combined database. After that, the solutions for $\texttt{P}$ are obtained by restricting to $\texttt{P}$ the neighborhood solutions for the combined instance. The neighborhood solution captures the minimal virtual updates that are necessary to satisfy the DECS and local ICs. As there may be several neighborhood solutions, several solutions for a peer are possible.

Example 3. (Example 1 and 2 continued) Consider the following instances of peers P1, P2 and P3 : $D(\texttt{P1}) = \{R^1(a,2)\}$, $D(\texttt{P2}) = \{R^2(c,4), R^2(d,5)\}$, $D(\texttt{P3}) = \{R^3(c,4)\}$ and $D(\texttt{P4}) = \{R^4(d,5,1)\}$. If we want the solutions for P1, the solutions for P2 are needed, who will need in turn the solutions for P1. Since P3 has no DECs with other peers, its only neighborhood solution is it local instance $D(\texttt{P3})$. This data is sent back to P2, who needs to repair $\{R^2(c,4), R^2(d,5), R^3(c,4)\}$ now wrt $\Sigma(\texttt{P2},\texttt{P3})$. As P2 trusts P3 the same as itself, it can modify its own data or the data it got from P3. There are two neighborhood solutions for P2: $\{R^2(c,4), R^2(d,5)\}$ and $\{R^2(d,5), R^3(c,4)\}$, that lead to two solutions for P2: $\{R^2(c,4), R^2(d,5)\}$ and $\{R^2(d,5)\}$. Peer P2 will send to P1 the intersection of its solutions: $\{R^2(d,5)\}$. Now, P1 has to repair $\{R^1(a,2), R^2(d,5)\}$ wrt $\Sigma(\texttt{P1},\texttt{P2}) = \{\forall xy\ (R^2(x,y) \rightarrow R^1(x,y))\}$. Since P1 trusts its own data less than the data of P2, it will solve inconsistencies by modifying its own data. There is only one neighborhood solution, $\{R^1(a,2), R^2(d,5), R^1(d,5)\}$, and the solution for P1 is $\{R^1(a,2), R^1(d,5)\}$.

To compute the solutions for P4, the solutions of P2 and P3 are computed as shown before. Neighborhood solutions for P4 are obtained by repairing $\{R^4(d,5,1), R^2(d,5), R^3(c,4)\}$ wrt $\Sigma(\texttt{P4},\texttt{P2})$, and $\Sigma(\texttt{P4},\texttt{P3})$. The DECs in $\Sigma(\texttt{P4},\texttt{P2})$ are already satisfied, but not the ones in $\Sigma(\texttt{P4},\texttt{P3})$. Since P4 trusts the data in P3 more, a repair is obtained by adding a tuple with *null* into P4. The unique neighborhood solution for P4 is $\{R^4(d,5,1), R^2(d,5), R^3(c,4), R^4(c,4,\mathit{null})\}$. Consequently, $S(\texttt{P4}) = \{\{R^4(d,5,1), R^4(c,4,\mathit{null})\}\}$. □

The peer consistent answers are the semantically correct answers to a query returned by a peer who consistently considers the data of- and trust relationships with its neighbors.

Definition 7. Given a FO query $\mathcal{Q}(\bar{x}) \in \mathcal{L}(\texttt{P})$ posed to P, a ground tuple $\bar{t}$ is a *peer consistent* answer (PCA) to $\mathcal{Q}$ from P iff $D \models \mathcal{Q}(\bar{t})$ for every solution instance D for P. □

Example 4. (Example 3 continued) If P2 is posed the query $\mathcal{Q}\colon\ R^2(x,y)$, from its first solution instance we get $\{(c,4),(d,5)\}$, and from the second, $\{(d,5)\}$. Therefore, the only PCA from P2 is $\{(d,5)\}$. □

Even in the absence of cycles in $\mathcal{G}(\mathfrak{P})$, there may be no solutions for a peer. Furthermore, still with acyclic $\mathcal{G}(\mathfrak{P})$, [illegible], i.e. deciding if a tuple is a PCA to a query, may be undecidable if consistency wrt RDECs is achieved using arbitrary values in the domain.[6] However, using *null* instead avoids this problem, making the problem decidable. Actually, by reduction from CQA to PCA and known results on the *data complexity* of CQA [12], we obtain

Theorem 1. The problem of peer consistent answering is Π_2^P-complete. □

---

[6] The undecidability result for CQA in [13] can be reconstructed in our framework, because even with $\mathcal{G}(\mathfrak{P})$ acyclic, DECs can have ref-cycles (cf. Example 5).

3 Answer Set Programs and the Solutions for a Peer

In order to define the solutions for a peer `P`, we have to consider `P`'s relevant peers, which are those in $\mathcal{AC}(\texttt{P})$. The presence of cycles, through trust relationships or constraints (DECs or ICs), have an impact on the semantics. The former cycles appear in a cyclic $\mathcal{G}(\texttt{P})$. The latter appear when the DECs and local ICs of peers in $\mathcal{AC}(\texttt{P})$ put together present cycles through the implications that involve an RDEC or a local referential IC (a RIC). Sets of local ICs of this kind are called *RIC-cyclic* in [12]. For example, $IC = \{\forall x(S(x) \rightarrow Q(x)), \forall x\ (Q(x) \rightarrow S(x)), \forall x(Q(x) \rightarrow \exists y\ T(x,y))\}$ is not RIC-cyclic, whereas $IC' = IC \cup \{\forall xy\ (T(x,y) \rightarrow Q(y))\}$ is, because there is a cycle involving the RIC $\forall x(Q(x) \rightarrow \exists y\ T(x,y))$. RIC-cyclicity at the level of local ICs may lead to more solutions than intended when capturing the repair semantics by means of logic programs [12].

In order to deal with the new issues arising in PDESs, we will assume that, for each peer `P`, $IC(\texttt{P})$ is RIC-acyclic (cf. Section 4 for a discussion). Cycles through DECs and ICs will be crucial for a logic programming-based specification of solutions for a peer. We will say that the PDES $\mathfrak{P}$ is *ref-acyclic* when in $\Sigma \cup IC$ there are no cycles that involve an RDEC or a RIC.

As Example 5 below shows, even assuming the acyclicity of $\mathcal{G}(\mathfrak{P})$, and RIC-acyclicity at the level of local ICs (or no local ICs at all), we may have ref-cycles in the set of all DECs. This is due to the generality of DECs, where we can have relations of any of the two peers on both sides of the implication.

Example 5. Peers `P1`, `P2` have relations R^1, R^2, resp. $\Sigma(\texttt{P1}) = \{\forall xz(R^1(x,z) \rightarrow \exists y R^2(x,y)), \forall xz(R^2(x,z) \rightarrow \exists y R^1(x,y)\}$, $\Sigma(\texttt{P2}) = \emptyset$; and $(\texttt{P1}, \mathit{less}, \texttt{P2}) \in \mathit{trust}$. Here, $\mathcal{AC}(\mathfrak{P})$ is acyclic, but $\Sigma(\texttt{P1}) \cup \Sigma(\texttt{P2})$ has a ref-cycle. □

Now we will show how to specify solutions for a peer, given instances for the other peers, as the stable models of disjunctive logic programs. These programs use annotation constants to indicate the atoms that may become true or false (virtually inserted or deleted) in order to satisfy the DECs and local ICs. For each database predicate P we generate a new copy $P\_$ with an extra argument to accommodate the annotation. In $P\_(\bar{a}, \mathbf{t_a})$, annotation $\mathbf{t_a}$ means that the atom is advised to be made true; and $\mathbf{f_a}$, that the atom should be made false. For each DEC and local IC ψ, a rule captures through its disjunctive head the alternative virtual updates that can be performed to satisfy ψ (cf. rules 2. and 3. in Definition 9 for DECs, and 4. and 5. for local ICs).

Annotation $\mathbf{t}^\star$ indicates that the atom is true or becomes true in the program. It is introduced in order to execute a sequence of virtual updates that is needed due to interacting DECs and ICs. Finally, atoms annotated with $\mathbf{t}^{\star\star}$ are those that become true in a solution. They are the relevant atoms, and are used to read off the database atoms in the solutions (rules 8. below).

The relevant attributes of a constraint are those where the occurrence of *null* is relevant for its satisfaction [12], and then, they receive a special treatment in the logic programs. For the DEC in $\Sigma(\texttt{P4}, \texttt{P3})$ in Example 1, the two attributes of R^3 and the first two of R^4 are relevant, but not the third attribute of R^4.

Definition 8. For a constraint $\psi \in \mathcal{L}(\mathcal{R})$ and a variable or a domain constant t, $pos^R(\psi, t)$ is the set of positions in predicate $R \in \mathcal{R}$ where t appears in ψ. The set of *relevant variables* for ψ is $\mathcal{V}(\psi) = \{x \mid x \text{ is a repeated variable in } \psi\}$. The set of *relevant attributes* for ψ is $\mathcal{A}(\psi) = \{R[i] \mid x \in \mathcal{V}(\psi) \text{ and } i \in pos^R(\psi, x)\} \cup \{R[i] \mid c \text{ is a constant in } \psi \text{ and } i \in pos^R(\psi, c)\}$, where $R[i]$ denotes the attribute in position i in R. □

Definition 9. Consider a PDES $\mathfrak{P}$, a peer $\mathtt{P} \in \mathcal{P}$ with $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P1}, \ldots, \mathtt{Pn}\}$, and $\mathcal{I} = \{I_1, \ldots, I_n\}$, where I_j is a database instance over the schema of $\mathtt{Pj}$. The *solution program* $\Pi(\mathfrak{P}, \mathtt{P}, \mathcal{I})$ for $\mathtt{P}$ contains:

1. $dom(a)$, for every $a \in (\mathcal{U} \smallsetminus \{null\})$. $R(\bar{a})$, for each atom $R(\bar{a}) \in D(\mathtt{P})$. $R(\bar{a})$, for each $R(\bar{a}) \in I$ with $I \in \mathcal{I}$.
2. For every UDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{Pj})$ of the form (1) with $\mathtt{Pj} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, \{same \text{ or } less\}, \mathtt{Pj}) \in trust$, the rule:
$$\bigvee_{R \in R_{\mathtt{P}}} R\_(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{Q \in Q_{\mathtt{P}}} Q\_(\bar{y}_j, \mathbf{t_a}) \;\leftarrow\; \bigwedge_{i=1}^{n} R_i\_(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{f}^\star), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq null,\ \bar{\varphi},$$
where $\mathcal{A}(\psi)$ is the set of relevant attributes of ψ, $\bar{\varphi}$ is a conjunction of built-ins that is equivalent to the negation of φ; and, given $\mathcal{R} = \{R_i \mid i = 1, \ldots, n,\ R_i \text{ appears in (1)}\}$, $R_{\mathtt{P}}$ is defined by $R_{\mathtt{P}} = \mathcal{R} \cap \mathcal{R}(\mathtt{P})$ if $(\mathtt{P}, less, \mathtt{Pj}) \in trust$; and $R_{\mathtt{P}} = \mathcal{R}$ if $(\mathtt{P}, same, \mathtt{Pj}) \in trust$. $Q_{\mathtt{P}}$ is defined analogously in terms of the Q_j predicates in (1).
3. For every RDEC $\psi \in \Sigma(\mathtt{P}, \mathtt{Pj})$ of the form (2) such that $\mathtt{Pj} \in \mathcal{N}(\mathtt{P})$ and $(\mathtt{P}, \{same \text{ or } less\}, \mathtt{Pj}) \in trust$:
    (a) If $(\mathtt{P}, same, \mathtt{Pj}) \in trust$, the rule:
    $R\_(\bar{x}, \mathbf{f_a}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R\_(\bar{x}, \mathbf{t}^\star),\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$
    (b) If $(\mathtt{P}, less, \mathtt{Pj}) \in trust$ and $R \in \mathcal{R}(\mathtt{P})$, the rule:
    $R\_(\bar{x}, \mathbf{f_a}) \leftarrow R\_(\bar{x}, \mathbf{t}^\star),\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$
    (c) If $(\mathtt{P}, less, \mathtt{Pj}) \in trust$ and $Q \in \mathcal{R}(\mathtt{P})$, the rule:
    $Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow R\_(\bar{x}, \mathbf{t}^\star),\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$
    Plus the auxiliary rules:
    $aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ not\ Q\_(\bar{x}', \overline{null}, \mathbf{f_a}),\ \bar{x}' \neq null.$
    For every $y_i \in \bar{y}$:
    $aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t}^\star),\ not\ Q\_(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq null,\ y_i \neq null.$
4. For every UIC $\psi \in IC(\mathtt{P})$ of the form (1), the rule:
$$\bigvee_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{f_a}) \vee \bigvee_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{t_a}) \;\leftarrow\; \bigwedge_{i=1}^{n} P_i\_(\bar{x}_i, \mathbf{t}^\star), \bigwedge_{j=1}^{m} Q_j\_(\bar{y}_j, \mathbf{f}^\star), \bigwedge_{x_l \in \mathcal{A}(\psi)} x_l \neq null,\ \bar{\varphi}.$$
5. For every RIC $\psi \in IC(\mathtt{P})$ of the form (2), the rules:
    $P\_(\bar{x}, \mathbf{f_a}) \vee Q\_(\bar{x}', \overline{null}, \mathbf{t_a}) \leftarrow P\_(\bar{x}, \mathbf{t}^\star),\ aux_\psi(\bar{x}'),\ \bar{x}' \neq null.$
    $aux_\psi(\bar{x}') \leftarrow Q(\bar{x}', \overline{null}),\ not\ Q\_(\bar{x}', \overline{null}, \mathbf{f_a}),\ \bar{x}' \neq null.$
    For every $y_i \in \bar{y}$:
    $aux_\psi(\bar{x}') \leftarrow Q\_(\bar{x}', \bar{y}, \mathbf{t}^\star),\ not\ Q\_(\bar{x}', \bar{y}, \mathbf{f_a}),\ \bar{x}' \neq null,\ y_i \neq null.$
6. For each predicate $R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))$, the annotation rules:
    $R\_(\bar{x}, \mathbf{f}^\star) \leftarrow dom(\bar{x}),\ not\ R(\bar{x}).$ $R\_(\bar{x}, \mathbf{f}^\star) \leftarrow R\_(\bar{x}, \mathbf{f_a}).$
    $R\_(\bar{x}, \mathbf{t}^\star) \leftarrow R(\bar{x}).$ $R\_(\bar{x}, \mathbf{t}^\star) \leftarrow R\_(\bar{x}, \mathbf{t_a}).$
7. For each predicate $R \in \mathcal{R}(\mathcal{N}(\mathtt{P}))$, the program constraint:
    $\leftarrow\ R\_(\bar{x}, \mathbf{t_a}),\ R\_(\bar{x}, \mathbf{f_a}).$

8. For each predicate $R \in \mathcal{R}(\texttt{P})$, the interpretation rule:
   $R\_(\bar{x}, \mathbf{t}^{\star\star}) \leftarrow R\_(\bar{x}, \mathbf{t}^{\star}),\ \mathbf{not}\ R\_(\bar{x}, \mathbf{f_a})$. □

In bodies of rules associated to DECs or ICs ψ, the conditions of the form $x \neq \mathbf{null}$, with x a variable appearing in a relevant attribute of ψ are used to capture the semantics of null values as used in SQL (cf. [12] for details). An atom of the form $P(\bar{x}, \overline{\mathbf{null}}, ...)$ in the program represents an atom with possibly several occurrences of $\mathbf{null}$, not necessarily in its last arguments, e.g. $P(x, \mathbf{null}, y, \mathbf{null}, ...)$. For $\bar{x} = x_1, \ldots, x_n$, $\bar{x} \neq \mathbf{null}$ abbreviates $x_1 \neq \mathbf{null}, \ldots, x_n \neq \mathbf{null}$. The program constraints in 7. discard models where an atom is both inserted and deleted.

The facts of the program are those in instance $D(\texttt{P})$ of P and those in instances $I(\texttt{Pi})$ for P's neighbors Pi. The instances $I(\texttt{Pi})$ used in the program may not coincide with the physical instances $D(\texttt{Pi})$. Actually, as shown below, if each $I(\texttt{Pi})$ is the intersection of the solutions of Pi, then the stable models of the program are in one-to-one correspondence with the solutions of peer P.

Since virtual updates are executed on the peers' instances, their local ICs have to be kept satisfied. That is the role of rules 4. (for universal ICs) and 5. (for referential ICs) above. We adopt the stable model semantics for the solution programs [24], i.e. their intended models are their stable models.

Example 2. (Example 1 continued) Consider $D(\texttt{P1}) = \{R^1(a, 2)\}$, and the instance $I_2 = \{R^2(d, 5)\}$ for P2, the neighbor of P1. The solution program $\Pi(\mathfrak{P}, \texttt{P1}, \{I_2\})$ contains: $dom(a)$. $dom(c)$. ... $R^1(a, 2)$. $R^2(d, 5)$.
$R^1\_(x, y, \mathbf{t_a}) \leftarrow R^2\_(x, y, \mathbf{t}^{\star}),\ R^1\_(x, y, \mathbf{f}^{\star}), x \neq \mathbf{null}, y \neq \mathbf{null}$.
$R^1\_(x, y, \mathbf{t}^{\star}) \leftarrow R^1\_(x, y, \mathbf{t_a})$. $R^1\_(x, y, \mathbf{t}^{\star}) \leftarrow R^1(x, y)$.
$R^1\_(x, y, \mathbf{f}^{\star}) \leftarrow R^1\_(x, y, \mathbf{f_a})$. $R^1\_(x, y, \mathbf{f}^{\star}) \leftarrow dom(x), dom(y), \mathbf{not}\ R^1(x, y)$.
$\leftarrow R^1\_(x, y, \mathbf{t_a}), R^1\_(x, y, \mathbf{f_a})$. $R^1\_(x, y, \mathbf{t}^{\star\star}) \leftarrow R^1\_(x, y, \mathbf{t}^{\star}),\ \mathbf{not}\ R^1\_(x, y, \mathbf{f_a})$.
With similar rules to the 2nd-6th for R^2. The first rule makes sure that an R^2-tuple that is not in R^1, is also virtually inserted into R^1. In this case, since P1 trusts P2 more than itself, virtual changes affect only peer P1. □

Example 3. Consider a PDES $\mathfrak{P}$ with $\mathcal{R}(\texttt{P1}) = \{R^1(\cdot, \cdot)\}$, $D(\texttt{P1}) = \{R^1(s, t), R^1(a, \mathbf{null})\}$, $\mathcal{R}(\texttt{P2}) = \{R^2(\cdot, \cdot)\}$, $D(\texttt{P2}) = \{R^2(c, d), R^2(a, e)\}$, $\Sigma(\texttt{P1}, \texttt{P2}) = \{\forall xy\ (R^2(x, y) \rightarrow \exists z\ R^1(x, z))\}$, $IC(\texttt{P1}) = \{\forall xyz\ (R^1(x, y) \wedge R^1(x, z) \rightarrow y = z)\}$, $\Sigma(\texttt{P2}, \texttt{P1}) = IC(\texttt{P2}) = \emptyset$, and $trust = \{(\texttt{P1}, \mathbf{less}, \texttt{P2})\}$. The program $\Pi(\mathfrak{P}, \texttt{P1}, \{I_2\})$ for P1 needs an instance for P2 that may be different from $D(\texttt{P2})$, but in this case we choose $I_2 = D(\texttt{P2})$, obtaining the following program (omitting rules 6., 7.): $dom(a)$. $dom(b)$. ... $R^1(a, \mathbf{null})$. $R^1(s, t)$. $R^2(c, d)$. $R^2(a, e)$.

$$\begin{aligned}
R^1\_(x, \mathbf{null}, \mathbf{t_a}) &\leftarrow R^2\_(x, \mathbf{t}^{\star}),\ \mathbf{not}\ aux(x), x \neq \mathbf{null}.\\
aux(x) &\leftarrow R^1(x, \mathbf{null}),\ \mathbf{not}\ R^1\_(x, \mathbf{null}, \mathbf{f_a}).\\
aux(x) &\leftarrow R^1(x, y, \mathbf{t}^{\star}),\ \mathbf{not}\ R^1\_(x, y, \mathbf{f_a}), x \neq \mathbf{null}, y \neq \mathbf{null}.\\
R^1(x, y, \mathbf{f_a}) \vee R^1(x, z, \mathbf{f_a}) &\leftarrow R^1(x, y, \mathbf{t}^{\star}), R^1(x, z, \mathbf{t}^{\star}),\ x \neq \mathbf{null},\ y \neq z.\\
R^1\_(x, y, \mathbf{t}^{\star\star}) &\leftarrow R^1\_(x, y, \mathbf{t}^{\star}),\ \mathbf{not}\ R^1\_(x, y, \mathbf{f_a}).
\end{aligned}$$

The first rule has the role of satisfying the RDEC by introducing a $\mathbf{null}$ into R^1. The fourth rule takes care of the local functional dependency. □

The atoms annotated with $\mathbf{t}^{\star\star}$ in a stable model of P's program have predicates of P only. They define a database instance for P. In Example 7, only $R^1\_$-atoms become annotated with $\mathbf{t}^{\star\star}$. The program has only one stable model, with associated instance $\{R^1(a,$ [illegible]$), R^1(s,t), R^1(c,$ [illegible]$)\}$.

Definition 10. The database instance for peer P associated to a stable model $\mathcal{M}$ of program $\varPi(\mathfrak{P}, \mathtt{P}, \mathcal{I})$ is $D_{\mathcal{M}} = \{R(\bar{a}) \mid R\_(\bar{a}, \mathbf{t}^{\star\star}) \in \mathcal{M}\}$. □

Theorem 2. Given a PDES $\mathfrak{P}$, $\mathtt{P} \in \mathcal{P}$, $\mathcal{N}(\mathtt{P}) = \{\mathtt{P}, \mathtt{P1}, \ldots, \mathtt{Pn}\}$, $n \geq 0$, $D(\mathtt{P})$ an instance for P, and $\mathcal{I}^\star = \{I_1, \ldots, I_n\}$ instances for P1, …, Pn, resp. If $\Sigma \cup$ [illegible] is ref-acyclic and each of the I_i is the intersection of the solution instances for peer Pi, then the instances of the form $D_{\mathcal{M}}$, where $\mathcal{M}$ is a stable model of $\varPi(\mathfrak{P}, \mathtt{P}, \mathcal{I}^\star)$, are the all and only solution instances for P. □

In Example 7, given that $D(\mathtt{P2})$ is already the only solution instance for P2 (P2 has neither DECs nor local ICs) and $\Sigma(\mathtt{P1}, \mathtt{P2})$ is ref-acyclic, the only solution instance for P is $D = \{R^1(s,t), R^1(a,$ [illegible]$), R^1(c,$ [illegible]$)\}$. However, if there are ref-cycles, the stable models may correspond to a strict superset of the solutions for a peer. In this case, post-processing that deletes models corresponding to non-minimal "solutions" is necessary.

Under the assumption that we have already computed the (intersection of the) solution instances for the neighbors of P, the program for P allows us to compute its solution instances. This generates a recursive process that can be applied because $\mathcal{G}(\mathtt{P})$ is acyclic. The terminal peers Pt, i.e. those with no outgoing edges in $\mathcal{G}(\mathtt{P})$, will become the base cases for the recursion. If a peer P' has Pt as one of its neighbors, the instance I_t to be used for Pt in the program for P' is simply $D(\mathtt{Pt})$, Pt's original local instance.

Theorem 2 still holds if peer P, instead of collecting the intersection I_i of the solutions of a neighbor Pi, uses the intersection of the solutions for Pi restricted to the subschema of Pi that contains Pi's relations that appear in $\Sigma(\mathtt{P}, \mathtt{Pi})$, which are those P needs to run its program.

With a solution program for P, PCAs for a query $\mathcal{Q}$ posed to P can be obtained by running under the stable model semantics a query program $\varPi(\mathcal{Q})$ that represents the query in combination with $\varPi(\mathfrak{P}, \mathtt{P}, \mathcal{I}^\star)$.

[illegible] (Example 6 continued) In order to obtain the PCAs to the query $\mathcal{Q}_1: R^1(x,y)$, asking for the tuples in R^1, the rule [illegible]$_1(x,y) \leftarrow R^1(x,y,\mathbf{t}^{\star\star})$ has to be added to $\varPi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ (assuming that I_2 is the intersection of the solution instances for P2). The ground [illegible]$_1$-atoms in the intersection of all stable models correspond to the PCAs. For the query $\mathcal{Q}_2: \exists y R^1(x,y)$, the projection of R^1 on its first attribute, the query rule is [illegible]$_2(x) \leftarrow R^1(x,y,\mathbf{t}^{\star\star})$. □

Our semantics could be naively implemented as follows. When P is posed a query, P has to run its program, for which it needs as facts those in the intersections of the solutions of its neighbors. So, P sends to each neighbor P' queries of the form $\mathcal{Q}$: $R(\bar{x})$, where R is a relation of P' that appears in $\Sigma(\mathtt{P}, \mathtt{P'})$. P expects to receive from P' the set of PCAs to $\mathcal{Q}$, because they corresponds to the extension of R in the intersection of solutions for P'. In order to return to P the PCAs to its queries, the neighboring peers have to run their own programs (except for the facts, each peer has a fixed solution program that can be used with any query). As before, they will need PCAs from their own neighbors; etc. This recursion will eventually reach peers that have no DECs (and its local ICs will be satisfied), who will offer answers from their original instances to queries by other peers. Now, propagation of PCAs goes backwards until reaching `P`, and `P` gets the facts to run its program and obtain the PCAs to the original query.

Example 7. (Example 6 continued) Consider local instances $D(\mathtt{P1}) = \{R^1(a,2)\}$, $D(\mathtt{P2}) = \{R^2(c,4), R^2(d,5)\}$, and $D(\mathtt{P3}) = \{R^3(c,4)\}$. A user poses the query $\mathcal{Q}_0 : R^1(x,y)$ to `P1`, expecting its PCAs. To run its program, `P1` needs the intersection of the solutions of peer `P2`. So, `P1` sends to `P2` the queries $\mathcal{Q}_1^1 : R^2(x,y)$ and $\mathcal{Q}_2^1 : S^2(x,y)$ (actually, `P1` does not need the latter, S^2 is not relevant to `P1`). In order to peer-consistently-answer these queries, `P2` needs from `P3` the PCAs to $\mathcal{Q}_3^2$: $R^3(x,y)$. Since `P3` has no neighboring peers, it returns to `P2` the entire extension in its local database of relation R^3: $I_3 = D(\mathtt{P3}) = \{R^3(c,4)\}$ is given to `P2`. Now, `P2` can run its solution program $\Pi(\mathfrak{P}, \mathtt{P2}, \{I_3\})$, containing: $dom(c)$. $dom(d)$. $\ldots$ $R^2(c,4)$. $R^2(d,5)$. $R^3(c,4)$.

$$R^2_{\_}(x,y,\mathbf{f_a}) \vee R^3_{\_}(x,y,\mathbf{f_a}) \leftarrow R^2_{\_}(x,y,\mathbf{t}^\star), R^3_{\_}(x,y,\mathbf{t}^\star), x \neq \mathit{null}, y \neq \mathit{null}.$$

$$R^2_{\_}(x,y,\mathbf{t}^{\star\star}) \leftarrow R^2_{\_}(x,y,\mathbf{t}^\star), \mathit{not}\ R^2_{\_}(x,y,\mathbf{f_a}).$$

$$S^2_{\_}(x,y,\mathbf{t}^{\star\star}) \leftarrow S^2_{\_}(x,y,\mathbf{t}^\star), \mathit{not}\ S^2_{\_}(x,y,\mathbf{f_a}).$$

We obtain two solutions: $\{\{R^2(d,5)\}$ and $\{R^2(c,4), R^2(d,5)\}\}$. So, the intersection of `P2`'s solutions is $I_2 = \{R^2(d,5)\}$. Finally, the program $\Pi(\mathfrak{P}, \mathtt{P1}, \{I_2\})$ given in Example 6 is run. It has only one solution, namely $\{R^1(a,2), R^1(d,5)\}\}$. Therefore, the peer consistent answers to $\mathcal{Q}_0$ are $(a,2)$ and $(d,5)$. □

4 Discussion

The domain predicate, dom, in the solution programs can always be instantiated in a finite active domain; actually dom can be eliminated by adding rules [11].

In the most common PDESs, let us call them the *unrestricted import case*, peers `P` only import data from other peers they trust more than themselves; and using DECs $\Sigma(\mathtt{P})$ of the form (1) or (2) that have only one database predicate in the consequent which belongs to $\mathcal{R}(\mathtt{P})$, and all predicates in the antecedent belonging to another peer's schema. In this case, the relevant part of the intersection of the solutions of each neighbor can be obtained as the PCAs to a single conjunctive query, namely the one in the antecedent of the DEC.

Proposition 1. For the unrestricted import case of PDES, a peer `P` with an empty set $\mathit{IC}(\mathtt{P})$ of local ICs always has a solution instance. □

This result still holds under rather weak conditions, e.g. if the ICs in $\mathit{IC}(\mathtt{P})$: (a) have a consequent that contains at least one database predicate (not a built-in); or (b) if only built-ins appear in the consequent, e.g. **false**, there is a predicate in the antecedent of the IC that does not appear in any of `P`'s DECs.

If a PDES has no local ICs, then it is easy to see that the solution program is head-cycle free [18], and we obtain

Proposition 2. In the unrestricted import case, the solution program for a peer with an empty set of local ICs is equivalent to a non-disjunctive program. □

The hypothesis on local ICs in this result can be much weakened by assuming that local ICs are of the form identified in [5, 12], that lead to head-cycle free repair programs. Since cautious reasoning from normal logic programs is [illegible]-complete [18], peer consistent answering is in [illegible] in data complexity.

We have assumed that $\mathcal{G}(\mathfrak{P})$ is acyclic. However, the peers, not being aware of being in a cyclic $\mathcal{G}(\mathfrak{P})$, could attempt to do data exchange as described above. In order not to detect an infinite loop, for each query a unique identifier can be created and kept in all the queries that have origin in it.

The assumption of acyclicity of the accessibility graph is quite cautious in the sense that it excludes cases where a reasonable semantics could be given and the logic programs would work correctly, because the cycles in $\mathcal{G}(\mathfrak{P})$ are not relevant.

We have also assumed that the sets [illegible]`(P)` of local ICs of peers `P` are each ref-acyclic. Even under this assumption, and also with Σ(`P`) ref-acyclic, [illegible]`(P)` $\cup$ Σ(`P`) can have ref-cycles. For example, with [illegible]`(P1)` $= \{\forall x(R^1(x) \rightarrow S^1(x))\}$, Σ(`P1`, `P2`) $= \{\forall x(S^1(x) \rightarrow \exists y R^2(x,y)), \forall x(R^2(x,y) \rightarrow R^1(x))\}$. There are also cases with an acyclic $\mathcal{G}(\mathfrak{P})$, but with ref-cycles in the DECs, where the logic programming counterpart of the semantics is correct due to the role of the trust relationships.

It becomes clear that it is possible to find more relaxed conditions, both on the accessibility graph and ref-cycles, under which a sensible semantics for solutions and semantically corresponding logic programs can be given. Also, with general cyclic accessibility graphs, [illegible] [32] could be used to detect cycles and prune certain DECs, making the graph acyclic if necessary; and then our semantics could be applied.

In [14, 16, 15], the semantics of a PDES is given in terms of epistemic logic. The mappings (our DECs) are of the form $cq_i \rightarrow cq_j$, with cq_i and cq_j conjunctive queries over `Pi` and `Pj`'s schemas, resp. Those DECs keep the schemas separate. It is implicitly assumed that peers trust themselves less than other peers. The semantics can be applied in the presence of cycles in the accessibility graph.

The treatment of local ICs differs from ours in two ways : (a) A peer that is inconsistent wrt its local ICs is not considered for data exchange, while in our case such a peer may apply a repair semantics, as in CQA. (b) Atoms are imported into a peer by interaction with other peers only if this does not produce a local IC violation. In our case, under the same trust conditions, the data is accepted and the peer applies again a local repair semantics.

In order to answer a query [15], a peer traverses the network eventually collecting at its site all DECs, ICs and data of other logically related peers. With these elements, the peer can construct its epistemic theory, that is used for query answering. An accessibility cycle can be detected by using request identifiers. The use of epistemic logic makes sure that [illegible], the one a peer really knows, is passed to another peer. In our case, a peer collects only data from its neighbors; and certainty is achieved by using the PCAs of a peer, or more generally, the intersection of its solutions. A more detailed comparison can be found in [11].

The semantics in [21, 22] coincides with the epistemic semantics in [16]. They provide a distributed algorithm, where peers' data is updated by instruction of a super peer. When a query is posed to a peer, it can answer the query right away with its data because the PDES is already updated.

5 Conclusions

We have introduced a framework for peer data exchange with trust relationships. Each peer solves its data and semantic conflicts at query time, when querying its own and other peers' data.

Logic programs can be used to specify solutions for a peer and to obtain peer consistent answers. Techniques to partially compute the solution instances can be useful, since we are not interested in them per se, but in the PCAs. Techniques used in CQA, such as magic sets for stable model semantics [20] and identification of predicates that are relevant to queries and constraints, could also be used in this setting, to restrict the number of rules and the amount of data that are needed to run the program [17, 19].

The problem of query evaluation from disjunctive programs is Π_2^P-complete [18], which matches the complexity of PCA. In spite of this, it is possible to identify syntactic classes of PDESs for which peer consistent query answering has a lower complexity, and specifically tailored mechanisms to solve this problem could be developed, as for CQA (cf. [10] for a survey).

The concepts and results presented in this paper smoothly extend the semantics for [illegible] for a peer as introduced in [7] to the transitive case. Basically, those local solutions correspond to the neighborhood solutions we introduced above. No general solution programs were presented in [7].

Semantics for PDESs have been introduced and analyzed in [27, 21, 29, 28, 22, 14, 16, 15, 23], but without considering trust relationships. In them, if there is a DEC from `P` to `Q`, it is implicitly assumed that `P` trusts itself less than `Q`. Also, all the research so far, has concentrated on the unrestricted import case. In our setting, a DEC may also restrict the data that can belong to a peer.

Acknowledgements. Research supported by NSERC and a CITO/IBM-CAS Student Internship. L. Bertossi is Faculty Fellow of IBM CAS (Toronto Lab.). Part of this work was done when L. Bertossi was visiting the Database Group at Edinburgh University. Their hospitality is much appreciated.

References

[1] Arenas, M., Bertossi, L., Chomicki, J.: Consistent Query Answers in Inconsistent Databases. In: PODS 1999. Proc. ACM Symposium on Principles of Database Systems, pp. 68–79. ACM Press, New York (1999)

[2] Arenas, M., Bertossi, L., Chomicki, J.: Answer Sets for Consistent Query Answers. Theory and Practice of Logic Programming 3(4&5), 393–424 (2003)

[3] Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)

[4] Barcelo, P., Bertossi, L.: Logic Programs for Querying Inconsistent Databases. In: Dahl, V., Wadler, P. (eds.) PADL 2003. LNCS, vol. 2562, pp. 208–222. Springer, Heidelberg (2002)

[5] Barceló, P., Bertossi, L., Bravo, L.: Characterizing and Computing Semantically Correct Answers from Databases with Annotated Logic and Answer Sets. In: Bertossi, L., Katona, G.O.H., Schewe, K.-D., Thalheim, B. (eds.) Semantics in Databases. LNCS, vol. 2582, pp. 7–33. Springer, Heidelberg (2003)

[6] Bertossi, L., Bravo, L.: Consistent Query Answers in Virtual Data Integration Systems. In: Bertossi, L., Hunter, A., Schaub, T. (eds.) Inconsistency Tolerance. LNCS, vol. 3300, pp. 42–83. Springer, Heidelberg (2005)

[7] Bertossi, L., Bravo, L.: Query Answering in Peer-to-Peer Data Exchange Systems. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 476–485. Springer, Heidelberg (2004)

[8] Bertossi, L., Bravo, L.: The Semantics of Consistency and Trust in Peer Data Exchange Systems (extended version). `http://www.scs.carleton.ca/~bertossi/papers/lparExt.pdf`

[9] Bertossi, L., Chomicki, J.: Query Answering in Inconsistent Databases. In: Logics for Emerging Applications of Databases, pp. 43–83. Springer, Heidelberg (2003)

[10] Bertossi, L.: Consistent Query Answering in Databases. ACM Sigmod Record 35(2), 68–76 (2006)

[11] Bravo, L.: Handling Inconsistency in Databases and Data Integration Systems. PhD. Thesis, Carleton University, Department of Computer Science (2007), `http://homepages.inf.ed.ac.uk/lbravo/Publications.htm`

[12] Bravo, L., Bertossi, L.: Semantically Correct Query Answers in the Presence of Null Values. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Mesiti, M., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijsen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 336–357. Springer, Heidelberg (2006)

[13] Calì, A., Lembo, D., Rosati, R.: On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In: PODS 2003. Proc. ACM Symposium on Principles of Database Systems, pp. 260–271. ACM Press, New York (2003)

[14] Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Logical Foundations of Peer-To-Peer Data Integration. In: PODS 2004. Proc. ACM Symposium on Principles of Database Systems, pp. 241–251. ACM Press, New York (2004)

[15] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Inconsistency Tolerance in P2P Data Integration: An Epistemic Logic Approach. In: Bierman, G., Koch, C. (eds.) DBPL 2005. LNCS, vol. 3774, pp. 90–105. Springer, Heidelberg (2005)

[16] Calvanese, D., Damaggio, E., De Giacomo, G., Lenzerini, M., Rosati, R.: Semantic Data Integration in P2P Systems. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) VLDB 2003. LNCS, vol. 2944, pp. 77–90. Springer, Heidelberg (2004)

[17] Caniupan, M., Bertossi, L.: Optimizing Repair Programs for Consistent Query Answering. In: SCCC 2005. Proc. International Conference of the Chilean Computer Science Society, pp. 3–12. IEEE Computer Society Press, Los Alamitos (2005)

[18] Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys 33(3), 374–425 (2001)

[19] Eiter, T., Fink, M., Greco, G., Lembo, D.: Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 163–177. Springer, Heidelberg (2003)

[20] Faber, W., Greco, G., Leone, N.: Magic Sets and their Application to Data Integration. J. Comp. and Sys. Sciences 73(4), 584–609 (2007)

[21] Franconi, E., Kuper, G., Lopatenko, A., Serafini, L.: A Robust Logical and Computational Characterisation of Peer-to-Peer Database Systems. In: Aberer, K., Koubarakis, M., Kalogeraki, V. (eds.) VLDB 2004. LNCS, vol. 2944, pp. 64–76. Springer, Heidelberg (2004)

[22] Franconi, E., Kuper, G., Lopatenko, A., Zaihrayeu, I.: A Distributed Algorithm for Robust Data Sharing and Updates in P2P Database Networks. In: Lindner, W., Mesiti, M., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 446–455. Springer, Heidelberg (2004)

[23] Fuxman, A., Kolaitis, Ph., Miller, R., Tan, W.: Peer Data Exchange. ACM Trans. Database Systems 31(4), 1454–1498 (2006)

[24] Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)

[25] Gelfond, M., Leone, N.: Logic Programming and Knowledge Representation: The A-Prolog Perspective. Artificial Intelligence 138(1-2), 3–38 (2002)

[26] Greco, G., Greco, S., Zumpano, E.: A Logical Framework for Querying and Repairing Inconsistent Databases. IEEE Transactions on Knowledge and Data Engineering 15(6), 1389–1408 (2003)

[27] Halevy, A., Ives, Z., Suciu, D., Tatarinov, I.: Schema Mediation in Peer Data Management Systems. In: ICDE 2003. Proc. International Conference on Data Engineering, pp. 505–518. IEEE Computer Society, Los Alamitos (2003)

[28] Halevy, A., Ives, Z., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The Piazza Peer Data Management System. IEEE Transactions on Knowledge and Data Engineering 16(7), 787–798 (2004)

[29] Kementsietsidis, A., Arenas, M., Miller, R.: Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In: SIGMOD 2003. Proc. ACM International Conference on Management of Data, pp. 325–336. ACM Press, New York (2003)

[30] Kolaitis, Ph.: Schema Mappings, Data Exchange, and Metadata Management. In: PODS 2005. Proc. of ACM Symposium on Principles of Database Systems, pp. 61–75. ACM Press, New York (2005)

[31] Levene, M., Loizou, G.: Null Inclusion Dependencies in Relational Databases. Information and Computation 136(2), 67–108 (1997)

[32] Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. In: ICDE 2003. Proc. International Conference on Data Engineering, p. 49. IEEE Computer Society, Los Alamitos (2003)

Completeness and Decidability in Sequence Logic

Marc Bezem[1], Tore Langholm[2], and Michał Walicki[1]

[1] Department of Informatics, University of Bergen
{bezem,michal}@ii.uib.no
[2] Department of Informatics, University of Oslo
tore.langholm@ifi.uio.no

Abstract. Sequence logic is a parameterized logic where the formulas are sequences of formulas of some arbitrary underlying logic. The sequence formulas are interpreted in certain linearly ordered sets of models of the underlying logic. This interpretation induces an entailment relation between sequence formulas which strongly depends on which orderings one wishes to consider. Some important classes are: all linear orderings, all dense linear orderings and all (or some specific) wellorderings.

For all these classes one can ask for a sound and complete proof system for the entailment relation, as well as for its decidability. For the class of dense linear orderings and all linear orderings we give sound and complete proof systems which also yield decidability (assuming that the underlying logic is sound, complete and decidable). We formulate some open problems on the entailment relation in the case of wellorderings.

1 Introduction

Sequence logic is a parameterized logic where the formulas are sequences of formulas of some underlying logic. It can be viewed as a subsystem of linear temporal logic where the temporal aspects are completely separated from other logical aspects. This separation makes it easy to change the underlying logic without having to rethink the temporal aspects. Sequence logic has recently been proposed in [11] and its virtues for modelling dynamically changing knowledge are studied in [9].

Examples can be provided by systems of interacting agents, in which communication involves state-changes, such as in protocols. A simple example is the Muddy Children Puzzle in epistemic logic, see [5]. Here each child's knowledge is dynamically extended every round, a process which can be described by the sequence formula $a_1; \ldots ; a_n$, where events causing changes are modelled exclusively as changes in the states expressed by the successive formulas. The problem is to prove that after m rounds, where m is the number of children, each child knows whether it is muddy. In sequence logic this problem can be cast as the entailment $a_1; \ldots ; a_m \models \top; b$, where $\top; b$ expresses that eventually each child knows whether it is muddy. The underlying logic will, most naturally, be some epistemic logic. The example is simple in that the children's knowledge increases

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 123–137, 2007.
© Springer-Verlag Berlin Heidelberg 2007

monotonically. As a consequence, the example is valid with respect to any ordering. Other examples discriminate between various classes of orderings and will be given at relevant places in the paper.

In this paper we give sound and complete proof systems for sequence logic and draw some conclusions on decidability of entailment. We start by reviewing some necessary preliminaries.

1.1 Orderings

An [illegible] $(D, <)$ is an irreflexive and transitive relation $<$ on a non-empty set D. The non-strict variant (reflexive closure) of $<$ is denoted by $\leq$. An ordering is [illegible] if, for any $x, y \in D$, either $x < y$ or $x = y$ or $y < x$. A [illegible] ([illegible]) element is an $x \in D$ such that $x \leq y$ ($y \leq x$) for all $y \in D$. A linear ordering is [illegible] if it has no greatest element. An ordering is [illegible] if for any $x, y \in D$ with $x < y$, there exists a $z \in D$ with $x < z < y$. A [illegible] is a linear ordering in which every non-empty subset has a least element. In a right-open wellordering every element $x \in D$ has a [illegible], that is, the least element $y \in D$ with $x < y$. Any element of a wellordering that is not the least element of the ordering nor the successor of another element, is called a [illegible]. Classes of wellorderings modulo isomorphy are called [illegible]. An important ordinal is ω, the class of the natural numbers equipped with their natural ordering.

Unless explicitly stated otherwise, we assume all orderings to be linear, right-open and to have a least element, and we will denote this class by LO. The subclasses of dense orderings and of wellorderings are denoted by DLO and WO, respectively. Note that right-openness excludes all successor ordinals and, in particular, all finite orderings.

Given an ordering $<$ and $d < d'$, we use the common notation $[d, d')$ to denote the left-closed, right-open interval from d to d', that is, the set $\{x \in D \mid d \leq x < d'\}$. We use $[d, \infty)$ to denote the set of elements that are greater than or equal to d. For a function $s : D \rightarrow X$ we use $s(I)$ to denote the s-image of interval I.

1.2 Sequence Logic

The formulas of sequence logic are non-empty finite sequences $a_1; \ldots; a_n$ where each a_i is a formula of some underlying logic, ul, which is a fixed parameter. We define $hd(a_1; \ldots; a_n) = a_1$.

Notation and Terminology 1

[illegible] ul [illegible] $a, b, \ldots$ [illegible] sequence formulas [illegible] $\alpha, \beta, \ldots$ [illegible] $\alpha; \beta$ [illegible] α [illegible] β [illegible] $\alpha; a; \beta$ [illegible] $\alpha; a; b; \beta$ [illegible] a [illegible] b [illegible] ul [illegible]

[illegible] A [illegible] ul [illegible] A^l [illegible] l [illegible] A [illegible] a [illegible] a^l [illegible] l [illegible] a [illegible] b [illegible] $\alpha = a_1; \ldots; a_n$

$b \circ \alpha$ $b \wedge a_1; \ldots ; b \wedge a_n$
b α ul $\wedge$
$\models$
ul
$s \models a$ a ul s. $s \models \alpha$
α s ul
$\mathcal{S}$ $\mathcal{S} \models a$. $s \models a$ $s \in \mathcal{S}$ $a \models b$ ul
$\mathcal{C}$
$\alpha \models_{\mathcal{C}} \beta$

A **structure** of sequence logic is a tuple $S = (d_0, D, <, s)$ where $(D, <)$ is an ordering, d_0 its least element and s a mapping from D to models of the underlying logic. We identify the structure S with the mapping s when the ordering is clear from the context.

A structure S **satisfies** a sequence formula $\alpha = a_1; \ldots ; a_n$ if there exist $d_1, \ldots , d_{n-1} \in D$, $d_0 < d_1 < \cdots < d_{n-1}$, such that for all $1 \leq i < n$ we have $s([d_{i-1}, d_i)) \models a_i$ and $s([d_{n-1}, \infty)) \models a_n$. Then $[d_{i-1}, d_i)$ ($[d_{n-1}, \infty)$) is called the interval of s satisfying a_i (a_n), or, equivalently, the interval that s uses to satisfy a_i. Note that this terminology refers to a specific instance of $S \models \alpha$ and that it is possible that the same S satisfies α using different intervals. Satisfaction of α in S will also be denoted by $s \models \alpha$ when the structure S is clear from the context.

The satisfaction relation defined in the previous paragraph gives rise to the following **entailment** relation: Given a class $\mathcal{C}$ of orderings we define $\alpha \models_{\mathcal{C}} \beta$ if for all structures $S = (d_0, D, <, s)$ with $(D, <) \in \mathcal{C}$ we have $s \models \beta$ whenever $s \models \alpha$.

The operator $\_;\_$ corresponds to the chop operator, e.g., [7,6]. Sequence logic can also be viewed as a fragment of linear-time temporal logic LTL. For example, $a_1; ...; a_n$ can be expressed in LTL as $a_1 U(a_2 U \ldots (a_{n-1} U \neg(\top U \neg a_n)) \ldots)$. Here U is an until-operator[1] and $\top$ is always true. We are not aware of the separate study of this fragment elsewhere. Complexity theoretic questions are not considered in the present paper, but it would be interesting to investigate whether the restricted expressivity of sequence logic results in lower complexity. To give a concrete example of such a question: in [8] it is proved that satisfiability of LTL based on ω is PSPACE-complete. This constitutes an upper bound for the complexity of $\models_\omega$, with co-NP as an obvious lower bound, both with classical propositional logic as ul. An open question is now: what is the exact computational complexity of $\models_\omega$?

The main novelty and strength of sequence logic is its parameterization by the underlying logic ul. The general assumption about the derivability relation

---

[1] There exists a rich variety of temporal operators. For convenience we have used an until-operator with semantics defined by $i \models \phi U \psi$ if $\exists k > i \ (k \models \psi \wedge \forall j \ (i \leq j < k \Rightarrow j \models \phi))$. This until-operator is definable by $\phi \wedge (\phi U' \psi)$ with U' the until-operator from [1], and by $\phi \wedge X(\phi U'' \psi)$ with U'' the until-operator and X the next-operator from [8].

$\vdash$ of ul is that it satisfies the classical closure properties, [10,2], namely (X, Y range over sets of formulas):

extension: $X \vdash a$ if $a \in X$
idempotence: $X \vdash a$ if $\{b \mid X \vdash b\} \vdash a$
monotonicity: $Y \vdash a$ if for some $X \subseteq Y : X \vdash a$

Besides these three we assume that ul contains a formula $\bot$ such that $\bot \vdash a$ for all a. A formula a is called **consistent** if $a \nvdash \bot$. Semantically we require that $\bot$ is unsatisfiable.

We will use classical propositional logic as the main example of ul, but soundness, completeness and decidability results below only use the fact that the underlying logic is sound, complete (strongly, that is, $\Gamma \vdash a$ if $\Gamma \models a$) and decidable, respectively. Combinations with other underlying logics can easily be conceived and some examples can be found in [11,9]. In some cases, we assume that ul is closed under propositional connectives with the usual semantics but even then the results apply to an arbitrary logic extending the classical propositional one.

Recall that $hd(\alpha)$ denotes the first ul formula of α. Table 1 gives some rules of inference for sequence logic based on a proof system $\vdash$ for the underlying logic. This minimal proof system, denoted $\Vdash_{\min}$, will be augmented with distinct rules depending on the class of orderings.

Table 1. Axioms and rules for $\Vdash_{\min}$

Ex Falso:	$a_1; \ldots; a_n \Vdash \beta$, if some $a_i \vdash \bot$ $(1 \leq i \leq n)$
Lift:	$a \Vdash b_1; \ldots; b_n$, if $n \geq 1$ and $a \vdash b_i$ for all b_i $(1 \leq i \leq n)$
Double;-Intro:	$\dfrac{\alpha \Vdash \beta}{a; \alpha \Vdash b; \beta}$, if $a \vdash b$
Left;-Intro:	$\dfrac{\alpha \Vdash \beta}{a; \alpha \Vdash \beta}$, if $a \vdash hd(\beta)$

Theorem 2. [illegible] $\Vdash_{\min}$ [illegible] $\models_{\mathrm{LO}}$ [illegible] $\vdash$ [illegible] $\models$ [illegible] ul

[illegible] Soundness of Ex Falso and Lift is trivial.
For Double;-Intro, assume $\alpha \models_{\mathrm{LO}} \beta$ and $a \vdash b$ and let $s \models a; \alpha$. Then the first interval which is used by s to satisfy a can also be used to satisfy b, relying on the soundness of the ul. We can restrict s to the ordering starting with the second interval and get a model $s' \models \alpha$, so $s' \models \beta$. It follows that $s \models b; \beta$, and hence $a; \alpha \models_{\mathrm{LO}} b; \beta$.

For Left;-Intro, assume $\alpha \models_{\mathrm{LO}} \beta$ and $a \vdash hd(\beta)$ and let $s \models a; \alpha$. Then the first interval which is used by s to satisfy a can also be used to satisfy $hd(\beta)$, relying on the soundness of the ul. We can restrict s to the ordering starting with the second interval and get a model $s' \models \alpha$, so $s' \models \beta$. It follows that $s \models \beta$ by joining the first two intervals.

As a corollary to the proof, we have soundness of the proof system $\Vdash_{\min}$ for any subclass of LO.

2 Dense Linear Orderings

In this section we give a proof system extending $\Vdash_{\min}$ which is sound and complete for the class of dense linear orderings. Inspection of the proof rules immediately gives the decidability of $\models_{\text{DLO}}$.

Note that joining two intervals, as done in the proof of Theorem 2, is always possible but the dual operation, splitting an interval in two, is not. For example, we do not have $a; \neg a \models_\omega a; a; \neg a$ since the first interval can have length one. Consequently, the following rule **Right;-Intro** is not sound for all orderings, but it is sound for the class of dense orderings.

$$\boxed{\text{Right;-Intro: } \frac{\alpha \Vdash \beta}{\alpha \Vdash b; \beta} \text{ provided } hd(\alpha) \vdash b}$$

The system $\Vdash_{\text{DLO}}$ is obtained by adding the rule Right;-Intro to $\Vdash_{\min}$.

Theorem 3. [illegible] $\Vdash_{\text{DLO}}$ [illegible] $\models_{\text{DLO}}$ [illegible] $\alpha \Vdash_{\text{DLO}} \beta$ [illegible] $\alpha \models_{\text{DLO}} \beta$

[illegible] In view of the proof of Theorem 2, which can be carried out with DLO instead of LO, it suffices to show that the rule Right;-Intro is sound. Assume $\alpha \models_{\text{DLO}} \beta$ and $hd(\alpha) \vdash b$ and let $s \models \alpha$. Then by density the first interval which is used by s to satisfy $hd(\alpha)$ can be split in two, say $[d_0, x)$ and $[x, d_1)$. We can use $[d_0, x)$ to satisfy b, since $hd(\alpha) \vdash b$. We can use $[x, d_1)$ to satisfy $hd(\alpha)$, and so s restricted to $[x, \infty)$ satisfies α and hence β. It follows that $s \models b; \beta$.

The following construction will play a central role in the proof of completeness. Given two sequence formulas α, β, of respective lenghts n, m, the structures of the underlying logic can be divided in at most 2^{n+m} equivalence classes, with two structures being in the same class iff they assign the same truth value to all formulas in α and β. By $\mathcal{S}(\alpha, \beta)$ we will denote some finite set of ul structures containing a member of each equivalence class. For any formula c of the underlying logic we let $[\![c]\!]_{\mathcal{S}(\alpha,\beta)} = \{S \in \mathcal{S}(\alpha, \beta) \mid S \models c\}$. For every pair of ul formulas a, b from α and β, respectively, we then have:

$$a \models b \iff [\![a]\!]_{\mathcal{S}(\alpha,\beta)} \models b. \tag{1}$$

We use $[\![c]\!]_{\mathcal{S}(\alpha,\beta)}$ to define special kinds of models of α:

Definition 1. [illegible] $\alpha = a_1; \dots; a_n$ [illegible] β [illegible] β **dense model** [illegible] α [illegible] $s \models \alpha$ [illegible] $1 \leq i \leq n$ $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)}$ [illegible] I [illegible] s [illegible] a_i [illegible] I' [illegible] I [illegible] $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)} \subseteq s(I')$

The idea behind the notion of β-dense model s of α is that on every interval of s satisfying a_i we have all possible behaviours of a_i models on every b_j of β. It can be proved that for all β-dense models s, s' of α:

$$s \models \beta \iff s' \models \beta \tag{2}$$

The existence of β-dense models of a satisfiable α follows, for instance, by distributing for each $1 \leq i \leq n$ the models $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)} = \{S_0, \ldots, S_{k-1}\}$, $k > 1$, in the following standard way on the rational interval $[i-1, i)$. We describe the construction only for $i = 1$, i.e., for $[0, 1)$. Put $s(\frac{j}{k}) = S_j$ for all $0 \leq j < k$. Put $s(\frac{j}{k} + \frac{j'}{k^2}) = S_{(j+j') mod\ k}$ for all $0 \leq j, j' < k$. Continue in this way, in the nth round all intervals of length k^{-n} are divided in intervals of length k^{-n-1} and the models are assigned in a cyclic way. If $s(q')$ has not been defined in the above procedure then $s(q')$ can be chosen arbitrarily from $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)}$. Note that every β-dense model of α is also a γ-dense model of α, for every subsequence γ of β, assuming $\mathcal{S}(\alpha, \gamma) \subseteq \mathcal{S}(\alpha, \beta)$. Of course, β-dense models of α are not unique, but the arguments below only depend on (1) and (2).

Lemma 1. [illegible] ul [illegible] $s \models \beta$ [illegible] β [illegible] s [illegible] α [illegible] $\alpha \Vdash_{\text{DLO}} \beta$

[illegible] By induction on $|\alpha| + |\beta|$. Base case: Assume $\alpha = a$, $\beta = b$ and let $s \models \beta$ be a β-dense model of α. Then $[\![a]\!]_{\mathcal{S}(a,b)} \models b$, which implies $a \models b$ by (1) and hence $a \vdash b$ by the completeness of ul. Hence $a \Vdash_{\text{DLO}} b$ by the Lift rule. For the induction step, let $|\alpha| + |\beta| > 2$ and assume the lemma has been proved for all smaller cases. Let $s \models \beta$ be a β-dense model of α. If $|\alpha| = 1$, that is, $\alpha = a$ for some ul formula a, then every b_j in β is true in $[\![a]\!]_{\mathcal{S}(\alpha,\beta)}$, and hence $a \models b_i$, by (1), and $a \vdash b_i$ by completeness of ul. We then get $a \Vdash_{\text{DLO}} \beta$ by one application of the rule Lift. The case in which $|\beta| = 1$ is proved analogously, with repeated applications of the rule Left;-Intro instead of Lift. Now assume $\alpha = a_1; \ldots; a_n$ and $\beta = b_1; \ldots; b_m$ with $n, m > 1$. In view of (2) we may assume without loss of generality that s has domain $[0, n)$ in the rationals, and uses $[i-1, i)$ to satisfy a_i. Let $[0, q)$ be the interval s uses to satisfy b_1. We distinguish the following three cases.

$q < 1$ Then the first interval that s uses to satisfy a_1 overlaps with the first and the second interval that s uses to satisfy $b_1; b_2$. Hence we have $a_1 \vdash b_1$ and $a_1 \vdash b_2$. Consequently, for the subsequence $\gamma = b_2; \ldots; b_m$ of β, s is a γ-dense model of α satisfying γ. By the induction hypothesis we get $\alpha \Vdash_{\text{DLO}} \gamma$ and by the rule Right;-Intro we get $\alpha \Vdash_{\text{DLO}} \beta$.

$q = 1$ Then $a_1 \vdash b_1$ and with γ as in the previous case, s restricted to $[1, n)$ is a γ-dense model of $a_2; \ldots; a_n$ satisfying γ. Now we get $\alpha \Vdash_{\text{DLO}} \beta$ by the induction hypothesis and an application of the rule Double;-Intro.

$q > 1$ Then $a_1 \vdash b_1$ and s restricted to $[1, n)$ is a β-dense model of $a_2; \ldots; a_n$ satisfying β. Now we get $\alpha \Vdash_{\text{DLO}} \beta$ by the induction hypothesis and an application of the rule Left;-Intro.

In all cases we have proved the conclusion of the lemma.

The completeness theorem follows directly from the above lemma.

Theorem 4. [illegible] $\alpha \models_{\text{DLO}} \beta$ [illegible] $\alpha \Vdash_{\text{DLO}} \beta$

[illegible] Assume $\alpha \models_{\text{DLO}} \beta$. If α is not satisfiable, then $\alpha \Vdash_{\text{DLO}} \beta$ by the Ex Falso rule. Otherwise, let s be a β-dense model of α. Then $s \models \beta$ since $\alpha \models_{\text{DLO}} \beta$, and so $\alpha \Vdash_{\text{DLO}} \beta$ by Lemma 1.

As an example, consider $a; a \vee b; b \not\models_{\text{DLO}} a; b; a \vee b$. Semantically we can see that the entailment doesn't hold by distributing models of $a \wedge b$ and $a \wedge \neg b$ in a dense way on the second interval. Using the completeness theorem we get the same result from the observation that $a \vee b$ proves neither a nor b, so that all applications of the ;-introduction rules are blocked.

Since the base cases Ex Falso and Lift only use provability in the underlying logic and the other rules decrease the length of the sequence when applied bottom-up, we obtain the following corollary.

Corollary 1. [illegible] ul [illegible] $\models_{\text{DLO}}$

In particular, decidability of ul gives also decidability of $\Vdash_{\min}$.

3 All Linear Orderings

As an appetizer, showing that even with classical propositional logic as ul the relation $\models_{\text{LO}}$ is far from trivial, consider the following entailment:

$$a; a \wedge b; c \wedge (a \vee b); b; b \wedge c \models_{\text{LO}} a; b; c; c; b \tag{3}$$

Intuitively, two consecutive c's needed to validate the conclusion can be found either in the interval for $c \wedge (a \vee b)$, provided that it is not a single point, or for $b \wedge c$. But it is far from obvious that every ordering satisfying the assumption can be chopped into intervals satisfying the conclusion.

In this section we give a sound, complete and decidable system $\Vdash_{\text{LO}}$ for $\models_{\text{LO}}$, under the assumption that ul is closed under boolean operators. We start by introducing a series of concepts and conventions which will be applied throughout this section.

Notation and Terminology 5

- [illegible] **convex set** A [illegible] $i \in A$ [illegible] $k \leq i \leq j$ [illegible] $k, j \in A$ [illegible] $\{k, \dots, k+j\}$ [illegible] k, j
- [illegible] R [illegible] A, B [illegible] $R[A]$ [illegible] $\{j \mid \exists i \in A\ iRj\}$ [illegible] $R^{-1}[B]$ [illegible] $\{i \mid \exists j \in B\ iRj\}$
- [illegible] n, m**-coupling** [illegible] $C \subseteq \{1, \dots, n\} \times \{1, \dots, m\}$ [illegible]
  - $C[\{1, ..., n\}] = \{1, ..., m\}$ [illegible] $C^{-1}[\{1, ..., m\}] = \{1, ..., n\}$ [illegible]
  - $i_1 C j_2$ [illegible] $i_2 C j_1$ [illegible] $i_1 < i_2$ [illegible] $j_1 < j_2$
- [illegible] n, m [illegible] C [illegible] **quasi-functional** [illegible] $\{1, \dots, n-1\}$ [illegible] $C[\{i\}]$ [illegible] $i \in \{1, \dots, n-1\}$

[illegible] $\alpha = a_1;\ldots;a_n$ [illegible] $\beta = b_1;\ldots;b_m$ [illegible] $R \subseteq \{1,\ldots,n\} \times \{1,\ldots,m\}$ [illegible] $\alpha \vdash_R \beta$ [illegible] $a_i \vdash b_j$ [illegible] i, j [illegible] iRj

An important consequence of the definition of coupling in point 3 is that for any n, m and n,m-coupling C the images $C[A]$ and $C^{-1}[B]$ are convex sets of natural numbers whenever A and B are.

The characterisations below are proved by straightforward induction.

Lemma 2. [illegible] α [illegible]

[illegible] $\alpha \Vdash_{\mathrm{DLO}} \beta$ [illegible] ff $\alpha \vdash_C \beta$ [illegible] $|\alpha|, |\beta|$ [illegible] C

[illegible] $\alpha \Vdash_{\min} \beta$ [illegible] ff $\alpha \vdash_C \beta$ [illegible] $|\alpha|, |\beta|$ [illegible] C

It can be seen that, with α and β as in (3), there exists a 5,5-coupling C such that $\alpha \vdash_C \beta$, but not a quasi-functional one. Hence $\Vdash_{\min}$ is not complete for $\models_{\mathrm{LO}}$. It is, however, complete for pairs α, β in a certain normal form which we now proceed to describe.

In the rest of this section we shall take particular interest in blocks of identical formulas occurring consecutively in a sequence formula. For this purpose we define $\equiv_\alpha$, for any sequence formula $\alpha = a_1;\ldots;a_n$, to be the smallest equivalence relation on $\{1,\ldots,n\}$ such that $i \equiv_\alpha i+1$ whenever $1 \leq i < n$ and $a_i = a_{i+1}$. The equivalence class of i relative to $\equiv_\alpha$ is written $[i]_{\equiv_\alpha}$, and is always a convex set. We refer to the cardinality of $[i]_{\equiv_\alpha}$ as the **padding** of i. Hence the padding of i is the size of the (maximal) block of consecutive, identical formulas in which a_i occurs:

Definition 2. [illegible] n, m [illegible] C [illegible] **sparse** [illegible] $C[\{i\}]$ [illegible] $i \in \{1,\ldots,n-1\}$ [illegible] m

We can now state the following Redistribution Lemma which will lead to the restricted completeness of $\Vdash_{\min}$.

Lemma 3 (Redistribution). [illegible] $\alpha \vdash_C \beta$ [illegible] $|\alpha|, |\beta|$ [illegible] C [illegible] $\alpha \vdash_{C'} \beta$ [illegible] $|\alpha|, |\beta|$ [illegible] C'

[illegible] Suppose $\alpha \vdash_C \beta$ for the sequence formulas $\alpha = a_1;\ldots;a_n$ and $\beta = b_1;\ldots;b_m$ and the sparse n,m-coupling C. The first step is to define the (possibly partial) function F on numbers $i \in \{1,\ldots,n\}$ by the following clauses.

- if $i \neq n$ and i has padding less than m, then $F(i)$ is the unique member of $C[\{i\}]$.
- if i has padding at least m, then $[i]_{\equiv_\alpha} = \{j,\ldots,j+k\}$ for some $j \leq i$ and $k \geq (m-1)$. Being the image of a convex set, $C[[i]_{\equiv_\alpha}]$ is also convex and hence equals $\{j',\ldots,j'+k'\}$ for some j' and k'. As $1 \leq j'$ and $j'+k' \leq m$, it follows that $k' \leq (m-1) \leq k$. Now define

$$F(j+r) = \begin{cases} j'+r & \text{for } 0 \leq r \leq k' \\ j'+k' & \text{for } k' < r \leq k \end{cases}$$

If n itself has padding at least m, then F is total on $\{1,\ldots,n\}$. Then let C' be F itself, considered as a binary relation, i.e., let C' be the graph of F. If n has padding less than m, then F is defined only on $\{1,\ldots,n-1\}$. In that case let C' be the union of F and $\{(n,j) \mid nCj\}$.

It is seen that $C'[[i]_{\equiv_\alpha}] = C[[i]_{\equiv_\alpha}]$ for any i with padding at least m, and that $C'[\{i\}]$ and $C[\{i\}]$ are the same singleton set for $i < n$ with padding less than m, and it follows that C' is a quasi-functional coupling.

Since $iC'j$ only if $i'Cj$ for some $i' \in [i]_{\equiv_\alpha}$, the assumption $\alpha \vdash_C \beta$ directly implies $\alpha \vdash_{C'} \beta$.

Definition 3. [illegible] $\alpha = a_1;\ldots;a_n$ [illegible] $\beta = b_1;\ldots;b_m$ [illegible] a_i [illegible] **β-definite** [illegible] $a_i \vdash b_j$ [illegible] $a_i \vdash \neg b_j$ [illegible] $j \in \{1,\ldots,m\}$

α [illegible] **β-expanded** [illegible] a_i [illegible] β [illegible] $i \in \{1,\ldots,n-1\}$ [illegible] m

Lemma 4. [illegible] ul [illegible] $\alpha \models_{\text{LO}} \beta$ [illegible] α [illegible] β [illegible] $\alpha \Vdash_{\min} \beta$

[illegible] By the Ex Falso rule, we may assume that each member of α is satisfiable.

Let $D = \bigcup_{i=1}^{i=n} D_i$, where each D_i is

- $\{i-1\}$ if a_i is β-definite and $i \neq n$,
- the rationals in $[i-1,i)$ otherwise.

Now consider the structure $S = (d_0, D, <, s)$, where $d_0 = 0$ and $<$ is the standard ordering and s is such that for each $i \in \{1,\ldots,n\}$,

- $s(i) \in [\![a_i]\!]_{\mathcal{S}(\alpha,\beta)}$ if a_i is β-definite and $i \neq n$,
- otherwise, s distributes the members of $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)}$ densely over D_i (cf. the construction following Definition 1).

Since $S \models \alpha$ so, by assumption, $S \models \beta$. Define $C \subseteq \{1,\ldots,n\} \times \{1,\ldots,m\}$ to be such that iCj iff D_i intersects with the interval for b_j. C is clearly an n,m-coupling, and the construction also guarantees that it is sparse: if $i < n$ has padding less than m, then by assumption a_i is β-definite. Hence D_i is a singleton and can only intersect with the interval for one b_j.

Finally $\alpha \vdash_C \beta$, i.e., iCj implies $a_i \vdash b_j$. We argue for this in cases:

- If $D_i = \{i-1\}$ then a_i is β-definite. Then $a_i \vdash b_j$ follows from the fact that the two have a common model.
- If $D_i = [i-1,i)$, and this interval intersects with the interval for b_j, then b_j is true in all members of $[\![a_i]\!]_{\mathcal{S}(\alpha,\beta)}$. Hence $a_i \models b_j$ by (1) and $a_i \vdash b_j$ by completeness of ul.

$\alpha \Vdash_{\min} \beta$ follows now by the Redistribution Lemma 3 and Lemma 2.

The above lemma is the restricted completeness referred to previously: when α is β-expanded then the pair α, β is in a normal form for which $\models_{\text{LO}}$ and $\Vdash_{\min}$ coincide. To obtain a general procedure for proving (and, in fact, deciding) whether $\alpha \models_{\text{LO}} \beta$ holds, we show how to compute, given the pair (α,β) a finite set $\{(\rho_1,\rho_1'),\ldots,(\rho_k,\rho_k')\}$ of pairs in normal form, such that $\alpha \models_{\text{LO}} \beta$ iff $\rho_i \models_{\text{LO}} \rho_i'$, and hence $\rho_i \Vdash_{\min} \rho_i'$, for every i. For this purpose, we introduce the following definition.

Definition 4. [illegible] $\Vdash_{\mathrm{LO}}$ [illegible] **Cut** [illegible] $\Vdash_{\min}$,

$$\text{Cut}\quad \frac{\alpha_1; a; a; \alpha_2 \Vdash \beta \quad \alpha_1; a \wedge c; \alpha_2 \Vdash \beta \quad \alpha_1; a \wedge \neg c; \alpha_2 \Vdash \beta}{\alpha_1; a; \alpha_2 \Vdash \beta}$$

[illegible] a [illegible] **expansion formula**

In view of Theorem 2, the following lemma is established by an easy verification of soundness of the Cut rule.

Lemma 5. $\Vdash_{\mathrm{LO}}$ [illegible] $\models_{\mathrm{LO}}$

With $\beta = a; b; c; c; b$, $\alpha_1 = a; a \wedge b$, $\alpha_2 = b; b \wedge c$, it can be seen that (3) follows by Cut from $\alpha_1; c \wedge (a \vee b); c \wedge (a \vee b); \alpha_2 \Vdash \beta$ and $\alpha_1; c \wedge (a \vee b) \wedge a; \alpha_2 \Vdash \beta$ and $\alpha_1; c \wedge (a \vee b) \wedge \neg a; \alpha_2 \Vdash \beta$, which are all provable in $\Vdash_{\min}$.

To show completeness (and decidability) of $\Vdash_{\mathrm{LO}}$, we first consider the following generalizations of Cut.

Definition 5. [illegible] $\delta = d_1; \ldots; d_l$ [illegible] (δ) [illegible] $c_1 \wedge \ldots \wedge c_l$ [illegible] c_j [illegible] d_j [illegible] $\neg d_j$ [illegible] **i- ii-** [illegible] **iii-Cut** [illegible] $l \geq 1$ [illegible] a^l [illegible] $a \circ \rho$ [illegible]

$$\text{i-Cut}\quad \frac{\alpha_1; a; a; \alpha_2 \Vdash \beta \quad \alpha_1; a \wedge \rho; \alpha_2 \Vdash \beta \text{ for all } \rho \in \ldots (\delta)}{\alpha_1; a; \alpha_2 \Vdash \beta}$$

$$\text{ii-Cut}\quad \frac{\alpha_1; a^{l+1}; \alpha_2 \Vdash \beta \quad \alpha_1; a \circ \rho; \alpha_2 \Vdash \beta \text{ for all } \rho \in \ldots (\delta)^l}{\alpha_1; a^l; \alpha_2 \Vdash \beta}$$

$$\text{iii-Cut}\quad \frac{\alpha_1; a^{l+1}; \alpha_2 \Vdash \beta \quad \alpha_1; a \circ \rho; \alpha_2 \Vdash \beta \text{ for all } \rho \in \ldots (\delta)^{\leq l}}{\alpha_1; a; \alpha_2 \Vdash \beta}$$

The four cut rules are closely related. Cut is the special case of i-Cut corresponding to δ being a single ul formula c, while i-Cut is the special case of ii-Cut, as well as of iii-Cut, corresponding to $l = 1$. The following lemma is easy to verify and is stated without a proof.

Lemma 6. [illegible] $\models_{\mathrm{LO}}$

Corollary 2. [illegible] ul [illegible] $\models_{\mathrm{LO}}$

[illegible] For any candidate entailment $a_1; \ldots; a_n \models_{LO} \beta$ apply iii-Cut bottom-up, with the first a_i which is not β-definite as expansion formula and with $\delta = \beta$ and $l = |\beta|$, to obtain $1 + 2^l + 2^{2l} + \ldots + 2^{l^2} < 2^{(l+1)^2}$ new items, which by Lemma 6 are all valid iff the original item was valid. Then proceed, for each of the new items, with a new bottom-up application of iii-Cut, this time using the next $a_{i'}$ which is not β-definite as the expansion formula, etc., to obtain eventually less than $2^{n(l+1)^2}$ items in normal form which are all valid iff the original item was. Validity of each of these items is decidable, provided that ul is, by Corollary 1.

From this proof we see that the system obtained by adding iii-Cut to $\Vdash_{\min}$ is complete and decidable with respect to $\models_{\mathrm{LO}}$. However, already $\Vdash_{\mathrm{LO}}$, i.e., $\Vdash_{\min}$ extended with Cut, has these properties and the rest of this Section is devoted to proving this fact by showing admissibility of iii-Cut in $\Vdash_{\mathrm{LO}}$. The proof proceeds stepwise by showing first admissibility of i-Cut and then of ii-Cut. First, we need the following auxiliary result.

Lemma 7. [illegible] $c \vdash a$ [illegible] $\alpha_1; a; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ [illegible] $\alpha_1; c; \alpha_2 \Vdash_{\mathrm{LO}} \beta$

[illegible] Proceeding by induction on proofs, we skip the trivial cases of the rules of $\Vdash_{\min}$ and consider only the final step being an application of the Cut rule. This gives three cases to consider, corresponding to whether the a mentioned in the lemma is the expansion formula itself, or it occurs to its left or to its right. In the two latter cases, the induction hypothesis is applied once to each of the three premises, always strengthening a to c, while in the former case the induction hypothesis is applied twice to the first premise, strengthening a to c, and once to each of the other premises, strengthening $a \wedge b$ and $a \wedge \neg b$ to $c \wedge b$ and $c \wedge \neg b$ respectively.

Lemma 8. [illegible] $\Vdash_{\mathrm{LO}}$

[illegible] We prove this by induction on the length of δ, which is always positive. The base case is just Cut itself; now suppose the result holds for δ, and that

(1) $\alpha_1; a; a; \alpha_2 \Vdash_{\mathrm{LO}} \beta$, and
(2) $\alpha_1; a \wedge \rho; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ for all ρ in [illegible] $(\delta; b)$.

Now let κ be an arbitrary member of [illegible] (δ), then from (1) we obtain $\alpha_1; a \wedge \kappa; a \wedge \kappa; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by Lemma 7, and from (2) $\alpha_1; a \wedge \kappa \wedge b; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ and $\alpha_1; a \wedge \kappa \wedge \neg b; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by definition. Hence by Cut we also obtain (3) $\alpha_1; a \wedge \kappa; \alpha_2 \Vdash_{\mathrm{LO}} \beta$. Since κ was arbitrary, we can now apply the induction hypothesis to (1) and (3), to obtain $\alpha_1; a; \alpha_2 \Vdash_{\mathrm{LO}} \beta$.

Lemma 9. [illegible] $\Vdash_{\mathrm{LO}}$

[illegible] We prove this by induction on l. The base case, for $l = 1$, is just an instance of i-Cut and was shown in the previous Lemma 8. Now suppose the result holds for $l \geq 1$, and that

(1) $\alpha_1; a^{l+2}; \alpha_2 \Vdash_{\mathrm{LO}} \beta$, and
(2) $\alpha_1; a \circ \rho; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ for all $\rho \in$ [illegible] $(\delta)^{l+1}$.

For an arbitrary $\kappa \in$ [illegible] $(\delta)^l$ we obtain from (1) $\alpha_1; a; a; a \circ \kappa; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by Lemma 7, and from (2) $\alpha_1; a \wedge \rho; a \circ \kappa; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ for all $\rho \in$ [illegible] (δ) by definition. Hence we also obtain (3) $\alpha_1; a; a \circ \kappa; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by i-Cut. Since κ was arbitrary, we are now in a position to apply the induction hypothesis to (1) and (3), treating the first occurrence of a in all the involved items as “passive”, to obtain $\alpha_1; a^{l+1}; \alpha_2 \Vdash_{\mathrm{LO}} \beta$.

Lemma 10. [illegible] $\Vdash_{\mathrm{LO}}$

[illegible] We prove this by induction on l. The base case ($l = 1$) is an instance of i-Cut shown in Lemma 8. So suppose the result holds for $l \geq 1$, and that

(1) $\alpha_1; a^{l+2}; \alpha_2 \Vdash_{\mathrm{LO}} \beta$, and
(2) $\alpha_1; a \circ \rho; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ for all ρ in [illegible] $(\tau)^{\leq l+1}$.

In particular $\alpha_1; a \circ \rho; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ then holds for all ρ in [illegible] $(\tau)^{l+1}$, and hence $\alpha_1; a^{l+1}; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by ii-Cut. Combining this with "the rest of (2)" we then obtain $\alpha_1; a; \alpha_2 \Vdash_{\mathrm{LO}} \beta$ by the induction hypothesis.

Theorem 6. $\Vdash_{\mathrm{LO}}$ [illegible] $\models_{\mathrm{LO}}$

The proof is immediate from the previous results. From the proofs it can also be seen that $\Vdash_{\mathrm{LO}}$ remains complete when the use of Cut is restricted to cases in which the cut formula is chosen from formulas occurring in β.

4 Wellorderings

Recall that $\alpha \models_\lambda \beta$ denotes entailment with respect to structures $(0, \lambda, <, s)$ with $<$ the ordering on a limit ordinal λ and s mapping ordinals $< \lambda$ to ul structures. Using the standard translation of modal logic into first-order logic (see for example [1]) one can express the entailment relation in the first-order theory of the ordering. It is known that the first-order theory of every countable ordinal is decidable, c.f. [3]. As a consequence, for every countable λ, the entailment relation $\alpha \models_\lambda \beta$ is decidable, provided that ul is decidable.

The decidability of $\alpha \models_\omega \beta$ also follows from the (much stronger) result from [8] that ω-based linear-time temporal logic is PSPACE-complete. This complexity theoretic result is extended to all countable ordinals in [4]. In the next section we give a simple argument for decidability of $\models_\omega$ based on a form of finite model property, Lemma 11, which may be of independent interest. In the concluding Section 4.2, we discuss some results concerning the entailment relation in the case of wellorderings and list some open problems.

4.1 Decidability of $\models_\omega$

The first step in our decidability proof is a simplification of the definition of $\models_\omega$. For this we use models s defined on finite initial segments of ω, the only place in this paper where we use orderings with a greatest element.

Definition 6. [illegible] $\alpha = a_1; \ldots; a_n$ [illegible] $\beta = b_1; \ldots; b_m$ [illegible]

$$k(\alpha, \beta) = \begin{cases} m & \text{[illegible]}\ a_n \models b_1 \wedge \cdots \wedge b_m \\ m - j & \text{[illegible]}\ a_n \not\models b_j \text{ [illegible] } a_n \models b_{j+1} \wedge \cdots \wedge b_m \end{cases}$$

[illegible] k [illegible] β [illegible] α [2]

---

[2] In this subsection, we use propositional conjunction, as in $a_n \models b_1 \wedge ... \wedge b_m$, only as an abbreviation for $a_n \models b_1$ and ... and $a_n \models b_m$. That is, ul need not contain propositional logic.

Lemma 11. [illegible] $\alpha = a_1;\ldots;a_n$ [illegible] $\beta = b_1;\ldots;b_m$ [illegible] $\alpha \models_\omega \beta$ [illegible] $a_n \models b_m$ [illegible] $s \models \beta$ [illegible] s [illegible] ω [illegible] $s \models \alpha$ [illegible] a_n [illegible] $k(\alpha, \beta)$

[illegible] Let α, β be as above. In the equivalence we have to prove, the implication from right to left is the easiest. Assume the rhs and let $s \models \alpha$ with s defined on ω. The last interval of s is infinite, let s^- be s with the last interval cut down to length $k(\alpha, \beta)$. Then s^- satisfies the condition of the rhs and hence $s^- \models \beta$. By $a_n \models b_m$ it follows that $s \models \beta$.

For the converse, assume $\alpha \models_\omega \beta$. Then in particular $a_n \models b_m$. Let $s \models \alpha$ be as assumed in the rhs, that is, defined on $[0, \ldots, i + k) \subseteq \omega$ and with the last interval $[i, i + k)$ of length $k = k(\alpha, \beta)$. In proving $s \models \beta$ we distinguish two cases.

$k = m$ Then $a_n \models b_1 \wedge \cdots \wedge b_m$. Let s^+ be s extended with ω (arbitrary) models of a_n. Then $s^+ \models \alpha$, so by the lhs we get $s^+ \models \beta$. Since $a_n \models b_1 \wedge \cdots \wedge b_m$ we can shift intervals that (possibly) occur to the right of i to the left and shorten them to length 1. In this way they all fit within the last interval of s. It follows that $s \models \beta$.

$k < m$ Then $a_n \not\models b_j$ with $j = m - k$ and $a_n \models b_{j+1} \wedge \cdots \wedge b_m$. In the argument we will use a propositional model V satisfying $a_n \wedge \neg b_j$. Let s' be s with the last interval replaced by ω copies of V. Then we still have $s' \models \alpha$, so by the lhs we get $s' \models \beta$. Since b_j is false in V, the jth interval of s' must be to the left of i. Intervals used to satisfy the remaining formulas $b_{j+1}, \ldots, b_m$ and (possibly) occurring to the right of i can be shortened and shifted to the left as in the previous case. With this new interval structure we still have $s' \models_\omega \beta$. Restoring the last interval of s, that is, replacing the ω copies of V by the models in the last interval of s, we get $s \models \beta$.

The last step in the last case relies on $a_n \models b_{j+1} \wedge \cdots \wedge b_m$.

Now that we have expressed $\models_\omega$ in terms of [illegible] sequences of models we use the fact these can be viewed as words over an alphabet, where the symbols are valuations. Model classes then become languages. Let $\alpha = a_1;\ldots;a_n$ and $\beta = b_1;\ldots;b_m$ and $k = k(\alpha, \beta)$. Let $V_1, \ldots, V_p$ be all possible valuations of the atoms occurring in α, β. For any proposition a, define $L(a) = \{V_i \mid V_i \models a\}$. Being a finite language consisting of one-letter words, $L(a)$ is regular. The finite models $m \models \alpha$ correspond one-to-one to words in the regular language $L(\alpha) = L(a_1)^+ \cdots L(a_n)^+$, where juxtaposition stands for concatenation and $^+$ for one or more iterations (Kleene $^+$). The finite models $s \models \alpha$ with last interval of length k correspond one-to-one to words in the regular language $L(\alpha, k) = L(a_1)^+ \cdots L(a_{n-1})^+ L(a_n)^k$. In this way we can rephrase the rhs of Lemma 11 as: $a_n \models b_m$ and $L(\alpha, k) \subseteq L(\beta)$. Since inclusion between regular languages is decidable we get the following result.

Theorem 7. [illegible] $\models_\omega$

We have $\alpha \models_\omega \beta$ if and only if either α is unsatisfiable, or $k(\alpha,\beta) > 0$ and $L(\alpha, k(\alpha,\beta)) \subseteq L(\beta)$. All ingredients of the rhs are computable/decidable.

As an example, consider $a; a\vee b; b \models_\omega a; b; a\vee b$. Semantically, given $m \models a; a\vee b; b$ we can see this by looking at the second interval of m. If all models in this interval satisfy a we are done. Otherwise, use the first model in the second interval that satisfies b as second interval (of length 1) for $a; b; a\vee b$. The more general method would be to apply the above theorem. We can actually take two-bit sequences as symbols representing valuations: 11 represents the valuation which makes both a and b true, 10 makes only a true, 01 only b, and 00 neither a nor b. Then $L(a)$ is the regular language $\{10, 11\}$, $L(b)$ is $\{01, 11\}$ and $L(a \vee b)$ is $\{01, 10, 11\}$. Obviously, $k(a; a \vee b; b\ ,\ a; b; a \vee b) = 2$. So by Lemma 11 we have $a; a\vee b; b \models_\omega a; b; a\vee b$ if and only if $L(a)^+L(a\vee b)^+L(b)^2 \subseteq L(a)^+L(b)^+L(a\vee b)^+$. The latter can be verified by a decision procedure for inclusion between regular languages.

4.2 Wellorderings and Open Problems

The following lemma states that $\models_\lambda$ is weakly decreasing in λ.

Lemma 12. $\lambda < \lambda'$: $\alpha \models_{\lambda'} \beta$ $\alpha \models_\lambda \beta$

Let $\alpha = a_1; \ldots; a_n$, $\beta = b_1; \ldots; b_m$ and $k = k(\alpha,\beta)$. Let $\lambda < \lambda'$ be limit ordinals and assume $\alpha \models_{\lambda'} \beta$. Let $s \models \alpha$ for some λ model s. Let V be a model of a_n that, in case $k < m$, also satisfies $\neg b_{m-k}$. Let $[o, \lambda)$ be the last interval used by s to satisfy a_n. Define $s_o^V(o') = s(o')$ if $o' < o$ and $s_o^V(o') = V$ for all $o \le o' < \lambda'$. In other words, s_o^V is s with the last interval replaced by sufficiently many copies of V in order to be a λ' model of α. As a consequence, $s_o^V \models \beta$. By the particular choice of V we have $V \models b_{m-k+1} \wedge \cdots \wedge b_m$. Since $\lambda > o$ is a limit ordinal we have $o + \omega \le \lambda$. Consequently, like in the proof of Lemma 11, we can shorten and shift to the left , i.e., into the interval $[o, \lambda)$, those of the last k intervals that occur in s_o^V to the right of o, obtaining a λ model which still satisfies β. But since $a_n \models b_{m-k+1} \wedge \cdots \wedge b_m$, restoring now back the original $[o, \lambda)$ interval from s, we obtain that $s \models \beta$.

The implication cannot be reversed: we have $a; a \wedge \neg b; a; a \wedge b \models_\omega a; a \wedge \neg b; a \wedge b$ (look at the third interval satisfying a!), but not $a; a \wedge \neg b; a; a \wedge b \models_{\omega+\omega} a; a \wedge \neg b; a \wedge b$. A counterexample to the latter is $(11\,10)^\omega(11)^\omega$, where we use the same representation of valuations by two-bit sequences as in the previous section. A trivial corollary of the previous theorem is: $\alpha \models_\omega \beta$ if and only if there exists a limit ordinal λ such that $\alpha \models_\lambda \beta$.

Since the wellorderings form a class, and $\models_\omega$ is a set, it can be expected that $\models_\lambda$ in Lemma 12 stabilizes. This can be made precise by the following argument. Assume by contradiction that for all λ there exists a $\lambda' > \lambda$ such that $\models_{\lambda'} \subset \models_\lambda$. Define a function f from ordinals to limit ordinals by $f(0) = \omega$, $f(o + 1) =$ the smallest λ such that $\models_\lambda \subset \models_{f(o)}$, and in the limit case $f(\lambda) =$ the smallest λ' such that $\models_{\lambda'} \subset \models_{\lambda''}$, where λ'' is the supremum of all $f(o)$, $o < \lambda$. Then we have that $o \mapsto \models_{f(o)}$ is a strictly decreasing mapping from the class of ordinals into the power set of $\models_\omega$, which is impossible. In fact, using results from [3], it can be shown that $\models_\lambda$ stabilizes for some $\lambda \leq \omega^\omega$ and then, by Lemma 12, coincides with $\models_{\mathrm{WO}}$. We finish by formulating some open problems.

Open Problem 8. [illegible] $\lambda < \lambda' < \omega^\omega$ [illegible] $\models_\lambda$ [illegible] $\models_{\lambda'}$ [illegible] $\lambda = \omega * 2,\ \lambda' = \omega * 3$ [illegible]

Open Problem 9. [illegible] $\models_\lambda$ [illegible] $\lambda \geq \omega$.

Acknowledgement. We are indebted to Stéphane Demri for pointing out the relevance of [3] to us.

References

1. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge University Press, Cambridge (2001)
2. Brown, D.J., Suszko, R.: Abstract logics. Dissertationes Mathematicae 102, 9–42 (1973)
3. Büchi, J.R., Siefkes, D.: VDM 1988. LNM, vol. 328. Springer, Heidelberg (1988)
4. Demri, S., Rabinovich, A.: The complexity of temporal logic with until and since over ordinals (submitted to LPAR 2007)
5. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning about Knowledge. MIT Press, Cambridge (1995)
6. Goranko, V., Montanari, A., Sciavicco, G.: Propositional interval neighborhood temporal logics. Journal of Universal Computer Science 9(9), 1137–1167 (2003)
7. Halpern, J., Shoham, Y.: A propositional modal logic of time intervals. Journal of the ACM 38(4), 935–962 (1991)
8. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. Journal of the ACM 32(3), 733–749 (1985)
9. Szajnkenig, W.: Sequence Logic. PhD thesis, Department of Informatics, University of Bergen, forthcoming
10. Tarski, A.: Logic, Semantics, Metamathematics. Oxford University Press, New York (1956)
11. Walicki, M., Bezem, M.A., Szajnkenig, W.: A strongly complete logic of dense time intervals. In: Alechina, N., Ågotnes, T. (eds.) Proceedings of the Workshop on Logics for Resource-Bounded Agents, ESSLLI, Malaga, Spain (2006)

HORPO with Computability Closure: A Reconstruction

Frédéric Blanqui[1], Jean-Pierre Jouannaud[2,⋆], and Albert Rubio[3]

[1] INRIA & LORIA, Protheo team, Campus Scientifique, BP 239, 54506
Vandœuvre-lès-Nancy Cedex, France
[2] LIX, École Polytechnique, 91400 Palaiseau, France
[3] Technical University of Catalonia, Pau Gargallo 5, 08028 Barcelona, Spain

Abstract. This paper provides a new, decidable definition of the higher-order recursive path ordering in which type comparisons are made only when needed, therefore eliminating the need for the computability closure, and bound variables are handled explicitly, making it possible to handle recursors for arbitrary strictly positive inductive types.

1 Introduction

The Higher-order Recursive Path ordering was first introduced in [3]. The goal was to provide a tool for showing strong normalization of simply typed lambda calculi in which higher-order constants were defined by higher-order recursive rules using plain pattern matching. Inspired by Dershowitz's recursive path ordering for first-order terms, comparing two terms started by comparing their types under a given congruence generated by equating given basic types, before to proceed recursively on the structure of the compared terms. In [4], the type discipline was generalized to a polymorphic type discipline with type constructors, the congruence on types was replaced by a well-founded quasi-ordering on types (in practice, a restriction of the recursive path ordering on types), and the recursive definition itself could handle new cases. There were two variants of the subterm case: in the first, following the recursive path ordering tradition, a subterm of the left-hand side was compared with the whole right-hand side; in the second, a term belonging to the computability closure of the left-hand side was used instead of a subterm. And indeed, a subterm is the basic case of the computability closure construction, whose fixpoint definition included various operations under which Tait and Girard's notion of computability is closed. The ordering and the computational closure definitions shared a lot in common, raising some expectations for a simpler and yet more expressive definition able to handle inductive types, as advocated in [2]. This paper meets these expectations (and goes indeed much further) with a new definition of HORPO that improves over the previous one [4] in several respects:

[⋆] Project LogiCal, Pôle Commun de Recherche en Informatique du Plateau de Saclay, CNRS, École Polytechnique, INRIA, Université Paris-Sud.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 138–150, 2007.
© Springer-Verlag Berlin Heidelberg 2007

1. There is a single decidable recursive definition, instead of a pair of mutually inductive definitions for the computability closure and the ordering itself;
2. In contrast with the definition of HORPO with computability closure, the new definition is decidable and syntax-directed (except, as usual, for the subterm case);
3. Type checking applies only when really needed, that is, when the comparison does not follow from computability arguments;
4. Bound variables are handled explicitly by the ordering, allowing for arbitrary abstractions in the right-hand sides;
5. Strictly positive inductive types are accommodated;
6. There is no need for flattening applications on the right-hand side.

This new definition appears to be powerful enough to prove strong normalization of recursors for arbitrary strictly positive inductive types. The two major technical innovations which make it possible are the integration of the computability closure within the ordering definition on the one hand, and the explicit handling of binders on the other hand. This integration of the computability closure is not obtained by adding new cases in the definition, as was suggested in [2], but instead by eliminating from the previous definition the unnecessary type checks.

2 Higher-Order Algebras

Polymorphic higher-order algebras are introduced in [4]. Their purpose is twofold: to define a simple framework in which many-sorted algebra and typed lambda-calculus coexist; to allow for polymorphic types for both algebraic constants and lambda-calculus expressions. For the sake of simplicity, we will restrict ourselves to monomorphic types in this presentation, but allow us for polymorphic examples. Carrying out the polymorphic case is no more difficult, but surely more painful.

Given a set $\mathcal{S}$ of [illegible] of a fixed arity, denoted by $s : *^n \rightarrow *$, the set of [illegible] is generated by the constructor $\rightarrow$ for [illegible]:

$$\mathcal{T}_{\mathcal{S}} := s(\mathcal{T}_{\mathcal{S}}^n) \mid \mathcal{T}_{\mathcal{S}} \rightarrow \mathcal{T}_{\mathcal{S}}$$
$$\text{for } s : *^n \rightarrow * \ \in \mathcal{S}$$

Types are [illegible] when headed by the $\rightarrow$ symbol, and [illegible] otherwise. $\rightarrow$ associates to the right. We use $\sigma, \tau, \rho, \theta$ for arbitrary types.

Function symbols are meant to be algebraic operators equipped with a fixed number n of arguments (called the [illegible]) of respective types $\sigma_1, \ldots, \sigma_n$, and an [illegible] σ. Let $\mathcal{F} = \biguplus_{\sigma_1,\ldots,\sigma_n,\sigma} \mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma}$. The membership of a given function symbol f to $\mathcal{F}_{\sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma}$ is called a [illegible] and written $f : \sigma_1 \times \ldots \times \sigma_n \rightarrow \sigma$.

The set $\mathcal{T}(\mathcal{F}, \mathcal{X})$ of [illegible] λ[illegible] is generated from the signature $\mathcal{F}$ and a denumerable set $\mathcal{X}$ of variables according to the grammar:

$$\mathcal{T} := \mathcal{X} \mid (\lambda \mathcal{X} : \mathcal{T}_{\mathcal{S}}.\mathcal{T}) \mid @(\mathcal{T}, \mathcal{T}) \mid \mathcal{F}(\mathcal{T}, \ldots, \mathcal{T}).$$

The raw term $\lambda x : \sigma.u$ is an [illegible] and $@(u, v)$ is an application. We may omit σ in $\lambda x : \sigma.u$ and write $@(u, v_1, \ldots, v_n)$ or $u(v_1, \ldots, v_n)$, $n > 0$, omitting applications. $\mathcal{V}ar(t)$ is the set of free variables of t. A raw term t is [illegible] if $\mathcal{V}ar(t) = \emptyset$. The notation $\overline{s}$ shall be ambiguously used for a list, a multiset, or a set of raw terms $s_1, \ldots, s_n$.

Raw terms are identified with finite labeled trees by considering $\lambda x : \sigma.u$, for each variable x and type σ, as a unary function symbol taking u as argument to construct the raw term $\lambda x : \sigma.u$. [illegible] are strings of positive integers. $t|_p$ denotes the [illegible] of t at position p. We use $t \trianglerighteq t|_p$ for the subterm relationship. The result of replacing $t|_p$ at position p in t by u is written $t[u]_p$.

An [illegible] Γ is a finite set of pairs written as $\{x_1 : \sigma_1, \ldots, x_n : \sigma_n\}$, where x_i is a variable, σ_i is a type, and $x_i \neq x_j$ for $i \neq j$. Our typing judgements are written as $\Gamma \vdash_\Sigma s : \sigma$. A raw term s has type σ in the environment Γ if the judgement $\Gamma \vdash_\Sigma s : \sigma$ is provable in the inference system given in Figure 1. An important property of our type system is that a raw term typable in a given environment has a unique type.

Variables:

$$\frac{x : \sigma \in \Gamma}{\Gamma \vdash_\Sigma x : \sigma}$$

Functions:

$$\frac{f : \sigma_1 \times \ldots \times \sigma_n \to \sigma \in \mathcal{F} \qquad \Gamma \vdash_\Sigma t_1 : \sigma_1 \; \ldots \; \Gamma \vdash_\Sigma t_n : \sigma_n}{\Gamma \vdash_\Sigma f(t_1, \ldots, t_n) : \sigma}$$

Abstraction:

$$\frac{\Gamma \cdot \{x : \sigma\} \vdash_\Sigma t : \tau}{\Gamma \vdash_\Sigma (\lambda x : \sigma.t) : \sigma \to \tau}$$

Application:

$$\frac{\Gamma \vdash_\Sigma s : \sigma \to \tau \qquad \Gamma \vdash_\Sigma t : \sigma}{\Gamma \vdash_\Sigma @(s, t) : \tau}$$

Fig. 1. The type system for monomorphic higher-order algebras

Typable raw terms are called [illegible]. We categorize terms into three disjoint classes:

1. [illegible], which are headed by λ;
2. [illegible], which are headed by a function symbol, assuming that the output type of $f \in \mathcal{F}$ is a base type;
3. [illegible], which are variables or headed by an application.

A [illegible] σ of domain $\mathcal{D}om(\sigma) = \{x_1, \ldots, x_n\}$ is a set of triples $\sigma = \{\Gamma_1 \vdash_\Sigma x_1 \mapsto t_1, \ldots, \Gamma_n \vdash_\Sigma x_n \mapsto t_n\}$, such that x_i and t_i have the same type in the environment Γ_i. Substitutions are extended to terms by morphism, variable capture being avoided by renaming bound variables when necessary. We use post-fixed notation for substitution application.

A rewrite rule is a triple $\Gamma \vdash_\Sigma l \to r$ such that $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$, and $\Gamma \vdash_\Sigma l : \sigma$ and $\Gamma \vdash_\Sigma r : \sigma$ for some type σ. Given a set of rules R,

$$s \xrightarrow[l \to r \in R]{p} t \text{ iff } s|_p = l\gamma \text{ and } t = s[r\gamma]_p \text{ for some substitution } \gamma$$

The notation $l \to r \in R$ assumes that the variables bound in l, r (resp. the variables free in l, r) are renamed away from the free variables of $s[]_p$ (resp. the bound variables of $s[]_p$), to avoid captures.

For simplicity, typing environments are omitted in the rest of the paper.

A [illegible] $\succ$ is a well-founded ordering of the set of typable terms which is

(i) [illegible] : $s \succ t$ implies that $u[s] \succ u[t]$;

(ii) [illegible] : $s \succ t$ implies that $s\gamma \succ t\gamma$ for all substitution γ.

(iii) [illegible] : $s \longrightarrow_\beta \cup \longrightarrow_\eta t$ implies $s \succ t$,

In [4], we show that the rewrite relation generated by $R \cup \{\beta, \eta\}$ can be proved by simply checking that $l > r$ for all $l \to r \in R$ with some higher-order reduction ordering.

3 The Improved Higher-Order Recursive Path Ordering

The improved higher-order recursive path ordering on higher-order terms is generated from four basic ingredients: a [illegible]; an [illegible] relationship; a [illegible] on functions symbols; and a [illegible] for the function symbols. Accessibility is a new ingredient originating from inductive types, while the other three were already needed for defining HORPO. We describe these ingredients before defining the improved higher-order recursive path ordering.

3.1 Ingredients

- A quasi-ordering on types $\geq_{\mathcal{T}_\mathcal{S}}$, called [illegible], satisfying the following properties (let $>_{\mathcal{T}_\mathcal{S}} = \geq_{\mathcal{T}_\mathcal{S}} \setminus \leq_{\mathcal{T}_\mathcal{S}}$ be its strict part and $=_{\mathcal{T}_\mathcal{S}} = \geq_{\mathcal{T}_\mathcal{S}} \cap \leq_{\mathcal{T}_\mathcal{S}}$ be its associated equivalence relation):
  1. [illegible] : $>_{\mathcal{T}_\mathcal{S}}$ is well-founded;
  2. [illegible] : $\tau \to \sigma =_{\mathcal{T}_\mathcal{S}} \alpha$ iff $\alpha = \tau' \to \sigma'$, $\tau' =_{\mathcal{T}_\mathcal{S}} \tau$ and $\sigma =_{\mathcal{T}_\mathcal{S}} \sigma'$;
  3. [illegible] : $\tau \to \sigma >_{\mathcal{T}_\mathcal{S}} \alpha$ implies $\sigma \geq_{\mathcal{T}_\mathcal{S}} \alpha$ or $\alpha = \tau' \to \sigma', \tau' =_{\mathcal{T}_\mathcal{S}} \tau$ and $\sigma >_{\mathcal{T}_\mathcal{S}} \sigma'$;
  4. [illegible] : $\tau \geq_{\mathcal{T}_\mathcal{S}} \sigma$ implies both $\alpha \to \tau \geq_{\mathcal{T}_\mathcal{S}} \alpha \to \sigma$ and $\tau \to \alpha \geq_{\mathcal{T}_\mathcal{S}} \sigma \to \alpha$;

  We denote by $\mathcal{T}_\mathcal{S}^{min}$ the set of minimal types with respect to $\geq_{\mathcal{T}_\mathcal{S}}^{\rightarrow} = (>_{\mathcal{T}_\mathcal{S}} \cup \rhd)^*$ (reflexive and transitive closure).

  We say that a data type σ occurs [illegible] (resp. [illegible]) in a type τ if τ is a data type (resp. τ is a data type non equivalent to σ in $=_{\mathcal{T}_\mathcal{S}}$), or if $\tau = \rho \to \theta$ and σ occurs positively (resp. negatively) in θ and negatively (resp. positively) in ρ.
- A set $Acc(f)$ of accessible arguments for every function declaration $f : \sigma_1 \ldots \sigma_n \to \sigma$ with σ being a data type, where $i \in [1..n]$ is said to be [illegible] if all data types occuring in σ_i are smaller than σ in the quasi-order $\geq_{\mathcal{T}_\mathcal{S}}$, and in case of equivalence (with $=_{\mathcal{T}_\mathcal{S}}$), they must occur only positively in σ_i. Note that the application operator $@ : (\alpha \to \beta) \times \alpha \to \beta$ can be seen as a

function symbol with an empty set of accessible positions, since its output type τ may occur negatively in any of its two argument types σ and $\sigma \rightarrow \tau$.

A term u is [illegible] in $f(\overline{s})$, $f \in \mathcal{F}$, iff there is $i \in Acc(f)$ such that $u = s_i$ or u is [illegible] in s_i. [illegible] for $f \in \mathcal{F} \cup \{@\}$ is now obtained by adding the minimal type subterms: $s \rhd_{acc} v : \tau$ iff v is accessible in s, or $\tau \in \mathcal{T}_\mathcal{S}^{min}$, v is a strict subterm of s and $\mathcal{V}ar(v) \subseteq \mathcal{V}ar(s)$. We denote by $\trianglerighteq_{acc}$ the reflexive closure of $\rhd_{acc}$.

- A [illegible] $\geq_\mathcal{F}$ on $\mathcal{F} \cup \{@\}$, with $f >_\mathcal{F} @$ for all $f \in \mathcal{F}$.
- A status (lexicographic or multiset) for all symbols in $\mathcal{F} \cup \{@\}$ with $@ \in Mul$. The status of the symbol f is denoted by $stat_f$.

We recall important properties of the type ordering [4]:

Lemma 1. [illegible] $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$, σ [illegible] iff τ [illegible]

Lemma 2. [illegible] $\geq_{\mathcal{T}_\mathcal{S}}$ [illegible] $>_{\mathcal{T}_\mathcal{S}}$ [illegible] $\geq^{\rightarrow}_{\mathcal{T}_\mathcal{S}}$ [illegible] fi [illegible] $(\geq_{\mathcal{T}_\mathcal{S}} \cup \rhd)^*$ [illegible] $\geq_{\mathcal{T}_\mathcal{S}}$ [illegible] $\rhd$, [illegible] $=_{\mathcal{T}_\mathcal{S}}$

Lemma 3. $\mathcal{T}_\mathcal{S}^{min}$ [illegible] $\mathcal{T}_\mathcal{S} \neq \emptyset$

3.2 Notations

- $s \succ^X t$ for the main ordering, with a finite set of variables $X \subset \mathcal{X}$, with the convention that X is omitted when empty;
- $s : \sigma \succ^X_{\mathcal{T}_\mathcal{S}} t : \tau$ for $s \succ^X t$ and $\sigma \geq_{\mathcal{T}_\mathcal{S}} \tau$;
- $s : \sigma \trianglerighteq_{acc} \succeq^X_{\mathcal{T}_\mathcal{S}} t : \tau$ for $s \trianglerighteq_{acc} w$ for some w and $@(w, \overline{x}) : \sigma' =_{\mathcal{T}_\mathcal{S}} \tau \succeq_{\mathcal{T}_\mathcal{S}} t$ for some $\overline{x} \in X$.

3.3 Ordering Definition

Definition 1. $s : \sigma \succ^X t : \tau$ iff [illegible]

$s = f(\overline{s})$ [illegible] $f \in \mathcal{F}$ [illegible]
( ) $s_i \trianglerighteq_{acc} \succeq^X_{\mathcal{T}_\mathcal{S}} t$ [illegible] i
( ) $t = g(\overline{t})$ [illegible] $f =_\mathcal{F} g \in \mathcal{F}$, $s \succ^X \overline{t}$ [illegible] $\overline{s}(\succ_{\mathcal{T}_\mathcal{S}} \cup \rhd_{acc} \succeq^X_{\mathcal{T}_\mathcal{S}})_{stat_f} \overline{t}$
( ) $t = g(\overline{t})$ [illegible] $f >_\mathcal{F} g \in \mathcal{F} \cup \{@\}$ [illegible] $s \succ^X \overline{t}$
$s = @(u, v)$ [illegible]
( ) $u \trianglerighteq_{acc} \succeq^X_{\mathcal{T}_\mathcal{S}} t$ [illegible] $v \trianglerighteq_{acc} \succeq^X_{\mathcal{T}_\mathcal{S}} t$
( ) $t = @(u', v')$ [illegible] $\{u, v\}(\succ^X_{\mathcal{T}_\mathcal{S}})_{mul}\{u', v'\}$
( ) $u = \lambda x : \alpha.w$ [illegible] $w\{x \mapsto v\} \succeq^X t$
$s = \lambda x : \alpha.u$ [illegible]
( ) $u\{x \mapsto z\} \succeq^X_{\mathcal{T}_\mathcal{S}} t$, [illegible] $z : \alpha$ [illegible]
( ) $t = \lambda y : \beta.v$, $\alpha =_{\mathcal{T}_\mathcal{S}} \beta$ [illegible] $u\{x \mapsto z\} \succ^X v\{y \mapsto z\}$ [illegible] $z : \beta$ [illegible]
( ) $u = @(v, x)$, $x \notin \mathcal{V}ar(v)$ [illegible] $v \succeq^X t$
( ) $s \notin \mathcal{X}$ [illegible] $t \in X$
( ) $s \notin \mathcal{X}, s \neq \lambda x : \alpha.u$, $t = \lambda y : \beta.w$ [illegible] $s \succ^{X \cup \{z\}} w\{y \mapsto z\}$ [illegible] $z : \beta$ [illegible]

Our ordering definition comes in four parts, the first three dealing with left-hand sides headed respectively by an algebraic symbol, the application symbol and an abstraction, while the fourth factors out those cases where the right-hand side is a previously bound variable or an abstraction.

Cases 1 are very similar (up to type checks) to those of Dershowitz's recursive path ordering with the subterm case 1a, the status case 1b and the precedence case 1c. So are Cases 2 and 3. One difference is that there is an additional case for handling respectively beta and eta. A more substantial difference is that variable renaming has become explicit.

The major innovation of this new definition is the annotation of the ordering by the set of variables X that were originally bound in the right-hand side term, but have become free by taking some subterm. This allows rule 4b to pull out abstractions from the right-hand side regardless of the left-hand side term, meaning that abstractions are smallest in the precedence. Note that freed variables become smaller than everything else but variables.

One may wonder why Case 1b is so complicated: the reason is that using recursively $\overline{s}(\succ^{X}_{\mathcal{T}_\mathcal{S}})_{stat_f}\overline{t}$ would yield to lose strong normalization of the ordering.

We give now an example of use of this new definition with the inductive type of Brouwer's ordinals, which constructor *lim* takes an infinite sequence of ordinals to build a new, limit ordinal, hence admits a functional argument of type $\mathbb{N} \rightarrow Ord$, in which Ord occurs positively. As a consequence, the recursor admits a much more complex structure than that of natural numbers, with an explicit abstraction in the right-hand side of the rule for *lim*.

[illegible] Brouwer's ordinals.

$0 : Ord \qquad S : Ord \Rightarrow Ord \qquad lim : (\mathbb{N} \rightarrow Ord) \Rightarrow Ord$

$n : \mathbb{N} \qquad N : \mathbb{N} \qquad F : \mathbb{N} \rightarrow Ord$

$rec : Ord \times \alpha \times (Ord \rightarrow \alpha \rightarrow \alpha) \times ((\mathbb{N} \rightarrow Ord) \rightarrow (\mathbb{N} \rightarrow \alpha) \rightarrow \alpha) \Rightarrow \alpha$

$$\begin{aligned}
&rec(0, U, V, W) && \rightarrow U \\
&rec(s(N), U, V, W) && \rightarrow @(V, N, rec(N, U, V, W)) \\
&rec(lim(F), U, V, W) && \rightarrow @(W, F, \lambda n.rec(@(F, n), U, V, W))
\end{aligned}$$

Although the strong normalization of such rules is known to be difficult to prove, it is checked automatically by our ordering. We only show how the third rule is included in the ordering.

$$rec(lim(F), U, V, W) \succ_{\mathcal{T}_\mathcal{S}} @(W, F, \lambda n.rec(@(F, n), U, V, W))$$

yields 2 subgoals according to Case 1c: $\alpha \geq_{\mathcal{T}_\mathcal{S}} \alpha$ and $rec(lim(F), U, V, W) \succ \{W, F, \lambda n.rec(@(F, n), U, V, W)\}$.
The first one is trivial and the second one simplifies to:

1. $rec(lim(F), U, V, W) \succ W$ which succeeds by Case 1a,
2. $rec(lim(F), U, V, W) \succ F$, which succeeds by Case 1a since F is accessible in $lim(F)$,
3. $rec(lim(F), U, V, W) \succ \lambda n.rec(@(F, n), U, V, W)$ which yields, by Case 4b, $rec(lim(F), U, V, W) \succ^{\{n\}} rec(@(F, n), U, V, W)$ yielding, by Case 1b, two goals

(a) $\{lim(F), U, V, W\}(\succ_{\mathcal{T}_\mathcal{S}} \cup \rhd_{acc} \succeq^{\{n\}}_{\mathcal{T}_\mathcal{S}})_{mul}\{@(F, n), U, V, W\}$,
which reduces to $lim(F) \rhd_{acc} \succeq^{\{n\}}_{\mathcal{T}_\mathcal{S}} @(F, n)$ which holds by Case 1a since F is accessible in $lim(F)$, and

(b) $rec(lim(F), U, V, W) \succ^{\{n\}} \{@(F, n), U, V, W\}$, that decomposes into three goals trivially solved by Case 1a, and
$rec(lim(F), U, V, W) \succ^{\{n\}} @(F, n)$ yielding, by Case 1c,
   i. $rec(lim(F), U, V, W) \succ^{\{n\}} F$, which holds by Case1a, since F is accessible in $lim(F)$, and
   ii. $rec(lim(F), U, V, W) \succ^{\{n\}} n$ which holds by Case 4a, therefore ending the computation.

4 Strong Normalization

Theorem 1. $\succ^+_{\mathcal{T}_\mathcal{S}}$ [illegible]

Contrasting with our previous proposal made of an ordering part and a computability closure part, our new ordering is a decidable inductive definition: $s \succ^X t$ is defined by induction on the triple (n, s, t), using the order $(>_{\mathbb{N}}, \longrightarrow_\beta \cup \rhd, \rhd)_{lex}$, where n is the number of abstractions in t. The quadratic time decidability follows since all operations used are clearly decidable in linear time. The fact that $\succ^X$ is quadratic comes from those cases that recursively compare one side with each subterm of the other side. This assumes of course that precedence and statuses are given, since inferring them yields NP-completeness as is well-known for the recursive path ordering on first-order terms.

The stability and monotonicity proofs are routine. As the old one, the new definition is not transitive, but this is now essentially due to the beta-reduction case 2c. We are left with strong normalization, and proceed as in [4]. The computability predicate differs however in case of data types, since it has to care about inductive type definitions.

4.1 Candidate Terms

Because our strong normalization proof is based on Tait and Girard's reducibility technique, we need to associate to each type σ, actually to the equivalence class of σ modulo $=_{\mathcal{T}_\mathcal{S}}$, a set of terms $[\![\sigma]\!]$ closed under term formation. In particular, if $s \in [\![\sigma \rightarrow \tau]\!]$ and $t \in [\![\sigma]\!]$, then the raw term $@(s, t)$ must belong to the set $[\![\tau]\!]$ even if it is not typable, which may arise in case t does not have type σ but $\sigma' =_{\mathcal{T}_\mathcal{S}} \sigma$. Relaxing the type system to type terms up to type equivalence $=_{\mathcal{T}_\mathcal{S}}$ is routine [4]. We use the notation $t :_C \sigma$ to indicate that the raw term t, called a [illegible] (or simply, a term), has type σ in the relaxed system.

4.2 Candidate Interpretations

In the coming sections, we consider the well-foundedness of the strict ordering $(\succ_{\mathcal{T}_\mathcal{S}})^+$, that is, equivalently, the strong normalization of the rewrite relation defined by the rules $s \longrightarrow t$ such that $s \succ_{\mathcal{T}_\mathcal{S}} t$. Note that the set X of previously bound variables is empty. We indeed have failed proving that the ordering $(\succ^X_{\mathcal{T}_\mathcal{S}})^+$ is well-founded for an arbitrary X, and we think that it [illegible]. As usual in this context, we use Tait and Girard's computability predicate method, with a definition of computability for candidate terms inspired from [4,1].

Definition 2. [illegible] candidate interpretations $\{[\![\sigma]\!]\}_{\sigma \in \mathcal{T}_\mathcal{S}}$ [illegible]

[illegible] σ [illegible] $s :_C \sigma$ [illegible] $s \in [\![\sigma]\!]$ [illegible] $t \in [\![\tau]\!]$ [illegible] t [illegible] $s \succ_{\mathcal{T}_\mathcal{S}} t :_C \tau$.

[illegible] σ [illegible] $s = f(\overline{s}) :_C \sigma$ [illegible] $f : \sigma_1 \ldots \sigma_n \Rightarrow \sigma' \in \mathcal{F}$ [illegible] $s \in [\![\sigma]\!]$ [illegible] $s_i \in [\![\sigma_i]\!]$ [illegible] $i \in Acc(f)$ [illegible] $t \in [\![\tau]\!]$ [illegible] t [illegible] $s \succ_{\mathcal{T}_\mathcal{S}} t :_C \tau$.

[illegible] σ [illegible] $\rho \to \tau$ [illegible] $s \in [\![\sigma]\!]$ [illegible] $@(s,t) \in [\![\tau]\!]$ [illegible] $t \in [\![\rho]\!]$.

[illegible] s [illegible] σ [illegible] computable [illegible] $s \in [\![\sigma]\!]$ [illegible] $\overline{s}$ [illegible] $\overline{\sigma}$ [illegible] ([illegible]) [illegible] γ [illegible] $\{x\gamma \mid x \in \mathcal{D}om(\gamma)\}$ [illegible]

Our definition of candidate interpretations is based on a lexicographic combination of an induction on the well-founded type ordering $>_{\mathcal{T}_\mathcal{S}}^{\rightarrow}$, and a fixpoint computation for data types.

4.3 Computability Properties

We start with a few elementary properties stated without proofs:

Lemma 4. [illegible] $\sigma =_{\mathcal{T}_\mathcal{S}} \tau$ [illegible] $[\![\sigma]\!] = [\![\tau]\!]$

Lemma 5. $s = @(u,v)$ [illegible] u [illegible] v [illegible]

Lemma 6. s [illegible] $s \in \mathcal{T}_\mathcal{S}^{min}$ [illegible]

Lemma 7. [illegible] $\overline{s}$ [illegible] $f(\overline{s}) \rhd_{acc} v$ [illegible] $f \in \mathcal{F} \cup \{@\}$ [illegible] v [illegible]

We now give the fundamental properties of the interpretations. Note that we use our term categorisation to define the computability predicates, and that this is reflected in the computability properties below.

(i) Every computable term is strongly normalizable for $\succ_{\mathcal{T}_\mathcal{S}}$;

(ii) If s is computable and $s \succeq_{\mathcal{T}_\mathcal{S}} t$, then t is computable;

(iii) A neutral term s is computable iff t is computable for all terms t such that $s \succ_{\mathcal{T}_\mathcal{S}} t$;

(iv) An abstraction $\lambda x : \sigma.u$ is computable iff $u\{x \mapsto w\}$ is computable for all computable terms $w :_C \sigma$;

(v) A prealgebraic term $s = f(\overline{s}) :_C \sigma$ such that $f : \overline{\sigma} \to \tau \in \mathcal{F}$ is computable if $\overline{s} :_C \overline{\sigma}$ is computable.

All proofs are adapted from [4], with some additional difficulties. The first four properties are proved together.

[illegible] Properties (i), (ii), (iii), (iv). Note first that the only if part of properties (iii) and (iv) is property (ii). We are left with (i), (ii) and the if parts of (iii) and (iv) which spell out as follows:

Given a type σ, we prove by induction on the definition of $[\![\sigma]\!]$ that

(i) Given $s :_C \sigma \in [\![\sigma]\!]$, then s is strongly normalizable;

(ii) Given $s :_C \sigma \in [\![\sigma]\!]$ such that $s \succeq_{\mathcal{T}_\mathcal{S}} t$ for some $t :_C \tau$, then $t \in [\![\tau]\!]$;

(iii) A neutral candidate term $u :_C \sigma$ is computable if $w :_C \theta \in [\![\theta]\!]$ for all w such that $u \succ_{\mathcal{T}_\mathcal{S}} w$; in particular, variables are computable;

(iv) An abstraction $\lambda x : \alpha.u :_C \sigma$ is computable if $u\{x \mapsto w\}$ is computable for all $w \in [\![\alpha]\!]$.

We prove each property in turn, distinguishing in each case whether σ is a data or functional type.

(ii)
1. Assume that σ is a data type. The result holds by definition of the candidate interpretations.
2. Let $\sigma = \theta \to \rho$. By arrow preservation and decreasingness properties, there are two cases:
    (a) $\rho \geq_{\mathcal{T}_\mathcal{S}} \tau$. Let $y :_C \theta \in \mathcal{X}$. By induction hypothesis (iii), $y \in [\![\theta]\!]$, hence $@(s, y) \in [\![\rho]\!]$ by definition of $[\![\sigma]\!]$. Since $@(s, y) :_C \rho \succ_{\mathcal{T}_\mathcal{S}} t :_C \tau$ by case 2a of the definition, t is computable by induction hypothesis (ii).
    (b) $\tau = \theta' \to \rho'$, with $\theta =_{\mathcal{T}_\mathcal{S}} \theta'$ and $\rho \geq_{\mathcal{T}_\mathcal{S}} \rho'$. Since s is computable, given $u \in [\![\theta]\!]$, then $@(s, u) \in [\![\rho]\!]$. By monotonicity, $@(s, u) \succ^X_{\mathcal{T}_\mathcal{S}} @(t, u)$. By induction hypothesis (ii) $@(t, u) \in [\![\rho']\!]$. Since $[\![\theta]\!] = [\![\theta']\!]$ by Lemma 4, t is computable by definition of $[\![\tau]\!]$.

(i)
1. Assume first that σ is a data type. Let $s \succ_{\mathcal{T}_\mathcal{S}} t$. By definition of $[\![\sigma]\!]$, t is computable, hence is strongly normalizable by induction hypothesis. It follows s is strongly normalizable in this case.
2. Assume now that $\sigma = \theta \to \tau$, and let $s_0 = s :_C \sigma = \sigma_0 \succ_{\mathcal{T}_\mathcal{S}} s_1 :_C \sigma_1 \ldots \succ_{\mathcal{T}_\mathcal{S}} s_n :_C \sigma_n \succ_{\mathcal{T}_\mathcal{S}} \ldots$ be a derivation issuing from s. Therefore $s_i \in [\![\sigma_i]\!]$ by induction on i, using the assumption that s is computable for $i = 0$ and otherwise by the already proved property (ii). Such derivations are of the following two kinds:
    (a) $\sigma >_{\mathcal{T}_\mathcal{S}} \sigma_i$ for some i, in which case s_i is strongly normalizable by induction hypothesis (i). The derivation issuing from s is therefore finite.
    (b) $\sigma_i =_{\mathcal{T}_\mathcal{S}} \sigma$ for all i, in which case $\sigma_i = \theta_i \to \tau_i$ with $\theta_i =_{\mathcal{T}_\mathcal{S}} \theta$. Then, $\{@(s_i, y :_C \theta) :_C \tau_i\}_i$ is a sequence of candidate terms which is strictly decreasing with respect to $\succ_{\mathcal{T}_\mathcal{S}}$ by monotonicity. Since $y :_C \theta$ is computable by induction hypothesis (iii), $@(s_i, y)$ is computable by definition of $[\![\tau_i]\!]$. By induction hypothesis, the above sequence is finite, implying that the starting sequence itself is finite.

    Therefore, s is strongly normalizing as well in this case.

(iii) 1. Assume that σ is a data type. The result holds by definition of $[\![\sigma]\!]$.

2. Assume now that $\sigma = \sigma_1 \to \sigma_2$. By definition of $[\![\sigma]\!]$, u is computable if the neutral term $@(u, u_1)$ is computable for all $u_1 \in [\![\sigma_1]\!]$. By induction hypothesis, $@(u, u_1)$ is computable iff all its reducts w are computable. Since u_1 is strongly normalizable by induction hypothesis (i), we show by induction on the pair $(u_1, |w|)$ ordered by $(\succ_{\mathcal{T}_\mathcal{S}}, >_{\mathbb{N}})$ that all reducts w of $@(u, u_1)$ are computable. Since u is neutral, hence is not an abstraction, there are three possible cases:

   (a) $@(u, u_1) \succ_{\mathcal{T}_\mathcal{S}} w$ by Case 2a, therefore $u \trianglerighteq_{acc} v \succeq_{\mathcal{T}_\mathcal{S}} w$ or $u_1 \trianglerighteq_{acc} v \succeq_{\mathcal{T}_\mathcal{S}} w$ for some v. Since the type of w is smaller or equal to the type of $@(u, u_1)$, it is strictly smaller than the type of u, hence $w \neq u$. Therefore, in case $v = u$, w is a reduct of u, hence is computable by assumption. Otherwise, v is u_1 or a minimal-type subterm of u_1, in which case it is computable by assumption on u_1 and Lemma 6, or a minimal-type subterm of u in which case $u \succ_{\mathcal{T}_\mathcal{S}} v$ by Case 1a or 2a since the neutral term u is not an abstraction, and therefore v is computable by assumption. It follows that w is computable by induction hypothesis (ii).

   (b) $@(u, u_1) \succ_{\mathcal{T}_\mathcal{S}} w$ by Case 2b, therefore $w = @(v, v_1)$ and also $\{u, u_1\} (\succeq_{\mathcal{T}_\mathcal{S}})_{mul} \{w_1, w_2\}$. For type reasons, there are again two cases:
   - w_1 and w_2 are strictly smaller than u, u_1, in which case w_1 and w_2 are computable by assumption or induction hypothesis (ii), hence w is computable by Lemma 5.
   - $u = w_1$ and $u_1 \succ_{\mathcal{T}_\mathcal{S}} w_2$, implying that w_2 is computable by assumption and induction hypothesis (ii). Then, since $(u_1, \_)$ $(\succ_{\mathcal{T}_\mathcal{S}}, >_{\mathbb{N}})_{lex} (w_2, \_)$, we conclude by induction hypothesis.

   (c) $@(u, u_1) \succ_{\mathcal{T}_\mathcal{S}} w$ by Case 4b, then $w = \lambda x : \beta.w'$, $x \notin \mathcal{V}ar(w')$ and $@(u, u_1) \succ w'$. By induction hypothesis (iv) and the fact that $x \notin \mathcal{V}ar(w')$, w is computable if w' is computable. Since the type of $\lambda x : \beta.w'$ is strictly bigger than the type of w', we get $@(u, u_1) \succ_{\mathcal{T}_\mathcal{S}} w'$. We conclude by induction hypothesis, since $(u_1, \lambda x.w') (\succ_{\mathcal{T}_\mathcal{S}}, >_{\mathbb{N}})_{lex}$ (u_1, w').

(iv) By definition of $[\![\sigma]\!]$, the abstraction $\lambda x : \alpha.u :_C \sigma$ is computable if the term $@(\lambda x.u, w)$ is computable for an arbitrary $w \in [\![\alpha]\!]$.

Since variables are computable by induction hypothesis (iii), $u = u\{x \mapsto x\}$ is computable by assumption. By induction hypothesis (i), u and w are strongly normalizable. We therefore prove that $@(\lambda x.u, w)$ is computable by induction on the pair (u, w) compared in the ordering $(\succ_{\mathcal{T}_\mathcal{S}}, \succ_{\mathcal{T}_\mathcal{S}})_{lex}$.

Since $@(\lambda x.u, w)$ is neutral, we need to show that all reducts v of $@(\lambda x.u, w)$ are computable. We consider the four possible cases in turn:

1. If $@(\lambda x.u, w) \succ_{\mathcal{T}_\mathcal{S}} v$ by Case 2a, there are two cases:

   - if $w \succeq_{\mathcal{T}_\mathcal{S}} v$, we conclude by induction hypothesis (ii) that v is computable.

   - if $\lambda x.u \succeq_{\mathcal{T}_\mathcal{S}} v$, then $\lambda x.u \succ_{\mathcal{T}_\mathcal{S}} v$ since the type of $\lambda x.u$ must be strictly bigger than the type of v. There are two cases depending on the latter comparison.

If the comparison is by Case 3a, then $u \succeq_{\mathcal{T}_S} v$, and we conclude by induction hypothesis (ii) that v is computable.

If the comparison is by Case 3b, then $v = \lambda x : \alpha'.u'$ with $\alpha =_{\mathcal{T}_S} \alpha'$. By stability, $u\{x \mapsto w\} \succ_{\mathcal{T}_S} u'\{x \mapsto w\}$, hence $u'\{x \mapsto w\}$ is computable by property (ii) for an arbitrary $w \in [\![\alpha]\!] = [\![\alpha']\!]$ by lemma 4. It follows that v is computable by induction hypothesis, since $(u, \_)(\succ_{\mathcal{T}_S}, \succ_{\mathcal{T}_S})_{lex}(u', \_)$.

2. If $@(\lambda x.u, w) \succ_{\mathcal{T}_S} v$ by case 2b, then $v = @(v_1, v_2)$, and by definition of $\succ$, $\{\lambda x.u, w\}(\succ_{\mathcal{T}_S})_{mul}\{v_1, v_2\}$. There are three cases:
   - $v_1 = \lambda x.u$ and $w \succ_{\mathcal{T}_S} v_2$. Then v_2 is computable by induction hypothesis (ii) and, since $u\{x \mapsto v_2\}$ is computable by the main assumption, $@(v_1, v_2)$ is computable by induction hypothesis, since $(\lambda x.u, w)$ $(\succ_{\mathcal{T}_S}, \succ_{\mathcal{T}_S})_{lex}(\lambda x.u, v_2)$.
   - Terms in $\{v_1, v_2\}$ are reducts of u and w. Therefore, v_1 and v_2 are computable by induction hypothesis (ii) and v is computable by Lemma 5.
   - Otherwise, for typing reason, v_1 is a reduct of $\lambda x.u$ of the form $\lambda x.u'$ with $u \succ_{\mathcal{T}_S} u'$, and v_2 is a reduct of the previous kind. By the main assumption, $u\{x \mapsto v''\}$ is computable for an arbitrary computable v''. Besides, $u\{x \mapsto v''\} \succ_{\mathcal{T}_S} u'\{x \mapsto v''\}$ by stability. Therefore $u'\{x \mapsto v''\}$ is computable for an arbitrary computable v'' by induction hypothesis (ii). Then $@(v_1, v_2)$ is computable by induction hypothesis, since $(u, \_)$ $(\succ_{\mathcal{T}_S}, \succ_{\mathcal{T}_S})_{lex}$ $(u', \_)$.
3. If $@(\lambda x.u, w) \succ_{\mathcal{T}_S} v$ by Case 4b, then $v = \lambda x.v'$, $x \notin \mathcal{V}ar(v')$ and $@(\lambda x.u, w) \succ_{\mathcal{T}_S} v'$. Since $\lambda x.v' \succ_{\mathcal{T}_S} v'$ by Case 3a, v' is computable by induction hypothesis. Since $x \notin \mathcal{V}ar(v')$, it follows that $\lambda x.v'$ is computable.
4. If $@(\lambda x.u, w) \succ_{\mathcal{T}_S} v$ by case 2c, then $u\{x \mapsto w\} \succeq_{horpo} v$. By assumption, $u\{x \mapsto w\}$ is computable, and hence v is computable by property (ii). □

We are left with property (v) whose proof differs from [4].

[illegible] Property (v). As we have seen, each data type interpretation $[\![\sigma]\!]$ is the least fixpoint of a monotone function G on the powerset of the set of terms. Hence, for every computable term $t \in [\![\sigma]\!]$, there exists a smallest ordinal $o(t)$ such that $t \in G^{o(t)}(\emptyset)$, where G^a is the a transfinite iteration of G. The relation $\sqsupset$, defined by $t \sqsupset u$ iff $o(t) > o(u)$, is a well-founded ordering which is compatible with $\succ_{\mathcal{T}_S}$: if $t \succ_{\mathcal{T}_S} u$ then $t \sqsupseteq u$. The proof is by induction on the type ordering. Therefore, $\succ_{\mathcal{T}_S} \cup \sqsupset$ is well-founded on computable terms. Note that the result would again hold for terms headed by a function symbol with a functional output.

We use this remark to build our outer induction argument: we prove that $f(\overline{s})$ is computable by induction on the pair $(f, \overline{s})$ ordered lexicographically by $(>_{\mathcal{F}}, (\succ_{\mathcal{T}_S} \cup \sqsupset)_{stat_f})_{lex}$. This is our outer statement (OH).

Since $f(\overline{s})$ is prealgebraic, it is computable if every subterm at an accessible position is computable (which follows by assumption) and reducts t of s are computable.

Since $\succ_{\mathcal{T}_S}$ is defined in terms of $\succ^X$, we actually prove by an inner induction on the recursive definition of $\succ^X$ the more general inner statement (IH) that

$t\gamma$ is computable for an arbitrary term t such that $f(\overline{s}) \succ^X t$ and computable substitution γ of domain X such that $X \cap \mathcal{V}ar(s) = \emptyset$. Since the identity substitution is computable by property (iii), our inner induction hypothesis implies our outer induction hypothesis.

1. If $f(\overline{s}) \succ^X u$ by Case 4a, Then $u \in X$ and we conclude by assumption on γ that $u\gamma$ is computable.
2. If $f(\overline{s}) \succ^X u$ by Case 1a, then $s_i \unrhd_{acc} t$ for some i and $@(t, \overline{x}) \succeq_{\mathcal{T_S}} u$ for some $\overline{x} \in X$. By assumption on $\overline{s}$ and Lemma 7, t is computable. Since t is a subterm of s and $X \cap \mathcal{V}ar(s) = \emptyset$, then $t\gamma = t$ is computable. It follows that $@(t, \overline{x}\gamma)$ is computable. Thus, by stability, $u\gamma$ is computable.
3. If $f(\overline{s}) \succ^X u$ by case 1b, then $u = g(\overline{u})$, $f =_{\mathcal{F}} g$, $s \succ^X \overline{u}$ and finally $\overline{s}\ (\succ_{\mathcal{T_S}} \cup \rhd_{acc} \succeq^X_{\mathcal{T_S}})_{stat}\ \overline{u}$. By the inner induction hypothesis, $\overline{u}\gamma$ is computable. Assume now that $s_i : \sigma_i \rhd_{acc} v$ and $@(v, \overline{x}) : \sigma'_i =_{\mathcal{T_S}} \sigma_i \succeq_{\mathcal{T_S}} u_j$. Using the fact that $X \cap \mathcal{V}ar(s) = \emptyset$, by stability we get $s_i\gamma = s_i \rhd_{acc} v\gamma = v$ and $@(v, \overline{x})\gamma = @(v, \overline{x}\gamma) : \sigma'_i =_{\mathcal{T_S}} \sigma_i \succeq_{\mathcal{T_S}} u_j\gamma$. Moreover, by definition of computability, $s_i \sqsupset @(v, \overline{x}\gamma)$. Therefore, $u\gamma = f(\overline{u}\gamma)$ is computable by the outer induction hypothesis.
4. If $f(\overline{s}) \succ^X_{\mathcal{T_S}} u$ by case 4b, then $u = \lambda x.v$ with $x \notin \mathcal{V}ar(s)$ and $f(\overline{s}) \succ^{X \cup \{x\}} v$. By the inner induction hypothesis, $v(\gamma \cup \{x \mapsto w\})$ is computable for an arbitrary computable w. Assuming without loss of generality that $x \notin \mathcal{R}an(\gamma)$, then $v(\gamma \cup \{x \mapsto w\}) = (v\gamma)\{x \mapsto w\}$. Therefore, $u = \lambda x.v\gamma$ is computable by computability property (iv).
5. If $f(\overline{s}) \succ^X u$ by Case 1c, then $u = g(\overline{u})$ with $g \in \mathcal{F} \cup \{@\}$ and $s \succ^X \overline{u}$. By the inner induction hypothesis, $\overline{u}\gamma$ is computable. We conclude by Lemma 5 in case $g = @$ and by the outer induction hypothesis if $g \in \mathcal{F}$. □

4.4 Strong Normalization

We are now ready for the strong normalization proof. From the previous properties, one can easily prove the following lemma by induction on the term structure:

Lemma 8. [illegible] γ [illegible] t [illegible] λ [illegible] $t\gamma$ [illegible]

The proof of our main theorem follows from Lemma 8 when using the identity substitution, and of computability property (i).

5 Conclusion

An implementation of the new definition with examples is available from the web page of the third author.

There are still a few possible improvements that we have not yet explored, such as ordering the abstractions according to their type, increasing the set of accessible terms for applications that satisfy the strict positivity restriction, and showing that the new definition is strictly more general that the general schema when adopting the same type discipline. A more difficult problem to be investigated then is the generalization of this new definition to the calculus of constructions along the lines of [5].

References

1. Blanqui, F. (HO)RPO revisited. Research Report 5972, INRIA (2006)
2. Blanqui, F., Jouannaud, J.-P., Rubio, A.: Higher order termination: from Kruskal to computability. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, Springer, Heidelberg (2006)
3. Jouannaud, J.-P., Rubio, A.: The higher-order recursive path ordering. In: 14th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Los Alamitos (1999)
4. Jouannaud, J.-P., Rubio, A.: Polymorphic higher-order recursive path orderings. Journal of the ACM 54(1), 1–48 (2007)
5. Walukiewicz-Chrzaszcz, D.: Termination of rewriting in the Calculus of Constructions. In: Proceedings of the Workshop on Logical Frameworks and Meta-languages (2000)

Zenon: An Extensible Automated Theorem Prover Producing Checkable Proofs

Richard Bonichon[1], David Delahaye[2], and Damien Doligez[3]

[1] LIP6/Paris 6, Paris, France
Richard.Bonichon@lip6.fr
[2] CEDRIC/CNAM, Paris, France
David.Delahaye@cnam.fr
[3] INRIA, Rocquencourt, France
Damien.Doligez@inria.fr

Abstract. We present Zenon, an automated theorem prover for first order classical logic (with equality), based on the tableau method. Zenon is intended to be the dedicated prover of the Focal environment, an object-oriented algebraic specification and proof system, which is able to produce OCaml code for execution and Coq code for certification. Zenon can directly generate Coq proofs (proof scripts or proof terms), which can be reinserted in the Coq specifications produced by Focal. Zenon can also be extended, which makes specific (and possibly local) automation possible in Focal.

1 Introduction

Theorem proving is generally separated into two distinct domains: automated theorem proving and interactive theorem proving. Even if these two domains are obviously connected, it seems that in practice, they have little interaction. Actually, the motivations are quite different: automated theorem proving focuses on heuristic concerns (complexity, efficiency, ...) to solve well-identified problems, whereas interactive theorem proving is more concerned with providing means (essentially tools) to achieve proofs of theorems. As a consequence, in automated theorem proving, it is quite difficult to produce formal proofs and in general, the corresponding tools only generate proof traces, which can be seen as abstractions of formal proofs and cannot be directly translated into machine checkable proofs. In this way, we can understand how complicated it is to integrate automated theorem proving features into interactive theorem provers, which tend to suffer from a certain lack of automation. Over the past ten years, some experiments have aimed to make these two kinds of theorem proving activities interact, such as between Gandalf and HOL by J. Hurd [5], between Otter and ACL2 by W. McCune and O. Shumsky [8], between Bliksem and Coq by M. Bezem, D. Hendriks and H. de Nivelle [2], or more recently between E, SPASS, Vampire and Isabelle by L. C. Paulson and K. W. Susanto [9]. However, these examples of integration are not fully satisfactory, since the design of the corresponding automated theorem provers is clearly separated from the automation that could be required

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 151–165, 2007.
© Springer-Verlag Berlin Heidelberg 2007

by the respective interactive theorem provers. In particular, it is impossible to extend the automated theorem prover to manage a very specific and local need for automation.

In this paper, we present Zenon, an automated theorem prover for first order classical logic (with equality), based on the tableau method. Zenon is not supposed to be only another general-purpose automated theorem prover, but is designed to be the reasoning support mechanism of the Focal [15] environment, initially conceived by T. Hardin and R. Rioboo. Focal is a language in which it is possible to build applications step by step, going from abstract specifications to concrete implementations. These different structures are combined using inheritance and parameterization, inspired by object-oriented programming; in addition, each of these structures is equipped with a carrier set, providing a typical algebraic specification flavor. Moreover, in this language, there is a clean separation between the activities of programming and proving. In particular, the compiler is able to produce OCaml [14] code for execution and Coq [13] code for certification. In this compilation scheme, Zenon is involved in the certification part, between the specification level and the generated Coq implementation. Zenon is intended to be the prover of Focal, whereas Coq is used only as a proof checker to ensure the soundness of the final output.

Beyond the automation itself, Zenon brings an effective help to the design of Focal. First, Zenon uses the tableau method. Even though, these days, this method is generally considered as not very efficient (compared to resolution, for example), it has the advantage of being very appropriate for building formal proofs. In this way, Zenon has a low-level format of proofs, which is very close to a sequent calculus. From this low-level format, Zenon can directly produce proofs for Coq (it could be easily done for other proof assistants). This feature can be seen as a guarantee of soundness for the implementation of Zenon, but it is also essential to Focal, where the Coq proofs produced by Zenon are reinserted in the Coq specifications generated by the Focal compiler and fully verified by Coq. In addition, Zenon is also able to produce proof terms for Coq (using its Curry-Howard isomorphism capability), so that Zenon verifies the De Bruijn criterion [1], i.e. it generates proof terms that can be checked independently by a relatively small and easily hand checked algorithm. This means that it is possible to verify Zenon's proofs without Coq, using another tool that would implement *only* the type-checking of Coq. Second, Zenon can be easily extended and this is directly related to the use of the tableau method, which is also very appropriate to handle additional rules. Thanks to this feature, it is possible to manage specific (and possibly local) needs of automation in Focal, such as arithmetic, induction, etc.

The paper is organized as follows: in Section 2, we give the rules of the search method used by Zenon, as well as the format of the generated proofs (in this part, we also point out some specific implementation techniques, such as the use of non-destructive rules and pruning, the management of lemmas or the extension mechanism); in Section 3, we describe the intermediate proof format produced by translating from the proof search rules; in Section 4, we give the translation from this intermediate format to Coq proofs; in Section 5, we provide some examples of use, coming from the TPTP library but also from Focal applications.

2 MLproof

The MLproof inference rules (Figures 1 and 2) are used by Zenon to search for a proof. These rules are applied with the normal tableau method: starting from the negation of the goal, apply the rules in top-down fashion to build a tree. When all branches are *closed* (i.e. end with an application of a closure rule), the tree is closed. The closed tree is a proof of the goal.

Note that this algorithm is applied in strict depth-first order: we close the current branch before starting work on another branch. Moreover, we work in a non-destructive way: working on one branch will never change the formulas present in any other branch.

We divide these rules into five distinct classes to be used for a more efficient proof search. This extends the usual sets of rules dealing with $\alpha, \beta, \delta, \gamma$-formulas and closure ($\odot$) with the specific rules of Zenon. We list below the five sets of rules and their elements:

α	$\alpha_{\neg\vee}, \alpha_{\wedge}, \alpha_{\neg\Rightarrow}, \alpha_{\neg\neg}, \neg_{\text{refl}}$ unfolding rules
β	$\beta_{\vee}, \beta_{\neg\wedge}, \beta_{\Rightarrow}, \beta_{\Leftrightarrow}, \beta_{\neg\Leftrightarrow}, \neq_{\text{func}}$ trans, pred, sym, transsym, transeq, transeqsym
δ	$\delta_{\exists}, \delta_{\neg\forall}$
γ	$\gamma_{\forall}, \gamma_{\neg\exists}, \gamma_{\forall\text{inst}}, \gamma_{\neg\exists\text{inst}}, \gamma_{\forall\text{un}}, \gamma_{\neg\exists\text{un}}$
$\odot$	$\odot_{\top}, \odot_{\bot}, \odot, \odot_{r}, \odot_{s}$

As hinted by the use of the ϵ symbol in the rules, the δ rules are handled with Hilbert's operator [7] rather than using skolemization.

The following subsections describe specific features of our theorem prover, starting with how metavariables are used in a non-destructive setting.

2.1 Handling of Metavariables

What we call here metavariables are often named *free variables* in tableau-related literature. They are not used as variables in Zenon as they are never substituted.

Instead of substitution, we use the following method: when we encounter a universal formula $\forall x\ P(x)$, we apply rule $\gamma_{\forall M}$, which introduces a new metavariable, linked to this universal formula. Then, when we have a potential contradiction such as $\neg R_r(t, X)$, we apply rule $\gamma_{\forall\text{inst}}$ (with the t given by the potential contradiction) *in the current branch* to our original universal formula. If this instantiation closes the subtree rooted at the $\gamma_{\forall\text{inst}}$ node, we know that pruning (see section 2.2) will remove the nodes between the two γ nodes, hence removing the need for substitution of the metavariable.

If the instantiation does not close the subtree, the formulas containing the metavariable are still available in the current branch to trigger other potential

Closure and cut rules

$$\frac{\bot}{\odot}\,\odot_{\bot} \qquad \frac{\neg\top}{\odot}\,\odot_{\neg\top} \qquad \frac{}{P \mid \neg P}\,\text{cut}$$

$$\frac{\neg R_r(t,t)}{\odot}\,\odot_r \qquad \frac{P \qquad \neg P}{\odot}\,\odot \qquad \frac{R_s(a,b) \qquad \neg R_s(b,a)}{\odot}\,\odot_s$$

Analytic rules

$$\frac{\neg\neg P}{P}\,\alpha_{\neg\neg} \qquad \frac{P \Leftrightarrow Q}{\neg P, \neg Q \mid P, Q}\,\beta_{\Leftrightarrow} \qquad \frac{\neg(P \Leftrightarrow Q)}{\neg P, Q \mid P, \neg Q}\,\beta_{\neg\Leftrightarrow}$$

$$\frac{P \wedge Q}{P, Q}\,\alpha_{\wedge} \qquad \frac{\neg(P \vee Q)}{\neg P, \neg Q}\,\alpha_{\neg\vee} \qquad \frac{\neg(P \Rightarrow Q)}{P, \neg Q}\,\beta_{\neg\Rightarrow}$$

$$\frac{P \vee Q}{P \mid Q}\,\beta_{\vee} \qquad \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}\,\beta_{\neg\wedge} \qquad \frac{P \Rightarrow Q}{\neg P \mid Q}\,\beta_{\Rightarrow}$$

$$\frac{\exists x\ P(x)}{P(\epsilon(x).P(x))}\,\delta_{\exists} \qquad \frac{\neg\forall x\ P(x)}{\neg P(\epsilon(x).P(x))}\,\delta_{\neg\forall}$$

γ-rules

$$\frac{\forall x\ P(x)}{P(X)}\,\gamma_{\forall M} \qquad \frac{\neg\exists x\ P(x)}{\neg P(X)}\,\gamma_{\neg\exists M} \qquad \frac{\forall x\ P(x)}{\forall x_1 ... x_n\ P(s(x_1, ..., x_n))}\,\gamma_{\forall\text{un}}$$

$$\frac{\forall x\ P(x)}{P(t)}\,\gamma_{\forall\text{inst}} \qquad \frac{\neg\exists x\ P(x)}{\neg P(t)}\,\gamma_{\neg\exists\text{inst}} \qquad \frac{\neg\exists x\ P(x)}{\neg\exists x_1 ... x_n\ P(s(x_1, ..., x_n))}\,\gamma_{\neg\exists\text{un}}$$

Relational rules

$$\frac{P(t_1, ..., t_n) \qquad \neg P(s_1, .., s_n)}{t_1 \neq s_1 \mid ... \mid t_n \neq s_n}\,\text{pred} \qquad \frac{f(t_1, ..., t_n) \neq f(s_1, ..., s_n)}{t_1 \neq s_1 \mid ... \mid t_n \neq s_n}\,\text{fun}$$

$$\frac{R_s(s,t) \qquad \neg R_s(u,v)}{t \neq u \mid s \neq v}\,\text{sym} \qquad \frac{\neg R_r(s,t)}{s \neq t}\,\neg_{\text{refl}}$$

$$\frac{R_t(s,t) \qquad \neg R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid t \neq v, \neg R_t(t,v)}\,\text{trans}$$

$$\frac{R_{ts}(s,t) \qquad \neg R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid t \neq u, \neg R_{ts}(t,u)}\,\text{transsym}$$

$$\frac{s = t \qquad R_t(u,v)}{u \neq s, \neg R_t(u,s) \mid \neg R_t(u,s), \neg R_t(t,v) \mid t \neq v, \neg R_t(t,v)}\,\text{transeq}$$

$$\frac{s = t \qquad R_{ts}(u,v)}{v \neq s, \neg R_{ts}(v,s) \mid \neg R_{ts}(v,s), \neg R_{ts}(t,u) \mid t \neq u, \neg R_{ts}(t,u)}\,\text{transeqsym}$$

where R_r, R_s, R_t, and R_{ts} are respectively reflexive, symmetric, transitive, and transitive-symmetric relations.

Fig. 1. MLproof rules (part 1)

Unfolding rules: if $P(x) \hat{=} \mathrm{Def}(x)$ and $f(x) \hat{=} \mathrm{def}(x)$ then

$$\frac{P(x)}{\mathrm{Def}(x)}\ \text{p-unfold} \qquad \frac{\neg P(x)}{\neg \mathrm{Def}(x)}\ \text{p-unfold}_{\neg}$$

$$\frac{f(x) = t}{\mathrm{def}(x) = t}\ \text{f-unfold}_{l=} \qquad \frac{t = f(x)}{t = \mathrm{def}(x)}\ \text{f-unfold}_{r=}$$

$$\frac{f(x) \neq t}{\mathrm{def}(x) \neq t}\ \text{f-unfold}_{l} \qquad \frac{t \neq f(x)}{t \neq \mathrm{def}(x)}\ \text{f-unfold}_{r}$$

Extension rule

$$\frac{C_1, ..., C_p}{H_{11}, ..., H_{1m} \mid ... \mid H_{n1}, ..., H_{nq}}\ \begin{array}{l}\text{ext}(\mathsf{name},\mathsf{args},\\ [C_i],[H_{1j}, ..., H_{nk}])\end{array}$$

where name is the name of a predefined lemma s.t.
$C_1 \land ... \land C_p \Rightarrow \bigvee_j (\bigwedge_i H_i j)$

Fig. 2. MLproof rules (part 2)

contradictions, hence we get as many instantiations as needed from a single application of the $\gamma_{\forall M}$ rule. This means that we do not need to use iterative deepening to ensure completeness.

Let us consider the following example:

$$\dfrac{\forall x, P(x) \lor Q(x) \quad \neg P(a) \quad \neg Q(a)}{\dfrac{P(X) \lor Q(X)}{\dfrac{P(X)}{\dfrac{P(a) \lor Q(a)}{\dfrac{P(a)}{\odot}\,\odot \quad \dfrac{Q(a)}{\odot}\,\odot}\,\beta_{\lor}}\,\gamma_{\forall \text{inst}} \quad Q(X)}\,\beta_{\lor}}\,\gamma_{\forall M}$$

In this case, the rule $\gamma_{\forall \text{inst}}$ is triggered by the match between $\neg P(a)$ and $P(X)$, which tells us to instantiate $\forall x, P(x) \lor Q(x)$ with the value a. This tree is not a complete proof because it has an open branch (under $Q(X)$). As we will see in Section 2.2, this open branch does not need to be explored because we can remove it (along with some nodes) to yield a closed proof tree of the original formulas.

2.2 Minimizing the Tree Size

For efficient proof search, a prover must minimize the size of the search tree. This is done in two ways. The first is by *choosing the order in which the rules are applied*: non-branching rules are tried first. It induces the following $\prec$ ordering on the application of the rules $\odot \prec \alpha \prec \delta \prec \beta \prec \gamma$, stating thereby that any applicable $\odot$ rule has priority over any of the other possible rules.

The second is by *pruning*. When a branching node N has a closed subtree as one of its branches B, we can examine this closed subtree to determine which formulas are useful. If the formula introduced by N in B is not in the set of useful formulas, we can remove N and graft the subtree in its place because the subtree is a valid refutation of B without N.

The notion of *useful formula* is defined as follows: a formula is useful in a subtree if it is one of the formulas appearing in the hypotheses (the upper side) of a rule application in that subtree.

Consider the example of section 2.1. There is a subtree rooted at the ∀inst node. This subtree does not *use* the formula $P(X)$ that appears just above it, because the premise of the ∀inst rule is the formula $\forall x, P(x) \lor Q(x)$ at the root of the proof tree, and none of the other subtree nodes uses $P(X)$. Because of this, we can remove the $\beta_\lor$ node above the subtree, and graft the subtree in its place. We can proceed in the same fashion to remove the $\gamma_{\forall M}$ node, and we get the following tree:

$$\dfrac{\dfrac{\forall x, P(x) \lor Q(x) \qquad \neg P(a) \qquad \neg Q(a)}{P(a) \lor Q(a)}\,\gamma_{\forall\mathrm{inst}}}{\dfrac{P(a)}{\odot}\,\odot \qquad \dfrac{Q(a)}{\odot}\,\odot}\,\beta_\lor$$

This time, the proof tree is closed and the proof search is over. The importance of this pruning is that we have completely avoided doing the proof search below the $Q(X)$ branch by carefully examining the result of the proof search in the $P(X)$ branch, thereby reducing the branching factor of the search tree. In the process, we have reduced the size of the resulting proof as compared to the proof search tree.

2.3 Extensions

Zenon offers the ability to extend its core of deductive rules to match certain specific requirements. For instance, the extension named `Coqbool` is regularly used in the setting of Focal, where a function $P(x, y)$ returning a boolean result is encapsulated into a `Is_true(P(x,y))` predicate as it is translated into the corresponding Coq file. In the case where P is transitive (for example), this prevents Zenon from using its specific inference rules, thereby reducing the efficiency of the proof search. Our solution is to transform all occurrences of `Is_true(P(x,y))` into a corresponding `Is_true__P(x,y)` predicate which will let Zenon make use of its transitivity property.

Concretely, extensions are arbitrary OCaml files that implement new inference rules; they are loaded through command-line options when Zenon is started, along with Coq files containing the lemmas used to translate the inference rules introduced by the extension.

2.4 Subsumption

Whenever the current branch contains a superset of the formulas used in an already-closed subtree, we can graft this subtree at the current node because it is a valid closure of the current branch. The implementation maintains a data structure with all the subtrees closed so far (indexed by their used formulas) and queries this data each time a formula is added to the current branch.

We can illustrate subsumption with the following example:

$$\dfrac{B \lor C \quad B \Rightarrow D \quad C \Rightarrow D \quad D \Rightarrow E \quad \neg E}{\dfrac{B}{\dfrac{\neg B}{\odot}\odot \quad \dfrac{D*}{\dfrac{\neg D}{\odot}\odot \quad \dfrac{E}{\odot}\odot}\beta_{\Rightarrow}}\beta_{\Rightarrow} \qquad \dfrac{C}{\dfrac{\neg C}{\odot}\odot \quad \mathrm{D}}\beta_{\Rightarrow}}\beta_{\lor}$$

Consider the $D*$ subtree in the left half of the tree and the open branch under D. The formulas used by the $D*$ subtree are D, $D \Rightarrow E$, and $\neg E$. The same formulas are already available in the open branch, thus we do not need to search for a proof: we can simply reuse the $D*$ subtree. In fact, the implementation does not copy the subtree, but uses sharing (hence turning the proof tree into a dag). Such shared subtrees appear as lemmas in the Coq proof output.

3 LLproof

LLproof is the low-level language of proofs produced by Zenon, which makes the generation of machine checkable proofs possible (see Section 4 for an example in the framework of Coq). Once a proof has been found with the MLproof rules, it is translated to this sequent-like language. We will sketch a proof of soundness and completeness of MLproof proofs w.r.t. LLproof proofs.

LLproof rules (Figures 3 and 4) indeed describe a one-sided sequent calculus with explicit contractions in every inference rule, which roughly resembles an upside-down non-destructive tableau method. This sequent calculus is extended to handle unfolding, lemmas and the extension mechanism of Zenon.

Translating mid-level to low-level proofs gives us a direct proof of soundness for MLproof w.r.t. LLproof. There is a one-to-one correspondence between parts of the two calculi, most notably those which do not introduce quantifiers in MLproof (quantifier-free fragment, axioms).

We can now proceed to prove the following proposition.

Theorem 1 (Soundness and completeness of MLproof w.r.t. LLproof)

1. *Every formula provable in* LLproof *has a proof in* MLproof*.*
2. *Every formula provable in* MLproof *has a proof in* LLproof*.*

Proof. Proof of (1) is immediate as every rule of LLproof has a direct equivalent in MLproof, except the lemma rule, but we can only apply the lemma rule when we have a proof of the lemma's statement, which we can handle in MLproof by grafting a copy of the lemma's proof in the place of the lemma rule.

The proof of (2) is not so immediate as we have to transform some MLproof rules which are the combination of two or more lower-level rules. It proceeds by induction on the size of the MLproof proofs; the details of the proof are not given here.

Closure and quantifier-free rules

$$\frac{}{\bot \vdash \bot}\,\bot \qquad \frac{}{\neg\top \vdash \bot}\,\neg\top \qquad \frac{}{\Gamma, P, \neg P \vdash \bot}\,\mathrm{ax}$$

$$\frac{}{t \neq t \vdash \bot}\,\neq \qquad \frac{\Gamma, P, \neg\neg P \vdash \bot}{\Gamma, P \vdash \bot}\,\neg\neg \qquad \frac{\Gamma, P \vdash \bot \qquad \Gamma, \neg P \vdash \bot}{\Gamma \vdash \bot}\,\mathrm{cut}$$

$$\frac{\Gamma, P \wedge Q, P, Q \vdash \bot}{\Gamma, P \wedge Q \vdash \bot}\,\wedge \qquad \frac{\Gamma, P \vee Q, P \vdash \bot \qquad \Gamma, P \vee Q, Q \vdash \bot}{\Gamma, P \vee Q \vdash \bot}\,\vee$$

$$\frac{\Gamma, P, \neg Q, \neg(P \Rightarrow Q) \vdash \bot}{\Gamma, \neg(P \Rightarrow Q) \vdash \bot}\,\neg\Rightarrow \qquad \frac{\Gamma, \neg P, P \Rightarrow Q \vdash \bot \qquad \Gamma, Q, P \Rightarrow Q \vdash \bot}{\Gamma, P \Rightarrow Q \vdash \bot}\,\Rightarrow$$

$$\frac{\Gamma, \neg P, \neg Q, \neg(P \vee Q) \vdash \bot}{\Gamma, \neg(P \vee Q) \vdash \bot}\,\neg\vee \qquad \frac{\Gamma, \neg P, \neg(P \wedge Q) \vdash \bot \quad \Gamma, \neg Q, \neg(P \wedge Q) \vdash \bot}{\Gamma, \neg(P \wedge Q) \vdash \bot}\,\neg\wedge$$

$$\frac{\Gamma, P \Leftrightarrow Q, \neg P, \neg Q \vdash \bot \qquad \Gamma, P \Leftrightarrow Q, P, Q \vdash \bot}{\Gamma, P \Leftrightarrow Q \vdash \bot}\,\Leftrightarrow$$

$$\frac{\Gamma, \neg P, Q, \neg(P \Leftrightarrow Q) \vdash \bot \qquad \Gamma, P, \neg Q, \neg(P \Leftrightarrow Q) \vdash \bot}{\Gamma, \neg(P \Leftrightarrow Q) \vdash \bot}\,\neg\Leftrightarrow$$

Fig. 3. LLproof rules (part 1)

4 Producing Coq Proofs

As we said in the introduction, Zenon is able to produce Coq [13] proofs, and this automatic generation is carried out from the LLproof format described in Section 3. From a theoretical point of view, this feature ensures the soundness of the LLproof formalism (w.r.t. a known theory), whereas from a practical point of view, this provides a (local) guarantee of Zenon's implementation. But especially, in the context of the Focal system [15], this allows us to produce homogeneous Coq code (where the Coq proofs built by Zenon are reinserted in the Coq specifications generated by the Focal compiler), that can be fully verified by Coq.

4.1 Translation

The translation consists in producing, from proofs provided in LLproof format, proofs in the theory of the theorem prover we chose to perform the validation, which is Coq in our case. This translation is not straightforward for some reasons inherent to the underlying theory of Coq, but also to Coq itself. One of them is that the theory of Coq is based on an intuitionistic logic, i.e. without the excluded middle, whereas LLproof is purely classical. To adapt the theory of Coq to LLproof, we have to add the excluded middle and the resulting theory is still consistent. But Coq does not provide a genuine classical mode (even if the classical library is loaded), i.e. with a classical sequent allowing several propositions on the right hand side, so that proofs must still be completed using an

Quantifier rules

$$\frac{\Gamma, P(c), \exists x\ P(x) \vdash \bot}{\Gamma, \exists x\ P(x) \vdash \bot}\ \exists \qquad \frac{\Gamma, \neg P(c), \neg\forall x\ P(x) \vdash \bot}{\Gamma, \neg\forall x\ P(x) \vdash \bot}\ \neg\forall$$ where c is a fresh constant

$$\frac{\Gamma, P(t), \forall x\ P(x) \vdash \bot}{\Gamma, \forall x\ P(x) \vdash \bot}\ \forall \qquad \frac{\Gamma, \neg P(t), \neg\exists x\ P(x) \vdash \bot}{\Gamma, \neg\exists x\ P(x) \vdash \bot}\ \neg\exists$$ where t is any closed term

Special rules

$$\frac{\Delta, t_1 \neq u_1 \vdash \bot \qquad \ldots \qquad \Delta, t_n \neq u_n \vdash \bot}{\Gamma, P(t_1, \ldots, t_n), \neg P(u_1, \ldots, u_n) \vdash \bot}\ \text{pred}$$

where $\Delta = \Gamma \cup \{P(t_1, \ldots, t_n), \neg P(u_1, \ldots, u_n)\}$

$$\frac{\Delta, t_1 \neq u_1 \vdash \bot \qquad \ldots \qquad \Delta, t_n \neq u_n \vdash \bot}{\Gamma, f(t_1, \ldots, t_n) \neq f(u_1, \ldots, u_n) \vdash \bot}\ \text{fun}$$

where $\Delta = \Gamma \cup \{f(t_1, \ldots, t_n) \neq f(u_1, \ldots, u_n)\}$

$$\frac{\Gamma, C, H \vdash \bot}{\Gamma, C \vdash \bot}\ \text{def}(\mathsf{name}, C, H)$$

if one can go from C to H by unfolding definition name.

$$\frac{\Delta, H_{11}, \ldots, H_{1m} \vdash \bot \qquad \ldots \qquad \Delta, H_{n1}, \ldots, H_{nq} \vdash \bot}{\Gamma, C_1, \ldots, C_p \vdash \bot}\ \text{ext}(\mathsf{name}, \mathsf{args}, [C_i], [H_{1j}, \ldots, H_{nk}])$$

where $\Delta = \Gamma \cup \{C_1, \ldots, C_p\}$
name is the name of a predefined lemma s.t.
$C_1 \wedge \ldots \wedge C_p \Rightarrow \bigvee_j (\bigwedge_i H_i j)$

$$\frac{}{C \vdash \bot}\ \text{lemma}(\mathsf{name}, \mathsf{args})$$

if C is the conclusion associated with name in the list of previously-done proofs. Arguments args are the parameters of name.

Fig. 4. LLproof rules (part 2)

intuitionistic sequent (with only one proposition to the right hand side) and the excluded middle must be added as an axiom. Such a system does not correspond to Gentzen's LK sequent calculus, which is normally used when doing classical proofs, but rather to Gentzen's LJ sequent calculus provided with an explicit excluded middle rule. From a practical point of view, doing proofs in this system is more difficult than in LK (where the right contraction rule is a good shortcut), but in our case this has little effect because all our proofs are produced automatically.

Beyond predicate calculus in general, Zenon, like most of first order automated deduction systems, considers equality as a special predicate and uses specific rules to deal with it. Thus, to translate equality proofs correctly, we have to extend the theory of LJ with equational logic rules. Such a theory will be called $\mathsf{LJ}_{\mathsf{eq}}$ (due to space constraints, we cannot give the corresponding rules, but this theory is quite standard and can be found in literature).

We have the following theorem:

Theorem 2 (Soundness of LLproof w.r.t. $\mathsf{LJ}_{\mathsf{eq}}$). *Every sequent provable in* LLproof *has a proof in* $\mathsf{LJ}_{\mathsf{eq}}$.

Proof. The proof is done by induction over the structure of the proof of the sequent in LLproof. Due to space constraints, we cannot detail the many cases, but as an example, we can consider the translation of the $\neg\wedge$ rule of LLproof, which is the following:

$$\dfrac{\dfrac{\pi_1}{\Gamma, \neg(P \wedge Q), \neg P \vdash \bot} \qquad \dfrac{\pi_2}{\Gamma, \neg(P \wedge Q), \neg Q \vdash \bot}}{\Gamma, \neg(P \wedge Q) \vdash \bot}\ \neg\wedge$$

where π_1 and π_2 are respectively the proofs of $\Gamma, \neg(P \wedge Q), \neg P \vdash \bot$ and $\Gamma, \neg(P \wedge Q), \neg Q \vdash \bot$.

This rule is translated in $\mathsf{LJ}_{\mathsf{eq}}$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\widehat{\pi_1}}{\Gamma, \neg(P \wedge Q), \neg P \vdash \bot}}{\Gamma, \neg(P \wedge Q) \vdash \neg\neg P}\ \neg_{\mathsf{right}}}{\Gamma, \neg(P \wedge Q) \vdash P}\ \mathsf{em} \qquad \dfrac{\dfrac{\dfrac{\widehat{\pi_2}}{\Gamma, \neg(P \wedge Q), \neg Q \vdash \bot}}{\Gamma, \neg(P \wedge Q) \vdash \neg\neg Q}\ \neg_{\mathsf{right}}}{\Gamma, \neg(P \wedge Q) \vdash Q}\ \mathsf{em}}{\Gamma, \neg(P \wedge Q) \vdash P \wedge Q}\ \wedge_{\mathsf{right}}}{\Gamma, \neg(P \wedge Q), \neg(P \wedge Q) \vdash \bot}\ \neg_{\mathsf{left}}}{\Gamma, \neg(P \wedge Q) \vdash \bot}\ \mathsf{cont}$$

where $\widehat{\pi_1}$ and $\widehat{\pi_2}$ are the translated proofs of π_1 and π_2, em the excluded middle rule, cont the left contraction rule, $\neg/\wedge_{\mathsf{right}}$ the right rule for $\neg/\wedge$, and $\neg_{\mathsf{left}}$ the left rule for $\neg$.

4.2 Implementation

General Scheme. The proof of Theorem 2 allows Zenon to produce Coq proofs from proofs in LLproof, since $\mathsf{LJ}_{\mathsf{eq}}$ is included in the underlying theory of Coq, i.e. the Calculus of Inductive Constructions (CIC for short). Actually, we have two kinds of translations: a first one generating proof scripts and a second one directly generating proof terms (thanks to the Curry-Howard isomorphism capability of Coq). In both translations, in order to factorize proofs and especially to minimize the size of the produced proofs, the idea is not to build the proof scripts corresponding to the translated rules, but to prove a lemma for each translated rule once and for all (a macro tactic in $\mathcal{L}_{tac}$ is not appropriate because the body of these macros is rerun each time a translated rule is used in a proof). Thus, the generated Coq proofs are just sequences of applications of these lemmas, and they are not only quite compact, but also quite efficient in the sense that the corresponding Coq checking is fast. For instance, if we consider the $\neg\wedge$ rule of LLproof translated in the proof of Theorem 2, the associated Coq lemma is the following:

```
Lemma zenon_notand : forall P Q : Prop,
  (~P → False) → (~Q → False) → (~(P ∧ Q) → False).
```

As an example of complete Coq proof produced by Zenon and involving the previous lemma, let us consider the proof of $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$, where P and Q are two propositional variables. For this proof, Zenon is able to generate a Coq proof script as follows:

```
Parameters P Q : Prop.
Lemma de_morgan : ~(P ∧ Q) → ~P ∨ ~Q.
Proof.
  apply NNPP. intro G.
  apply (notimply_s _ _ G). zenon_intro H2. zenon_intro H1.
  apply (notor_s _ _ H1). zenon_intro H4. zenon_intro H3.
  apply H3. zenon_intro H5.
  apply H4. zenon_intro H6.
  apply (notand_s _ _ H2);
    [ zenon_intro H8 | zenon_intro H7 ].
  exact (H8 H6).
  exact (H7 H5).
Qed.
```

where NNPP is the excluded middle, *rule*_s (where *rule* is notimply, notor, etc) a definition which allows us to apply partially the corresponding lemma *rule* providing the arguments at any position (not only beginning by the leftmost position), and zenon_intro a macro tactic to introduce (in the context) hypotheses with possibly fresh names if the provided names are already used.

For the same example, Zenon is also able to directly produce the following proof term (without the help of Coq):

```
Parameters P Q : Prop.
Lemma de_morgan : ~(P ∧ Q) → ~P ∨ ~Q.
Proof.
  exact (NNPP _ (fun G : ~(~(P ∧ Q) → ~P ∨ ~Q) ⇒ (notimply
    (~(P ∧ Q)) (~P ∨ ~Q) (fun (H5 : ~(P ∧ Q))
    (H8 : ~(~P ∨ ~Q)) ⇒ (notor (~P) (~Q) (fun (H6 : ~~P)
    (H7 : ~~Q) ⇒ (H7 (fun H1 : Q ⇒ (H6
    (fun H3 : P ⇒ (notand P Q (fun H4 : ~P ⇒ (H4 H3))
    (fun H2 : ~Q ⇒ (H2 H1)) H5)))))) H8)) G))).
Qed.
```

As said in the introduction, this possibility of generating proof terms is particularly important in the sense that Zenon verifies the De Bruijn criterion [1], i.e. it generates a proof format that can be checked by Coq but also independently, by means of another program or proof system which implements the same type theory. For example, as an alternative and with an appropriate printer, we can imagine using the Matita [16] theorem prover, which has the same underlying theory (CIC) as Coq.

Difficulties. In this implementation, we have to be aware of some difficulties. One of them is that we plug first order logic, which is a priori untyped, into a typed calculus (CIC). To deal with this problem, we consider that we have a mono-sorted first order logic, of sort U, and we provide types to variables, constants, predicates and functions explicitly (the type inference offered by Coq does not always allow us to guess these types). Obviously, this must be done only when dealing with purely first order propositions, but can be avoided with propositions coming from Coq or Focal, which are possible inputs for Zenon, since these systems are strongly typed and Zenon keeps the corresponding type information (this is possible since Zenon works in a non-destructive way, see Section 2); in this case, we generally have a multi-sorted first order logic.

Another difficulty, probably deeper, is that mono/multi-sorted first order logic implicitly supposes that each sort is not empty, while in the CIC, types may be not inhabited. This problem is fixed by skolemizing the theory and considering at least one element for each sort, e.g. E for U. Thus, for example, it is possible to prove Smullyan's drinker *paradox* with Zenon as follows:

```
Parameter U : Set.
Parameter E : U.
Parameter d : U → Prop.
Lemma drinker_paradox :
  exists X : U, (d X) → forall Y : U, (d Y).
Proof.
  apply NNPP. intro G.
  apply G. exists E. apply NNPP. zenon_intro H3.
  apply (notimply_s _ _ H3). zenon_intro H5. zenon_intro H4.
  apply H4. zenon_intro T0. apply NNPP. zenon_intro H6.
  apply G. exists T0. apply NNPP. zenon_intro H7.
  apply (notimply_s _ _ H7). zenon_intro H8. zenon_intro H4.
  exact (H6 H8).
Qed.
```

5 Using Zenon in Practice

In this section, we consider the effectiveness of Zenon through benchmarks and applications. The interested reader can get the distribution of Zenon, which is available either as part of the Focal environment at `http://focal.inria.fr/`, or directly (as a separate tool) at `http://focal.inria.fr/zenon/`.

5.1 Benchmarks

In order to see how Zenon fares w.r.t. available first-order theorem provers, we benchmarked it against parts of the latest TPTP library [12] release (v3.2.0).

The Zenon runs were made on an Apple Power Mac Core 2 Duo 2 GHz, with Zenon's default timeout of 5 min and size limit of 400 Mbytes. The set of TPTP syntactic problems SYN was chosen as representative of Zenon's typical target problems, and indeed we get good results. We also tried Zenon against the problems of the FOF category for the latest CASC competition [11].

Problems	Proof found	No proof		
		time	size	other
SYN theorems (282)	264	10	7	1
CASC-J3 (150)	48	46	56	0

Some of the formulas proved by Zenon in CASC have a rather high rating, such as SWV026+1 (0.79), SWV038+1 (0.71), or MSC010+1 (0.57). This last one consists in proving $\neg\neg P$, assuming P, where P is a large first-order formula. Thanks to the tableau method, Zenon does not need to decompose the formula, and the proof is found immediately. All the proofs found by Zenon were verified by Coq.

5.2 The EDEMOI Project

In the framework of the EDEMOI[1] [10] project, Zenon was used to certify the formal models of two regulations related to airport security: the first one is the international standard Annex 17 produced by the International Civil Aviation Organization (ICAO), an agency of the United Nations; the second one is the European Directive Doc 2320 produced by the European Civil Aviation Conference (ECAC) and which is supposed to refine the first one at the European level. The EDEMOI project aims to integrate and apply several requirements engineering and formal methods techniques to analyze standards in the domain of airport security. The novelty of the methodology developed in this project, resides in the application of techniques, usually reserved for safety-critical software, to the domain of regulations (in which no implementation is expected).

The two formal models of the two considered standards were completed using the Focal [15] environment and can be found in [3], where the reader can also find a brief description of Focal. In this formalization, Zenon was used to prove the several identified theorems ensuring the correctness and the completeness of both regulations (consistency was not studied formally). Concretely, the development represents about 10,000 lines of Focal and 200 proofs (2 years to be completed). Regarding the validation part, Zenon allowed us to discharge most of the proof obligations automatically (about 90% of them). Actually, Zenon also succeeded in completing the remaining 10% automatically but beyond the default timeout (set to 3 min in Focal). This tends to show that Zenon is quite appropriate when dealing with abstract specifications (no concrete types and very few definitions).

---

[1] The EDEMOI project is supported by the French National "Action Concertée Incitative Sécurité Informatique".

Zenon also helped us to study the consistency of the regulations from a practical point of view. The idea is to try to derive False from the set of security properties and to let Zenon work on it for a while. If the proof succeeds then we have a contradiction, otherwise we can only have a certain level of confidence. This approach may seem rather naive but appears quite pertinent when used to identify the correlation between the several security measures according to specific attack scenarios. The principle is to falsify an existing hypothesis or to add an inconsistent hypothesis and to study its impact over the entire regulation, i.e. where the potential conflicts are located and which security properties are concerned. For more information regarding this experiment with Zenon, the reader can refer to [4].

6 Conclusion

Zenon is an experiment in progress, but we already have a reasonably powerful prover (see the benchmarks) that can output actual proofs in Coq format (proof scripts or proof terms) for use in a skeptic-style system, such as the Focal environment for example. In addition, the help provided by Zenon in the EDEMOI project framework, where most of the proofs were discharged (and even all the proofs with an extended timeout), tends to show how this tool is appropriate for real-world applications, so that we can be quite optimistic regarding its use, in particular in the context of Focal.

Future work will focus on improving the handling of metavariables in order to get better heuristics for finding the right instantiations, and on implementing some theory-based reasoning by using the extension mechanism of Zenon. Amongst other extensions, we plan to add a theory of arithmetic, but also reasoning by induction (this feature is under development), which is crucial when dealing with specifications close to implementations involving, in particular, concrete datatypes. Finally, it is quite important to apply Zenon to other case-studies, not only to get a relative measure of its automation power, but also to understand the practical needs of automation. For example, proofs provided by Zenon are progressively integrated into the Focal standard library [15] (which mainly consists of a large kernel of Computer Algebra), and a certified development regarding security policies [6] is in progress.

References

1. Barendregt, H., Barendsen, E.: Autarkic Computations in Formal Proofs. Journal of Automated Reasoning (JAR) 28(3), 321–336 (2002)
2. Bezem, M., Hendriks, D.H., de Nivelle, H.: Automated Proof Construction in Type Theory Using Resolution. Journal of Automated Reasoning (JAR) 29(3–4), 253–275 (2002)
3. Delahaye, D., Étienne, J.-F., Donzeau-Gouge, V.V.: Certifying Airport Security Regulations using the Focal Environment. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 48–63. Springer, Heidelberg (2006)

4. Delahaye, D., Étienne, J.-F., Donzeau-Gouge, V.V.: Reasoning about Airport Security Regulations using the Focal Environment. In: International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA), Paphos (Cyprus) (November 2006)
5. Hurd, J.: Integrating Gandalf and HOL. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs 1999. LNCS, vol. 1690, pp. 311–322. Springer, Heidelberg (1999)
6. Jaume, M., Morisset, C.: Formalisation and Implementation of Access Control Models. In: Information Assurance and Security (IAS), International Conference on Information Technology (ITCC), Las Vegas (USA), pp. 703–708. IEEE Computer Society Press, Los Alamitos (2005)
7. Leisenring, A.C.: Mathematical Logic and Hilbert's ϵ-Symbol. MacDonald Technical and Scientific, London (1969) ISBN 0356026795
8. McCune, W., Shumsky, O.: System Description: IVY. In: McAllester, D. (ed.) CADE-17. LNCS, vol. 1831, pp. 401–405. Springer, Heidelberg (2000)
9. Paulson, L.C., Susanto, K.W.: Source-Level Proof Reconstruction for Interactive Theorem Proving. In: Theorem Proving in Higher Order Logics (TPHOLs). LNCS, Springer, Heidelberg (2007)
10. The EDEMOI Project (2003), `http://www-lsr.imag.fr/EDEMOI/`
11. Sutcliffe, G.: CASC-J3 - The 3rd IJCAR ATP System Competition. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 572–573. Springer, Heidelberg (2006)
12. Sutcliffe, G., Suttner, C.B.: The TPTP Problem Library: CNF Release v1.2.1. Journal of Automated Reasoning (JAR) 21(2), 177–203 (1998)
13. The Coq Development Team. Coq, version 8.1. INRIA (November 2006), available at: `http://coq.inria.fr/`
14. The Cristal Team. Objective Caml, version 3.10. INRIA (May 2007), available at: `http://caml.inria.fr/`
15. The Focal Development Team. Focal, version 0.3.1. CNAM/INRIA/LIP6 (May 2005), available at: `http://focal.inria.fr/`
16. The HELM Team. Matita, version 0.1.0. Computer Science Department, University of Bologna (July 2006), available at: `http://matita.cs.unibo.it/`

Matching in Hybrid Terminologies

Sebastian Brandt

School of Computer Science, Manchester, UK
`brandt@cs.manchester.ac.uk`

Abstract. In the area of Description Logic (DL) based knowledge representation, hybrid terminologies have been proposed as a means to make non-standard inference services available to knowledge bases that contain general concept inclusion (GCI) axioms. Building on existing work on subsumption in hybrid terminologies, the present paper provides the first in-depth investigation of the non-standard inferences least-common subsumer, and matching in hybrid $\mathcal{EL}$-TBoxes; providing sound and complete algorithms for both inference services.

1 Motivation

In Description Logic (DL) based knowledge representation (KR), intensional knowledge of a given domain is represented by a terminology (TBox) that defines properties of concepts relevant to the domain [1]. A TBox usually comprises *definitions* of the form $A \equiv C$ by which a *concept name* A is assigned to a *concept description* C. Concept descriptions are terms built from atomic concepts by means of a set of constructors provided by the DL under consideration. TBoxes are interpreted with a model-theoretic *semantics* which allows to reason over the terminology in a formally well-defined way. Our DL of interest is $\mathcal{EL}$ which provides top concept ($\top$), conjunction ($\sqcap$), and existential restriction ($\exists r.C$).

General TBoxes additionally allow for *general concept inclusion (GCI)* axioms of the form $C \sqsubseteq D$, where both C and D may be complex concept descriptions. GCIs define implications ("D holds whenever C holds") relevant to the terminology as a whole. The utility of GCIs for practical KR applications has been examined in depth; see, e.g., [2,3,4]. In addition to constraining (admissible models of) terminologies further without explicitly changing all its definitions, using GCIs can lead to smaller, more readable TBoxes, and can facilitate the re-use of data in applications of different levels of detail. Consequently, GCIs are supported by most modern DL reasoners such as FaCT [5], Racer [6], Pellet [7], and Cel [8].

One of the most important reasoning services provided by such DL systems is *classification*, i.e., computing the subsumption hierarchy. Before DL systems can be deployed for reasoning over terminologies in an application area, however, the relevant TBoxes must be built-up and maintained. In order to support these knowledge engineering tasks, additional so-called 'non-standard' inference services have been proposed, most notably *least common subsumer (lcs)* [9,10,11,12] and *matching* [13,14,15]. As discussed in [16], the lcs facilitates the build-up of

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 166–180, 2007.
© Springer-Verlag Berlin Heidelberg 2007

DL knowledge bases in a 'bottom-up' fashion suitable for domain experts with limited KR background. Among other applications, matching can be used as a means of querying TBoxes for concepts of a certain structure [17]. This can be utilized to construct new concepts by retrieving and modifying structurally similar ones in the TBox.

Unfortunately, non-standard inferences are not straightforwardly available for general TBoxes: it has been shown in [18] that lcs need not always exist, even for cyclic $\mathcal{EL}$-TBoxes interpreted with descriptive semantics, the standard semantics for DL systems. This result carries over to general $\mathcal{EL}$-TBoxes and any extension of $\mathcal{EL}$. The same holds for matching which relies on the lcs.

In order to provide non-standard inferences in the presence of GCIs, so-called [illegible] have been proposed [19]. A hybrid $\mathcal{EL}$-TBox is a pair $(\mathcal{F}, \mathcal{T})$ of a general TBox $\mathcal{F}$ ('foundation') and a possibly cyclic TBox $\mathcal{T}$ ('terminology') defined over the same set of atomic concepts and roles. $\mathcal{F}$ serves as a foundation of $\mathcal{T}$ in that the GCIs in $\mathcal{F}$ define relationships between concepts used as atomic concept names in the definitions in $\mathcal{T}$. Hence, $\mathcal{F}$ lays a foundation of general implications constraining $\mathcal{T}$. The semantics of hybrid TBoxes is different from the usual descriptive semantics: while the foundation of a hybrid TBox is interpreted with descriptive semantics, the terminology is interpreted with so-called greatest-fixpoint (gfp) semantics to be introduced in detail in Section 2.

With respect to non-standard inferences for hybrid $\mathcal{EL}$-TBoxes, our point of departure is as follows: it has been sketched in [19] how an equivalence-preserving reduction from hybrid to cyclic $\mathcal{EL}$-TBoxes with gfp-semantics can be exploited to utilize the lcs defined for cyclic $\mathcal{EL}$-TBoxes with gfp-semantics in [18]. The lcs algorithm thus obtainable for hybrid $\mathcal{EL}$-TBoxes has not yet been studied, though. In case of matching, the above mentioned reduction appears useful as well, only that no matching algorithm for cyclic $\mathcal{EL}$-TBoxes with descriptive semantics exists as yet. The present paper closes both gaps before turning to matching in hybrid TBoxes: after introducing matching in cyclic $\mathcal{EL}$-TBoxes with gfp-semantics in Section 3 and the least-common subsumer for hybrid TBoxes in Section 4.1, our matching algorithm for hybrid TBoxes is presented in Section 4.2. It should be noted that matching problems have not yet been defined for cyclic or hybrid $\mathcal{EL}$ TBoxes. Hence, Sections 3 and 4 start by introducing the relevant notions for the cyclic and hybrid case, respectively. Given that hybrid TBoxes may be viewed as a rather exotic KR formalism, we conclude by discussing the utility of our results for common general $\mathcal{EL}$-TBoxes.

All details and complete proofs can be found in our technical report [20].

2 Formal Preliminaries

[illegible] are inductively defined with the help of a set of concept [illegible], starting with arbitrary but fixed disjoint sets $\mathsf{N}_{\mathrm{prim}} \uplus \mathsf{N}_{\mathrm{def}} =: \mathsf{N}_{\mathrm{con}}$ of [illegible] ($\mathsf{N}_{\mathrm{prim}}$) and [illegible] ($\mathsf{N}_{\mathrm{def}}$), respectively, and a set $\mathsf{N}_{\mathrm{role}}$ of [illegible]. The DL $\mathcal{EL}$ provides the concept constructors top-concept ($\top$), conjunction ($\sqcap$), and existential restrictions ($\exists r.C$). Concept descriptions using only these constructors are called $\mathcal{EL}$-concept descriptions.

As usual, the semantics of concept descriptions is defined in terms of an [illegible] $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is a non-empty set and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $P \in \mathsf{N}_{\text{prim}} \cup \mathsf{N}_{\text{def}}$ to a subset $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each role name $r \in \mathsf{N}_{\text{role}}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary $\mathcal{EL}$-concept descriptions is defined inductively as follows: $\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and

$$(\exists r.C)^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid \exists y\colon (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}.$$

The main purpose of DLs is to be used as underlying representation language for knowledge bases. Two common kinds of DL knowledge bases, TBoxes and general TBoxes, are defined as follows.

For every $A \in \mathsf{N}_{\text{def}}$ and every $\mathcal{EL}$-concept description C over N_{con} and N_{role}, $A \equiv C$ is a [illegible] A. Every finite set of definitions is an $\mathcal{EL}$ [illegible] *($\mathcal{EL}$ [illegible])* over N_{def}, N_{prim}, and N_{role} iff it contains at most one definition of A for every $A \in \mathsf{N}_{\text{def}}$. An $\mathcal{EL}$-TBox $\mathcal{T}$ is [illegible] iff $\mathcal{T}$ is of the form $\{A_i \equiv C_i \mid 1 \leq i \leq n\}$ such that for every $i \in \{1, \ldots, n\}$, only defined names from $\{A_1, \ldots, A_{i-1}\}$ occur in C_i. For concept descriptions C, D over N_{con} and N_{role}, $C \sqsubseteq D$ is a [illegible] *(GCI)* [illegible]. Every finite set of GCIs is a [illegible] $\mathcal{EL}$ [illegible]. For every $\mathcal{EL}$-TBox $\mathcal{T}$, denote by $\mathsf{N}_{\text{con}}^{\mathcal{T}}$ ($\mathsf{N}_{\text{def}}^{\mathcal{T}}$, $\mathsf{N}_{\text{prim}}^{\mathcal{T}}$) and $\mathsf{N}_{\text{role}}^{\mathcal{T}}$ the sets of all (defined, primitive) concept names and role names, respectively, occurring in $\mathcal{T}$. For general $\mathcal{EL}$-TBoxes, only $\mathsf{N}_{\text{con}}^{\mathcal{T}}$ and $\mathsf{N}_{\text{role}}^{\mathcal{T}}$ apply. For the sake of brevity, we may write TBox instead of $\mathcal{EL}$-TBox.

Descriptive semantics: an interpretation $\mathcal{I}$ is a [illegible] $\mathcal{T}$ *($\mathcal{I} \models \mathcal{T}$)* iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every GCI $C \sqsubseteq D \in \mathcal{T}$. Every (non-general) TBox can be viewed as a general TBox since every definition $A \equiv C$ is equivalent to the pair of GCIs $A \sqsubseteq C$, $C \sqsubseteq A$. This semantics is usually called [illegible] [22].

One of the most basic inference services provided by DL systems is computing the subsumption hierarchy. For concept descriptions C, D defined in a TBox $\mathcal{T}$, C [illegible] D [illegible] $\mathcal{T}$ *($C \sqsubseteq_{\mathcal{T}} D$)* iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model of $\mathcal{T}$. C [illegible] D [illegible] $\mathcal{T}$ *($C \equiv_{\mathcal{T}} D$)* iff $C \sqsubseteq_{\mathcal{T}} D$ and $D \sqsubseteq_{\mathcal{T}} C$. Explicit reference to the empty TBox may be omitted: if $\mathcal{T} = \emptyset$, write $C \sqsubseteq D$ instead of $C \sqsubseteq_{\mathcal{T}} D$, and analogously for equivalence.

Greatest-fixpoint semantics: for (non-general) TBoxes, we additionally introduce greatest-fixpoint semantics. We begin by interpreting only primitive concepts and roles occurring: for every TBox $\mathcal{T}$, a [illegible] *$(\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$* of $\mathcal{T}$ interprets all primitive concepts $P \in \mathsf{N}_{\text{prim}}$ by subsets of $\Delta^{\mathcal{J}}$ and all roles $r \in \mathsf{N}_{\text{role}}$ by binary relations on $\Delta^{\mathcal{J}}$. An Interpretation $\mathcal{I} := (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is [illegible] $\mathcal{J}$ iff $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$ and $\cdot^{\mathcal{J}}$ and $\cdot^{\mathcal{I}}$ coincide on N_{role} and N_{prim}. The set of all interpretations based on $\mathcal{J}$ is denoted by $\text{Int}(\mathcal{J})$. On $\text{Int}(\mathcal{J})$, a binary relation $\preceq_{\mathcal{J}}$ is defined for all $\mathcal{I}_1, \mathcal{I}_2 \in \text{Int}(\mathcal{J})$ by $\mathcal{I}_1 \preceq_{\mathcal{J}} \mathcal{I}_2$ iff $A^{\mathcal{I}_1} \subseteq A^{\mathcal{I}_2}$ for all $A \in \mathsf{N}_{\text{def}}^{\mathcal{T}}$.

The pair $(\text{Int}(\mathcal{J}), \preceq_{\mathcal{J}})$ is a complete lattice, so that every subset of $\text{Int}(\mathcal{J})$ has a least upper bound (lub) and a greatest lower bound (glb) w.r.t. $\preceq_{\mathcal{J}}$. Hence, by

Tarski's fixpoint theorem [23], every monotonic function on $\mathrm{Int}(\mathcal{J})$ has a fixpoint. In particular, this applies to the function $O_{\mathcal{T},\mathcal{J}}$ defined by $O_{\mathcal{T},\mathcal{J}}\colon \mathrm{Int}(\mathcal{J}) \to \mathrm{Int}(\mathcal{J})$ with $\mathcal{I}_1 \mapsto \mathcal{I}_2$ iff $A^{\mathcal{I}_2} = C^{\mathcal{I}_1}$ for all $A \equiv C \in \mathcal{T}$.

As shown in [21], $O_{\mathcal{T},\mathcal{J}}$ is in fact a fixpoint operator on $\mathrm{Int}(\mathcal{J})$. Moreover, it holds that $\mathcal{I}$ is a fixpoint of $O_{\mathcal{T},\mathcal{J}}$ iff $\mathcal{I}$ is a model of $\mathcal{T}$. As a consequence, an interpretation $\mathcal{I}$ is called a [illegible] of $\mathcal{T}$ iff there is a primitive interpretation $\mathcal{J}$ such that $\mathcal{I} \in \mathrm{Int}(\mathcal{J})$ is the greatest fixpoint of $O_{\mathcal{T},\mathcal{J}}$.

As $(\mathrm{Int}(\mathcal{J}), \preceq_{\mathcal{J}})$ is a complete lattice, the gfp-model is uniquely determined for a given TBox $\mathcal{T}$ and a primitive interpretation $\mathcal{J}$. We may thus refer to [illegible] gfp-model $\mathrm{gfp}(\mathcal{T}, \mathcal{J})$ for any given $\mathcal{T}$ and $\mathcal{J}$. With this preparation, we define gfp-subsumption by: for concept names A, B defined in $\mathcal{T}$, A [illegible] B [illegible] *($A \sqsubseteq_{\mathrm{gfp},\mathcal{T}} B$)* iff $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ for all gfp-models $\mathcal{I}$ of $\mathcal{T}$.

Note that descriptive semantics considers a superset of the set of gfp-models, implying that descriptive subsumption entails gfp-subsumption. Hence, all subsumption relations w.r.t. $\sqsubseteq_{\mathcal{T}}$ also hold w.r.t. $\sqsubseteq_{\mathrm{gfp},\mathcal{T}}$. Moreover, both semantics coincide on acyclic TBoxes. For $\mathcal{EL}$, our DL of interest, least-fixpoint semantics is inappropriate w.r.t. cyclic TBoxes [21] and hence is not considered.

See [20] for details of how gfp-models can actually be computed.

Deciding subsumption w.r.t. cyclic $\mathcal{EL}$-TBoxes with gfp-semantics: a decision procedure for the subsumption problem w.r.t. cyclic $\mathcal{EL}$-TBoxes with descriptive semantics has been presented in [21]. We repeat the notions central to this procedure in so far as they are required for our matching algorithms w.r.t. cyclic and hybrid $\mathcal{EL}$-TBoxes.

An $\mathcal{EL}$-TBox $\mathcal{T}$ is [illegible] iff $A \equiv D \in \mathcal{T}$ implies that D is of the form $P_1 \sqcap \cdots \sqcap P_m \sqcap \exists r_1.B_1 \sqcap \ldots \exists r_\ell.B_\ell$, where for $m, \ell \geq 0$, $P_1, \ldots, P_m \in \mathsf{N}_{\mathrm{prim}}$ and $B_1, \ldots, B_\ell \in \mathsf{N}_{\mathrm{def}}$. If $m = \ell = 0$ then $D = \top$. The subsumption algorithm in [21] represents normalized $\mathcal{EL}$-TBoxes by means of [illegible]. Given a normalized $\mathcal{EL}$ TBox $\mathcal{T}$, the $\mathcal{EL}$-description graph $\mathcal{G}_{\mathcal{T}} = (\mathsf{N}^{\mathcal{T}}_{\mathrm{def}}, E_{\mathcal{T}}, L_{\mathcal{T}})$ of $\mathcal{T}$ is defined as follows:

- the nodes of $\mathcal{G}_{\mathcal{T}}$ are the defined concepts of $\mathcal{T}$;
- if A is defined in $\mathcal{T}$ and $A \equiv P_1 \sqcap \cdots \sqcap P_m \sqcap \exists r_1.B_1 \sqcap \cdots \sqcap \exists r_\ell.B_\ell$ is its definition then $L_{\mathcal{T}}(A) := \{P_1, \ldots, P_m\}$, and A is the source of the edges $(A, r_1, B_1), \ldots, (A, r_\ell, B_\ell) \in E_{\mathcal{T}}$.

Any primitive interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ can be represented by an $\mathcal{EL}$-description graph as well, see [20] for details.

In preparation for the characterization of subsumption we need to introduce simulation relations on description graphs. Given two $\mathcal{EL}$-description graphs $\mathcal{G}_i = (V_i, E_i, L_i)$, $i = 1, 2$, the binary relation $Z \subseteq V_1 \times V_2$ is a [illegible] from $\mathcal{G}_1$ to $\mathcal{G}_2$ ($Z\colon \mathcal{G}_1 \rightleftarrows \mathcal{G}_2$) iff (S1) $(v_1, v_2) \in Z$ implies $L_1(v_1) \subseteq L_2(v_2)$; and (S2) if $(v_1, v_2) \in Z$ and $(v_1, r, v_1') \in E_1$ then there exists a node $v_2' \in V_2$ such that $(v_1', v_2') \in Z$ and $(v_2, r, v_2') \in E_2$.

It has been shown in [21] that simulation relations are closed under concatenation. Moreover, one of the main results in [21] is a characterization of gfp-subsumption w.r.t. cyclic $\mathcal{EL}$-TBoxes by simulation relations over description graphs. The following results provide the relevant characterizations.

Theorem 1. [illegible] $\mathcal{T}$ [illegible] $\mathcal{EL}$ [illegible] A, B [illegible] $\mathcal{T}$ [illegible] $A \sqsubseteq_{\text{gfp},\mathcal{T}} B$ [illegible] $Z\colon \mathcal{G}_\mathcal{T} \underset{\sim}{\rightarrow} \mathcal{G}_\mathcal{T}$ [illegible] $(B, A) \in Z$

Since the description graph of a TBox is of polynomial size in the size of the TBox and since the existence of simulation relations with the required properties can be tested in polynomial time, subsumption w.r.t. cyclic $\mathcal{EL}$-TBoxes with gfp-semantics is decidable in polynomial time. [21].

The least-common subsumer w.r.t. cyclic $\mathcal{EL}$-TBoxes: a main preparatory step towards matching w.r.t. cyclic $\mathcal{EL}$-TBoxes is to introduce the lcs for cyclic $\mathcal{EL}$-TBoxes. The relevant definitions are due to [18].

Let $\mathcal{T}_1$ be a cyclic $\mathcal{EL}$-TBox and let $A, B \in \mathsf{N}_{\text{def}}^{\mathcal{T}_1}$. Let $\mathcal{T}_2$ be a conservative extension of $\mathcal{T}_1$ with $E \in \mathsf{N}_{\text{def}}^{\mathcal{T}_2} \setminus \mathsf{N}_{\text{def}}^{\mathcal{T}_1}$. Then E is the [illegible] A [illegible] B [illegible] $\mathcal{T}_1$ [illegible] ([illegible]) iff the following conditions hold:

1. $A \sqsubseteq_{\text{gfp},\mathcal{T}_2} E$ and $B \sqsubseteq_{\text{gfp},\mathcal{T}_2} E$;
2. if $\mathcal{T}_3$ is a conservative extension of $\mathcal{T}_2$ and F a defined concept in $\mathcal{T}_3$ such that $A \sqsubseteq_{\text{gfp},\mathcal{T}_3} F$ and $B \sqsubseteq_{\text{gfp},\mathcal{T}_3} F$ then $E \sqsubseteq_{\text{gfp},\mathcal{T}_3} F$.

In order to be able to actually compute the lcs, we need to compute the product of description graphs. Let $\mathcal{G}_i := (V_i, E_i, L_i)$, $i = 1, 2$ be two description graphs. Their [illegible] is the description graph $\mathcal{G}_1 \times \mathcal{G}_2 := (V, E, L)$ with $V := V_1 \times V_2$; $E := \{((v_1, v_2), r, (v_1', v_2')) \mid \forall i \in \{1, 2\}\colon (v_i, r, v_i') \in E_i\}$; and $L(v_1, v_2) := L_1(v_1) \cap L_2(v_2)$. For a description graph $\mathcal{G} = (V, E, L)$, the n-ary graph product is inductively defined in the obvious way, i.e., $\mathcal{G}^1 := \mathcal{G}$ and $\mathcal{G}^{n+1} := \mathcal{G}^n \times \mathcal{G}$.

In order to transform product graphs back to TBoxes, we define TBoxes induced by description graphs. Let $\mathcal{G} := (V, E, L)$ be a description graph. Then the [illegible] $\mathcal{G}$ is defined by

$$\mathsf{tbox}(\mathcal{G}) := \{A \equiv \bigsqcap_{P \in L(A)} P \sqcap \bigsqcap_{(A,r,B) \in E} \exists r.B \mid A \in V\}.$$

Two of the main results from [18] prove that the gfp-lcs w.r.t. cyclic $\mathcal{EL}$-TBoxes always exists and can in fact be computed by means of the graph product: for concept names A, B defined in $\mathcal{T}$, the concept (A, B) defined in $\mathcal{T} \cup \mathsf{tbox}(\mathcal{G}_\mathcal{T} \times \mathcal{G}_\mathcal{T})$ is the gfp-lcs of A and B w.r.t. $\mathcal{T}$. Hence, the gfp-lcs can be computed in polynomial time in the binary case can and in exponential time in the general case. We are now prepared to introduce hybrid TBox, our main TBox formalism of interest.

2.1 Hybrid TBoxes

Definition 1. ([illegible]) [illegible] $\mathcal{EL}$ [illegible] $\mathcal{F}$ [illegible] N_{prim} [illegible] N_{role} [illegible] $\mathcal{EL}$ [illegible] $\mathcal{T}$ [illegible] N_{def}, N_{prim} [illegible] N_{role} [illegible] $(\mathcal{F}, \mathcal{T})$ [illegible] hybrid $\mathcal{EL}$-TBox

In order to simplify the presentation of our subsumption algorithm, we introduce a normal form for hybrid $\mathcal{EL}$-TBoxes. Analogous to the case of cyclic TBoxes, we normalize hybrid TBoxes in order to simplify the presentation of our proofs. See [20] for an example of what an actual hybrid $\mathcal{EL}$-TBox looks like and for details about normalization. The semantics of hybrid TBoxes can now be defined as follows.

Let $(\mathcal{F}, \mathcal{T})$ be a hybrid TBox over N_{prim}, N_{role}, and N_{def}. A primitive interpretation $\mathcal{J}$ is a [illegible] $\mathcal{F}$ *(*$\mathcal{J} \models \mathcal{F}$*)* iff $C^{\mathcal{J}} \subseteq D^{\mathcal{J}}$ for all GCIs $C \sqsubseteq D$ in $\mathcal{F}$. A model $\mathcal{I} \in \text{Int}(\mathcal{J})$ is a [illegible] of $(\mathcal{F}, \mathcal{T})$ iff $\mathcal{J} \models \mathcal{F}$ and $\mathcal{I}$ is a gfp-model of $\mathcal{T}$.

Note that $\mathcal{F}$ ("foundation") is interpreted with descriptive semantics while $\mathcal{T}$ ("terminology") is interpreted with gfp-semantics. Note also that every gfp-model of $(\mathcal{F}, \mathcal{T})$ can be expressed as the greatest fixpoint $\text{gfp}(\mathcal{T}, \mathcal{J})$ for some primitive interpretation $\mathcal{J}$ with $\mathcal{J} \models \mathcal{F}$.

In order to complete the semantics of hybrid TBoxes, we still have to introduce an appropriate notion of subsumption: Let A, B be defined concepts in $\mathcal{T}$. Then A [illegible] B [illegible] $(\mathcal{F}, \mathcal{T})$ *(*$A \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}} B$*)* iff $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ for all gfp-models $\mathcal{I}$ of $(\mathcal{F}, \mathcal{T})$.

Hybrid TBoxes generalize cyclic TBoxes with gfp-semantics in the sense that every cyclic $\mathcal{EL}$-TBox $\mathcal{T}$ can be viewed as a hybrid TBox with an empty foundation. Thus, gfp-subsumption w.r.t. $\mathcal{T}$ coincides with subsumption w.r.t. the hybrid TBox $(\emptyset, \mathcal{T})$. Also note that every general TBox $\mathcal{T}'$ can be seen as a hybrid TBox $(\mathcal{T}', \emptyset)$. In this case, a descriptive subsumption $P \sqsubseteq_{\mathcal{T}'} Q$ holds iff A_P is subsumed by A_Q w.r.t. the normalized instance of $(\mathcal{T}', \emptyset)$.

Deciding subsumption w.r.t. hybrid $\mathcal{EL}$-TBoxes: in order to decide subsumption of concepts defined in a $\mathcal{EL}$-hybrid TBox, an equivalence preserving reduction from hybrid to cyclic $\mathcal{EL}$-TBoxes with gfp-semantics has been proposed in [19]. After the reduction, subsumption can be decided as described above.

The idea underlying the reduction is to use the [illegible] subsumption relations induced by the GCIs in $\mathcal{F}$ to extend the definitions in $\mathcal{T}$ accordingly. To this end, we view the union of $\mathcal{F}$ and $\mathcal{T}$ as a general TBox and ask for all descriptive implications in $\mathcal{T}$ directly involving names from $\mathcal{F}$. These implications are then added to the definitions in $\mathcal{T}$. This notion is formalized as follows: for a given normalized hybrid $\mathcal{EL}$-TBox $(\mathcal{F}, \mathcal{T})$, the $\mathcal{F}$ [illegible] $f(\mathcal{T})$ extends the definitions in $\mathcal{T}$ to $f(\mathcal{T}) := \{A \equiv C \sqcap f(A) \mid A \equiv C \in \mathcal{T}\}$, where for every $A \in \mathsf{N}_{\text{def}}^{\mathcal{T}}$, the concept description $f(A)$ is defined as follows.

$$f(A) := \bigsqcap_{P \in \{P' \in \mathsf{N}_{\text{prim}}^{\mathcal{F}} \mid A \sqsubseteq_{\mathcal{F} \cup \mathcal{T}} P'\}} P \sqcap \bigsqcap_{r \in \mathsf{N}_{\text{role}}^{\mathcal{T}}} \bigsqcap_{Q \in \{Q' \in \mathsf{N}_{\text{prim}}^{\mathcal{F}} \mid A \sqsubseteq_{\mathcal{F} \cup \mathcal{T}} \exists r.Q'\}} \exists r.A_Q \ .$$

Note that $f(\mathcal{T})$ is still a normalized $\mathcal{EL}$-TBox. To preserve normalization, $f(A)$ adds $\exists r.A_Q$ instead of $\exists r.Q$ whenever A implies $\exists r.Q$. It has been shown in [19] that the above reduction yields a cyclic TBox equivalent to the original hybrid one in the following sense.

Theorem 2. [illegible] $(\mathcal{F}, \mathcal{T})$ [illegible] $\mathcal{EL}$ [illegible] $A, B \in \mathsf{N}_{\text{def}}^{\mathcal{T}}$ [illegible] $A \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}} B$ *iff* $A \sqsubseteq_{\text{gfp},f(\mathcal{T})} B$

It has been shown in [19] that subsumption w.r.t. hybrid $\mathcal{EL}$-TBoxes can be decided in polynomial time in the size of the hybrid TBox.

In the following sections, we introduce matching problems w.r.t. cyclic and hybrid TBoxes and present appropriate matching algorithms for both cases.

3 Matching w.r.t. Cyclic $\mathcal{EL}$-TBoxes

Our first step towards defining matching w.r.t. cyclic TBoxes is to extend concept descriptions to concept patterns by admitting concept variables.

Denote by N_{var} a finite set of [illegible] pairwise disjoint to N_{con} and N_{role}. The set of [illegible] over N_{con}, N_{role}, and N_{var} is inductively defined as follows: every $\mathcal{EL}$-concept description over N_{con} and N_{role} is a concept pattern; every variable $X \in \mathsf{N}_{\text{var}}$ is a concept pattern; and if $r \in \mathsf{N}_{\text{role}}$ and D_1, D_2 are concept patterns then so are $D_1 \sqcap D_2$ and $\exists r.D_1$.

Trivially, every concept description is a concept pattern. The following definition similarly extends cyclic TBoxes to pattern TBoxes in which the right-hand side of a definition may be a concept pattern.

Definition 2. ([illegible]) [illegible] $\mathcal{EL}$-pattern TBox $\mathcal{T}$ [illegible] $A \equiv C$, [illegible] $A \in \mathsf{N}_{\text{def}}$ [illegible] C [illegible] N_{prim}, N_{def}, N_{role}, [illegible] N_{var}. A [illegible] defined in $\mathcal{T}$ [illegible] $\mathcal{T}$ [illegible] $\mathsf{N}_{\text{var}}^{\mathcal{T}}$ [illegible] $\mathcal{T}$.

Note that variables do not occur on left-hand sides of definitions. Denote by $\mathsf{N}_{\text{var}}^{\mathcal{T}}(A)$ the set of variables in $\mathcal{T}$ 'reachable' from A. Matching problems over cyclic TBoxes can now be defined as follows.

Definition 3. ([illegible]) [illegible] $\mathcal{T}$ [illegible] $\mathcal{EL}$ [illegible] $A, B \in \mathsf{N}_{\text{def}}^{\mathcal{T}}$ [illegible] $\mathsf{N}_{\text{var}}^{\mathcal{T}}(A) = \emptyset$ [illegible] $A \equiv^{?}_{\text{gfp},\mathcal{T}} B$ [illegible] $\mathcal{EL}$-matching problem modulo equivalence w.r.t. $\mathcal{T}$ with gfp-semantics

Throughout this section, we shall refer to '$\mathcal{EL}$-matching problem modulo equivalence with gfp-semantics' by '$\mathcal{EL}$-matching problem'. In order to define solutions to matching problems appropriately, some preparation is necessary. The following definition introduces conservative extensions for pattern TBoxes.

Definition 4. ([illegible]) [illegible] $\mathcal{T}_1$ [illegible] $\mathcal{EL}$ [illegible] N_{prim}, N_{def}, N_{role}, [illegible] N_{var} [illegible] $\mathcal{EL}$ [illegible] $\mathcal{T}_2$ [illegible] conservative extension [illegible] $\mathcal{T}_1$ [illegible] $\mathsf{N}_{\text{prim}}^{\mathcal{T}_2} = \mathsf{N}_{\text{prim}}^{\mathcal{T}_1}$, $\mathsf{N}_{\text{role}}^{\mathcal{T}_2} = \mathsf{N}_{\text{role}}^{\mathcal{T}_1}$, $\mathsf{N}_{\text{var}}^{\mathcal{T}_1} \supseteq \mathsf{N}_{\text{var}}^{\mathcal{T}_2}$, [illegible] $\mathcal{T}_1 \subseteq \mathcal{T}_2$

The above definition coincides on ordinary TBoxes with the definition of conservative extensions from [18]. Moreover, since $\mathcal{T}_2$ is a pattern TBox, $\mathsf{N}_{\text{def}}^{\mathcal{T}_1}$ and $\mathsf{N}_{\text{def}}^{\mathcal{T}_2 \setminus \mathcal{T}_1}$ are disjoint. In contrast to concept matching (as, e.g., in [24]), we do not use substitutions to instantiate variables. Instead, we simply extend pattern TBoxes by appropriate definitions for the occurring variables. This leads to the notion of [illegible]

Definition 5. *( )* $\mathcal{T}_1$ $\mathcal{EL}$, N_{prim}, N_{def}, N_{role}, N_{var} $\mathcal{T}_2$, $\mathcal{T}_1$ $X \in \mathsf{N}_{\text{var}}^{\mathcal{T}_1}$, D_X , , N_{prim}, N_{def}, N_{role}, $\mathsf{N}_{\text{var}}^{\mathcal{T}_1}$ $\mathcal{T}_3 := \mathcal{T}_2 \cup \{X \equiv D_X \mid X \in \mathsf{N}_{\text{var}}^{\mathcal{T}_1}\}$ instantiation , $\mathcal{T}_1$

Intuitively, an instantiation turns variables into defined concepts, and thus turns a pattern TBox into an ordinary TBox. Using these notions, it is particularly simple to define solutions to matching problems.

Definition 6. *( )* $A \equiv^?_{\text{gfp},\mathcal{T}} B$ $\mathcal{EL}$, $\mathcal{T}'$, $\mathcal{T}$ $\mathcal{T}'$ matcher of $A \equiv^?_{\text{gfp},\mathcal{T}} B$ ff $A \equiv_{\text{gfp},\mathcal{T}'} B$

Hence, a matcher to $A \equiv^?_{\text{gfp},\mathcal{T}} B$ extends the pattern TBox $\mathcal{T}$ by definitions for all variables reachable from B such that A and B become equivalent. Clearly, we may restrict ourselves to matching problems over names because it holds for every concept description C and every concept pattern D defined over a pattern TBox $\mathcal{T}$ that the matching problem $C \equiv^?_{\text{gfp},\mathcal{T}} D$ can be simulated by $A \equiv^?_{\text{gfp},\mathcal{T}\cup\{A\equiv C, B\equiv D\}} B$ with A, B fresh defined names.

We are now ready to show how to solve matching problems w.r.t. cyclic $\mathcal{EL}$-TBoxes as defined above.

3.1 Solving Matching Problems w.r.t. Cyclic $\mathcal{EL}$-TBoxes

By treating variables as primitive concepts, pattern TBoxes can, syntactically, be regarded as ordinary TBoxes. This allows us to define normalized pattern TBoxes analogously to normalized cyclic TBoxes, and to transform pattern TBoxes into description graphs and vice versa. Similarly, we adopt the notion of a product TBox. For an $\mathcal{EL}$-pattern TBox and $n \in \mathbb{N}$, let $\mathcal{T}^n := \mathsf{tbox}(\mathcal{G}_{\mathcal{T}}^n)$. In order to extend the notion of simulation relations to graphs of pattern TBoxes, variables are simply ignored. We can now define our matching algorithm w.r.t. cyclic $\mathcal{EL}$-TBoxes as follows.

Definition 7. *(match)* $\mathcal{T}$ $\mathcal{EL}$, $A \equiv^?_{\text{gfp},\mathcal{T}} B$ $\mathcal{EL}$, $Z\colon \mathcal{G}_{\mathcal{T}} \mathrel{\overset{\rightarrow}{\sim}} \mathcal{G}_{\mathcal{T}}$ $X \in \mathsf{N}_{\text{var}}^{\mathcal{T}}$, fi

$$Z(X) := \{A' \in \mathsf{N}_{\text{def}} \mid \exists B' \in \mathsf{N}_{\text{def}}\colon (B', A') \in Z \wedge X \in L_{\mathcal{T}}(B')\}$$

$\mathsf{match}(A \equiv^?_{\text{gfp},\mathcal{T}} B)$ fi

Upon input $A \equiv^?_{\text{gfp},\mathcal{T}} B$, our matching algorithm match returns all instantiations $\mathcal{T}_Z$ for which, firstly, Z is a simulation relation on $\mathcal{G}_{\mathcal{T}}$ with $(B, A) \in Z$; and secondly, A subsumes B w.r.t. $\mathcal{T}_Z$ interpreted with gfp-semantics.

For a given Z, $\mathcal{T}_Z$ is defined as an instantiation of a conservative extension of $\mathcal{T}$. We discuss the conservative extension first and the additional definitions for variables afterwards. For every variable $X \in \mathsf{N}_{\text{var}}^{\mathcal{T}}$, $\mathcal{T}$ is extended by the $|Z(X)|$-ary graph product of $\mathcal{T}$. For every X, the set $Z(X)$ contains all 'destination' vertices onto which vertices in $\mathcal{G}_{\mathcal{T}}$ labeled by X are mapped. Hence, whenever

Input: matching problem $\mathcal{P} := A \equiv^{?}_{\mathrm{gfp},\mathcal{T}} B$ with normalized $\mathcal{EL}$-pattern TBox $\mathcal{T}$
Output: set of matchers of $\mathcal{P}$

Return $\{\mathcal{T}_Z \mid Z\colon \mathcal{G}_\mathcal{T} \underset{\sim}{\rightarrow} \mathcal{G}_\mathcal{T} \wedge (B, A) \in Z \wedge A \sqsupseteq_{\mathrm{gfp},\mathcal{T}_Z} B\}$,
where, for every $Z\colon \mathcal{G}_\mathcal{T} \underset{\sim}{\rightarrow} \mathcal{G}_\mathcal{T}$, $\mathcal{T}_Z$ is defined by:

$$
\begin{aligned}
\mathcal{T}_Z := \mathcal{T} \cup & \bigcup_{i \in \{|Z(X)| \mid X \in \mathsf{N}^{\mathcal{T}}_{\mathrm{var}}(B)\} \setminus \{1\}} (\mathcal{T}[X/\top \mid X \in \mathsf{N}^{\mathcal{T}}_{\mathrm{var}}])^i \\
& \cup \{X \equiv (A_1, \ldots, A_n) \mid X \in \mathsf{N}^{\mathcal{T}}_{\mathrm{var}}(B) \\
& \qquad \wedge Z(X) = \{A_1, \ldots, A_n\} \wedge |Z(X)| = n\} \\
& \cup \{X \equiv \top \mid X \notin \mathsf{N}^{\mathcal{T}}_{\mathrm{var}}(B)\}.
\end{aligned}
$$

Fig. 1. The algorithm match for cyclic $\mathcal{EL}$-TBoxes

Z maps vertices labeled by X onto n different vertices then $\mathcal{T}$ is extended by the n-ary graph product of $\mathcal{T}$. More precisely, the graph product is computed after removing variables from $\mathcal{T}$. Note that this removal is only done for convenience to simplify the notation in our proofs and not necessary for correctness or completeness of the algorithm.

As a result, the relevant conservative extension of $\mathcal{T}$ for every X contains a definition of the lcs over all destination vertices of vertices labeled by X: if $Z(X)$ contains n pairwise distinct destination vertices $\{A_1, \ldots, A_n\}$ then the relevant lcs is the vertex $(A_1, \ldots, A_n)$ in the n-ary product of $\mathcal{T}$.

As the second line of the definition of $\mathcal{T}_Z$ shows, X is finally assigned the lcs over all destinations of X: $X \equiv (A_1, \ldots, A_n)$. Note that the condition $|Z(X)| = n$ only ensures pairwise distinctness of the vertices $A_1, \ldots, A_n$. Without this condition, X might be assigned to vertices not existing in the relevant extension. Note also that variables unreachable from B are assigned $\top$.

In order to get an impression how the above matching algorithm works, see our example in [20].

We can show that the above algorithm is sound and complete and that the set of all matchers of a given matching problem can be computed in exponential time, see [20] for details. More precisely, we show that our matching algorithm is [illegible]. Intuitively, this means that the set of matchers computed by the algorithm contains all 'interesting' solutions which contain as much information about the input matching problem as possible; see [20] for details.

In addition to that, we obtain that our matching algorithm for cyclic $\mathcal{EL}$-TBoxes with greatest-fixpoint semantics generalizes the $\mathcal{EL}$-matching algorithm w.r.t. the empty TBox presented in [24]. This immediately implies several complexity lower bounds: Firstly, deciding the solvability of matching problems modulo equivalence w.r.t. cyclic $\mathcal{EL}$-TBoxes is NP-hard. Secondly, the minimal matchers to matching problems w.r.t. cyclic $\mathcal{EL}$-TBoxes can be of exponential size in the input TBox. Moreover, the number of minimal matchers can also be exponential in the input TBox. Any algorithm solving matching problems w.r.t. cyclic $\mathcal{EL}$-TBoxes is therefore necessarily worst-case exponential. In this sense, our algorithm is worst-case optimal. It is open whether deciding the solvability of matching problems modulo equivalence w.r.t. cyclic $\mathcal{EL}$-TBoxes with gfp-semantics is in NP.

4 Matching w.r.t. Hybrid TBoxes

The main ingredient of the matching algorithm presented in the previous section has been the gfp-lcs w.r.t. cyclic $\mathcal{EL}$-TBoxes with gfp-semantics from [18]. Our aim now is to extend the algorithm from cyclic to hybrid TBoxes. We begin by extending the notion of a pattern TBox from Definition 2 to hybrid TBoxes.

Definition 8. *([illegible])* [illegible] $\mathcal{EL}$-pattern TBox $\mathcal{T}$ [illegible] $(\mathcal{F},\mathcal{T})$ [illegible] $\mathcal{EL}$ [illegible] $\mathcal{F}$ [illegible] *fi* [illegible] N_{prim} [illegible] N_{role} [illegible] $\mathcal{EL}$ [illegible] *fi* [illegible] N_{def} N_{prim} [illegible] N_{role}

Hence, hybrid pattern TBoxes extend ordinary pattern TBoxes by adding a 'foundation' general TBox. Conservative extensions and instantiations of hybrid pattern TBoxes are defined analogous to their counterpart cyclic TBoxes, i.e., they affect only $\mathcal{T}$ and leave $\mathcal{F}$ unchanged. We can now immediately extend the notion of matching problems to hybrid pattern TBoxes.

Definition 9. *([illegible])* [illegible] $(\mathcal{F},\mathcal{T})$ [illegible] $\mathcal{EL}$ [illegible] $A, B \in \mathsf{N}_{\text{def}}^{\mathcal{T}}$ [illegible] $\mathsf{N}_{\text{var}}^{\mathcal{T}}(A) = \emptyset$ [illegible] $A \equiv^?_{\text{gfp},\mathcal{F},\mathcal{T}} B$ [illegible] hybrid $\mathcal{EL}$-matching problem modulo equivalence w.r.t. $(\mathcal{F},\mathcal{T})$

Note that, despite the restriction of A to defined concept names from $\mathcal{T}$, concept patterns can also be matched against concept names defined in $\mathcal{F}$. For instance, in order to match a concept pattern B defined in $\mathcal{T}$ against some $P \in \mathsf{N}_{\text{con}}^{\mathcal{T}}$ from $\mathcal{F}$, it suffices to extend $\mathcal{T}$ by a definition of the form $A_P \equiv P$, with A_P a fresh concept name, and solve the matching problem $A_P \equiv^?_{\text{gfp},\mathcal{F},\mathcal{T}} B$. Clearly, one can also define concept patterns using only names from $\mathcal{F}$.

Solutions to hybrid $\mathcal{EL}$-matching problems can now be defined analogous to matchers for matching problems w.r.t. cyclic TBoxes.

Definition 10. *([illegible])* [illegible] $A \equiv^?_{\text{gfp},\mathcal{F},\mathcal{T}} B$ [illegible] $\mathcal{EL}$ [illegible] $(\mathcal{F},\mathcal{T}')$ [illegible] $(\mathcal{F},\mathcal{T})$ [illegible] $(\mathcal{F},\mathcal{T}')$ [illegible] matcher of $A \equiv^?_{\text{gfp},\mathcal{F},\mathcal{T}} B$ [illegible] *ff* $A \equiv_{\text{gfp},\mathcal{F},\mathcal{T}'} B$

In preparation to solving matching problems w.r.t. hybrid TBoxes, we extend the lcs algorithm to hybrid TBoxes in the following section. In Section 4.2, the actual matching algorithm for hybrid TBoxes is presented.

4.1 The Least-Common Subsumer w.r.t. Hybrid $\mathcal{EL}$-TBoxes

Our aim is to extend the lcs w.r.t. cyclic $\mathcal{EL}$-TBoxes to hybrid $\mathcal{EL}$-TBoxes. To this end, we begin by extending the notion of conservative extensions of $\mathcal{EL}$-TBoxes from cyclic to hybrid TBoxes. A hybrid TBox $(\mathcal{F},\mathcal{T}_2)$ is a conservative extension of $(\mathcal{F}, \mathcal{T}_1)$ iff $\mathcal{T}_2$ is a conservative extension of $\mathcal{T}_1$ in the sense of Definition 4. Hence, a conservative extension of $(\mathcal{F}, \mathcal{T})$ is obtained by fixing $\mathcal{F}$ and extending $\mathcal{T}$ in the usual way. We can now define the lcs w.r.t. hybrid TBoxes analogously to the case of cyclic ones.

Definition 11. *(* [illegible] *)* [illegible] $(\mathcal{F}, \mathcal{T}_1)$ [illegible] $A, B \in \mathsf{N}_{\text{def}}^{\mathcal{T}_1}$ [illegible] $(\mathcal{F}, \mathcal{T}_2)$ [illegible] $(\mathcal{F}, \mathcal{T}_1)$ [illegible] $C \in \mathsf{N}_{\text{def}}^{\mathcal{T}_2}$ [illegible] C [illegible] $(\mathcal{F}, \mathcal{T}_2)$ [illegible] hybrid least-common subsumer (lcs) of A, B in $(\mathcal{F}, \mathcal{T}_1)$ [illegible] *ff* [illegible]

$A \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}_2} C$ [illegible] $B \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}_2} C$ [illegible]
[illegible] $(\mathcal{F}, \mathcal{T}_3)$ [illegible] $(\mathcal{F}, \mathcal{T}_2)$ [illegible] $D \in \mathsf{N}_{\text{def}}^{\mathcal{T}_3}$ [illegible]
$A \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}_3} D$ [illegible] $B \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}_3} D$ [illegible] $C \sqsubseteq_{\text{gfp},\mathcal{F},\mathcal{T}_3} D$

In order to compute the lcs w.r.t. hybrid $\mathcal{EL}$-TBoxes, we again utilize the reduction from hybrid to cyclic TBoxes from [19] and the usual gfp-lcs algorithm for cyclic $\mathcal{EL}$-TBoxes from [18]. We show in [20] that the hybrid lcs algorithm thus obtained in fact yields the correct results: (A, B) in $(\mathcal{F}, f(\mathcal{T}) \cup f(\mathcal{T})^2)$ is the hybrid lcs of any concepts A, B defined in a given hybrid TBox $(\mathcal{F}, \mathcal{T})$.

As the lcs of arbitrary arity can be reduced to the binary lcs, the above results immediately carry over to the n-ary lcs. As the reduction from hybrid to cyclic $\mathcal{EL}$-TBoxes can be computed in polynomial time and as the lcs algorithm for cyclic $\mathcal{EL}$-TBoxes with gfp-semantics has already been studied [18], we find that the lcs of concepts defined in a hybrid TBox $(\mathcal{F}, \mathcal{T})$ always exists and (in the binary case) can be computed in polynomial time in the size of $(\mathcal{F}, \mathcal{T})$. Moreover, the lcs of arbitrary arity w.r.t. hybrid $\mathcal{EL}$-TBoxes can be computed in exponential time in the size of the input and is of exponential size in the size of the input in the worst-case. In particular, our lcs algorithm is worst-case optimal.

4.2 Solving Matching Problems w.r.t. Hybrid $\mathcal{EL}$-TBoxes

We are now prepared to introduce our matching algorithm for hybrid TBoxes.

Definition 12. *(*$\mathsf{match}_{\text{hy}}$*)* [illegible] $(\mathcal{F}, \mathcal{T})$ [illegible] $\mathcal{EL}$ [illegible] $A \equiv^?_{\text{gfp},\mathcal{F},\mathcal{T}} B$ [illegible] $\mathcal{EL}$ [illegible] *fi*

$$\mathsf{match}_{\text{hy}}(A \equiv_{\text{gfp},\mathcal{F},\mathcal{T}} B) := \{(\mathcal{F}, (\mathcal{T}' \setminus f(\mathcal{T})) \cup \mathcal{T}) \mid \mathcal{T}' \in \mathsf{match}(A \equiv_{\text{gfp},f(\mathcal{T})} B)\}$$

In the above definition, $f(\mathcal{T})$ denotes the $\mathcal{F}$-completion of $\mathcal{T}$ from Section 2.1 and match the matching algorithm for cyclic $\mathcal{EL}$-TBoxes from Definition 7. Hence, the algorithm $\mathsf{match}_{\text{hy}}$ proceeds in three main steps. Firstly, the input hybrid pattern TBox $(\mathcal{F}, \mathcal{T})$ is translated into an equivalent[1] cyclic pattern TBox $f(\mathcal{T})$. Secondly, for the translated matching problem $A \equiv_{\text{gfp},f(\mathcal{T})} B$, the algorithm match computes all minimal solutions and returns them in the form of instantiations $\mathcal{T}'$ of $f(\mathcal{T})$. Thirdly, the solution is returned as a set of instantiations of [illegible] pattern TBoxes. How exactly these hybrid instantiations are defined deserves a closer look.

---

[1] Treating variables as atomic concepts.

As every instantiation $\mathcal{T}'$ returned by the algorithm match is a conservative extension of $f(\mathcal{T})$ and not $\mathcal{T}$, $\mathcal{T}'$ already completely specifies a solution to the initial hybrid matching problem. Or, in other words, $\mathcal{F}$ becomes redundant. As we are interested in [illegible] instantiations of $(\mathcal{F}, \mathcal{T})$, and not of $(\mathcal{F}, f(\mathcal{T}))$, we modify every $\mathcal{T}'$ by removing $f(\mathcal{T})$ and replacing it by the original TBox $\mathcal{T}$, i.e., compute $(\mathcal{T}' \setminus f(\mathcal{T})) \cup \mathcal{T}$. This modification preserves equivalence as a direct consequence of the correctness of the $\mathcal{F}$-completion shown in [19]. Together with the correctness of our hybrid lcs algorithm, we immediately obtain soundness and completeness of the hybrid matching algorithm.

Corollary 1. [illegible] $(\mathcal{F}, \mathcal{T})$ [illegible] $\mathcal{EL}$ [illegible] $A \equiv_{\mathrm{gfp},\mathcal{F},\mathcal{T}} B$ [illegible] $\mathcal{EL}$ [illegible] $(\mathcal{F}, \mathcal{T})$ [illegible] $\mathsf{match}_{\mathrm{hy}}(A \equiv_{\mathrm{gfp},\mathcal{F},\mathcal{T}} B)$ [illegible] $A \equiv_{\mathrm{gfp},\mathcal{F},\mathcal{T}} B$

The complexity results obtained in the previous section together with the fact that $f(\mathcal{T})$ can be computed in polynomial time in the size of $(\mathcal{F}, \mathcal{T})$ [19] imply the following complexity results: Deciding the solvability of matching problems modulo subsumption w.r.t. hybrid $\mathcal{EL}$-TBoxes is tractable. Deciding the solvability of matching problems modulo equivalence w.r.t. hybrid $\mathcal{EL}$-TBoxes is NP-hard. The solutions to a matching problem w.r.t. hybrid $\mathcal{EL}$-TBoxes can be exponential in number and of exponential size in the input matching problem. They can be computed by a deterministic exponential-time algorithm. The computation algorithm is worst-case optimal. See [20] for details.

It is open whether deciding the solvability of matching problems modulo equivalence w.r.t. hybrid $\mathcal{EL}$-TBoxes is in NP. Note that that additional rewriting might be desirable in order to present the solutions of $\mathsf{match}_{\mathrm{hy}}$ more succinctly: $\mathcal{T}'$ can contain the n-ary product of $f(\mathcal{T})$ which might contain information already implied by $\mathcal{F}$.

5 Conclusion and Outlook

In the present paper, we have proposed the notion of matching problems in cyclic $\mathcal{EL}$-TBoxes with gfp-semantics and have devised a sound and s-complete exponential time algorithm for that case. Using an existing reduction from hybrid $\mathcal{EL}$-TBoxes to cyclic ones, we have shown that the lcs w.r.t. hybrid $\mathcal{EL}$-TBoxes always exists and have devised a sound and complete exponential time algorithm to compute it. Utilizing both the reduction and the result on the hybrid lcs, we could devise a sound and complete exponential time matching algorithm for matching problems w.r.t. hybrid TBoxes. All computation algorithms are worst-case optimal. Optimality of the relevant algorithms for the decision problem, i.e., existence of a matcher, remains an open problem.

Apart from the fact that reasoning over $\mathcal{EL}$-TBoxes has an attractive computational complexity, ontologies based on $\mathcal{EL}$-TBoxes are of some significance to the life sciences. For instance, the widely used medical terminology SNOMED [27] corresponds to an $\mathcal{EL}$-Tbox [28]. Similarly, the Gene Ontology [29] can be represented by an $\mathcal{EL}$-TBox with one transitive role, and large parts of the medical knowledge base Galen [30] can be expressed by a general $\mathcal{EL}$-TBox with transitive roles. Similarly, the widely used International Classification for Nursing Practice (ICNP) [31] corresponds to a general $\mathcal{EL}$-TBox.

Matching in general $\mathcal{EL}$-TBoxes: the apparent popularity of 'common' general $\mathcal{EL}$-TBoxes motivates the question to which extent the above results have any potential to be used for that KR formalism.

It has been shown in [18] that the least-common subsumer w.r.t. cyclic $\mathcal{EL}$-TBoxes with descriptive semantics need not exist[2], a result that carries over to general $\mathcal{EL}$-TBoxes. Moreover, as every lcs can be expressed as a minimal solution to some matching problem, minimal matchers need not always exist likewise.

On the other hand, we have pointed out in Section 2.1 that every general $\mathcal{EL}$-TBox $\mathcal{T}$ can be viewed as a hybrid TBox $(\mathcal{T}, \emptyset)$ with empty terminology. Hence, we can define matching problems in general TBoxes (with descriptive semantics) and use our hybrid matching algorithm to compute a set of solutions S with gfp-semantics. As descriptive subsumption entails gfp-subsumption, every 'descriptive' solution to the matching problem is obtained by a gfp-matching algorithm. All matchers w.r.t. descriptive semantics can thus be computed by first computing S with our hybrid matching algorithm and then removing every matcher from S that is not valid w.r.t. descriptive semantics.

The pure decision problem for general TBoxes might be even more interesting for our hybrid matching algorithm. As pointed out in [17], matching can be utilized as a retrieval mechanism over TBoxes in a straightforward way. The user specifies a concept pattern with the syntactic structure he has in mind. The matching algorithm is then used to retrieve all concepts in the TBox for which a matcher exists. The fact that variables in concept patterns are named, in contrast to, e.g., wildcards ('$*$') known from standard database queries, allows us to search the TBox for concepts with very specific structural properties.

In the application scenario sketched above, two ways of dealing with 'descriptive' results suggest themselves. The first option is to solve the full computation problem in the background and return only those concepts for which the matcher is also valid with descriptive semantics. Queries of the above kind, however, are motivated by structural properties of concepts defined in the TBox. Therefore, a viable second option might be to just present all solutions retrieved with gfp-semantics.

In order to substantiate the claim that the above query mechanism driven by our hybrid matching algorithm is useful for the task of knowledge engineering, we plan to implement our matching algorithm as a plugin to the widely used ontology editor Protégé [32]. One way to achieve this might be to integrate the query functionality into the system Sonic [33], a plug-in specifically designed for the purpose to bring non-standard inferences to users of Protégé.

---

[2] Nevertheless, the existence of the lcs under these circumstances is decidable, see [26].

Acknowledgements

We would like to thank Hongkai Liu for his valuable contribution to the results for matching in cyclic $\mathcal{EL}$-TBoxes.

References

1. Nardi, D., Brachmann, R.J.: An introduction to description logics. In: The Description Logic Handbook: Theory, Implementation, and Applications, pp. 1–40. Cambridge University Press, Cambridge (2003)
2. Rector, A., Nowlan, W., Glowinski, A.: Goals for concept representation in the GALEN project. In: Proc. of SCAMC, Washington, USA, pp. 414–418 (1993)
3. Rector, A.: Medical informatics. In: The Description Logic Handbook: Theory, Implementation, and Applications, pp. 406–426. Cambridge University Press, Cambridge (2003)
4. Horrocks, I., Rector, A.L., Goble, C.A.: A description logic based schema for the classification of medical data. In: Proc. of KRDB 1996 (1996)
5. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Proc. of KR 1998, pp. 636–645. Morgan-Kaufmann Publishers, San Francisco (1998)
6. Haarslev, V., Möller, R.: Racer system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–712. Springer, Heidelberg (2001)
7. Sirin, E., Parsia, B.: Pellet: An OWL DL reasoner. In: Proc. of DL 2004. CEUR-WS (2004) Proceedings (2004), `http://CEUR-WS.org/Vol-104/`
8. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL—a polynomial-time reasoner for life science ontologies. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 287–291. Springer, Heidelberg (2006)
9. Cohen, W.W., Borgida, A., Hirsh, H.: Computing least common subsumers in description logics. In: Proc. of AAAI 1992, pp. 754–760. The MIT Press, USA (1992)
10. Cohen, W.W., Hirsh, H.: The learnability of description logics with equality constraints. Machine Learning 17(2/3), 169–199 (1994) Special Issue for COLT 1992.
11. Frazier, M., Pitt, L.: CLASSIC learning. Machine Learning 25, 151–193 (1996) Was in COLT 1994.
12. Baader, F., Küsters, R.: Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$-concept descriptions. In: Herzog, O. (ed.) KI 1998. LNCS (LNAI), vol. 1504, pp. 129–140. Springer, Heidelberg (1998)
13. McGuinness, D.: Explaining Reasoning in Description Logics. Ph.D. dissertation, Department of Computer Science, Rutgers University, USA (1996)
14. Borgida, A., McGuinness, D.L.: Asking queries about frames. In: Proc. of KR 1996, pp. 340–349. Morgan-Kaufmann Publishers, San Francisco (1996)
15. Baader, F., Küsters, R., Borgida, A., McGuinness, D.: Matching in description logics. Journal of Logic and Computation 9(3), 411–447 (1999)
16. Baader, F., Küsters, R., Molitor, R.: Computing least common subsumers in description logics with existential restrictions. In: Proc. of IJCAI 1999, pp. 96–101. Morgan-Kaufmann Publishers, San Francisco (1999)
17. Brandt, S., Turhan, A.Y.: Using non-standard inferences in description logics — what does it buy me? In: Proc. of KIDLWS 2001. CEUR-WS (September 2001) Proceedings online available from `http://CEUR-WS.org/Vol-44/`

18. Baader, F.: Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In: Proc. of IJCAI 2003, pp. 319–324. Morgan-Kaufmann Publishers, San Francisco (2003)
19. Brandt, S., Model, J.: Subsumption in $\mathcal{EL}$ w.r.t. hybrid TBoxes. In: Furbach, U. (ed.) KI 2005. LNCS (LNAI), vol. 3698, pp. 34–48. Springer, Heidelberg (2005)
20. Brandt, S.: Matching and general concept inclusion axioms. Technical report (2007), See `http://personalpages.manchester.ac.uk/staff/Sebastian-philipp.Brandt/tr0707.pdf`
21. Baader, F.: Terminological cycles in a description logic with existential restrictions. In: Proc. of IJCAI 2003, pp. 325–330. Morgan-Kaufmann Publishers, San Francisco (2003)
22. Nebel, B.: Terminological cycles: Semantics and computational properties. In: Proc. of Principles of Semantic Networks, pp. 331–361. Morgan Kaufmann, San Francisco (1991)
23. Tarski, A.: A lattice-theoretic fixpoint theorem and its applications. Pacific Journal of Mathematics 5(2), 285–309 (1955)
24. Baader, F., Küsters, R.: Matching in description logics with existential restrictions. In: Proc. of KR 2000, pp. 261–272. Morgan-Kaufmann Publishers, San Francisco (2000)
25. Küsters, R.: Non-Standard Inferences in Description Logics. In: Küsters, R. (ed.) Non-Standard Inferences in Description Logics. LNCS (LNAI), vol. 2100, Springer, Heidelberg (2001)
26. Baader, F.: A graph-theoretic generalization of the least common subsumer and the most specific concept in the description logic $\mathcal{EL}$. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 177–188. Springer, Heidelberg (2004)
27. Côté, R., Rothwell, D., Palotay, J., Beckett, R., Brochu, L.: The systematized nomenclature of human and veterinary medicine. Technical report, SNOMED International, Northfield, IL (1993)
28. Spackman, K.: Normal forms for description logic expressions of clinical concepts in SNOMED RT. Journal of the American Medical Informatics Association (Symposium Supplement) (2001)
29. Consortium, T.G.O.: Gene Ontology: Tool for the unification of biology. Nature Genetics 25, 25–29 (2000)
30. Rector, A., Bechhofer, S., Goble, C.A., Horrocks, I., Nowlan, W.A., Solomon, W.D.: The GRAIL concept modelling language for medical terminology. Artificial Intelligence in Medicine 9, 139–171 (1997)
31. International council of Nurses, Geneva, CH. See `http://www.icn.ch/icnp.html`
32. Horridge, M., Tsarkov, D., Redmond, T.: Supporting early adoption of OWL 1.1 with Protege-OWL and FaCT++. In: Proc. of OWL-ED 2006 (2006)
33. Turhan, A.Y.: Pushing the SONIC border—SONIC 1.0. In: Proc. of FTP 2005. Technical Report, University of Koblenz (2005)

Verifying Cryptographic Protocols with Subterms Constraints

Yannick Chevalier[1], Denis Lugiez[2], and Michaël Rusinowitch[3,⋆]

[1] IRIT, Team LiLac, Université de Toulouse, France
ychevali@irit.fr
[2] LIF, CNRS, Aix-Marseille Université, France
lugiez@lif.univ-mrs.fr
[3] LORIA-INRIA-Lorraine, France
rusi@loria.fr

Abstract. Many analysis techniques and decidability results have been obtained for cryptographic protocols. However all of them consider protocols with limited procedures for the processing of messages by agents or intruders: Information expected in a protocol message has to be located at a fixed position. However this is too restrictive for instance to model web-service protocols where messages are XML semi-structured documents and where significant information (name, signature, ...) has to be extracted from some nodes occurring at flexible positions. Therefore we extend the standard Dolev Yao intruder model by a subterm predicate that allows one to express a larger class of protocols that employs data extraction by subterm matching. This also allows one to detect so-called *rewriting attacks* that are specific to web-services. In particular we show that protocol insecurity is decidable with complexity NP for finite sessions in this new model. The proof is not a consequence of the standard finite sessions case; on the contrary, it provides also a new short proof for this case.

1 Introduction

Cryptographic protocols have been applied to securing communications over an insecure network for many years. However, the underlying difficulties in properly designing cryptographic protocols are reflected by repeated discovery of [illegible] bugs in these protocols. As an attempt to solve the problem, there has been a sustained effort to devise formal methods for specifying and verifying the security goals of protocols. Various symbolic approaches have been proposed to represent protocols and reason about them, and to attempt to verify security properties such as confidentiality and authenticity, or to discover bugs. Such approaches include process algebra, model-checking, modal logics, equational reasoning, constraint solving and resolution theorem-proving (e.g., [20,1,5,2]).

⋆ This work has been supported by ARA SSIA Cops.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 181–195, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Although some of these approaches have been successful in detecting security flaws or showing their absence in many protocols, their scope remains limited. Typically in all this work the processing of messages by agents or intruders is very limited: Information expected in a protocol message has to be located at a fixed position. However this is too restrictive for instance to model web-service protocols where messages are XML semi-structured documents and where significant information (name, signature, . . .) has to be extracted from some nodes occurring at flexible positions. Also protocols for searching databases (such as the LDAP protocol for internet directories) cannot be modelled properly in the previous approaches.[1]

As a first step to relax these restrictions, we consider an extension of the standard Dolev Yao intruder model by a subterm predicate that allows one to express protocols applying subterm matching for extracting data in a received message. This extended model also allows one to detect some [illegible] that are specific to web services. In particular we show that protocol insecurity (i.e. whether the protocol preserves the confidentiality of some data) is decidable with complexity NP for fixed number of protocol sessions. The proof is short and does not follow from previous ones. As a matter of fact our result gives also as a by-product a new short proof that insecurity is in NP for the standard Dolev Yao case with non-atomic keys.

[illegible] Several decidability and complexity results have been obtained for cryptographic protocols [1,16,5,18]. These results have been extended to handle algebraic properties of basic cryptographic primitives [3,15,10]. In particular we have shown in [6] how to handle an associative-commutative message constructor. The result in [6] relies on unification and combination techniques [7] and allows the modelling of an immediate subterm relation: a is an immediate subterm of $a.f(b)$ where . is associative-commutative, but b is not. Here we consider an unrestricted subterm relation which allows to detect attacks on XML protocols that are out of the scope of [6] as shown in Section 2. In [9] the authors consider ordering constraints on atomic keys, but do not give precise complexity analysis. The results of the present paper are in some respects extensions of [9] and rely on a different proof technique.

The previous works on web service security protocols (e.g. [13,8]) have rather focused on encoding XML messages and the design of attack preserving abstraction. To our knowledge no specific decidability results have been provided yet in this context.

[illegible] In Section 2 we give an example of an attack on a web service that motivates our result. In Section 3 we introduce the needed basic notions on terms and subterm constraints. Then we present in Section 4 the protocol model we consider. In Section 5 we introduce the constraints we have to solve to verify security properties of a protocol in the given model. Then Sections 6, 8, 9, 10, 11 correspond to the different steps of the algorithm for solving these constraints. We conclude in Section 12.

---

[1] Since honest agents in usual models [4] can only search a fixed part of the database.

2 Motivating Example

Web services promise to be a standard technology for Internet and enterprise networks. They require the ability to securely transmit messages in XML syntax using the SOAP protocol. Messages that travel over the networks can be observed and modified by intruders. Hence the protocol was extended by W3C for allowing one to sign and encrypt some parts of the contents. They can be subject to the same attacks as classical cryptographic protocols, but the XML syntax and the specific way messages are processed (e.g. not examining the full content) also gives the opportunity to mount a new class of attacks as shown in [8] and illustrated by the following example. These attacks are sometimes called [illegible] since the intruder modifies some message contents for his purpose.

[illegible] Let us consider an abstraction of the security protocol $\mathcal{P}$ that supports a travel agency service and composes two roles: a client A and a server B. Each role consists of send (!) and receive (?) actions combined with some pattern-matching process.

$$
\begin{array}{lll}
\textit{Client } A: & !x_1^A = \langle \mathrm{se}(h(\textit{order}(x,y,z)), K_{A,B}), \textit{order}(x,y,z)\rangle & \\
& ?x_2^A = \langle u_h, u_b\rangle \ \text{WHERE} & \mathrm{se}(h(\textit{order}(x,y,z)), K_{A,B}) \prec u_h, \\
& & \mathrm{se}(\textit{accesscode}(z), K_{x,y}) \prec u_b \\
\textit{Server } B: & ?x_1^B = \langle v_h, v_b\rangle \ \text{WHERE} & \mathrm{se}(h(\textit{order}(x,y,z)), K_{A,B}) \prec v_h, \\
& & \textit{order}(x,y,z) \prec v_b \\
& !x_2^B = \langle \mathrm{se}(h(\textit{order}(x',y',z')), K_{A,B}), \mathrm{se}(\textit{accesscode}(z'), K_{x',y'})\rangle & \\
& \qquad \text{WHERE } \textit{order}(x',y',z') \prec v_b &
\end{array}
$$

where $\prec$ denotes the subtree relation, h denotes a hashing function, [illegible] (x, y, z) denotes the request for trip z for beneficiary x, with y the account to charge, [illegible] is the code requested to get the ticket from an automaton, $\mathrm{se}(u, v)$ denotes the encryption of u using key v, $K_{x,y}$ denotes a private key shared by x and y. The symbol $\langle \_, \_ \rangle$ is a free binary symbol that denotes pairing. The intruder deductive power is given by the classical Dolev-Yao rules (see [12]) extended by a rule that allows to select any argument of a free symbol. For some realistic implementations of the service, the following attack (described in [17]) will be possible:

$$
\begin{array}{rl}
A \to I(B): & \langle \mathrm{se}(h(\textit{order}(A,A,\textit{Erevan})), K_{A,B}), \textit{order}(A,A,\textit{Erevan})\rangle \\
I(A) \to B: & \langle \langle \mathrm{se}(h(\textit{order}(A,A,\textit{Erevan})), K_{A,B}), \textit{order}(C,A,\textit{Hawai})\rangle, \\
& \textit{Bogus}(\textit{order}(A,A,\textit{Erevan}))\rangle \\
B \to I(A): & \langle \mathrm{se}(h(\textit{order}(A,A,\textit{Erevan})), K_{A,B}), \mathrm{se}(\textit{accesscode}(\textit{Hawaii}), K_{C,B})\rangle
\end{array}
$$

where $I(A)$ (resp. $I(B)$) denotes a malicious agent I masquerading as the honest participant A (resp. B), and the secret key SK_C is known by I ($C = I$ or C has been compromised). The [illegible] symbol mimicks an encapsulation `<Bogus>...</>` with a Bogus header. The `<Bogus>` element and its content are ignored by the receiver since the header is unknown. However the signature is still acceptable since the element that is linked to the signature (via its URI) remains in the message.

[illegible] The subterm relation can be quite useful to model database search as in the following simple protocol (inspired by the Lightweight Directory Access Protocol) for retrieving public keys of users in some directory D owned by some server B. We assume that the term D represents a tree structured directory and that the records are leaves of type $r(name, pkey)$. K is a term representing the knowledge set of A.

$$\begin{array}{l} \textit{Client } A: \ !x_1^A \ \text{WHERE } x_1^A \prec K \quad ?x_2^A \quad \text{WHERE } x_2^A \ = \mathrm{se}(Y', K_{A,B}) \\ \textit{Server } B: \ ?x_1^B \ \ !x_2^B = \mathrm{se}(Y, K_{A,B}) \quad \text{WHERE } (x_1^B, Y) \prec D \end{array}$$

3 Terms, Unification and Subterm Relation

We refer to [11] for all notions on terms, substitutions,... and recall only the main ones. Terms are constructed from a finite set of function symbols $\mathcal{F}$, a denumerable set of variables $\mathcal{X}$ and $T_{\mathcal{F}}(\mathcal{X})$ denotes the set of terms. The set of ground terms is denoted by $T_{\mathcal{F}}$. Constants are functions of arity 0. For finite sets of terms, we abbreviate $E \cup F$ by E, F, the union $E \cup \{t\}$ by E, t and $E \setminus \{t\}$ by $E \setminus t$. The set of subterms of a term t, denoted by $\mathrm{Sub}(t)$, is the smallest set such that $\mathrm{Sub}(t) = \{t\}$ if $t \in \mathcal{X}$ or t is a constant, and $\mathrm{Sub}(t) = \{t\} \cup \bigcup_{i=1}^{i=n} \mathrm{Sub}(t_i)$ if $t = f(t_1, \ldots, t_n)$. A subterm s of t is strict if $s \neq t$. The size of a term t, denoted by [illegible], is the cardinality of $\mathrm{Sub}(t)$. A position is a sequence of integers and the subterm of t at position p, denoted by $t_{|p}$, is defined by $t_{|p} = t$ if $p = \epsilon$, $t_{|p.i} = t_i$ if $t_{|p} = f(t_1, \ldots, t_n)$, $i \leq n$. The set of positions in a term t is denoted by $\mathrm{Pos}(t)$. $\mathrm{Var}(t)$ denotes the set of variables occurring in the term t. A substitution σ is defined by $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ where $\{x_1, \ldots, x_n\}$ is the domain of σ, denoted by $\mathrm{Dom}(\sigma)$. The substitution is ground if $\mathrm{Var}(t_i) = \emptyset$ for $i = 1, \ldots, n$.

A unification system $\mathcal{S}$ is a finite set of equations $(u_i \stackrel{?}{=} v_i)_{i \in \{1,..,n\}}$ where $u_i, v_i \in T_{\mathcal{F}}(\mathcal{X})$ for $i = 1, \ldots, n$. A ground substitution σ is a solution of $\mathcal{S}$, denoted by $\sigma \models \mathcal{S}$, iff for $i = 1, \ldots, n$ we have $u_i\sigma = v_i\sigma$.

A subterm constraint is an expression $s \prec t$ where $\prec$ denotes the strict subterm relation on terms. The notation $s \preceq t$ stands for $s \prec t \vee s = t$. Let σ be a substitution, we write $\sigma \models s \prec t$ iff $s\sigma \in \mathrm{Sub}(t\sigma)$ and $s\sigma \neq t\sigma$ and we say that σ is a solution of or satisfies $s \prec t$. A subterm constraint system $\mathcal{T}$ is a finite set of subterm constraints. A substitution is a solution of a subterm constraint system iff it is a solution of each constraint of the system.

There exist polynomial time algorithms for solving unification systems and a NPTIME procedure to solve subterm constraint systems [19].

4 Protocol Model

4.1 Dolev Yao Model

A protocol is defined by a finite set of [illegible] (denoted by A, B,...) acting in this protocol. Each role is specified by a finite sequence of receive/send actions to perform according to its current state and is parameterized by some variables that represent the arguments passed to the program at the beginning of its execution. An [illegible] is defined by a set of values (its identity, its public/private keys,…) and a [illegible] is the execution of a role with its values as parameters. In the Dolev-Yao model [12], attacks on protocols are modelled by the addition of a malicious participant, called the intruder, that controls the network. It can intercept, block and/or redirect all messages sent by honest agents. It can also masquerade its identity and take part in the protocol under the identity of an honest participant. Its control of the communication network is modelled by assuming that all messages sent by honest agents are sent directly to the intruder and that all messages received by the honest agents are always sent by the intruder. Besides the control on the net, the intruder has specific rules to deduce new values and compute messages. From the intruder's point of view a finite execution of a protocol is therefore the interleaving of a finite sequence of messages it has to send and a finite sequence of messages it receives (and add to its knowledge). Therefore the intruder is simply an additional role that runs concurrently with the honest participants. The protocol is said to be [illegible] if some secret knowledge is revealed to the intruder during the execution of the protocol. Deciding whether a protocol is insecure (for a single or a fixed number of sessions) has been shown equivalent to solving constraints in a term algebra [16,1,5].

4.2 Deduction Rules

Deduction rules are introduced to describe the operational behavior of roles (including the intruder). They are used to define deduction systems and derivations on sets of ground terms. A [illegible] d is a rule $s_1, \dots, s_n \to s$ where $s_1, \dots, s_n, s$ are terms of $T_{\mathcal{F}}(\mathcal{X})$, and $\mathrm{Var}(s) \subseteq \bigcup_{i=1,..,n} \mathrm{Var}(s_i)$. The set $\mathrm{GI}(d)$ is the set of instances of d, i.e. $\mathrm{GI}(d) = \{l_1, \dots, l_n \to r \mid l_i = s_i\sigma, i = 1, ..., n, r = s\sigma, \sigma \text{ ground substitution}\}$. An instance of a rule pattern is called a [illegible].

The Dolev-Yao model is defined by the following deduction system. The signature is $\mathcal{F} = \{se(,), \langle,\rangle, \dots\}$, where $se(x, y)$ denotes the symmetric encryption of x by the key y and $\langle x, y\rangle$ denotes the pair of two messages x and y, and the rule patterns are:

Composition rule patterns:	Decomposition rule patterns:
$x, y \to se(x, y)$	$se(x, y), y \to x$
$x, y \to \langle x, y\rangle$	$\langle x, y\rangle \to x$
	$\langle x, y\rangle \to y$

The model can be extended with free symbols and the Composition rule and Decomposition rule patterns $x_1, \dots, x_n \to f(x_1, \dots, x_n)$ and $f(x_1, \dots, x_n) \to x_i$ $(i = 1, \dots, n)$ for each symbol f of arity n. These new symbols and rules behave like the pairing operation and we shall not consider these symbols for the sake of simplicity.

The set GI_c is the union of all ground instances of composition rule patterns, and the set GI_d is the union of all ground instances of decomposition rule patterns.

The set GI is the union of GI_c and GI_d. We shall also call a ([illegible]) [illegible] a ground instance of a (de)composition rule pattern.

Given two sets of ground terms E, F and a deduction rule $l \to r \in \mathrm{GI}$ we write $E \to_{l \to r} F$ iff $F = E \cup \{r\}$ and $l \subseteq E$. Recall that l is a set of terms. We write $E \to F$ (resp. $E \to_d F$, resp. $E \to_c F$) if there exists a deduction rule $l \to r$ in GI (resp. GI_d, resp. GI_c) such that $E \to_{l \to r} F$.

A [illegible] D of length $n \geq 0$, is a sequence of finite sets of ground terms $E_0, \ldots, E_n$ such that $E_0 \to E_1 \to \cdots \to E_n$ where $E_i = E_{i-1}, t_i$ for every $i \in \{1, \ldots, n\}$. The term t_n is called the [illegible] of the derivation. A derivation is without stutter iff $t_i \in E_j$ implies $j \geq i$ for $i, j \in \{1, .., n\}$.

A term is derivable from E if there exists a derivation starting from E of goal t. The set $Der(E)$ is the set of terms derivable from E and the set $Der_c(E)$ is the set of terms derivable from E using only composition rules.

5 Constraint Systems

The insecurity problem of protocols with subterm predicates can be reduced to solving special constraint systems to be defined below. The process of translating a security problem to a constraint system will not be detailed here since it is similar to the standard case [16,5].

An expression $E \rhd t$ where E is a finite set of terms (not necessarily ground ones) and $t \in T_{\mathcal{F}}(\mathcal{X})$ will be called a [illegible], and means that t can be deduced from E. A ground substitution σ is a solution of a deduction constraint, denoted by $\sigma \models E \rhd s$ iff $s\sigma \in Der(E\sigma)$. We shall consider also a slight variant of deduction constraints denoted by $E \rhd_c t$. A ground substitution σ is a solution of $E \rhd_c t$ iff $s\sigma \in Der_c(E\sigma)$.

A [illegible] $\mathcal{C}$ is a triple $((E_i \rhd t_i)_{i \in \{1,\ldots,n\}}; \mathcal{S}; \mathcal{T})$ where

1. E_i are finite sets of terms such that i) $E_{i-1} \subseteq E_i$, and ii) for each x occuring in a term of E_i, there exist $j < i$ such that $x \in \mathrm{Var}(t_j)$ or there exist s such that $x \in \mathrm{Var}(s)$ and $s \preceq t_i \in T$ or there exists a ground term t such that $s \prec t \in \mathcal{T}$. Property ii) is called [illegible];
2. $\mathcal{S}$ is a unification system;
3. $\mathcal{T}$ is a subterm constraint system.

A ground substitution is a solution of $\mathcal{C}$, denoted by $\sigma \models \mathcal{C}$, iff $\sigma \models E \rhd s$ for each $E \rhd s \in \mathcal{E}$, $\sigma \models \mathcal{S}$ and $\sigma \models \mathcal{T}$.

[illegible] For instance if we consider one session of the protocol described in Example 2:

$$\begin{array}{l} \textit{Client } A: \ !x_1^A \ \text{WHERE } x_1^A \prec K \quad ?x_2^A \quad \text{WHERE } x_2^A \ = \mathrm{se}(Y', K_{A,B}) \\ \textit{Server } B: \ ?x_1^B \ \ !x_2^B = \mathrm{se}(Y, K_{A,B}) \quad \text{WHERE } r(x_1^B, Y) \prec D \end{array}$$

The secrecy of Y, i.e. whether the intruder initially knowing $\{A, B\}$ can view Y, can be reduced to solving the following constraint system:

$$((\{A, B, w\} \rhd x, \{A, B, w, \mathrm{se}(y, K_{A,B})\} \rhd y) \ ; \ \emptyset \ ; \ \{w \prec K, \ (x, y) \prec D\})$$

In other words the intruder should build some well-chosen term x and send it to server B so that he receives back a term $\mathrm{se}(y, K_{A,B})$ satisfying $(x, y) \prec D$ and from this term and his previous knowledge he can derive the secret term y.

As mentioned above, we may also consider constraint systems where $\rhd$ is replaced by $\rhd_c$. A constraint system $\mathcal{C} = (\mathcal{E}; \mathcal{S}; \mathcal{T})$ is a [illegible] iff the following conditions are satisfied:

1. each deduction constraint has the form $E \rhd x$ where x is a variable,
2. $\mathcal{S} = \emptyset$ and
3. each constraint of $\mathcal{T}$ has the form $s \prec x$.

A solved form is [illegible] iff for each $s \prec x \in \mathcal{T}$, for each $y \in \mathrm{Var}(s)$, there exists a constraint $E_y \rhd y$ occurring before the first constraint $E_x \rhd x$ in $\mathcal{E}$.

The main result of this paper is the following theorem.

Theorem 1. [illegible]

The rest of the paper is devoted to the description and the proof of correctness and completeness of the successive steps of an algorithm deciding the satisfiability of constraint systems in NPTIME. This algorithm is non-deterministic and applies 6 steps for transforming a constraint system $\mathcal{C}_0 = (\mathcal{E}_0; \mathcal{S}_0; \mathcal{T}_0)$ into a normalized solved form. Finally the satisfiability of the normalized solved form is checked in the last step.

6 Guessing Unification and Subterm Ordering

The first two steps of the algorithm guess identifications between subterms and subterm contraints.

[illegible]: Guess a subset S' of $\{s \overset{?}{=} t \mid s, t \in \mathrm{Sub}(\mathcal{C}_0)\}$ and guess $\mathcal{T}'$ a finite set of subterm constraints $s \prec x$ for $s \in \mathrm{Sub}(\mathcal{C}_0), x \in Var(\mathcal{C}_0)$.
Check that $\mathcal{S}_0 \cup \mathcal{S}'$ defines a congruence and that $\mathcal{T}_0 \cup \mathcal{T}'$ defines a transitive antisymmetric relation. Let $\mathcal{E}_1 = \mathcal{E}_0$, let $\mathcal{S}_1 = \mathcal{S}_0 \cup \mathcal{S}'$, let $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{T}'$, and finally let $\mathcal{C}_1 = (\mathcal{E}_1; \mathcal{S}_1; \mathcal{T}_1)$. We check easily:

Lemma 1. [illegible] $\sigma = mgu(\mathcal{S}_1)$ [illegible] $\mathcal{C}_1\sigma$ [illegible]

[illegible]: Let $\sigma = mgu(\mathcal{S}_1)$. Let $\mathcal{T}_2$ be obtained from $\mathcal{T}_1\sigma$ by applying the simplification rules: $s \prec t \rightarrow true$ if $t_{|p} = s$ for some position p in t
$s \prec t \rightarrow s \prec x$ for $x \in \mathrm{Var}(t)$, if there is no p such that $t_{|p} = s$
Let $\mathcal{E}_2 = \mathcal{E}_1\sigma$ and let $\mathcal{C}_2 = (\mathcal{E}_2; \mathcal{S}_2; \mathcal{T}_2)$ where $\mathcal{S}_2 = \emptyset$.

Steps 1,2 are non-deterministic and there are finitely many possible outcomes $\mathcal{C}_2$ since the number of guesses is finite and the number of possible results of simplification rules is finite.

[illegible] The simplification rules for $\prec$ are correct and complete because we perform all possible guesses of equalities between terms. For instance, given a constraint $g(a) \prec f(g(x))$, one guess is $g(a) = g(x)$ and the subterm constraint becomes $g(a) \prec f(g(a))$ and the other guesses make $g(a)$ and $g(x)$ different and the subterm constraint is equivalent to $g(a) \prec x$.

Proposition 1. [illegible] σ [illegible] $\mathcal{C}_0$ [illegible] $\mathcal{C}_2$ [illegible] σ [illegible] $\mathcal{C}_2$ [illegible] σ [illegible] $\mathcal{C}_2$ [illegible] $\mathcal{C}_0$

[illegible] A solution σ of the initial constraint system defines a congruence $\equiv_\sigma$ on the subterms of $\mathcal{C}_0$, namely $s \equiv_\sigma t$ iff $s\sigma = t\sigma$. It also defines a transitive and anti-symmetric relation $\prec_\sigma$ on $\mathrm{Sub}(\mathcal{C}_0)$, namely $s \prec_\sigma t$ if $s\sigma \in \mathrm{Sub}(t\sigma) \setminus \{t\}$. In Step 1 of the algorithm, we guess any possible choices for congruence and the transitive and anti-symmetric relations $\equiv_\sigma$ and $\prec_\sigma$. Therefore, in the following we consider solutions that match exactly the congruence and the ordering guessed in Step 1, i.e. for all terms in $\mathrm{Sub}(\mathcal{C}_2)$ if the equality $u = v$ is not induced by $\mathcal{S}_2$, we shall assume that $u\sigma \neq v\sigma$, and if $u \prec v$ is not induced by $\mathcal{S}_2$ and $\mathcal{T}_2$, we shall assume $u\sigma \not\prec v\sigma$.

This first guessing phase permits us to obtain the following two lemmas. We denote by $s \sqsubseteq t$ the reflexive transitive closure of the binary relation on terms defined by $s \prec t \in \mathcal{T}_2$ or $s \in \mathrm{Sub}(t)$.

Lemma 2. [illegible] $\mathcal{C}_2 = (\mathcal{E}_2; \mathcal{S}_2; \mathcal{T}_2)$ [illegible] $\mathcal{E}_2 = (E_i \rhd s_i)_{i \in \{1,\ldots,n\}}$ [illegible] x [illegible] E_i [illegible] $j < i$ [illegible] $x \sqsubseteq s_j$, $x \notin \mathrm{Var}(E_j)$ [illegible] $x\sigma \notin$ [illegible] $(E_j\sigma)$

[illegible] Let j be minimal such that $x \sqsubseteq s_j$. The guessings at Step 1 imply that there is no $u \in \mathrm{Sub}(E_j) \setminus x$ with $u\sigma = x\sigma$. By contradiction, if there is $y \in \mathrm{Var}(E_j)$ such that $x \sqsubseteq y$, by definition of constraint systems either there is $j' < j$ such that $y \sqsubseteq s_{j'}$ or there is a ground term t with $y \prec t$. The latter is impossible since y has not been unified with a subterm of $\mathcal{C}_0$ at Step 1. Hence $y \sqsubseteq s_{j'}$ and $x \sqsubseteq y$. By transitivity we also have $x \sqsubseteq s_{j'}$, which contradicts the minimality of j.

Lemma 3. [illegible] σ [illegible] $\mathcal{C}_2 = (\mathcal{E}_2; \mathcal{S}_2; \mathcal{T}_2)$ [illegible] $(E \rhd s) \in \mathcal{E}_2$ [illegible] $t \in$ [illegible] $(\mathcal{C}_2)$ [illegible] t [illegible] (E) [illegible] $t\sigma \in$ [illegible] $(s\sigma)$ [illegible] $t\sigma$ [illegible] *Der*$(E\sigma)$ [illegible] $t\sigma$ [illegible]

[illegible] Since the sequence $\mathcal{E}_2 = (E_i \rhd s_i)_{i=1,\ldots,n}$ is such that $E_i \subseteq E_j$ if $i \leq j$, we may choose $E \rhd s$ as the first deduction constraint such that $t\sigma \in \mathrm{Sub}(s\sigma)$, and thus, given the guessing at Step 1, such that $t \sqsubseteq s$.

We first show that there is no $x \in \mathrm{Var}(E)$ such that $t\sigma \in \mathrm{Sub}(x\sigma)$, and thus $t \sqsubseteq x$. By contradiction if there exists such a variable x, by Lemma 2 there exists a constraint $E' \rhd s'$ prior to $E \rhd s$ such that $x \sqsubseteq s'$. By transitivity of $\sqsubseteq$ this implies that $t \sqsubseteq s'$, thereby contradicting the choice of the deduction constraint $E \rhd s$. Thus there is no variable $x \in \mathrm{Var}(E)$ such that $t\sigma$ is a subterm of $x\sigma$.

Thus if $t\sigma$ is in $\mathrm{Sub}(E\sigma)$, there existsy $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ such that $u\sigma = t\sigma$. Given the choice at Step 1, this implies in turn that $u = t$, and thus that

$t \in \mathrm{Sub}(E)$. The lemma is then trivially true since its hypotheses are not met on E.

Assume now $t\sigma \notin \mathrm{Sub}(E\sigma)$. Let D be a minimal derivation of goal $s\sigma$, say

$$F_0 = E\sigma \to \ldots \to F_j = E\sigma, t_1, \ldots, t_j \to \ldots \to F_m = E\sigma, t_1, \ldots, t_m = s\sigma$$

Let j be the smallest index such that $t\sigma \in \mathrm{Sub}(F_j)$. This index is defined since $t\sigma \in \mathrm{Sub}(t_n)$, and is not 0 since $t\sigma \notin \mathrm{Sub}(E\sigma)$. The minimality implies that $t\sigma \notin \mathrm{Sub}(F_{j-1})$, and thus $t\sigma \in \mathrm{Sub}(t_j)$. Since the right-hand side of a decomposition rule is always a subterm of its left-hand side, the deduction rule applied must be a composition rule. In this case, $\mathrm{Sub}(t_j) \setminus \{t_j\} \subseteq \mathrm{Sub}(F_{j-1})$ and thus we must have $t_j = t\sigma$. Truncating the derivation yields a derivation starting from $E\sigma$ of goal $t\sigma$ ending with a composition rule.

7 Adding Deduction Constraints for Variables

Given a constraint system $\mathcal{C} = ((E_i \rhd s_i)_{i\in\{1,\ldots,n\}}; \mathcal{S}; \mathcal{T})$, a variable x may occur in $\mathrm{Var}(E_i)$ but not in $\mathrm{Var}(s_j)$ for $j < i$. The next step transforms the constraint system into a constraint system where all variables of $\mathrm{Var}(E_i)$ occur in $\mathrm{Var}(s_j)$ for $j < i$.

[illegible]: For each variable x such that $x \in \mathrm{Var}(E_i)$ and $x \notin \mathrm{Var}(s_j)$ for $j < i$, replace $(E_i \rhd s_i)$ in $\mathcal{E}_2$ by $E_j \rhd x, E_i \rhd s_i$ for some $j < i$.
Let $\mathcal{C}_3 = (\mathcal{E}_3; \mathcal{S}_3; \mathcal{T}_3)$ be the resulting constraint system.

The correctness of the transformation relies on the following lemmas.

Lemma 4. [illegible] $\mathcal{C}_2 = (\mathcal{E}_2; \mathcal{S}_2; \mathcal{T}_2)$ [illegible] $E_i \rhd s_i \in \mathcal{E}_2$ [illegible] $x \in \mathrm{Var}(E_i)$ [illegible] $j < i$ [illegible] σ [illegible] $\mathcal{C}_2$ [illegible] $E_j\sigma$ [illegible] $x\sigma$ [illegible]

[illegible] By Lemma 2, the hypothesis $x \in \mathrm{Var}(E_i)$ implies that there exists $j < i$ with $x \notin \mathrm{Var}(E_j)$ and such that $x \sqsubseteq s_j$. By Lemma 3 applied on x, we derive this Lemma 4.

8 Eliminating Decomposition Rules

We prove in this section that for any deduction constraint $E \rhd m$ belonging to a constraint system $\mathcal{C}_2$ produced by Step 2, if there exists a ground substitution σ solution of $\mathcal{C}_2$, then there exists a derivation starting from $E\sigma$ of goal $m\sigma$ such that any decomposition rule $s_1, s_2 \to s$ or $s_1 \to s$ applied in this derivation is such that there exists $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ with $u = se(t, d)$ or $u = \langle t, t' \rangle$ with $s_1 = u\sigma$, $s_2 = d\sigma$ and $s = t\sigma$ or $s = t'\sigma$. This implies that there exists a subset of $\mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ such that, once these terms have been decomposed, all terms derivable from E can be derived by applying only composition rules.

In Step 3, we guess this set of subterms of E. The derivation contains several instances of decomposition rules that derive a new term t_j (a message below an encryption for instance) using a term d_j (a term used as an encryption key for instance) which is derived using composition rules.

[illegible]: For each $(E \rhd s) \in \mathcal{E}_3$, guess $t_1, \ldots, t_n, d_1, \ldots, d_n \in \mathrm{Sub}(E)$ such that for $j = 1, \ldots, n$,

$$E, t_1, \ldots, t_{j-1}, d_j \rightarrow_d E, t_1, \ldots, t_j, d_j$$

Replace $(E \rhd s)$ in $\mathcal{E}_3$ by

$$E \rhd_c d_1,\ E, t_1 \rhd_c d_2, \ldots,\ E, t_1, \ldots, t_{n-1} \rhd_c d_n, E, t_1, \ldots, t_n \rhd_c s$$

Let $\mathcal{C}_4 = (\mathcal{E}_4; \mathcal{S}_4; \mathcal{T}_4)$ be the resulting constraint system.

To state the correctness and completeness of this step, we need to prove several lemmas.

Lemma 5. [illegible] $\mathcal{C}_3 = (\mathcal{E}_3; \mathcal{S}_3; \mathcal{T}_3)$ [illegible] σ [illegible] $\sigma \models \mathcal{C}_3$ [illegible] $\sigma \models E \rhd s$ [illegible] E [illegible] $\mathcal{E}_3$ [illegible] $s \in$ [illegible] $(\mathcal{C}_3)$ [illegible] $E\sigma$ [illegible] $s\sigma$ [illegible] $x\sigma$ [illegible] $x \in \mathrm{Var}(E)$

[illegible] By contradiction let us assume there exists E and s as specified such that the minimal number of decompositions of a term $x\sigma$ (with $x \in \mathrm{Var}(E)$) in a derivation starting from $E\sigma$ of goal $s\sigma$ is $n > 0$. Without loss of generality let us consider this is the first such E in the order of the deduction constraints. Notice that this cannot be the leftmost E since the first left-hand side contains only ground terms (as a result of Step 3). Let $x \in \mathrm{Var}(E)$ be such that there exists a derivation with n decompositions of terms $y\sigma$ with $y \in \mathrm{Var}(E)$, and x is one of those variables. By Lemma 4 there exists $E_x \rhd s_x$ before $E \rhd s$ such that there is a derivation starting from $E_x\sigma$ with goal $x\sigma$ ending with a composition rule. By minimality of E we can assume that this derivation does not contain any decomposition of a term $y\sigma$ for $y \in \mathrm{Sub}(E_x)$. Since it ends with a composition rule, the strict maximal subterms of $x\sigma$, and thus the result of its decomposition, are also deduced by this derivation. Since $E_x \subseteq E$ by definition of constraint systems, we can replace the decomposition of $x\sigma$ by this derivation. This permits us to obtain a derivation in which $n-1$ variable instances are decomposed, thus contradicting the minimality of n.

Lemma 6. [illegible] $\mathcal{C}_3 = (\mathcal{E}_3; \mathcal{S}_3; \mathcal{T}_3)$ [illegible] σ [illegible] $\sigma \models \mathcal{C}_3$ [illegible] $E \rhd s \in \mathcal{E}_3$ [illegible]

$S_E(s) = \{\epsilon\} \cup \{p \in \mathrm{Pos}(s) \mid \forall q < p, \sigma \models E \rhd s_{|q}$ [illegible] $E\sigma$ [illegible] $s_{|q}\sigma$ [illegible] $\}$

[illegible] p [illegible] $S_E(s)$ [illegible] $s_{|p} \in$ [illegible] $(E) \cup \mathcal{X}$ [illegible] $s_{|p} \notin \mathcal{X} \cup E$ [illegible] $s_{|p}$ [illegible] (E)

[illegible] Let p be maximal in $S_E(s)$. Then either $p = \epsilon$ or there exists a position p' and an integer i such that $p = p' \cdot i$.

First, if $p = \epsilon$ then $s\sigma$ cannot be obtained by a composition rule. Since $\sigma \models E \rhd s$, By Lemma 3 we must have $s \in \mathrm{Sub}(E)$.

Now if $p \neq \epsilon$, by definition of $S_E(s)$ we must have $s_{|p'}\sigma$ is in $Der(E\sigma)$ and there exists a derivation starting from $E\sigma$ of goal $s_{|p'}\sigma$ ending with a composition rule. This implies that this derivation contains the term $s_{|p' \cdot i}\sigma$ in its next to last term set, and thus that $s_{|p}\sigma$ is in $Der(E\sigma)$. By maximality of p either $s_{|p}$ is a variable or there is no derivation starting from $E\sigma$ of goal $s_{|p}\sigma$ ending with a composition rule, and thus in this last case by Lemma 3 we must have $s_{|p} \in \mathrm{Sub}(E)$.

Thus, if p is maximal in $S_E(s)$ and is not a variable, we have $s_{|p}\sigma \in Der(E\sigma)$, and $s_{|p} \in \mathrm{Sub}(E)$, and there exists no derivation starting from $E\sigma$ of goal $s_{|p}\sigma$ ending with a composition rule.

Let us now consider a derivation starting from $E\sigma$ of goal $s_{|p}\sigma$. Without loss of generality we can consider it is without stutter (and thus no decomposition is applied on a term that has been composed before) and such that no variable instance is decomposed (this is possible by Lemma 5.) Let us prove that in this derivation, for any decomposition rule $t_1, t_2 \to t$ applied (with $t \in \mathrm{Sub}(t_1)$), there exists $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ such that $u\sigma = t$. By contradiction, let us assume this is not the case, and let $t_1, t_2 \to t$ of a decomposition rule such that there does not exist $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ with $u\sigma = t_1$. Since there is no stutter, t_1 has not been obtained by a composition rule. Thus either there exists $u \in E$ with $u\sigma = t_1$ or there exists a previous decomposition rule $t'_1, t'_2 \to t_1$. Since we have taken the first decomposition rule $t_1, t_2 \to t$ where t_1 is not an instance of a non-variable subterm of E, there exists $u' \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ such that $u'\sigma = t'_1$. This implies there exists $u \in \mathrm{Sub}(E)$ such that $t_1 = u\sigma$. But then either u is a variable, and we contradict the fact that the derivation does not contain any decomposition of the instance of a variable, or $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ and we contradict the fact that there does not exists $u \in \mathrm{Sub}(E) \setminus \mathrm{Var}(E)$ such that $u\sigma = t_1$. This terminates the proof of the lemma.

Proposition 2. [illegible] $\mathcal{C}_3$ [illegible] $t_1, \dots, t_n, d_1, \dots, d_n$ [illegible] $\mathcal{C}_4$ [illegible] $t \in$ [illegible] $(\mathcal{C}_4)$ [illegible] E [illegible] $\mathcal{C}_4$ [illegible] $\sigma \models E \rhd t$ [illegible] $\sigma \models E \rhd_c t$

[illegible] Let $\mathcal{C}_3 = (\mathcal{E}_3; \mathcal{S}_3; \mathcal{T}_3)$ with $\mathcal{E}_3 = (E_i \rhd s_i)_{i=1,\dots,m}$ and let us assume that $\mathcal{C}_3$ has a solution σ.

Firstly we prove that $\mathcal{C}_4$ is satisfiable. Let

$$\mathcal{M}_i = \{t \mid \exists s \in \mathrm{Sub}(\mathcal{C}_4), \sigma \models E_i \rhd s \text{ and } t = s_{|p} \text{ with } p \text{ maximal in } S_{E_i}(t)\}$$

First note that $\mathcal{M}_i$ is finite and included in $\mathrm{Sub}(E_i) \cup \mathcal{X}$ by Lemma 6.

Let $\mathcal{M}_i \setminus E_i = \{t_1, \dots, t_n\}$ and let us choose $t_1, \dots, t_n$ as the sequence of guesses used in Step 3 for the t_i's. By definition there is a derivation that constructs $t_1\sigma, \dots, t_n\sigma$ from $E_i\sigma$. W.l.o.g. we may assume that the indices of the t_i are in the order in which they appear in this derivation.

By Lemma 6, $t_i\sigma$ is obtained by the decomposition of a term $u_i\sigma$ with $u_i \in \mathrm{Sub}(E_i)$. The decomposition is $u_i\sigma \to t_i\sigma$ (rule $\langle x, y\rangle \to x$) or $u_i\sigma, d_i\sigma \to t_i\sigma$ (rule $se(x, y), y \to x$).

Since all the terms that are not among the t_i can be obtained by composition we can assume, by considering a derivation with a minimal number of decomposition rules, that all terms but the t_i, are deduced by a composition rule or are already present.

Thus $d_i\sigma$ is composable using only composition rules from $E_i\sigma, t_1\sigma, \ldots, t_{i-1}\sigma$.

Provided that $u_i, d_i \to t_i$ is a decomposition rule, this implies that σ is a solution of $\mathcal{C}_4$.

Since the $t_i\sigma$ are obtained by decomposition of a term $u_i\sigma$ with $u_i \in \mathrm{Sub}(E_i)$, the deduction rule instances permitting the deduction of $t_i\sigma$ are $u_i\sigma \to t_i\sigma$ for the pair and $u_i\sigma, d_i\sigma \to t_i\sigma$ for deciphering.

Now we prove that $\sigma \models E \rhd t$ iff $\sigma \models E \rhd_c t$.

⇒ This direction comes from the identification of subterms at Step 1 and on the definition of a right guess at Step 3, the guess at the beginning of this proof.

⇐ Straightforward.

9 Computing Normalized Solved Forms

: For each $E \rhd_c s$ in $\mathcal{E}_4$, check that $s \in Der_c(E \cup \mathrm{Var}(s))$ and replace $E \rhd_c s$ by deduction constraints $E \rhd_c x$ for all $x \in \mathrm{Var}(s)$.
Let $\mathcal{C}_5 = (\mathcal{E}_5; \mathcal{S}_5; \mathcal{T}_5)$ be the resulting normalized solved form.

The soundness and completeness of Step 5 is a direct consequence of the next proposition.

Proposition 3. $\mathcal{C}_4 = (\mathcal{E}_4; \mathcal{S}_4; \mathcal{T}_4)$ $m \in$ $(\mathcal{C}_4)$ $\tau \models E \rhd_c m$ iff $\tau \models E \rhd_c x$ $x \in \mathrm{Var}(m)$ $m \in Der_c(E, \mathrm{Var}(m))$

Assume first that $\tau \models E \rhd_c m$. Let Π be the set of minimal positions in $S(m)$. By Lemma 6 we know that $m_{|p}$ is either a variable or a term among the t_j terms guessed in Step 3 or in a knowledge set at Step 2. By definition of Dolev-Yao composition rules, we have $m \in Der_\mathrm{c}(\{t \mid m_{|p} = t \text{ for } p \in \Pi\})$, and thus, after the guess in Step 3, we have $m \in Der_\mathrm{c}(E \cup \mathrm{Var}(m))$. Given a variable $x \in \mathrm{Var}(m)$, by the determinacy of constraint systems there exists a constraint $E_x \rhd m_x$ in $\mathcal{C}_4$ with $x \in \mathrm{Var}(m_x) \setminus \mathrm{Var}(E_x)$. By Lemma 3 this implies that $x\tau \in Der(E_x\tau)$. By the inclusion of knowledge sets we have $E_x \subseteq E$ and thus $x\tau \in Der(E\tau)$. Given the guess at Step 3 this implies $x\tau \in Der_\mathrm{c}(E\tau)$ and thus $\tau \models E \rhd_c x$.

Conversely, if $\tau \models E \rhd_c x$ for all $x \in \mathrm{Var}(m)$ then starting from $E\tau$ one can first construct a set of terms F containing $E\tau \cup \mathrm{Var}(m)\tau$. Then one can instantiate with τ a derivation starting from $E \cup \mathrm{Var}(m)$ of goal m. These two derivations employ only composition rules, therefore $m\tau \in Der_\mathrm{c}(E\tau)$.

10 Permuting Deduction Constraints

This step aims at providing a compatibility between the ordering on variables induced by $\mathcal{T}$ and the ordering on variables induced by the deduction constraints. It is not necessary for the algorithm, but simplifies the proof of the last step. The variables X, Y and Z stand for (possibly empty) sequences of deduction constraints.

```
[illegible]  $\mathcal{E} \leftarrow \mathcal{E}_5$
    For all $x, y \in \mathrm{Var}(\mathcal{C}_5)$ with
            $\mathcal{E} = (X, E_x \rhd_c x, Y, E_y \rhd_c y, Z)$ and $y \prec x \in \mathcal{T}$
    do
        $\mathcal{E} \leftarrow (X, E_x \rhd_c y, E_x \rhd_c x, Y, Z)$
    od
```

Let $\mathcal{C}_6 = (\mathcal{E}_6; \mathcal{S}_6; \mathcal{T}_6)$ be the resulting normalized solved form.

Proposition 4. [illegible] σ [illegible] $\mathcal{C}_5$ [illegible] $\mathcal{C}_6$

[illegible] Since the E_i are nondecreasing sets, it is obvious that if σ is a solution of $\mathcal{C}_6$ then it is a solution of $\mathcal{C}_5$.

Assume now that a ground substitution σ is a solution of $\mathcal{C}_5$. It suffices to prove that if σ is a solution before a swap, it will remain a solution of the constraint system after the swap. There are two cases:

- If there exists a constraint $E'_y \rhd_c y$ before $E_x \rhd_c$, then $E'_y \subseteq E_x$ implies that if σ is a solution before the swap, it will be a solution after;
- Else, the determinacy of constraint systems implies that $y \notin \mathrm{Sub}(E_x)$, and the σ solution implies that $y\sigma \in \mathrm{Sub}(x\sigma)$. Thus by Lemma 3 this implies $y\sigma \in Der(E_x\sigma)$. By Proposition 2 this implies in turn $y\sigma \in Der_c(E_x\sigma)$, and thus the resulting constraint system is still satisfied by σ.

11 Decision of Normalized Solved Forms

This is the last step of the algorithm.

[illegible] : For each $E_x \rhd_c x \in \mathcal{E}_6, s \prec x \in \mathcal{T}_6$, check that there exists $t \in E$ such that $s \prec t$ or $s \in Der_c(E \cup \mathrm{Var}(s))$ and return true otherwise return fail.

Proposition 5. [illegible] $\mathcal{C}_6$ [illegible]

[illegible] Let $\mathcal{C}_6 = (\mathcal{E}_6; \mathcal{S}_6; \mathcal{T}_6)$.
$\Rightarrow$ direction: Let us assume that the variables occur in deduction constraints in the order $x_1, x_2, \ldots$. We construct a solution σ inductively according to this ordering. For simplicity we still call σ the restriction of σ to $\{x_1, \ldots, x_n\}$.

Assume that we have constructed $\sigma = \{x_1 \leftarrow x_1\sigma, \ldots, x_{n-1} \leftarrow x_{n-1}\sigma\}$ such that $\sigma \models E \rhd_c x_i$ and $\sigma \models s \prec x_i$ for all deduction and subterm constraints of $\mathcal{C}_6$ with $i < n$.

Let $s_1 \prec x_n, \ldots, s_m \prec x_n$ be the subterm constraints of $\mathcal{T}_6$ which have x_n as a right-hand side. Let $E \rhd_c x_n$ be the first deduction constraint of $\mathcal{E}$ containing x_n as a right-hand side.

Let t_i be such that $t_i \in E$ and $s_i \prec t_i$ or $t_i = s_i$ if $s_i \in Der_c(E \cup \mathrm{Var}(s))$. By hypothesis t_i always exists for $i = 1, \ldots, m$.

We extend σ by setting $x_n\sigma = \langle t_1\sigma, \langle t_2\sigma, \ldots \langle t_{m-1}\sigma, t_m\sigma\rangle\rangle\rangle$.

By construction $\sigma \models E \rhd_c x_n$ (hence $\sigma \models E' \rhd_c x_n$ since $E \subseteq E'$) and $\sigma \models \mathcal{T}$.

$\Leftarrow$ direction: Let σ be a solution of $\mathcal{C}_6$. Let $E \rhd_c x \in \mathcal{E}_6$ and $s \prec x \in \mathcal{T}_6$.

Let $F_0 = E\sigma \rightarrow E\sigma, t_1 \ldots \rightarrow E\sigma, t_1, \ldots, t_j \rightarrow \ldots \rightarrow x\sigma$ be a derivation of goal $x\sigma$ using composition rules only. Let j be the first index such that $s\sigma$ occurs in t_j.

Either there is $t \in E$ such that $t\sigma = s\sigma$ then $s = t \in E$ and the check succeeds.

Or, $t_j = s\sigma$ for some $j \geq 1$ and $\sigma \models E \rhd_c s$. By Proposition 3, we have $s \in Der_c(E \cup Vars)$ and the check succeeds.

12 Conclusion

We have shown how to decide secrecy for cryptographic protocols that can check a subterm relation. The proof is short (no appendix!) and shows that the complexity of the problem is in NP. In future work we will investigate the case of negative subterm constraints and also whether the results can be combined with known results on protocols that employ associative-commutative message constructors: this would enlarge the scope of techniques for addressing XML protocol security. We also plan to investigate whether our result generalizes to more general queries than subterm.

References

1. Amadio, R., Lugiez, D., Vanackère, V.: On the symbolic reduction of processes with cryptographic functions. Theor. Comput. Sci. 290(1), 695–740 (2003)
2. Armando, A., Compagna, L.: Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In: Foundation of Computer Security & Verification Workshops, Copenhagen, Denmark (2002)
3. Basin, D.A., Mödersheim, S., Viganò, L.: Algebraic intruder deductions. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 549–564. Springer, Heidelberg (2005)
4. Cervesato, I., Durgin, N.A., Lincoln, P., Mitchell, J.C., Scedrov, A.: A meta-notation for protocol analysis. In: CSFW, pp. 55–69 (1999)
5. Chevalier, Y., Vigneron, L.: A Tool for Lazy Verification of Security Protocols. In: ASE 2001. Proceedings of the Automated Software Engineering Conference, IEEE Computer Society Press, Los Alamitos (2001)
6. Chevalier, Y., Lugiez, D., Rusinowitch, M.: Towards an automatic analysis of web services security. In: Konev, B., Wolter, F. (eds.) FroCoS 2007. LNCS (LNAI), Springer, Heidelberg (2007)
7. Baader, F., Schulz, K.U.: Unification in the union of disjoint equational theories. combining decision procedures. J. Symb. Comput. 21(2), 211–243 (1996)

8. Bhargavan, K., Fournet, C., Gordon, A.D., Pucella, R.: Tulafale: A security tool for web services. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2003. LNCS, vol. 3188, pp. 197–222. Springer, Heidelberg (2004)
9. Cortier, V., Zalinescu, E.: Deciding Key Cycles for Security Protocols. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 317–331. Springer, Heidelberg (2006)
10. Delaune, S., Jacquemard, F.: A decision procedure for the verification of security protocols with explicit destructors. In: CCS 2004. Proceedings of the 11th ACM Conference on Computer and Communications Security, pp. 278–287. ACM Press, Washington, D.C., USA (2004)
11. Dershowitz, N., Jouannaud, J.P.: Rewrite systems. In: Handbook of Theoretical Computer Science, vol. B, pp. 243–320. Elsevier, Amsterdam (1990)
12. Dolev, D., Yao, A.: On the Security of Public-Key Protocols. IEEE Transactions on Information Theory 2(29) (1983)
13. Kleiner, E., Roscoe, A.: On the Relationship Between Web Services Security and Traditional Protocols. Electr. Notes Theor. Comput. Sci. 155, 583–603 (2006)
14. Lynch, L., Meadows, C.: On the Relative Soundness of the Free Algebra Model for Public Key Encryption. In: Proc. 4th Workshop on Issues in the Theory of Security (WITS) (2004)
15. Meadows, C., Narendran, P.: A unification algorithm for the group Diffie-Hellman protocol. In: Workshop on Issues in the Theory of Security (in conjunction with POPL'02), Portland, Oregon, USA, pp. 14–15 (January 2002)
16. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: ACM Conference on Computer and Communications Security, pp. 166–175 (2001)
17. Rits, M., Rahaman, M.A.: Secure SOAP Requests in Enterprise SOA. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186, Springer, Heidelberg (2006)
18. Rusinowitch, M., Turuani, M.: Protocol insecurity with finite number of sessions is NP-complete. In: Proc.14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia (2001)
19. Venkataraman, K.N.: Decidability of the purely existential fragment of the theory of term algebras. J. ACM 34(2), 492–510 (1987)
20. Weidenbach, C.: Towards an automatic analysis of security protocols in first-order logic. In: Ganzinger, H. (ed.) CADE-16. LNCS (LNAI), vol. 1632, pp. 314–328. Springer, Heidelberg (1999)

Deciding Knowledge in Security Protocols for Monoidal Equational Theories*

Véronique Cortier and Stéphanie Delaune

LORIA, CNRS & INRIA project Cassis, Nancy, France

Abstract. In formal approaches, messages sent over a network are usually modeled by terms together with an equational theory, axiomatizing the properties of the cryptographic functions (encryption, exclusive or, ...). The analysis of cryptographic protocols requires a precise understanding of the attacker knowledge. Two standard notions are usually used: deducibility and indistinguishability. Only few results have been obtained (in an ad-hoc way) for equational theories with associative and commutative properties, especially in the case of static equivalence. The main contribution of this paper is to propose a general setting for solving deducibility and indistinguishability for an important class (called *monoidal*) of these theories. Our setting relies on the correspondence between a monoidal theory E and a semiring $\mathcal{S}_{\mathsf{E}}$ which allows us to give an algebraic characterization of the deducibility and indistinguishability problems. As a consequence we recover easily existing decidability results and obtain several new ones.

1 Introduction

Security protocols are paramount in today's secure transactions through public channels. It is therefore essential to obtain as much confidence as possible in their correctness. Formal methods have proved their usefulness for precisely analyzing the security of protocols. Understanding security protocols often requires reasoning about knowledge of the attacker. In formal approaches, two main kind of definitions have been given in the literature for this knowledge. They are known as message deducibility and indistinguishability relations.

Most often, the knowledge of the attacker is described in terms of message deducibility [15,18,16]. Given some set of messages ϕ representing the knowledge of the attacker and another message M, intuitively the secret, one can ask whether an attacker is able to compute M from ϕ. To obtain such a message he uses his deduction capabilities. For instance, he may encrypt and decrypt using keys that he knows.

This concept of deducibility does not always suffice for expressing the knowledge of an attacker. For example, if we consider a protocol that transmits an encrypted Boolean value (e.g., the value of a vote), we may ask whether an attacker can learn this value by eavesdropping on the protocol. Of course, it seems

---

* This work has been partly supported by the RNTL project POSÉ and the ARA SSIA Formacrypt.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 196–210, 2007.
© Springer-Verlag Berlin Heidelberg 2007

to be completely unrealistic to say that the Boolean true and false are not deducible. We need to express the fact that the two transcripts of the protocol, one running with the Boolean value true and the other one with false are . Besides allowing more careful formalization of secrecy properties, indistinguishability can also be used for proving the more involved notion of cryptographic indistinguishability (e.g. [6]): two sequences of messages are cryptographically indistinguishable if their distributions are indistinguishable to any attacker, that is to any probabilistic polynomial Turing machine.

In both cases, deduction and indistinguishability apply to observations on messages at a particular point in time. They do not take into account the dynamic behavior of the protocol. For this reason the indistinguishability relation is called . Nevertheless those relations are quite useful to reason about the dynamic behavior of a protocol. For instance, the deducibility relation is often used as a subroutine of many decision procedures [19,8,10]. In the applied-pi calculus framework [2], it has been shown that observational equivalence (relation which takes into account the dynamic behavior) coincides with labeled bisimulation which corresponds to checking static equivalences and some standard bisimulation conditions.

Both of these relations rely on an underlying equational theory axiomatizing the properties of the cryptographic functions (encryption, exclusive or, . . .). Many decision procedures have been provided to decide these relations under a variety of equational theories. For instance algorithms for deduction are provided for exclusive or [10], homomorphic operators [11] and subterm theories [1]. These theories allow basic equations for functions such as encryption, decryption and digital signature. There are also results for static equivalence. For instance, a general decidability result for the class of subterm convergent equational theories is given in [1]. This class contains classical cryptographic primitives like encryption, signatures and hashes. Also in [1] some abstract conditions on the underlying equational theory are proposed to ensure decidability of deduction and static equivalence. Note that the use of this result requires checking some assumptions, which might be difficult to prove. Regarding theories with associative and commutative properties (AC), they only obtain decidability for pure AC and exclusive or. A weakness of most of these approaches is their lack of generality since each new theory requires a new proof. Homomorphic properties occur in many protocols and cannot be dealt with by a simple adaptation of the techniques that have been developed so far.

In this paper, we consider the axioms of Associativity-Commutativity (AC), Unit element (U), Nilpotency (N), Idempotency (I), homomorphism (h), and more especially the combinations of these axioms that constitute monoidal theories. We propose a general approach to handle theories that covers several cases already studied, and furthermore includes some new decidability and complexity results on homomorphic operators. Monoidal theories have been extensively studied by F. Baader and W. Nutt [17,4,5] who have provided a complete survey of unification in these theories. More recently, these theories have been studied in the context of security protocols. S. Delaune . have shown that deduction is decidable for a subclass of monoidal equational theories, also considering active attacks [12]. However, they do not address static equivalence.

Studying monoidal theories might seem very restricted since they do not contain the equational theories for classical operators like encryption or signatures. However, it has been shown in [3] that equational theories can easily be combined for both deduction and static equivalence, provided the signatures are disjoint. That is why it is sufficient to focus on the important case of monoidal theories. As a consequence of our general approach, we recover many existing results and we obtain several new ones (10 new decidability or complexity results) for static equivalence or deduction.

O . In Section 2 we recall some basic notation and the central notion of monoidal theory. Then, in Section 3, we define the two notions of knowledge we are interested in. In Section 4 we show how to represent terms and substitutions by means of vectors and matrices over semirings. Then Sections 5 and 6 are devoted to the study of deduction and static equivalence respectively. In Section 7, we sum up our results and provide new results obtained as a consequence of our main theorems.

2 Preliminaries

2.1 Terms

A Σ consists of a finite set of function symbols, each with an arity. A function symbol with arity 0 is a constant symbol. We assume given a signature Σ, an infinite set of names $\mathcal{N}$, and an infinite set of variables $\mathcal{X}$. The concept of names is borrowed from the applied pi calculus [2] and corresponds to the notion of free constant used for instance in [9]. Let $\mathcal{M}$ be a set of names and variables, we denote by $\mathcal{T}(\Sigma, \mathcal{M})$ the set of over $\Sigma \cup \mathcal{M}$. $\mathcal{T}(\Sigma, \mathcal{N})$ is called the set of terms while $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ is simply called the set of terms. We write (M) (resp. (M)) for the set of names (resp. variables) that occur in the term M. A σ is a mapping from a finite subset of $\mathcal{X}$ called its domain and written (σ) to $\mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$. Substitutions are extended to endomorphisms of $\mathcal{T}(\Sigma, \mathcal{X})$ as usual. We use a postfix notation for their application. Given two terms N_1 and N_2, the of N_1 by N_2, denoted by $[N_1 \mapsto N_2]$, maps every term M to the term $M[N_1 \mapsto N_2]$ which is obtained by replacing all occurrences of N_1 in M by N_2.

2.2 Monoidal Theories

Equational theories are very useful for modeling the algebraic properties of the cryptographic primitives. Given a signature Σ, an equational theory E is a set of equations (i.e., a set of unordered pairs of terms in $\mathcal{T}(\Sigma, \mathcal{X})$). Given two terms M and N such that $M, N \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$, we write $M =_{\mathsf{E}} N$ if the equation $M = N$ is a consequence of E. In this paper, we are particularly interested in the class of monoidal theories introduced by W. Nutt [17]. It captures many theories with AC properties, which are known to be difficult to deal with.

Definition 1 (monoidal theory). *A* E Σ monoidal :

1. *T* Σ + - 0, Σ .
2. *T* + - 0. *T* $x + (y + z) = (x + y) + z$, $x + y = y + x$ $x + 0 = x$ E.
3. *E* $\mathsf{h} \in \Sigma$ + 0, . . $\mathsf{h}(x + y) = \mathsf{h}(x) + \mathsf{h}(y)$ $\mathsf{h}(0) = 0$.

Note that a monoidal theory on a given signature Σ may contain arbitrary additional equalities over Σ. The only requirement is, that at least the laws given above hold.

E *1.* Suppose + is a binary function symbol and 0 is nullary. Moreover assume that the others symbols, . . −, h, are unary symbols. The equational theories below are monoidal.

- The theory ACU over $\Sigma = \{+, 0\}$ which consists of the axioms of associativity and commutativity with unit 0.
- The theories ACUI and ACUN ( ) over $\Sigma = \{+, 0\}$ which consist of the axioms (AC) and (U) with in addition Idempotency (I) $x + x = x$, or Nilpotency (N) $x + x = 0$.
- The theory AG (*A*) over $\Sigma = \{+, -, 0\}$ which is generated by the axioms (AC), (U) and $x + -(x) = 0$ (Inv). Indeed, the equations $-(x + y) = -(x) + -(y)$ and $-0 = 0$ are consequences of the others.
- The theories ACUh, ACUIh, ACUNh over $\Sigma = \{+, \mathsf{h}, 0\}$ and AGh over $\Sigma = \{+, -, \mathsf{h}, 0\}$: these theories correspond to the ones described above extended by the homomorphism laws (h) for the symbol h, i.e., $\mathsf{h}(x + y) = \mathsf{h}(x) + \mathsf{h}(y)$ and $\mathsf{h}(0) = 0$ (if it is not a consequence of the other equations).

Note that there are two homomorphisms in the theory AGh, namely − and h. These two homomorphisms commute: $\mathsf{h}(-x) = -(\mathsf{h}(x))$ is a consequence of the others. Other examples of monoidal theories can be found in [17].

3 Deduction and Static Equivalence

We now describe our two notions of knowledge for an intruder.

3.1 Assembling Terms into Frames

At a particular point in time, while engaging in one or more sessions of one or more protocols, an attacker may know a sequence of messages $M_1, \ldots, M_\ell$. This means that he knows each message but he also knows in which order he obtained the messages. So it is not enough for us to say that the attacker knows the set of terms $\{M_1, \ldots, M_\ell\}$ since the information about the order is lost. Furthermore, we should distinguish those names that the attacker knows from those that were freshly generated by others and which are secret from the attacker; both kinds of names may appear in the terms. In the applied pi calculus [2], such a sequence of messages is organized into a $\phi = \nu\tilde{n}.\sigma$, where $\tilde{n}$ is a finite set of names (intuitively the fresh ones), and σ is a substitution of the form:

$$\{^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}\} \quad \text{with} \quad (\sigma) = \{x_1, \ldots, x_\ell\}.$$

The variables enable us to refer to each M_i and we always assume that the terms M_i are ground. The names $\tilde{n}$ are bound to ϕ and can be renamed. Moreover names that do not appear in the names of ϕ can be added or removed from $\tilde{n}$. In particular, we can always assume that two frames share the same set of restricted names.

3.2 Deduction

Given a frame ϕ that represents the information available to an attacker, we may ask whether a given ground term M may be deduced from ϕ. Given a theory E over Σ, this relation is written $\phi \vdash_{\mathsf{E}} M$ and is axiomatized by the rules:

$$\frac{}{\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M} \text{ if } \exists x \in dom(\sigma) \text{ s.t. } x\sigma = M \qquad\qquad \frac{}{\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} s}\ s \in \mathcal{N} \smallsetminus \tilde{n}$$

$$\frac{\phi \vdash_{\mathsf{E}} M_1 \quad \ldots \quad \phi \vdash_{\mathsf{E}} M_\ell}{\phi \vdash_{\mathsf{E}} f(M_1, \ldots, M_\ell)}\ f \in \Sigma \qquad\qquad \frac{\phi \vdash_{\mathsf{E}} M}{\phi \vdash_{\mathsf{E}} M'}\ M =_{\mathsf{E}} M'$$

Intuitively, the deducible messages are the messages of ϕ and the names that are not protected in ϕ, closed by equality in E and closed by application of function symbols. Since the deducible messages depend on the underlying equational theory, we write $\vdash_{\mathsf{E}}$ and simply $\vdash$ when E is clear from the context. When $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$, any occurrence of names from $\tilde{n}$ in M is bound by $\nu\tilde{n}$. So $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$ could be formally written $\nu\tilde{n}.(\sigma \vdash_{\mathsf{E}} M)$. It is easy to prove by induction the following characterization of deduction.

Lemma 1 (characterization of deduction). *L* M $\nu\tilde{n}.\sigma$. *T* $\nu\tilde{n}.\sigma \vdash_{\mathsf{E}} M$ $\zeta \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X})$ $(\zeta) \cap \tilde{n} = \emptyset$ $\zeta\sigma =_{\mathsf{E}} M$. *S* ζ recipe M.

E *2.* Consider $\Sigma = \{+, 0\}$ and the equational theory ACUN given in Example 1. Let $\phi = \nu n_1, n_2, n_3.\{^{n_1+n_2+n_3}/_{x_1}, {}^{n_1+n_2}/_{x_2}, {}^{n_2+n_3}/_{x_3}\}$. We have that $\phi \vdash n_2 + n_4$. Indeed $x_1 + x_2 + x_3 + n_4$ is a recipe of the term $n_2 + n_4$.

Deduction problem for the equational theory E built over Σ.

E : A frame ϕ and a term M (both built over Σ)
Q : $\phi \vdash_{\mathsf{E}} M$?

3.3 Static Equivalence

Deduction does not always suffice for expressing the knowledge of an attacker. Sometimes, the attacker can deduce exactly the same set of terms from two different frames but he could still be able to tell the difference between these two frames. Static equivalence is particularly important when defining for example the confidentiality of a vote or anonymity-like properties.

Definition 2 (static equivalence). *L* ϕ M,N .
W M N equal in ϕ E, $(M =_{\mathsf{E}} N)\phi$,
$\tilde{n}$ $\phi = \nu\tilde{n}.\sigma$, ($(M) \cup$ $(N)) \cap \tilde{n} = \emptyset$ $M\sigma =_{\mathsf{E}} N\sigma$.
W $\phi_1 = \nu\tilde{n}.\sigma_1$ $\phi_2 = \nu\tilde{n}.\sigma_2$ statically equivalent
. . . E, $\phi_1 \approx_{\mathsf{E}} \phi_2$ $(\phi_1) =$ (ϕ_2),

$$\forall M, N \in \mathcal{T}(\Sigma, \mathcal{N} \cup \mathcal{X}) \qquad (M =_{\mathsf{E}} N)\phi_1 \Leftrightarrow (M =_{\mathsf{E}} N)\phi_2.$$

E *3.* Consider the equational theory ACU given in Example 1 and let $\phi = \nu n_1, n_2, n_3.\{^{3n_1+2n_2+3n_3}/_{x_1}, {}^{n_2+3n_3}/_{x_2}, {}^{3n_2+n_3}/_{x_3}, {}^{3n_1+n_2+4n_3}/_{x_4}\}$ where the notation kn with $k \in \mathbb{N}$ denotes $n + \cdots + n$ (k times). Let $M = 2x_1 + x_2$ and $N = x_3 + 2x_4$. We have that $(M =_{\mathsf{E}} N)\phi$.

Static equivalence problem for the equational theory E built over Σ.

E : Two frames ϕ_1 and ϕ_2 (both built over Σ)
Q : $\phi_1 \approx_{\mathsf{E}} \phi_2$?

In what follows, we consider decidability and complexity issues for deduction and static equivalence for monoidal theories.

4 Monoidal Theories

It has been shown that the deduction problem for ACU amounts to solving linear equations over the semiring $\mathbb{N}$ whereas for AGh this problem amounts to solving linear equations over the ring $\mathbb{Z}[\mathsf{h}]$, the ring of polynomials in one indeterminate with coefficients over $\mathbb{Z}$ [11]. Some results of this kind also exist in the case of static equivalence. For instance, static equivalence has been shown decidable for the equational theories ACUN and AC [1]. By using an algebraic characterization of the problem, we will generalize these results by associating to every monoidal theory E a semiring $\mathcal{S}_{\mathsf{E}}$, that will be used to solve the deduction and the static equivalence problems in E.

4.1 Monoidal Theories Define Semirings

Monoidal theories have an algebraic structure close to rings except that elements might not have an inverse. Such a structure is called a .

Definition 3 (semiring). *A* semiring $\mathcal{S}$ *(* universe
) 0 1 -
$+$ $\cdot$ $(\mathcal{S}, +, 0)$, $(\mathcal{S}, \cdot, 1)$,
$\alpha, \beta, \gamma \in \mathcal{S}$*:*

- $(\alpha+\beta)\cdot\gamma = \alpha\cdot\gamma+\beta\cdot\gamma$ ( )
- $\alpha\cdot(\beta+\gamma) = \alpha\cdot\beta+\alpha\cdot\gamma$ ( )
- $0\cdot\alpha = \alpha\cdot 0 = 0$ ( ).

We call the binary operations + and · respectively the and the - of the semiring. The elements 0 and 1 are called respectively and . A semiring is if its multiplication is commutative. Semirings are different from rings in that they need not be groups with respect to addition. Every ring is a semiring. In a ring, we will denote by $-\alpha$ the additive inverse of α.

It has been shown in [17] that for any monoidal theory E there exists a corresponding semiring $\mathcal{S}_{\mathsf{E}}$. We can rephrase the definition of $\mathcal{S}_{\mathsf{E}}$ as follows. Let $\mathbf{1}$ be a free constant ($\mathbf{1} \notin \Sigma$), the universe of $\mathcal{S}_{\mathsf{E}}$ is $\mathcal{T}(\Sigma,\{\mathbf{1}\})/\mathsf{E}$, that is the set of equivalence classes of terms built over Σ and $\mathbf{1}$ under equivalence by the equational axioms E. The constant 0 and the sum + of the semiring are defined as in the algebra $\mathcal{T}(\Sigma,\{\mathbf{1}\})/\mathsf{E}$. The multiplication in the semiring is defined by $M \cdot T := M[\mathbf{1} \mapsto T]$. Recall that $M[\mathbf{1} \mapsto T]$ denotes the term M where any occurrence of $\mathbf{1}$ has been replaced by T. As a consequence, $\mathbf{1}$ acts as a neutral element of multiplication in $\mathcal{S}_{\mathsf{E}}$. This is the reason why we call this new generator $\mathbf{1}$ instead of, say, x, as it is often done in the literature. It can be shown [17] that $\mathcal{S}_{\mathsf{E}}$ is a ring if, and only if, E is a group theory, and also that $\mathcal{S}_{\mathsf{E}}$ is commutative if, and only if, E has commuting homomorphisms, i.e., $\mathsf{h}_1(\mathsf{h}_2(x)) =_{\mathsf{E}} \mathsf{h}_2(\mathsf{h}_1(x))$ for any two homomorphisms h_1 and h_2. For instance, we have that

1. The semiring $\mathcal{S}_{\mathsf{ACU}}$ is isomorphic to $\mathbb{N}$, the semiring of natural numbers.
2. The semiring $\mathcal{S}_{\mathsf{ACUN}}$ consists of the two elements 0 and $\mathbf{1}$ and we have $0+\mathbf{1} = \mathbf{1}+0 = \mathbf{1}$, $0+0 = \mathbf{1}+\mathbf{1} = 0$, $0\cdot 0 = \mathbf{1}\cdot 0 = 0\cdot\mathbf{1} = 0$, and $\mathbf{1}\cdot\mathbf{1} = \mathbf{1}$. Hence, $\mathcal{S}_{\mathsf{ACUN}}$ is isomorphic to the commutative ring (field) $\mathbb{Z}/2\mathbb{Z}$.
3. The semiring $\mathcal{S}_{\mathsf{AGh}}$ is isomorphic to $\mathbb{Z}[\mathsf{h}]$ which is a commutative ring.

Let b be a free symbol (name or variable). We denote by $\psi_b \colon \mathcal{T}(\Sigma,\{b\}) \to \mathcal{S}_{\mathsf{E}}$ the function which maps any term $M \in \mathcal{T}(\Sigma,\{b\})$ to $M[b \mapsto \mathbf{1}]$ considered as an element of the semiring $\mathcal{S}_{\mathsf{E}}$.

E *4.* Let $\mathsf{E} = \mathsf{ACUN}$ and $t = b+b+b$. We have $\psi_b(t) = \mathbf{1}+\mathbf{1}+\mathbf{1} = \mathbf{1}$.

4.2 Representation of Terms and Frames

A $\mathcal{B}$ is a sequence $[b_1,\ldots,b_m]$ of free symbols (names or variables). We say that $\mathcal{B}$ is a when $b_1,\ldots,b_m$ are names.

Definition 4 (decomposable in a base). *A* $M \in \mathcal{T}(\Sigma,\mathcal{N}\cup\mathcal{X})$ decomposable $\mathcal{B}$ $(M) \cup$ $(M) \subseteq \mathcal{B}$. *L* $\phi = \nu\tilde{n}.\{^{M_1}/_{x_1},\ldots,{}^{M_\ell}/_{x_\ell}\}$. *W* ϕ decomposable $\mathcal{B}$ M_i $\mathcal{B}$.

Let $\mathcal{B} = [b_1,\ldots,b_m]$. We generalize the construction of the previous section and obtain a function which assigns to any term in $\mathcal{T}(\Sigma,\mathcal{B})$ a tuple in $\mathcal{S}_{\mathsf{E}}^m$, that is a tuple of m elements over $\mathcal{S}_{\mathsf{E}}$. The function $\psi_{\mathcal{B}} \colon \mathcal{T}(\Sigma,\{b_1,\ldots,b_m\}) \to \mathcal{S}_{\mathsf{E}}^m$ is defined as follows: Any term $M \in \mathcal{T}(\Sigma, \{b_1, \ldots, b_m\})$ has a unique decomposition $M_1, \ldots, M_m$ such that $M = M_1 + \ldots + M_m$ with $M_i \in \mathcal{T}(\Sigma, \{b_i\})$ [17]. We define $\psi_{\mathcal{B}}(M) = (\psi_{b_1}(M_1), \ldots, \psi_{b_m}(M_m))$. Given a vector $X \in \mathcal{S}_{\mathsf{E}}^m$ of size m, $\psi_{\mathcal{B}}^{-1}(X)$ is a term $M \in \mathcal{T}(\Sigma, \mathcal{B})$ such that $\psi_{\mathcal{B}}(M) = X$. This term is uniquely defined modulo E.

E *5.* Taking into account that the semiring $\mathcal{S}_{\mathsf{AGh}}$ is (isomorphic to) $\mathbb{Z}[\mathsf{h}]$, we have that $\psi_{[b_1,b_2,b_3]}(b_1 + b_1 + \mathsf{h}(b_3) + \mathsf{h}(\mathsf{h}(\mathsf{h}(b_3)))) = (2, 0, \mathsf{h} + \mathsf{h}^3)$. Indeed, we have that $\psi_{b_1}(b_1 + b_1) = 2$, $\psi_{b_2}(0) = 0$ and $\psi_{b_3}(\mathsf{h}(b_3) + \mathsf{h}(\mathsf{h}(\mathsf{h}(b_3)))) = \mathsf{h} + \mathsf{h}^3$.

A term can be uniquely decomposed on a base $\mathcal{B}$. This can be extended to associate a (unique) matrix to a frame. Let $\phi = \nu\tilde{n}.\sigma$ be a frame and $\mathcal{B} = [b_1, \ldots, b_m]$ be a base of names in which ϕ is decomposable. Let $\sigma = \{^{M_1}/_{x_1} \ldots {}^{M_\ell}/_{x_\ell}\}$. We denote by $\psi_{\mathcal{B}}(\phi)$ the matrix of size $\ell \times m$ (ℓ rows and m columns) defined by $(\psi_{\mathcal{B}}(M_1); \ldots; \psi_{\mathcal{B}}(M_\ell))$. This matrix is the decomposition of ϕ in $\mathcal{B}$.

E *6.* Consider the frame ϕ given in Example 3 and let $\mathcal{B} = [n_1, n_2, n_3]$. We have that

$$\psi_{\mathcal{B}}(\phi) = \begin{pmatrix} 3 & 2 & 3 \\ 0 & 1 & 3 \\ 0 & 3 & 1 \\ 3 & 1 & 4 \end{pmatrix} \quad \text{since} \quad \begin{array}{l} -\ \psi_{\mathcal{B}}(3n_1 + 2n_2 + 3n_3) = (3, 2, 3), \\ -\ \psi_{\mathcal{B}}(n_2 + 3n_3) = (0, 1, 3), \\ -\ \psi_{\mathcal{B}}(3n_2 + n_3) = (0, 3, 1), \text{ and} \\ -\ \psi_{\mathcal{B}}(3n_1 + n_2 + 4n_3) = (3, 1, 4). \end{array}$$

Applying a recipe to a frame is equivalent to multiplying the corresponding matrices.

Lemma 2. *L* $\phi = \nu\tilde{n}.\sigma$ ζ $\mathcal{T}(\Sigma,$ $(\phi))$. *L* $\mathcal{B}$ ϕ. *W*

$$\psi_{\mathcal{B}}(\zeta\sigma) = \psi_{dom(\phi)}(\zeta) \cdot \psi_{\mathcal{B}}(\phi).$$

Note that to apply the equation stated in Lemma 2, the recipe ζ has to be built without names. To ensure that such kind of recipes always exist, we will work with frames saturated w.r.t. $\mathcal{B}$ (base of names in which the frames are decomposable).

Definition 5 (frame saturated w.r.t. $\mathcal{B}$). *L* $\phi = \nu\tilde{n}.\sigma$ $\mathcal{B}$ $[b_1, \ldots, b_m]$ ϕ . *W* ϕ saturated . . . $\mathcal{B}$ $b_i \in \mathcal{B}$ $b_i \notin \tilde{n}$ $b_i = x\sigma$ $x \in$ (ϕ).

Given a frame $\phi = \nu\tilde{n}.\{^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}\}$ and a base of names $\mathcal{B} = [b_1, \ldots, b_k]$ in which ϕ is decomposable, we denote by $\overline{\phi}^{\mathcal{B}}$ the frame defined as follows:

$$\overline{\phi}^{\mathcal{B}} = \nu\tilde{n}.\{^{M_1}/_{x_1}, \ldots, {}^{M_\ell}/_{x_\ell}, {}^{b_{i_1}}/_{y_1}, \ldots, {}^{b_{i_p}}/_{y_p}\}$$

where $b_{i_1}, \ldots, b_{i_p}$ is a subsequence of $\mathcal{B}$ such that $b_{i_j} \notin \tilde{n}$ and $b_{i_j} \neq x\sigma$ for every $x \in$ (ϕ). The variables $y_1, \ldots y_p$ are fresh, which means that they do not appear in (ϕ). Note that the resulting frame $\overline{\phi}^{\mathcal{B}}$ is saturated w.r.t. $\mathcal{B}$.

E *7.* Let ϕ be the frame given in Example 3. Let $\mathcal{B} = [n_1, n_2, n_3]$. We have that ϕ is decomposable on $\mathcal{B}$ and also that ϕ is saturated w.r.t. $\mathcal{B}$. However, note that ϕ is not saturated w.r.t. $\mathcal{B}' = [n_1, n_2, n_3, n_4]$. We have that $\overline{\phi}^{\mathcal{B}'} = \nu n_1, n_2, n_3.\{^{3n_1+2n_2+3n_3}/_{x_1}, {}^{n_2+3n_3}/_{x_2}, {}^{3n_2+n_3}/_{x_3}, {}^{3n_1+n_2+4n_3}/_{x_4}, {}^{n_4}/_{y_1}\}$.

5 Deduction

We show that solving a deduction problem can be reduced to solving a linear system of equations in the corresponding semiring.

Theorem 1. *L* E $\mathcal{S}_\mathsf{E}$. *D* E :

Entries*: A* A $\mathcal{S}_\mathsf{E}$ $\ell \times m$ b $\mathcal{S}_\mathsf{E}$ ℓ
Question*: D* X *(* $\mathcal{S}_\mathsf{E}$ ℓ*)* $X \cdot A = b$*?*

Note that when $\mathcal{S}_\mathsf{E}$ is commutative, this problem is equivalent to the problem of deciding whether $A^\mathsf{T} \cdot Y = b^\mathsf{T}$, i.e., whether b^T is in the image of A^T where M^T is the transpose of M. Before proving the reduction we need to establish that we can restrict our attention to saturated frames. Moreover, for such frames, it is sufficient to consider recipes without names, i.e., such that $(\zeta) = \emptyset$.

Lemma 3. *L* $\phi = \nu\tilde{n}.\sigma$ M *. L* $\mathcal{B}$ ϕ M *. W* $\phi \vdash_\mathsf{E} M$ $\overline{\phi}^{\mathcal{B}} \vdash_\mathsf{E} M$*. M* $\overline{\phi}^{\mathcal{B}} \vdash_\mathsf{E} M$ ζ M $(\zeta) = \emptyset$*.*

R *.* Let $\phi = \nu\tilde{n}.\sigma$ be a frame and M be a ground term. Let $\mathcal{B}$ be a base of names in which ϕ and M are decomposable. We will also assume w.l.o.g. that ϕ is saturated w.r.t. $\mathcal{B}$. Let $A = \psi_\mathcal{B}(\phi)$, matrix of size $\ell \times m$ over $\mathcal{S}_\mathsf{E}$, and $b = \psi_\mathcal{B}(M)$, vector of size m over $\mathcal{S}_\mathsf{E}$.

P *.* (of Theorem 1) The construction described above is such that $X \cdot A = b$ has a solution over $\mathcal{S}_\mathsf{E}$ if and only if $\phi \vdash_\mathsf{E} M$.
($\Rightarrow$) We know that there exists $X \in \mathcal{S}_\mathsf{E}^\ell$ such that $X \cdot A = b$. Consider the recipe $\zeta = \psi^{-1}_{dom(\phi)}(X)$. By construction, we have that $(\zeta) \cap \tilde{n} = \emptyset$. It remains to show that $\zeta\sigma =_\mathsf{E} M$. For this, we establish that $\psi_\mathcal{B}(\zeta\sigma) = \psi_\mathcal{B}(M)$. Thanks to Lemma 2, we have that $\psi_\mathcal{B}(\zeta\sigma) = \psi_{dom(\phi)}(\zeta) \cdot \psi_\mathcal{B}(\phi)$. Hence we deduce that $\psi_\mathcal{B}(\zeta\sigma) = X \cdot A = b = \psi_\mathcal{B}(M)$. Hence the result.

($\Leftarrow$) Assume that $\phi \vdash_\mathsf{E} M$. Thanks to Lemma 3 and by the fact that ϕ is saturated w.r.t. $\mathcal{B}$, we know that there exists $\zeta \in \mathcal{T}(\Sigma, (\phi))$ such that $\zeta\sigma =_\mathsf{E} M$. Let $Y = \psi_{dom(\phi)}(\zeta)$. It remains to establish that $Y \cdot A = b$. Since $\zeta\sigma =_\mathsf{E} M$, we have $\psi_\mathcal{B}(\zeta\sigma) = \psi_\mathcal{B}(M)$. By Lemma 2, we have $\psi_{dom(\phi)}(\zeta) \cdot \psi_\mathcal{B}(\phi) = \psi_\mathcal{B}(M)$, i.e., $Y \cdot A = b$ witnessing the fact that $X \cdot A = b$ has a solution over $\mathcal{S}_\mathsf{E}$. $\square$

E *8.* Consider the theory ACUNh and the term $M = n_1 + \mathsf{h}(\mathsf{h}(n_1))$. Let $\phi = \nu n_1, n_2.\{^{n_1+\mathsf{h}(n_1)+\mathsf{h}(\mathsf{h}(n_1))}/_{x_1}, {}^{n_2+\mathsf{h}(\mathsf{h}(n_1))}/_{x_2}, {}^{\mathsf{h}(n_2)+\mathsf{h}(\mathsf{h}(n_1))}/_{x_3}\}$. We have:

$$A = \begin{pmatrix} 1+\mathsf{h}+\mathsf{h}^2 & \mathsf{h}^2 & \mathsf{h}^2 \\ 0 & 1 & \mathsf{h} \end{pmatrix} \text{ and } b = \begin{pmatrix} 1+\mathsf{h}^2 \\ 0 \end{pmatrix}$$

The equation $X \cdot A = b$ has a solution over $\mathbb{Z}/2\mathbb{Z}[\mathsf{h}] : (1+\mathsf{h}, \mathsf{h}, 1)$. The term M is deducible from ϕ by using the recipe $x_1 + \mathsf{h}(x_1) + \mathsf{h}(x_2) + x_3$.

As a consequence, decidability/complexity results for deduction can be deduced from decidability/complexity results for solving linear system of equations (see Section 7).

6 Static Equivalence

We show that deciding whether two frames are equivalent can be reduced to deciding whether two matrices satisfy the same set of equalities.

Theorem 2. *L* E $\mathcal{S}_\mathsf{E}$. *S* E :

Entries*: T* A_1 A_2 $\mathcal{S}_\mathsf{E}$ $\ell \times m$
Question*: D* *?*

$$\{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_1 = Y \cdot A_1\} = \{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_2 = Y \cdot A_2\}$$

Similarly to deduction, we first show that we can restrict our attention to saturated frames. Moreover, we show that it is sufficient to consider recipes, i.e., tests (M, N), without names.

Lemma 4. *L* $\phi_1 = \nu\tilde{n}.\sigma_1, \phi_2 = \nu\tilde{n}.\sigma_2$. $\mathcal{B}$ ϕ_1 ϕ_2 . *W* $\phi_1 \approx_\mathsf{E} \phi_2$ $\overline{\phi_1}^{\mathcal{B}} \approx_\mathsf{E} \overline{\phi_2}^{\mathcal{B}}$. *M* , $\overline{\phi_1}^{\mathcal{B}} \not\approx_\mathsf{E} \overline{\phi_2}^{\mathcal{B}}$ $M, N \in \mathcal{T}(\Sigma,$ $(\overline{\phi_1}^{\mathcal{B}}))$ $(M =_\mathsf{E} N)\overline{\phi_1}^{\mathcal{B}} \not\Leftrightarrow (M =_\mathsf{E} N)\overline{\phi_2}^{\mathcal{B}}$.

R . Let $\phi_1 = \nu\tilde{n}.\sigma_1$ and $\phi_2 = \nu\tilde{n}.\sigma_2$ be two frames having the same domain. Let $\mathcal{B}$ be a base of names in which the two frames are decomposable. We assume w.l.o.g. that ϕ_1 and ϕ_2 are saturated w.r.t. $\mathcal{B}$. Let $m = |\mathcal{B}|$. Let $A_1 = \psi_\mathcal{B}(\phi_1)$ and $A_2 = \psi_\mathcal{B}(\phi_2)$, two matrices of size $\ell \times m$, over $\mathcal{S}_\mathsf{E}$.

P . (of Theorem 2) The construction is such that $\phi_1 \approx_\mathsf{E} \phi_2$ if and only if

$$\{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_1 = Y \cdot A_1\} = \{(X,Y) \in \mathcal{S}_\mathsf{E}^\ell \times \mathcal{S}_\mathsf{E}^\ell \mid X \cdot A_2 = Y \cdot A_2\}.$$

($\Rightarrow$) Assume by contradiction that there exists (X_M, X_N) such that $X_M \cdot A_1 = X_N \cdot A_1$ and $X_M \cdot A_2 \neq X_N \cdot A_2$ (or the converse). Let $M = \psi^{-1}_{dom(\phi_1)}(X_M)$ and $N = \psi^{-1}_{dom(\phi_1)}(X_N)$. We have that

- $(M =_{\mathsf{E}} N)\phi_1$. For this, it is sufficient to show that $\psi_{\mathcal{B}}(M\sigma_1) = \psi_{\mathcal{B}}(N\sigma_1)$, i.e., $\psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1) = \psi_{dom(\phi_1)}(N) \cdot \psi_{\mathcal{B}}(\phi_1)$ thanks to Lemma 2. Now to conclude, it is sufficient to notice that we have $X_M = \psi_{dom(\phi_1)}(M)$, $X_N = \psi_{dom(\phi_1)}(N)$ and $A_1 = \psi_{\mathcal{B}}(\phi_1)$ and to rely on the hypothesis.
- $(M \neq_{\mathsf{E}} N)\phi_2$ can be shown similarly.

($\Leftarrow$) Assume that $\phi_1 \not\approx_{\mathsf{E}} \phi_2$. We have that there exists a test (M, N) such that $(M =_{\mathsf{E}} N)\phi_1$ and $(M \neq_{\mathsf{E}} N)\phi_2$ (or the converse). Thanks to Lemma 4 and the fact that the frames are saturated, we can assume that $M, N \in \mathcal{T}(\Sigma, \quad (\phi_1))$. Let $X_M = \psi_{dom(\phi_1)}(M)$ and $X_N = \psi_{dom(\phi_1)}(N)$. We have

- $X_M \cdot A_1 = X_N \cdot A_1$. We have $M\sigma_1 =_{\mathsf{E}} N\sigma_1$, hence $\psi_B(M\sigma_1) = \psi_{\mathcal{B}}(N\sigma_1)$. By Lemma 2, we have that $\psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1) = \psi_{dom(\phi_1)}(M) \cdot \psi_{\mathcal{B}}(\phi_1)$, i.e., $X_M \cdot A_1 = X_N \cdot A_1$.
- $X \cdot A_2 \neq Y \cdot A_2$ can be established in a similar way. □

Going further. Thanks to Theorem 2, we give a way to decide static equivalence in monoidal equational theories provided we can decide whether two sets of linear equations over $\mathcal{S}_{\mathsf{E}}$ have the same set of solutions. Actually, when $\mathcal{S}_{\mathsf{E}}$ is a ring or when we can extend the semiring $\mathcal{S}_{\mathsf{E}}$ into a ring $\mathcal{R}_{\mathsf{E}}$, the static equivalence problem is equivalent to the problem of deciding whether the following equality holds.

$$\{Z \in \mathcal{R}_{\mathsf{E}}^{\ell} \mid Z \cdot A_1 = 0\} = \{Z \in \mathcal{R}_{\mathsf{E}}^{\ell} \mid Z \cdot A_2 = 0\}$$

When $\mathcal{R}_{\mathsf{E}}$ is commutative, it is equivalent to deciding whether $\mathsf{Ker}(A_1) = \mathsf{Ker}(A_2)$, where $\mathsf{Ker}(M)$ denotes the kernel of the matrices M, i.e., the set $\{X \mid M \cdot X = 0\}$. The ring associated to a given monoidal theory E, denoted by $\mathcal{R}_{\mathsf{E}}$, is equal to $\mathcal{S}_{\mathsf{E}}$ when E is a group theory. Otherwise, it might be possible to extend the equational theory E with a new unary symbol $-$ and the law $x + -(x) = 0$ in order to obtain a theory E' that is consistent with E, i.e., for all $u, v \in \mathcal{S}_{\mathsf{E}}$ such that $u =_{\mathsf{E}'} v$, we have also that $u =_{\mathsf{E}} v$. In such a case, the ring $\mathcal{R}_{\mathsf{E}}$ is the semiring $\mathcal{S}_{\mathsf{E}'}$ associated to E' as explained in Section 4.1.

Example 9. We have seen that the semiring associated to AG is isomorphic to $\mathbb{Z}$ which is a commutative ring. Hence, we have that $\mathcal{R}_{\mathsf{E}}$ is isomorphic to $\mathbb{Z}$. The associated semiring to the monoidal equational theory ACU is isomorphic to $\mathbb{N}$ whereas its associated ring is $\mathbb{Z}$.

Note that the transformation described above does not allow us to associate a ring to any semiring. For instance, if we consider the theory ACUI and the theory E' obtained by the transformation described above, we have that $0 =_{\mathsf{E}'} (\mathbf{1} + \mathbf{1}) + -(\mathbf{1}) =_{\mathsf{E}'} \mathbf{1} + (\mathbf{1} + -(\mathbf{1})) =_{\mathsf{E}'} \mathbf{1}$ whereas this equality does not hold in ACUI.

7 Applications and Discussion

In this section we show that several interesting monoidal equational theories induce a ring or a semiring for which solving linear systems or checking for equalities of sets of solutions of linear systems are decidable. A summary is given in Figure 1. Note that any of these decidability results for deduction and static equivalence can be combined with any existing ones provided the signatures of the equational theories are disjoint [3]. For example, let E be a monoidal equational theory for which deduction and static equivalence are decidable (e.g., ACU, ACUNh, ...) then deduction and static equivalence are also decidable for the theory $\mathsf{E}_{\mathsf{enc}} \cup \mathsf{E}$ where $\mathsf{E}_{\mathsf{enc}}$ is defined by the following equations:

$$\mathsf{dec}(\mathsf{enc}(x, y), y) = x, \quad \mathsf{proj}_1(\langle x, y\rangle) = x \text{ and } \mathsf{proj}_2(\langle x, y\rangle) = y.$$

Theory ACU. This equational theory is the simplest monoidal theory. The semiring corresponding to this theory is $\mathbb{N}$ whereas its associated ring is $\mathbb{Z}$. This equational theory has been particularly studied. Since the problem of solving linear equations over $\mathbb{N}$ is strongly NP-complete, we obtain that deduction is a NP-complete problem. The problem of static equivalence for this theory has been shown decidable in [1]. Actually thanks to the algebraic characterization given in this paper, this problem can be solved in polynomial time [20].

At first sight, it might seem surprising since it has been shown [1] that deduction in a given theory E can be reduced in polynomial time to static equivalence in E. However, this reduction required the presence of a free function symbol and such a function symbol is not available in the theory ACU. Hence, the polynomial reduction provided in [1] does not apply in this setting.

Theories ACUI and ACUN (Exclusive Or). The semirings corresponding to these equational theories are respectively the Boolean semiring $\mathbb{B}$, which is finite, and the finite field $\mathbb{Z}/2\mathbb{Z}$. The theory ACUN has already been studied in terms of deduction [10,8] and static equivalence [1]. Deduction and static equivalence are both decidable in polynomial time. As far as we know the theory ACUI has only been studied in term of deduction [12]. Actually, since its associated semiring is finite, we easily deduce that deduction and static equivalence are decidable.

Theory AG (Abelian Groups). The semiring associated to this equational theory is in fact a ring, namely the ring $\mathbb{Z}$ of all integers. There exist several algorithms to compute solutions of linear equations over $\mathbb{Z}$ and to compute a base of the set of solutions (see for instance [20]). Hence, we easily deduce that both problems are decidable in PTIME. Deduction for this theory has already been studied in [10] and [7].

Theories ACUh, ACUNh and AGh. The semiring associated to ACUh is $\mathbb{N}[\mathsf{h}]$, the semiring of polynomial in one indeterminate over $\mathbb{N}$ whereas the ring associated to ACUh is $\mathbb{Z}[\mathsf{h}]$. For the theory ACUNh (resp. AGh) the associated semiring is $\mathbb{Z}/2\mathbb{Z}[\mathsf{h}]$ (resp. $Z[\mathsf{h}]$). Deduction for these three equational theories has already been studied in [13,11]. However, results obtained on static equivalence are new.

1. ACUh and AGh: Deciding static equivalence for both these theories is reducible to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$ where A and B

are matrices built over $\mathsf{N}[\mathsf{h}]$ in the case of ACUh and $\mathsf{Z}[\mathsf{h}]$ in the case of AGh. This problem has been solved by F. Baader to obtain a unification algorithm for the theory AGh (see [4]). This is done by the help of Gröbner Base methods in a more general settings. Actually, he provides an algorithm even in the case of several commutating homomorphisms.

2. ACUNh: Deciding static equivalence in ACUNh is reducible to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$ where A and B are matrices built over $\mathsf{Z}/2\mathsf{Z}[\mathsf{h}]$. This is achieved in [14] with an automata-theoretic approach.

Theory ACUIh. The semiring associated to ACUIh is $\mathbb{B}[\mathsf{h}]$. Deduction for this theory has never been studied but is clearly decidable. Indeed, to find a solution to $A \cdot X = b$, it is easy to see that each component of a solution to $A \cdot X = b$ has a degree smaller than the degree of b. Hence, the question of deciding whether there exists X such that $A \cdot X = b$ can be reduced to solving a system of linear equations over $\mathbb{B}$. Theorem 2 does not help us to provide an algorithm to solve static equivalence. Note also that we cannot reduce the problem to the problem of deciding whether $\mathsf{Ker}(A) = \mathsf{Ker}(B)$ since, as for ACUI, we are not able to associate a ring to this theory.

Adding more equations. A monoidal theory on a signature Σ may contain arbitrary additional equalities over Σ. Hence, the techniques developed in Section 5 and 6 can be applied to many different theories.

Example 10. Consider the theory E_1 over $\Sigma_1 = \{+, 0, -, \mathsf{h}\}$ which consists of the equalities of AGh and the additional equality $\mathsf{h}(\mathsf{h}(x)) = x$ which states that h is an involution. The theory E_1 is a monoidal theory and its associated semiring $\mathcal{S}_{\mathsf{E}_1}$ that is actually a ring is isomorphic to $\mathbb{Z}[\mathsf{h}]/_{(\mathsf{h}^2-1)}$, i.e., the ring $\mathbb{Z}[\mathsf{h}]$ quotiented by the ideal generated by the polynomial $\mathsf{h}^2 - 1$.

We can also consider more complex equational theories by simply associating each equation to a polynomial. This is illustrated in the next example.

Example 11. Consider the signature $\Sigma_2 = \{+, 0, -, \mathsf{h}_1, \mathsf{h}_2\}$ and the theory E_2 made up of the axioms of AG extending by $\mathsf{h}_1(\mathsf{h}_2(x)) = \mathsf{h}_2(\mathsf{h}_1(x))$ and the following laws:

$$\begin{array}{lll} \mathsf{h}_1(x+y) = \mathsf{h}_1(x) + \mathsf{h}_1(y) & \mathsf{h}_1(0) = 0 & \mathsf{h}_1(\mathsf{h}_1(\mathsf{h}_2(x))) + \mathsf{h}_2(\mathsf{h}_2(x)) = 0 \\ \mathsf{h}_2(x+y) = \mathsf{h}_2(x) + \mathsf{h}_2(y) & \mathsf{h}_2(0) = 0 & \mathsf{h}_1(x) + \mathsf{h}_1(\mathsf{h}_2(\mathsf{h}_2(x))) = 0 \end{array}$$

The theory E_2 is a monoidal theory and it is easy to see that its associated semiring $\mathcal{S}_{\mathsf{E}_2}$ is isomorphic to $\mathbb{Z}[\mathsf{h}_1, \mathsf{h}_2]/_{(\mathsf{h}_1^2\mathsf{h}_2+\mathsf{h}_2^2, \mathsf{h}_1+\mathsf{h}_1\mathsf{h}_2^2)}$, i.e., the ring $\mathbb{Z}[\mathsf{h}]$ quotiented by the ideal generated by the polynomials $\mathsf{h}_1^2\mathsf{h}_2 + \mathsf{h}_2^2$ and $\mathsf{h}_1 + \mathsf{h}_1\mathsf{h}_2^2$.

Thus decidability of deduction and static equivalence can be reduced to solving linear equations in the corresponding semiring and deciding the equalities of kernels of matrices in the corresponding ring. Hence, we can reduced our problems to rather classical problems of Algebra, which can often be solved using Gröbner basis for example.

Theory E	$\mathcal{S}_\mathsf{E}$	$\mathcal{R}_\mathsf{E}$	Deduction	Static Equivalence
ACU	$\mathbb{N}$	$\mathbb{Z}$	NP-complete	decidable [1], PTIME (*new*)
ACUI	$\mathbb{B}$	–	decidable [12]	decidable (*new*)
ACUN	$\mathbb{Z}/2\mathbb{Z}$		PTIME [8]	decidable [1], PTIME (*new*)
AG	$\mathbb{Z}$		PTIME [7]	PTIME (*new*)
ACUh	$\mathbb{N}[\mathsf{h}]$	$\mathbb{Z}[\mathsf{h}]$	NP-complete [13]	decidable (*new*)
ACUIh	$\mathbb{B}[\mathsf{h}]$	–	decidable (*new*)	?
ACUNh	$\mathbb{Z}/2\mathbb{Z}[\mathsf{h}]$		PTIME [11]	decidable (*new*)
AGh	$\mathbb{Z}[\mathsf{h}]$		PTIME [11]	decidable (*new*)
$\mathsf{AGh}_1 \ldots \mathsf{h}_n$	$\mathbb{Z}[\mathsf{h}_1, \ldots, \mathsf{h}_n]$		decidable (*new*)	decidable (*new*)

Fig. 1. Summary of the results

8 Conclusion

We have proposed a general schema for deciding deduction and static equivalence algorithms. This schema has to be filled with procedures for linear equations in order to yield complete algorithms. Such algorithms strongly depend on the structure of the semiring. In this paper, we have mentioned and used several existing results of Algebra. But Algebra can still provide useful techniques that allow us to deduce some new results. Moreover, efficient existing tools for solving algebraic problems can also be used to implement our algorithms.

Acknowledgment. We wish to thank Jean-Charles Faugère, Daniel Lazard and Paul Zimmermann for fruitful discussions.

References

1. Abadi, M., Cortier, V.: Deciding knowledge in security protocols under equational theories. Theoretical Computer Science 387(1-2), 2–32 (2006)
2. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: POPL 2001. Proc. 28th ACM Symposium on Principles of Programming Languages, London (UK), pp. 104–115. ACM Press, New York (2001)
3. Arnaud, M., Cortier, V., Delaune, S.: Combining algorithms for deciding knowledge in security protocols. In: Konev, B., Wolter, F. (eds.) FroCoS 2007. LNCS (LNAI), vol. 4720, pp. 103–117. Springer, Heidelberg (2007)
4. Baader, F.: Unification in commutative theories, Hilbert's basis theorem, and Gröbner bases. Journal of the ACM 40(3), 477–503 (1993)
5. Baader, F., Nutt, W.: Combination problems for commutative/ monoidal theories or How algebra can help in equational unification. Applicable Algebra Engineering Communication and Computing 7(4), 309–337 (1996)

6. Baudet, M., Cortier, V., Kremer, S.: Computationally sound implementations of equational theories against passive adversaries. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 652–663. Springer, Heidelberg (2005)
7. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: Deciding the security of protocols with Diffie-Hellman exponentiation and product in exponents. In: Pandya, P.K., Radhakrishnan, J. (eds.) FST TCS 2003. LNCS, vol. 2914, pp. 124–135. Springer, Heidelberg (2003)
8. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: LICS 2003. Proc. 18th IEEE Symposium on Logic in Computer Science, Ottawa (Canada), pp. 261–270. IEEE Computer Society Press, Los Alamitos (2003)
9. Chevalier, Y., Rusinowitch, M.: Combining intruder theories. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 639–651. Springer, Heidelberg (2005)
10. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS 2003. Proc. 18th IEEE Symposium on Logic in Computer Science, Ottawa (Canada), pp. 271–280. IEEE Computer Society Press, Los Alamitos (2003)
11. Delaune, S.: Easy intruder deduction problems with homomorphisms. Information Processing Letters 97(6), 213–218 (2006)
12. Delaune, S., Lafourcade, P., Lugiez, D., Treinen, R.: Symbolic protocol analysis for monoidal equational theories. Information and Computation (to appear)
13. Lafourcade, P., Lugiez, D., Treinen, R.: Intruder deduction for AC-like equational theories with homomorphisms. In: Giesl, J. (ed.) RTA 2005. LNCS, vol. 3467, pp. 308–322. Springer, Heidelberg (2005)
14. Lafourcade, P., Lugiez, D., Treinen, R.: ACUNh: Unification and disunification using automata theory. In: UNIF 2006. Proc. 20th Int. Workshop on Unification, Seattle (Washington, USA), pp. 6–20 (2006)
15. Lowe, G.: Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In: Margaria, T., Steffen, B. (eds.) TACAS 1996. LNCS, vol. 1055, Springer, Heidelberg (1996)
16. Millen, J., Shmatikov, V.: Constraint solving for bounded-process cryptographic protocol analysis. In: CCS 2001. Proc. 8th ACM Conference on Computer and Communications Security, ACM Press, New York (2001)
17. Nutt, W.: Unification in monoidal theories. In: Stickel, M.E. (ed.) 10th International Conference on Automated Deduction. LNCS, vol. 449, pp. 618–632. Springer, Heidelberg (1990)
18. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
19. Rusinowitch, M., Turuani, M.: Protocol insecurity with a finite number of sessions, composed keys is NP-complete. Theoretical Computer Science 299(1-3), 451–475 (2003)
20. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)

Mechanized Verification of CPS Transformations

Zaynah Dargaye and Xavier Leroy

INRIA Paris-Rocquencourt
B.P. 105, 78153 Le Chesnay, France
`Zaynah.Dargaye@inria.fr`, `Xavier.Leroy@inria.fr`

Abstract. Transformation to continuation-passing style (CPS) is often performed by optimizing compilers for functional programming languages. As part of the development and proof of correctness of a compiler for the mini-ML functional language, we have mechanically verified the correctness of two CPS transformations for a call-by-value λ-calculus with n-ary functions, recursive functions, data types and pattern-matching. The transformations generalize Plotkin's original call-by-value transformation and Danvy and Nielsen's optimized transformation, respectively. We used the Coq proof assistant to formalize the transformations and conduct and check the proofs. Originalities of this work include the use of big-step operational semantics to avoid difficulties with administrative redexes, and of two-sorted de Bruijn indices to avoid difficulties with α-conversion.

1 Introduction

Continuation-passing style (CPS) is a programming style in the λ-calculus and related functional languages where a function never returns directly the result of its computations, but instead passes it to another function, the [illegible], received as an extra argument and representing the meaning of the rest of the program. For instance, the successor function, written $\lambda x.\ x + 1$ in direct style, becomes $\lambda x.\lambda k.\ k(x+1)$ in continuation-passing style, where k is the continuation parameter. Programs can be systematically translated to semantically equivalent programs in CPS using a variety of CPS transformation algorithms (see Sect. 2 for examples).

CPS and the related CPS transformations play an important role in three domains relevant to programming languages: semantics, programming, and compilation.

As a semantic device, CPS makes it possible to use the pure λ-calculus (without a fixed evaluation strategy) as a meta-language to describe faithfully the semantics of functional or imperative programming languages. After translation to CPS, the evaluation strategy of these languages is encoded in the structure of the resulting λ-term. Additionally, CPS makes it easy to give formal semantics to advanced control structures such as exceptions, backtracking, coroutines and control operators.

As a programming device, CPS enables functional programmers to define advanced, application-specific control structures such as coroutines or non-blind

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 211–225, 2007.
© Springer-Verlag Berlin Heidelberg 2007

backtracking. These control structures need not be supported natively by the programming language.

As a compilation device, programs in CPS lend themselves to aggressive optimizations that are significantly harder to perform on direct-style programs. CPS has several features that facilitate optimizations: all intermediate results are named, and compile-time β-reductions are always semantically valid. Several optimizing compilers for functional languages, such as Orbit Scheme [15], Standard ML of New Jersey [2], and SML.NET [13], use CPS as an intermediate language.

In this paper, we describe the formal verification, using the Coq proof assistant [7,4], of the correctness (semantic preservation) of two CPS transformations for a realistic, pure, call-by-value functional language. This language features n-ary functions, recursive functions, and ML/Haskell-style data types and pattern-matching. The two CPS transformations are extensions of Plotkin's original call-by-value transformation [22] and Danvy and Nielsen's optimized transformation [8,9], respectively.

This work is part of a larger project that aims at mechanically verifying the correctness of a whole compiler for mini-ML, a pure, call-by-value functional language rich enough to be used as a target language for automatic extraction of functional programs from Coq specifications [19]. In a context where formal methods are increasingly being applied to critical software, it becomes important to guarantee that compilers preserve the semantics of the programs they compile: a bug in a compiler could result in incorrect executable code being produced from correct, formally verified source programs. One way to obtain this guarantee is to formally verify the compiler itself, using theorem provers to prove that it is correct, i.e. preserves the semantics of source programs. Several non-trivial compilers have been formally verified along these lines, for assembly languages [21], imperative languages [16,18,5], object-oriented languages [14] and functional languages [6]. The work presented here is part of the development and verification of a front-end compiler from mini-ML to the Cminor intermediate language [18]. Our long-term plan is to combine this front-end with the verified back-end for Cminor described in [18] and with a future verification of the extraction mechanism from Coq functional specifications to mini-ML to obtain a trusted execution path for programs directly written in the Coq specification language.

Related Work

Many on-paper proofs of correctness for various CPS transformations have been published already, starting with Plotkin's seminal article [22]. We are aware of three earlier on-machine formalizations and correctness proofs for CPS transformations: one by Minamide and Okuma [20], using Isabelle/HOL; one by Tian [24], using Twelf; and one by Chlipala [6], using Coq.

A recurring difficulty in mechanizing programming language semantics, type systems and program transformations is the handling of binders and α-conversion (the fact that $\lambda x.x$ and $\lambda y.y$ are equivalent terms). Most existing proof assistants provide no native support for working with terms modulo α-conversion like we routinely do on paper. (The only exception is Urban's Isabelle/HOL implementation of nominal logic [25].) The POPLmark challenge [3] gives an excellent summary of the difficulties this raises when mechanizing properties of programming languages and of the known techniques to circumvent these difficulties: de Bruijn indices, higher-order abstract syntax, locally nameless representations, ... In the case of CPS transformations, Minamide and Okuma use named variables with no α-conversion for Plotkin's naive CPS transformation, and with explicit renamings for Danvy and Nielsen's optimized transformation. Tian uses higher-order abstract syntax to reason about Danvy and Nielsen's CPS transformation. Like Chlipala, we use de Bruijn indices [11] to provide unique representatives for λ-terms. We avoid some of the difficulties associated with standard de Bruijn indices by using two kinds of de Bruijn indices, independently numbered: one for source variables and one for continuation variables introduced by the CPS transformation.

Earlier work also differs on the kind of operational semantics used to prove the correctness of CPS transformations. Following Plotkin's original proof, Minamide and Okuma use small-step semantics, while Tian uses a combination of big-step semantics and small-step semantics for the source and target languages, respectively, and Chlipala uses a form of denotational semantics directed by the types of the simply-typed λ-calculus. We use (untyped) big-step semantics for the source and target languages. A strength of big-step semantics is that it avoids the well-known difficulties caused by administrative redexes in Plotkin's original, small-step proof of CPS transformations. A weakness of big-step semantics is that it captures only terminating executions, and therefore cannot be used to prove semantic preservation for diverging source programs. This limitation is unproblematic in our intended usage scenario, since programs extracted from Coq functional specifications are strongly normalizing.

Finally, earlier mechanizations handle only the pure λ-calculus, while we cover a larger, more realistic functional language including n-ary functions, recursive functions, data types and pattern-matching. These extensions are conceptually easy but technically not entirely obvious. Mechanizing the correctness proof is especially useful to ensure that we do not overlook the small difficulties raised by these extensions.

Outline

The remainder of this paper is organized as follows. Section 2 reviews some of the known CPS transformations. Section 3 defines the source and target languages for our transformations. We define and outline the correctness proof of two CPS transformations in Sect. 4 and 5. Section 6 gives some practical information on the Coq mechanization of these results. Concluding remarks are given in Sect. 7.

2 Examples of CPS Transformations

We start by reviewing some of the many known variants of CPS transformation for call-by-value λ-calculus. One of the earliest and simplest transformations is that of Plotkin [22]:

$$\begin{aligned}
\llbracket x \rrbracket_1 &= \lambda k.\ k\ x\\
\llbracket \lambda x.M \rrbracket_1 &= \lambda k.\ k\ (\lambda x.\ \llbracket M \rrbracket_1)\\
\llbracket M\ N \rrbracket_1 &= \lambda k.\ \llbracket M \rrbracket_1\ (\lambda m.\ \llbracket N \rrbracket_1\ (\lambda n.\ m\ n\ k))
\end{aligned}$$

Each source term is transformed into an abstraction $\lambda k \ldots$ over the continuation for this term. A weakness of this transformation is that it generates many [illegible], that is, β-redexes that correspond to no redex in the original source term. For instance, the translation of $x\ y$ contains four such redexes, outlined below:

$$\begin{aligned}
\llbracket x\ y \rrbracket_1 &= \lambda k.\ \underline{(\lambda k.\ k\ x)\ (\lambda m.\ (\lambda k.\ k\ y)\ (\lambda n.\ m\ n\ k))}\\
&\xrightarrow{\beta} \lambda k.\ \underline{(\lambda m.\ (\lambda k.\ k\ y)\ (\lambda n.\ m\ n\ k))\ x}\\
&\xrightarrow{\beta} \lambda k.\ \underline{(\lambda k.\ k\ y)\ (\lambda n.\ x\ n\ k)} \xrightarrow{\beta} \lambda k.\ \underline{(\lambda n.\ x\ n\ k)\ y} \xrightarrow{\beta} \lambda k.\ x\ y\ k
\end{aligned}$$

When CPS transformation is used as part of a compiler, these administrative redexes introduce inefficiencies that must be eliminated by a later pass of compile-time β-reduction.

The following variant of Plotkin's transformation avoids the generation of some, but not all administrative redexes. Here, instead of λ-abstracting over the continuation variable k, we turn k into an additional parameter of the (mathematical) function that defines the translation. The translation therefore becomes $\llbracket M \rrbracket_2 \rhd k$ where M is the source term and k a continuation term.

$$\begin{aligned}
\llbracket x \rrbracket_2 \rhd k &= k\ x\\
\llbracket \lambda x.M \rrbracket_2 \rhd k &= k\ (\lambda x.\lambda k.\ \llbracket M \rrbracket_2 \rhd k)\\
\llbracket M\ N \rrbracket_2 \rhd k &= \llbracket M \rrbracket_2 \rhd \lambda m.\ \llbracket N \rrbracket_2 \rhd \lambda n.\ m\ n\ k
\end{aligned}$$

We now have $\llbracket x\ y \rrbracket_2 \rhd k = (\lambda m.\ (\lambda n.\ m\ n\ k)\ y)\ x$, which contains only two administrative β-redexes.

Danvy and Nielsen [8] present the following refinement of the two-place translation above that avoids generating any administrative redex. It distinguishes λ-terms that are [illegible] $A, B ::= x \mid \lambda x.M$ from the other λ-terms $P, Q ::= M\ N$. The transformation is presented as two mutually recursive functions, $\Psi_3(A)$ for atoms A and $\llbracket M \rrbracket_3 \rhd k$ for arbitrary terms M.

$$\begin{aligned}
\llbracket A \rrbracket_3 \rhd k &= k\ \Psi_3(A) & \Psi_3(x) &= x\\
\llbracket A\ B \rrbracket_3 \rhd k &= \Psi_3(A)\ \Psi_3(B)\ k & \Psi_3(\lambda x.M) &= \lambda x.\lambda k.\ \llbracket M \rrbracket_3 \rhd k\\
\llbracket P\ B \rrbracket_3 \rhd k &= \llbracket P \rrbracket_3 \rhd \lambda p.\ p\ \Psi_3(B)\ k\\
\llbracket A\ Q \rrbracket_3 \rhd k &= \llbracket Q \rrbracket_3 \rhd \lambda q.\ \Psi_3(A)\ q\ k\\
\llbracket P\ Q \rrbracket_3 \rhd k &= \llbracket P \rrbracket_3 \rhd \lambda p.\ \llbracket Q \rrbracket_3 \rhd \lambda q.\ p\ q\ k
\end{aligned}$$

We now have $\llbracket x\ y \rrbracket_3 \rhd k = x\ y\ k$, as desired. However, the case for applications was split in 4 different cases, depending on whether the function and its argument are atoms or not. This combinatorial explosion makes it difficult to extend this transformation to n-ary function applications and data constructor applications.

To circumvent this difficulty, we use (in Sect. 5) the following alternate presentation of Danvy and Nielsen's transformation. We define the "smart application" constructor $@_\beta$ that reduces (on the fly) administrative redexes that would arise if the first argument is a lambda-abstraction and the second argument is an atom:

$$(\lambda x.M) @_\beta A = M\{x \leftarrow A\} \qquad\qquad M @_\beta N = M\ N \text{ otherwise}$$

We then use $@_\beta$ instead of regular applications in the variable and abstraction cases of the translation $[\![M]\!]_2 \rhd k$, obtaining:

$$\begin{aligned} [\![x]\!]_4 \rhd k &= k @_\beta x \\ [\![\lambda x.M]\!]_4 \rhd k &= k @_\beta (\lambda x.\lambda k.\ [\![M]\!]_4 \rhd k) \\ [\![M\ N]\!]_4 \rhd k &= [\![M]\!]_4 \rhd \lambda m.\ [\![N]\!]_4 \rhd \lambda n.\ m\ n\ k \end{aligned}$$

We have $[\![x\ y]\!]_4 \rhd k = (\lambda m.\ (\lambda n.\ m\ n\ k) @_\beta y) @_\beta x = x\ y\ k$ as expected. More generally, this transformation is extensionally equivalent to that of Danvy and Nielsen: $[\![M]\!]_4 \rhd k = [\![M]\!]_3 \rhd k$ if k is a λ-abstraction. Therefore, just like Danvy and Nielsen's transformation, it produces CPS terms that are free of administrative redexes.

3 Source and Target Languages

The source language for the CPS transformation has the following grammar:[1]

Source terms:

$M, N, P ::= x_0 \mid x_1 \mid \ldots$	variables (de Bruijn)
$\mid \lambda^n.\ M$	function of $n+1$ arguments
$\mid \mu^n.\ M$	recursive function ($n+1$ args.)
$\mid M(N_1, \ldots, N_k)$	function application
$\mid$ `let` M `in` N	bind x_0 to M in N
$\mid C(N_1, \ldots, N_k)$	data constructor application
$\mid$ `match` M `with` $\pi_1, \ldots, \pi_k$	pattern-matching

Match cases:

$\pi ::= C^n \to M$	n is the arity of constructor C

Variables x_i are identified by their de Bruijn indices i. Indices start at 0. The abstraction $\lambda^n.\ M$ has arity $n+1$; it binds variables $x_n, \ldots, x_0$ in M. A recursive abstraction $\mu^n.\ M$ is similar, but in addition x_{n+1} is bound within M to the abstraction itself. In the right-hand side M of a match case $C^n \to M$, variables $x_{n-1}, \ldots, x_0$ are bound to the n arguments of the matched constructor C.

---

[1] Our Coq development also supports numeric constants and arithmetic and relational operators over numbers. These are omitted in this paper for brevity.

$$\lambda^n.\ M \Rightarrow \lambda^n.\ M \qquad\qquad \mu^n.\ M \Rightarrow \mu^n.\ M$$

$$\frac{M \Rightarrow \lambda^n.\ P \quad N_i \Rightarrow v_i \quad P\{v_n, \ldots, v_0\} \Rightarrow v}{M(N_0, \ldots, N_n) \Rightarrow v}$$

$$\frac{M \Rightarrow \mu^n.\ P \quad N_i \Rightarrow v_i \quad P\{v_n, \ldots, v_0, \mu^n.\ P\} \Rightarrow v}{M(N_0, \ldots, N_n) \Rightarrow v} \qquad \frac{M \Rightarrow v_1 \quad N\{v_1\} \Rightarrow v}{(\texttt{let}\ M\ \texttt{in}\ N) \Rightarrow v}$$

$$\frac{M \Rightarrow C(v_1, \ldots, v_n) \quad \pi_i = (C^n \to N) \quad N\{v_n, \ldots, v_1\} \Rightarrow v}{(\texttt{match}\ M\ \texttt{with}\ \pi_1, \ldots, \pi_k) \Rightarrow v}$$

Fig. 1. Big-step semantics for the source language

The dynamic semantics of this language is given in big-step operational style by the rules in Fig. 1. The rules define the predicate $M \Rightarrow v$, "the term M evaluates to the value v". Values are

$$v ::= \lambda^n.\ M \mid \mu^n.\ M \mid C(v_1, \ldots, v_n).$$

We write $M\{N_0, \ldots, N_k\}$ for the simultaneous substitution of terms $N_0, \ldots, N_k$ for variables $x_0, \ldots, x_k$ in term M. Note the two rules for function application $M(N_0, \ldots, N_n)$, depending on whether M evaluates to a recursive or non-recursive abstraction. In the evaluation rule for the `match` construct, the selected case π_i is the first case that matches constructor C with arity n.

The target language for the CPS transformation is similar, except that it has two kinds of variables, independently numbered by de Bruijn indices: variables x_n correspond to variables already present in the source term, while variables κ_n correspond to variables introduced by the transformation to hold continuations and intermediate evaluation results. The grammar of the target language is therefore:

Target terms:

M', N', P'	$::= x_n$	source-level variables
	$\mid \kappa_n$	continuation variables
	$\mid \lambda^n.\ M'$	function of $n+1$ arguments
	$\mid \mu^n.\ M'$	recursive function ($n+1$ args.)
	$\mid M'(N'_1, \ldots, N'_k)$	function application
	$\mid \texttt{let}\ M'\ \texttt{in}\ N'$	bind x_0 to M in N
	$\mid C(N'_1, \ldots, N'_k)$	data constructor application
	$\mid \texttt{match}\ M'\ \texttt{with}\ \pi'_1, \ldots, \pi'_k$	pattern-matching

Match cases:

π'	$::= C^n \to M'$	n is the arity of constructor C

Conventionally, every function takes its continuation as first argument. Therefore, in $\lambda^n.M'$, the first argument is bound to κ_0 in M', and the remaining n arguments are bound to $x_{n-1}, \ldots, x_0$. For a recursive abstraction $\mu^n.M'$, the variable x_n is additionally bound to the abstraction itself. Match cases and the `let` binding bind source-level variables x_n exactly as in the source language.

$$\lambda^n.\ M' \Rightarrow \lambda^n.\ M' \qquad\qquad \mu^n.\ M' \Rightarrow \mu^n.\ M'$$

$$\frac{M' \Rightarrow \lambda^n.\ P' \qquad N'_i \Rightarrow v_i \qquad P'\{v_0\}\{v_n, \ldots, v_1\} \Rightarrow v}{M'(N'_0, \ldots, N'_n) \Rightarrow v}$$

$$\frac{M' \Rightarrow \mu^n.\ P \qquad N'_i \Rightarrow v_i \qquad P'\{v_0\}\{v_n, \ldots, v_1, \mu^n.\ P'\} \Rightarrow v}{M'(N'_0, \ldots, N'_n) \Rightarrow v}$$

$$\frac{M' \Rightarrow v_1 \qquad N'\{\,\}\{v_1\} \Rightarrow v}{(\texttt{let}\ M'\ \texttt{in}\ N') \Rightarrow v}$$

$$\frac{M' \Rightarrow C(v_1, \ldots, v_n) \qquad \pi'_i = (C^n \rightarrow N') \qquad N'\{\,\}\{v_n, \ldots, v_1\} \Rightarrow v}{(\texttt{match}\ M'\ \texttt{with}\ \pi'_1, \ldots, \pi'_k) \Rightarrow v}$$

Fig. 2. Big-step semantics for the target language

The reason why we use two kinds of de Bruijn indices is to simplify the definition of CPS transformations. As observed by Minamide and Okuma [20], if regular de Bruijn indices are used, the transformations need to shift indices of source-level variables to reflect the additional bindings that it inserts. For instance, the naive CPS transformation of $x_i\ x_j$ in regular de Bruijn notation is

$$\lambda^0.\ (\lambda^0.\ x_0(x_{i+2}))\ (\lambda^0.\ (\lambda^0.\ x_0(x_{j+3}))\ (\lambda^0.\ x_1(x_2, x_0)))$$

where the indices i and j of the two source variables are shifted by 2 and 3, respectively. This shifting makes it delicate to define and reason about CPS transformations. Using two kinds of variables avoids this difficulty: the CPS transformation of $x_i\ x_j$ is, then,

$$\lambda^0.\ (\lambda^0.\ \kappa_0(x_i))\ (\lambda^0.\ (\lambda^0.\ \kappa_0(x_j))\ (\lambda^0.\ \kappa_1(\kappa_2, \kappa_0)))$$

The source variables x_i and x_j need not be shifted because all bindings introduced by the translation bind continuation variables $\kappa_0, \kappa_1, \ldots$ but not source variables.

Figure 2 defines the big-step semantics for the target language. The evaluation rules are direct adaptations of those for the source language. We write $M\{N_0, \ldots, N_n\}\{P_0, \ldots, P_p\}$ for the double simultaneous substitution of terms $N_0, \ldots, N_n$ for variables $\kappa_0, \ldots, \kappa_n$ and of terms $P_0, \ldots, P_p$ for variables $x_0, \ldots, x_p$ in term M.

As the semantics use substitution, we will need some standard properties over substitution and the lifting operation such as commutation between lifting and substitution, or neutrality of substitution over closed terms.

The $\Uparrow$ operator denotes lifting of free de Bruijn indices: $\Uparrow_x^n M'$ replaces all x_i variables free in M' by x_{i+n}, and similarly $\Uparrow_\kappa^n M'$ replaces all κ_i variables free in M' by κ_{i+n}.

The following two lemmas about compositions of substitutions play a crucial role in proving semantic preservation for the CPS transformation.

Lemma 1. $(M\{\vec{N}\}\{\vec{P}\})\{\vec{Q}\}\{\vec{R}\} \;=\; (M\{\Uparrow_{\kappa}^{|\vec{N}|}\Uparrow_{x}^{|\vec{P}|}\,\vec{Q}\}\{\Uparrow_{\kappa}^{|\vec{N}|}\Uparrow_{x}^{|\vec{P}|}\,\vec{R}\})\{\vec{N}\}\{\vec{P}\}$

Lemma 2. $(M\{\Uparrow_{\kappa}^{|\vec{N}|}\Uparrow_{x}^{|\vec{P}|}\,\vec{Q}\}\{\Uparrow_{\kappa}^{|\vec{N}|}\Uparrow_{x}^{|\vec{P}|}\,\vec{R}\})\{\vec{N}\}\{\vec{P}\} \;=\; M\{\vec{N},\vec{Q}\}\{\vec{P},\vec{R}\}$

4 Verification of a Non-optimizing CPS Transformation

The non-optimizing CPS transformation for our source language is a straightforward extension of Plotkin's original call-by-value CPS transformation. We define two mutually recursive transformations, Ψ for atoms and $[\![\cdot]\!]$ for arbitrary terms. Atoms are defined by the following grammar:

Atoms: $A ::= x_n \mid \lambda^n.\ M \mid \mu^n.\ M \mid C(A_1,\dots,A_n)$

The transformation is defined by the following equations:

$$\begin{aligned}
\Psi(x_n) &= x_n\\
\Psi(\lambda^n.\ M) &= \lambda^{n+1}.\ [\![M]\!](\kappa_0)\\
\Psi(\mu^n.\ M) &= \mu^{n+1}.\ [\![M]\!](\kappa_0)\\
\Psi(C(A_1,\dots,A_n)) &= C(\Psi(A_1),\dots,\Psi(A_n))
\end{aligned}$$

$$\begin{aligned}
[\![A]\!] &= \lambda^0.\ \kappa_0(\Psi(A))\\
[\![M(N_1,\dots,N_n)]\!] &= \lambda^0.\ [\![M.N_1\dots N_n \texttt{ then } \kappa_n(\kappa_{n+1},\kappa_{n-1},\dots,\kappa_0)]\!]\\
[\![\texttt{let } M \texttt{ in } N]\!] &= \lambda^0.\ [\![M]\!](\lambda^0.\ \texttt{let } \kappa_0 \texttt{ in } [\![N]\!](\kappa_1))\\
[\![C(N_1,\dots,N_n)]\!] &= \lambda^0.\ [\![N_1\dots N_n \texttt{ then } \kappa_n(C(\kappa_{n-1},\dots,\kappa_0)]\!]\\
&\quad\ \text{if } C(N_1,\dots,N_n) \text{ is not an atom}\\
[\![\texttt{match } M \texttt{ with } \pi_1,\dots,\pi_n]\!] &= \lambda^0.\ [\![M]\!](\lambda^0.\ \texttt{match } \kappa_0 \texttt{ with } [\![\pi_1]\!],\dots,[\![\pi_n]\!])\\
[\![M_1\dots M_n \texttt{ then } N']\!] &= [\![M_1]\!](\lambda^0.\dots[\![M_n]\!](\lambda^0.\ N')\dots)\\
[\![C^n \to M]\!] &= C^n \to [\![M]\!](\kappa_1)
\end{aligned}$$

The translation $[\![M]\!]$ of a source term M is always a one-argument abstraction $\lambda^0\dots$ that will receive the current continuation and bind it to variable κ_0. A source function of arity $n+1$ becomes a function of arity $n+2$ that expects the continuation of the call as first argument (bound to variable κ_0), along with $n+1$ regular arguments (bound to variables $x_n,\dots,x_0$). For n-ary applications of functions and constructors, we use an auxiliary transformation for lists of expressions, written $[\![M_1\dots M_n \texttt{ then } N']\!]$. The generated term evaluates the translations $[\![M_1]\!],\dots,[\![M_n]\!]$ and binds them to $\kappa_{n-1},\dots,\kappa_0$ (respectively) before evaluating N'. In the case of a function application $M(N_1,\dots,N_n)$, we translate the list $M.N_1\dots N_n$ and finish with κ_n bound to the translation of M, $\kappa_{n-1},\dots,\kappa_0$ bound to the translations of $N_1,\dots,N_n$, and κ_{n+1} bound to the outer continuation for the application. We therefore finish the computation by evaluating $\kappa_n(\kappa_{n+1},\kappa_{n-1},\dots,\kappa_0)$.

The case of a constructor application is similar. However, if all arguments to the constructor are atoms, the constructor application itself is an atom and we force it to be translated as such. This not only improves the efficiency of the generated CPS term, but more importantly this is necessary for the proof of correctness to go through.

The CPS transformation satisfies the following syntactic properties, which play a crucial role in the proof of semantic preservation. We say that a term is κ-closed if no κ_i variables appear free in this term.

Lemma 3. $[\![M]\!]$ $\Psi(A)$ κ- . A , κ- :

$$[\![M]\!]\{\vec{N}\}\{\vec{P}\} = [\![M]\!]\{\,\}\{\vec{P}\}$$

P . By structural induction over M and A. For the n-ary applications, notice that κ_i is free in $[\![M_1 \ldots M_n \ \texttt{then} \ N']\!]$ only if κ_{i+n} is free in N'.

Lemma 4. T x- :

$$[\![M\{A_1, \ldots, A_n\}]\!] = [\![M]\!]\{\,\}\{\Psi(A_1) \ldots \Psi(A_n)\}$$
$$\Psi(A\{A_1, \ldots, A_n\}) = \Psi(A)\{\,\}\{\Psi(A_1) \ldots \Psi(A_n)\}$$

P . By structural induction over M and A. Notice that atoms are stable by substitution: $A\{A_1, \ldots, A_n\}$ is an atom whenever $A, A_1, \ldots, A_n$ are atoms.

To show that the CPS transformation preserves the semantics of the source program, we would like to show that if the source program P evaluates to the value v, then the CPS program $[\![P]\!]$ applied to the initial continuation $\lambda^0.\ \kappa_0$ (the identity function) evaluates to the value $\Psi(v)$, which has the same shape as v and differs only on the bodies of functions contained in v. Of course, this result cannot be proved by induction over P: we need to generalize the result to continuations other than the initial continuation.

The intuition for this generalization is simple: if $M \Rightarrow v$, the intended effect for the transformation $[\![M]\!]$ applied to a continuation K is to compute the value $\Psi(v)$, then apply K to this value. Therefore, whenever $K\ \Psi(v) \Rightarrow v'$, it should be the case that $[\![M]\!](K) \Rightarrow v'$.

Lemma 5. L $K = \lambda^0.P$ κ- , - . I $M \Rightarrow v$, $P\{\Psi(v)\}\{\,\} \Rightarrow v'$, $[\![M]\!](K) \Rightarrow v'$.

P . The proof proceeds by induction on the evaluation derivation of $M \Rightarrow v$ and case analysis over the term M. To give an idea of the proof, we sketch one case of intermediate difficulty: the case where $M = \texttt{let}\ M_1\ \texttt{in}\ M_2$. We have $M_1 \Rightarrow v_1$ and $M_2\{v_1\} \Rightarrow v$. We need to show

$$(\lambda^0.\ [\![M_1]\!](\lambda^0.\ \texttt{let}\ \kappa_0\ \texttt{in}\ [\![M_2]\!](\kappa_1)))\ (K) \Rightarrow v' \qquad (1)$$

under the assumptions that $K = \lambda^0.P$, K is κ-closed, and $P\{\Psi(v)\}\{\,\} \Rightarrow v'$.

Applying the induction hypothesis to the second premise $M_2\{v_1\} \Rightarrow v$ and the continuation K, we obtain:

$$[\![M_2\{v_1\}]\!](K) \Rightarrow v' \tag{2}$$

By Lemma 4 and the fact that v_1 is a value and therefore also an atom, (2) is equivalent to:

$$([\![M_2]\!]\{\,\}\{\Psi(v_1)\})(K) \Rightarrow v' \tag{3}$$

Take $P_1 = \texttt{let}\ \kappa_0\ \texttt{in}\ [\![M_2]\!](\Uparrow^1_x K)$. By the evaluation rule for `let`, Lemma 3, and some calculation over substitutions, (3) implies

$$P_1\{\Psi(v_1)\}\{\,\} \Rightarrow v' \tag{4}$$

The expected result (1) follows from (4) and the induction hypothesis applied to the first premise $M_1 \Rightarrow v_1$ and to the continuation $K_1 = \lambda^0.P_1$, which is κ-closed by Lemma 3.

Theorem 1. *I* $M \Rightarrow v$, $[\![M]\!](\lambda^0.\ \kappa_0) \Rightarrow \Psi(v)$. M , v - (,), $[\![M]\!](\lambda^0.\ \kappa_0) \Rightarrow v$.

P . We apply Lemma 5 to the initial continuation $K = \lambda^0.\ \kappa_0$, obtaining $[\![M]\!](\lambda^0.\ \kappa_0) \Rightarrow \Psi(v)$. For the corollary, we observe that $\Psi(v) = v$ for any first-order data structure v.

5 Verification of an Optimizing CPS Transformation

We now define an optimized CPS transformation that does not generate administrative redexes. This transformation generalizes transformation $[\![\cdot]\!]_4 \rhd \cdot$ from Sect. 2, namely the transformation of Danvy and Nielsen [8,9] presented using a "smart application" constructor $@_\beta$. This constructor is defined over terms of the target language by

$$(\lambda^0.M)\ @_\beta\ A = M\{A\}\{\,\} \qquad\qquad M\ @_\beta\ N = M(N) \text{ otherwise}$$

The optimizing transformation is presented as two mutually recursive functions, a one-place function Ψ for atoms and a two-place function $[\![\cdot]\!] \rhd \cdot$ for arbitrary terms.

$$\begin{aligned}
\Psi(x_n) &= x_n \\
\Psi(\lambda^n.\ M) &= \lambda^{n+1}.\ [\![M]\!] \rhd \kappa_0 \\
\Psi(\mu^n.\ M) &= \mu^{n+1}.\ [\![M]\!] \rhd \kappa_0 \\
\Psi(C(A_1,\ldots,A_n)) &= C(\Psi(A_1),\ldots,\Psi(A_n))
\end{aligned}$$

$$
\begin{aligned}
[\![A]\!] \rhd k &= k \mathbin{@_\beta} \Psi(A) \\
[\![M(N_1, \ldots, N_n)]\!] \rhd k &= [\![M.N_1 \ldots N_n \ \texttt{then}\ \kappa_n(\Uparrow_\kappa^{n+1} k, \kappa_{n-1}, \ldots, \kappa_0)]\!] \\
[\![\texttt{let}\ M\ \texttt{in}\ N]\!] \rhd k &= [\![M]\!] \rhd \lambda^0.\ \texttt{let}\ \kappa_0\ \texttt{in}\ [\![N]\!] \rhd \Uparrow_\kappa^1 \Uparrow_x^1 k \\
[\![C(N_1, \ldots, N_n)]\!] \rhd k &= [\![N_1 \ldots N_n \ \texttt{then}\ \ \Uparrow_\kappa^n k(C(\kappa_{n-1}, \ldots, \kappa_0)]\!] \\
&\qquad \text{if } C(N_1, \ldots, N_n) \text{ is not an atom} \\
[\![\texttt{match}\ M\ \texttt{with}\ \pi_1, \ldots, \pi_n]\!] \rhd k &= [\![M]\!] \rhd \lambda^0.\ \texttt{match}\ \kappa_0\ \texttt{with} \\
&\qquad\qquad [\![\pi_1]\!] \rhd k, \ldots, [\![\pi_n]\!] \rhd k
\end{aligned}
$$

$$[\![M_1 \ldots M_n \ \texttt{then}\ \ N']\!] = [\![M_1]\!] \rhd \lambda^0. \ldots [\![M_n]\!] \rhd \lambda^0.\ N'$$

$$[\![C^n \to M]\!] \rhd k = C^n \to [\![M]\!] \rhd \Uparrow_\kappa^1 k$$

To show that the optimizing CPS transformation preserves semantics, we would like to prove an analogue of Theorem 1: if $M \Rightarrow v$ and v is a first-order data structure, then $[\![M]\!] \rhd \lambda^0.\ \kappa_0 \Rightarrow v$. However, a direct proof of this theorem in the style of Lemma 5 is difficult. The root of the problem is that the "smart application" $@_\beta$ does not commute with substitutions. For example,

$$(x_0 \mathbin{@_\beta} C)\{x_0 \leftarrow \lambda^0.\ \kappa_0\} \ = \ (x_0(C))\{x_0 \leftarrow \lambda^0.\ \kappa_0\} \ = \ (\lambda^0.\ \kappa_0)(C)$$

while

$$(x_0\{x_0 \leftarrow \lambda^0.\ \kappa_0\}) \mathbin{@_\beta} (C\{x_0 \leftarrow \lambda^0.\ \kappa_0\}) \ = \ (\lambda^0.\ \kappa_0) \mathbin{@_\beta} C \ = \ C$$

Consequently, the optimizing transformation does not commute with substitutions of atoms for x-variables, as was the case for the non-optimizing transformation (Lemma 4).

To avoid these difficulties, we do not attempt to directly prove the correctness of the optimizing transformations, but instead show a semantic equivalence result between the naive and the optimizing transformations. This equivalence builds on the intuition that $[\![M]\!] \rhd k$ is identical to $[\![M]\!](k)$ modulo the contraction of some administrative redexes. These contractions are instances of β_v reductions:

$$(\lambda^0.\ M)(A) \to M\{A\}\{\,\} \qquad (\beta_v)$$

where the argument A must be an atom. It is well known that β_v reductions are valid in call-by-value semantics [22].

We first formally define parallel β_v reduction between terms of the target language. This parallel reduction relation, written $\rightsquigarrow$, is defined by the inference rules in Fig. 3. The first rule corresponds to one β_v reduction. The other rules build the congruence closure of this reduction, enabling zero, one or several β_v redexes to be reduced simultaneously at any position in the term. The reflexive transitive closure of $\rightsquigarrow$ is written $\overset{*}{\rightsquigarrow}$.

We then formalize the intuition that the term $[\![M]\!] \rhd K$ can be obtained by contracting β_v redexes in the term $[\![M]\!](K)$.

Lemma 6. *F* [illegible] A, $\Psi(A) \overset{*}{\rightsquigarrow} \Psi(A)$. *F* [illegible] M [illegible] K_1 [illegible] K_2, [illegible] $K_1 \overset{*}{\rightsquigarrow} K_2$ [illegible] K_1 [illegible], [illegible] $[\![M]\!]K_1 \overset{*}{\rightsquigarrow} [\![M]\!] \rhd K_2$.

$$\frac{A_1 \rightsquigarrow A_2 \qquad A_1 \text{ is an atom} \qquad M \rightsquigarrow N}{(\lambda^0.\ M)\ A_1 \rightsquigarrow N\{A_2\}\{\,\}}$$

$$x_i \rightsquigarrow x_i \qquad \kappa_i \rightsquigarrow \kappa_i \qquad \frac{M \rightsquigarrow N}{\lambda^n.\ M \rightsquigarrow \lambda^n.\ N} \qquad \frac{M \rightsquigarrow N}{\mu^n.\ M \rightsquigarrow \mu^n.\ N}$$

$$\frac{M \rightsquigarrow N \qquad M_1 \rightsquigarrow N_1 \qquad \dots \qquad M_n \rightsquigarrow N_n}{M(M_1, \dots, M_n) \rightsquigarrow N(N_1, \dots, N_n)} \qquad \frac{M_1 \rightsquigarrow N_1 \qquad M_2 \rightsquigarrow N_2}{(\texttt{let}\ M_1\ \texttt{in}\ M_2) \rightsquigarrow (\texttt{let}\ N_1\ \texttt{in}\ N_2)}$$

$$\frac{M_1 \rightsquigarrow N_1 \qquad \dots \qquad M_n \rightsquigarrow N_n}{C(M_1, \dots, M_n) \rightsquigarrow C(N_1, \dots, N_n)}$$

$$\frac{M \rightsquigarrow N \qquad \pi_1 \rightsquigarrow \pi_1' \qquad \dots \qquad \pi_n \rightsquigarrow \pi_n'}{(\texttt{match M with}\ \pi_1 \dots \pi_{\mathtt{n}}) \rightsquigarrow (\ \texttt{match M with}\ \pi_1' \dots \pi_{\mathtt{n}}')} \qquad \frac{M \rightsquigarrow N}{C^n \to M \rightsquigarrow C^n \to N}$$

Fig. 3. Definition of the parallel β_v reduction $\rightsquigarrow$

P . By structural induction over A and M.

We then show that the $\rightsquigarrow$ relation preserves semantics, in the following sense:

Lemma 7. *I* $M \Rightarrow v$ $M \rightsquigarrow N$, w $N \Rightarrow w$ $v \rightsquigarrow w$.

P . By induction on the derivation of $M \Rightarrow v$. We use the following substitution lemma: if $M_1 \rightsquigarrow M_2$, $\vec{N} \rightsquigarrow \vec{Q}$ and $\vec{P} \rightsquigarrow \vec{R}$ where $\vec{N}$ and $\vec{P}$ are lists of values, then $M_1\{\vec{N}\}\{\vec{P}\} \rightsquigarrow M_2\{\vec{Q}\}\{\vec{R}\}$.

Combining these results, we obtain the correctness of the optimizing CPS transformation.

Theorem 2. *I* $M \Rightarrow v$, w $\Psi(v) \overset{*}{\rightsquigarrow} w$ $[\![M]\!] \rhd \lambda^0.\ \kappa_0 \Rightarrow w$. M , v - , $[\![M]\!] \rhd \lambda^0.\ \kappa_0 \Rightarrow v$.

P . By Theorem 1, we know that $[\![M]\!](\lambda^0.\ \kappa_0) \Rightarrow \Psi(v)$. Lemma 6 shows that $[\![M]\!](\lambda^0.\ \kappa_0) \overset{*}{\rightsquigarrow} [\![M]\!] \rhd \lambda^0.\ \kappa_0$. Applying Lemma 7 repeatedly, we obtain the desired value w. If, moreover, v is a first-order data structure, then $\Psi(v) = v$, and $v \overset{*}{\rightsquigarrow} w$ implies $w = v$ by definition of the $\rightsquigarrow$ relation.

6 The Coq Development

The Coq mechanization of the results presented here is mostly standard. The CPS transformations, as well as the substitution and lifting operations, are presented as structural recursive functions. An advantage of this style is that Coq's extraction mechanism can generate executable Caml code directly from these functional specifications — there is no need to manually implement these functions in a programming language. Coq puts strong syntactic restrictions on recursive functions to ensure that they always terminate. As presented in Sect. 4 and 5, our transformations violate these restrictions; we had to locally expand the transformations of values and lists at point of use in the transformations of general terms. The operational semantics for the languages are presented as inductive predicates where each constructor corresponds exactly to one inference rule in Fig. 1 and 2.

Concerning the integration of the CPS transformations into the mini-ML to Cminor compiler that we are developing and verifying, only the optimizing CPS transformation of Sect. 5 is actually used in the compiler. The naive transformation of Sect. 4 appears only as an intermediate step in its proof of correctness.

In the compiler chain, CPS transformation comes after an uncurrying optimization (described in [10]) and before closure conversion. The output language of the uncurrying pass, and the input language of the closure conversion pass, are identical to the source language of the CPS transformations as defined in Sect. 3. However, the correctness proofs of these passes are conducted against a big-step semantics for this language that uses environments and closures instead of simultaneous substitutions. To resolve these mismatches between the CPS transformation and the surrounding passes, we also formalized and proved correct a translation from the CPS target language (with two kinds of de Bruijn indices) back to the CPS source language (with a single kind of indices), as well as a semantic equivalence result between substitution-based and environment-based semantics.

The whole development took about 4 person.months and represents approximately 9000 lines of Coq, decomposed as follows:

	Specifications	Proofs
Languages and their semantics (Sect. 3)	624 lines	—
Substitutions and their properties	447 lines	1 685 lines
Non-optimizing CPS transformation (Sect. 4)	609 lines	1 676 lines
Parallel β_v reductions (Sect. 5)	413 lines	689 lines
Optimizing CPS transformation (Sect. 5)	113 lines	237 lines
Connecting uncurrying with CPS transformation	303 lines	516 lines
Connecting CPS transformation with closure conversion	644 lines	803 lines

Among the specifications, only 300 lines correspond to definitions of executable functions which will be integrated into the compiler itself after extraction.

As usual with mechanizations using de Bruijn indices, the definitions of substitution and lifting plus the proofs of their properties take up a large part of our development. Finding the correct statements of these properties is now well understood in the case of elementary substitutions, but required some trial and error in the case of simultaneous substitutions. The theory of the $\lambda\sigma$-calculus [1] helped us find the correct statements before attempting to prove them. Their proofs are large but mostly routine. We were able to partially automate these proofs using special-purpose tactics defined within Coq's `ltac` language.

7 Conclusions

The work presented in this paper shows that an optimizing CPS transformation defined for a realistic core functional language can be, with some effort, mechanically proved correct using a proof assistant. Our mechanization uses only elementary techniques (no higher-order abstract syntax, no nominal logic) and should therefore be adaptable to most proof assistants. We used several non-standard technical devices: de Bruijn notation with two kinds of indices; proving the CPS transformation against big-step operational semantics instead of small-step semantics; and proving the optimizing transformation by reduction to the non-optimizing one. These devices do not significantly reduce the overall size of the proof, but enable us to decompose it into mostly-independent sub-proofs of more manageable size. For instance, using two kinds of indices requires an additional transformation and a separate correctness proof for it, but minimizes the amount of index management performed during CPS transformation and keeps its correctness proof simple.

A natural extension of this work is to mechanically verify the correctness of transformations to A-normal forms [12] and monadic normal forms [23], two intermediate representations that share many of the features of CPS. We have not attempted to do so, but believe that the techniques presented here could be effective in these other settings.

Although the intended use for our mini-ML compiler is to compile strongly normalizing programs, it would be interesting to try to prove the correctness of CPS transformations for diverging programs using the co-inductive big-step semantics of [17].

Another direction for further work is to investigate the usability of Urban's Isabelle/HOL implementation of nominal logic [25] for proving the correctness of CPS transformations.

References

1. Abadi, M., Cardelli, L., Curien, P.-L., Lévy, J.-J.: Explicit substitutions. Journal of Functional Programming 1(4), 375–416 (1991)
2. Appel, A.W.: Compiling with continuations. Cambridge University Press, Cambridge (1992)
3. Aydemir, B.E., Bohannon, A., Fairbairn, M., Foster, J.N., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.: Mechanized metatheory for the masses: The POPLmark challenge. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 50–65. Springer, Heidelberg (2005)
4. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development – Coq'Art: The Calculus of Inductive Constructions. In: EATCS Texts in Theoretical Computer Science, Springer, Heidelberg (2004)
5. Blazy, S., Dargaye, Z., Leroy, X.: Formal verification of a C compiler front-end. In: Misra, J., Nipkow, T., Sekerinski, E. (eds.) FM 2006. LNCS, vol. 4085, pp. 460–475. Springer, Heidelberg (2006)
6. Chlipala, A.: A certified type-preserving compiler from lambda calculus to assembly language. In: Programming Language Design and Implementation 2007, pp. 54–65. ACM Press, New York (2007)

7. Coq development team. The Coq proof assistant. Software and documentation (1989–2007), available at http://coq.inria.fr/
8. Danvy, O., Nielsen, L.R.: A first-order one-pass CPS transformation. Theoretical Computer Science 308(1-3), 239–257 (2003)
9. Danvy, O., Nielsen, L.R.: CPS transformation of beta-redexes. Information Processing Letters 94(5), 217–224 (2005)
10. Dargaye, Z.: Décurryfication certifiée. In: Journées Francophones des Langages Applicatifs (JFLA 2007), INRIA (2007)
11. de Bruijn, N.G.: Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indag. Math. 34(5), 381–392 (1972)
12. Flanagan, C., Sabry, A., Duba, B., Felleisen, M.: The essence of compiling with continuations. In: Programming Language Design and Implementation 1993, pp. 237–247. ACM Press, New York (1993)
13. Kennedy, A.: Compiling with continuations, continued. In: International Conference on Functional Programming, ACM Press, New York (2007)
14. Klein, G., Nipkow, T.: A machine-checked model for a Java-like language, virtual machine and compiler. ACM Transactions on Programming Languages and Systems 28(4), 619–695 (2006)
15. Kranz, D., Adams, N., Kelsey, R., Rees, J., Hudak, P., Philbin, J.: ORBIT: an optimizing compiler for Scheme. In: SIGPLAN 1986. symposium on Compiler Construction, pp. 219–233. ACM Press, New York (1986)
16. Leinenbach, D., Paul, W., Petrova, E.: Towards the formal verification of a C0 compiler: Code generation and implementation correctness. In: SEFM 2005. Int. Conf. on Software Engineering and Formal Methods, pp. 2–11. IEEE Computer Society Press, Los Alamitos (2005)
17. Leroy, X.: Coinductive big-step operational semantics. In: Sestoft, P. (ed.) ESOP 2006 and ETAPS 2006. LNCS, vol. 3924, pp. 54–68. Springer, Heidelberg (2006)
18. Leroy, X.: Formal certification of a compiler back-end, or: programming a compiler with a proof assistant. In: 33rd symposium Principles of Programming Languages, pp. 42–54. ACM Press, New York (2006)
19. Letouzey, P.: A new extraction for Coq. In: Geuvers, H., Wiedijk, F. (eds.) TYPES 2002. LNCS, vol. 2646, pp. 200–219. Springer, Heidelberg (2003)
20. Minamide, Y., Okuma, K.: Verifying CPS transformations in Isabelle/HOL. In: MERLIN 2003. Proc. workshop on Mechanized reasoning about languages with variable binding, pp. 1–8. ACM Press, New York (2003)
21. Moore, J.S.: Piton: a mechanically verified assembly-language. Kluwer Academic Publishers, Dordrecht (1996)
22. Plotkin, G.D.: Call-by-name, call-by-value and the lambda-calculus. Theoretical Computer Science 1(2), 125–159 (1975)
23. Sabry, A., Wadler, P.: A reflection on call-by-value. ACM Transactions on Programming Languages and Systems 19(6), 916–941 (1997)
24. Tian, Y.H.: Mechanically verifying correctness of CPS compilation. In: CATS 2006. Proceedings of the 12th Computing: The Australasian Theory Symposium, pp. 41–51. Australian Computer Society (2006)
25. Urban, C.: Nominal techniques in Isabelle/HOL. Journal of Automated Reasoning (to appear, 2007)

Operational and Epistemic Approaches to Protocol Analysis: Bridging the Gap

Francien Dechesne[1], MohammadReza Mousavi[1,2], and Simona Orzan[1]

[1] Department of Computer Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600MB, Eindhoven, The Netherlands
[2] Department of Computer Science, Reykjavík University,
Kringlan 1, IS-103, Reykjavík, Iceland

Abstract. Operational models of protocols, on one hand, are readable and conveniently match their implementation, at a certain abstraction level. Epistemic models, on the other hand, are appropriate for specifying knowledge-related properties such as anonymity. These two approaches to specification and analysis have so far developed in parallel and one has either to define ad hoc correctness criteria for the operational model or use complicated epistemic models to specify the operational behavior. We work towards bridging this gap by proposing a combined framework which allows modeling the behavior of a protocol in a process language with an operational semantics and supports reasoning about properties expressed in a rich logic with temporal and epistemic operators.

1 Introduction

Knowledge-related aspects are currently being recognized as very relevant when expressing and analyzing correctness requirements of complex distributed algorithms and communication protocols, from the fundamental ones like consensus in a network, to applications like information flow control and security protocols (secrecy, anonymity, fair exchange). Many approaches based on epistemic logics have been developed for the analysis of such protocols: BAN logic [8], the theory of function views [20], interpreted systems [14,16,25] etc.

They allow for natural and effective representations of subtle effects of communication acts such as classified information leaking to attackers or participants gaining the common knowledge that the protocol they were running meets its goal. But on the other hand, modeling protocols using epistemic-logic-based approaches requires a high degree of expertise and verification of functional properties is often very complex. The information updates generating the transitions between epistemic states are especially tedious to specify, because logics are geared to expressing properties rather than operational steps of a protocol.

The operational behavior of protocols is, however, easily and conveniently specified in languages such as process algebras [7,22,2] and message sequence charts [9]. Functional requirements such as liveness and safety are then easily verified by model checking applied on the underlying transition systems. Unfortunately, these standard and successful verification schemes use temporal logics

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 226–241, 2007.
© Springer-Verlag Berlin Heidelberg 2007

that are not well-suited for expressing knowledge-related properties, therefore complex specialized solutions need to be sought in order to make process algebras suitable for the analysis of epistemic-flavored properties like anonymity [26,11]. See [20,13] for a more detailed comparison of epistemic-based vs. process-based protocol verification.

In this paper, we propose a framework that allows one to benefit the best of the two worlds, i.e., one can specify the behavior of a protocol in a process language and verify properties expressed in a logic with both temporal and epistemic operators. To achieve this, the key idea is to introduce explicit identities in our process language *PA* and allow every action to be annotated with a visibility range — i.e., a set of identities that may observe it and a "public appearance" — i.e., an alternative action that is observed by the identities outside the visibility range. We give an operational semantics for *PA* in terms of annotated labeled transition systems (ALTSs), which are LTSs with, for every identity, an extra indistinguishability relation on states. These relations model the uncertainties of the identities (typically principals in a protocol) about the current state, similar to the way uncertainties are represented in standard possible-world semantics for epistemic logics [14]. Thanks to the combination of transitions and indistinguishability relations, ALTSs naturally support verification of logic formulae containing both temporal and epistemic operators. We introduce a rich logic, $E\overline{\mu}$ (epistemic μ-calculus with past) and give it an interpretation on ALTSs.

Due to the explicit use of identities, *PA* allows a precise specification of the information hiding behavior within protocols, and it is therefore more expressive and flexible than traditional process algebras. It is also more intuitive and more formal than epistemic logics, when it comes to behavior modeling. Also $E\overline{\mu}$ is more expressive than the usual temporal logics used in traditional protocol verification. The resulting model checking framework $PA + E\overline{\mu}$ soundly extends the traditional process-based and epistemic model checking settings.

Related Work. The fact that the two verification approaches, process algebraic and epistemic, are complementary and that they should ideally be combined has already been recognized in [20], where the aim is, just as here, to provide a framework in which both protocol specification and correctness criteria can be specified succinctly and intuitively (and the authors indeed put the two approaches in sharp contrast). They introduce the notion of [illegible] to represent partial information and uses it to precisely formalize several subtle information hiding properties. Since the focus of that theory is proper formalization of requirements, we believe that it is complementary to ours and that it could possibly be used in our *PA* models, for defining suitable visibility ranges.

BAN-logic [8], designed for the analysis of authentication in security protocols, is very popular, but it is a known problem that a clear semantics, linking the high-level BAN-specification to runs of the protocol, is still missing. Also in other interesting recent work concerning Dynamic Epistemic Logic [15,3,19] with an operational flavor, just as in tool-supported temporal epistemic approaches [25,18], where existing temporal specification languages are used, but the embedding of the epistemic aspects remains (for a large part) informal. We start from the other side - a process specification language with a formal semantics, and work towards properly integrating epistemic aspects.

Interpreted Systems [14,25,16] are close to the operational semantics of our process language. In fact, it is possible to translate ALTSs to ISs. Our key improvement is the introduction of a process specification language with a formal semantics, which enables the modeling of systems at a reasonable abstraction level. In [16], interpreted systems are used to model different complex notions of (probabilistic) anonymity, using also an epistemic logic. Our approach is related to and complements that one, by providing a way of verifying, on process-based specifications, anonymity notions as defined by [16].

The concept of indistinguishability used here bears resemblances to the data independence technique in [6]. We consider runs of a protocol indistinguishable if they appear equal to a principal (as defined by the visibility range of actions). It is worthwhile to extend our framework along the lines of [6], by allowing the visibility range of actions to be dynamically updated.

Concurrently with our work, a rich language C^3 [5] and a powerful logic CPL [21] have been developed for analyzing cryptographic protocols. The aim there is integrating a wide range of features, from deontic and spatial operators to probabilities, in one unified setting. C^3+CPL is therefore very expressive, but complex and seems difficult to implement, while our basic language with an easy to grasp operational semantics can immediately lead to a practical verification toolset. In fact, a prototype implementation already exists [1]. Furthermore, there is a fundamental difference between our underlying logics: that of [21] is a state-based logic (à la LTL) and ours is action-based (à la modal μ-calculus).

Overview. Section 2 introduces our generic process language for specifying protocols and a transition-system semantics for it. Section 3 defines our temporal epistemic logic $E\overline{\mu}$ and the interpretation of $E\overline{\mu}$ formulas on the transition systems. Then we show that this construction does indeed bridge the gap between process-based and epistemic-logic-based approaches to protocol analysis, by proving that its projections on the two worlds are consistent with established definitions in the two worlds separately (Section 4). Section 5 shows an example and Section 6 concludes the paper and presents directions for future research.

2 *PAi*: Syntax and Operational Semantics

In this section, we present the syntax and the operational semantics of a simple modeling language which we call process algebra with identities (*PA*). *PA* has generic features, that can be adapted to match constructs of any classical operational modeling language (such as CCS [22], CSP [7] or Spi-Calculus [2]). It mostly resembles Milner's CCS, but we deviate from CCS in a few ways. Apart from adding identities, we use sequential composition instead of action prefixing (and thus, we also introduce a termination predicate), since this is very handy in writing protocol specifications. Also, we do not hide the result of a communication automatically and leave this, if at all desired, to the renaming function since the communicated message can be of relevance in the correctness specification of the protocol.

***PAi* : Syntax.** Let $\mathcal{A}ct$ be a finite set of [illegible] which will be ranged over by $a, b, a_0, ?a, !a, \ldots$, and let $\mathcal{I}d$ be a finite set of [illegible] typically denoted by by $i, j, \ldots i_1, i_2, \ldots$. We designate an action $\tau \in \mathcal{A}ct$ to denote the internal (silent) action; in addition to its common process-algebraic meaning, an internal action here represents a message that offers no new information to the observer principal. Question mark and exclamation mark (preceding actions) represent the receiving and the sending parts of a communication, respectively, and an action without such marks is the outcome of the communication.

$$\begin{array}{lcl} Proc & ::= & 0 \mid D \mid Proc; Proc \mid Proc + Proc \mid Proc || Proc \\ D & ::= & (\mathtt{J})\alpha \end{array}$$

0 denotes inaction (the process that has terminated). $d = (\mathtt{J})\alpha \in D$ denotes a decorated action and has the following intuitive meaning: action $\alpha \in \mathcal{A}ct$ is taken and is visible to principals $i \in \mathtt{J} \subseteq \mathcal{I}d$, while principals $j \notin \mathtt{J}$ observe $\rho(\alpha)$ being taken, where $\rho : \mathcal{A}ct \to \mathcal{A}ct$ is a global renaming function, which assigns to every action its "public appearance". The renaming function ρ should be defined by the specifier of a protocol but we assume that $\rho(\tau)$ is always defined to be τ. For any other action a, if $\rho(a) = \tau$, then $(\mathtt{J})a$ becomes unobservable to the principals not in $\mathtt{J}$. The combination of identity annotations on actions and the action renaming provides different views on the behavior of the system, according to different principals. Modeling passive observation of a system by hiding parts of it to specific principals is already done in the literature [26], but we will generate the views for all principals simultaneously. This enables talking about properties such as "i knows that j knows that k has communicated message a". $Proc; Proc$ denotes sequential composition, $Proc + Proc$ denotes nondeterministic choice, and $Proc \,||\, Proc$ denotes parallel composition.

E[illegible] *1.* Take $P = (\mathtt{1})a\,; (\mathtt{1,2})d + (\mathtt{1})b + (\mathtt{1})c$, with the renaming function $\rho(a) = \rho(b) = \rho(c) = dum$ where dum is a dummy basic action and over the identity set $\mathcal{I}d = \{1, 2\}$. P denotes the process that executes one of the actions a,b,c, but only principal 1 is aware of the exact action taking place. 1 is the principal making a choice between actions a, b and c, and 2 is an observer who only notices that a choice has been made, but not what the outcome was. This is a process-style formalization of the [illegible] from epistemic modeling, where a party learns something while other parties are watching and learn that the party learned something, but not precisely what. After the first step, the process terminates or, if the first step was a, continues with the execution of d. Since principal 2 is allowed to observe the execution of d, she may now conclude that the first step must have been a, although 2 was not actually allowed to observe the a. This is exactly the type of information leaks that we aim at capturing with our verification framework.

***PAi* : Operational Semantics.** We introduce the notion of *A*[illegible] *L*[illegible]-*T*[illegible] *S*[illegible] (ALTS) as labeled transition systems extended with

$$
(\mathbf{0})\,\frac{}{(0,\pi)\checkmark} \qquad (\mathbf{a})\,\frac{}{(d,\pi)\overset{d}{\Rightarrow}(0,\pi\frown d)} \qquad (\mathbf{s0})\,\frac{(x_0,\pi)\overset{d}{\Rightarrow}(y_0,\pi')}{(x_0;x_1,\pi)\overset{d}{\Rightarrow}(y_0;x_1,\pi')}
$$

$$
(\mathbf{s1})\,\frac{(x_0,\pi)\checkmark \quad (x_1,\pi)\overset{d}{\Rightarrow}(y_1,\pi')}{(x_0;x_1,\pi)\overset{d}{\Rightarrow}(y_1,\pi')} \qquad (\mathbf{s2})\,\frac{(x_0,\pi)\checkmark \quad (x_1,\pi')\checkmark}{(x_0;x_1,\pi'')\checkmark}
$$

$$
(\mathbf{n0})\,\frac{(x_0,\pi)\overset{d}{\Rightarrow}(y_0,\pi')}{(x_0+x_1,\pi)\overset{d}{\Rightarrow}(y_0,\pi')} \qquad (\mathbf{n2})\,\frac{(x_0,\pi)\checkmark}{(x_0+x_1,\pi')\checkmark} \qquad (\mathbf{p0})\,\frac{(x_0,\pi)\overset{d}{\Rightarrow}(y_0,\pi')}{(x_0\,||\,x_1,\pi)\overset{d}{\Rightarrow}(y_0\,||\,x_1,\pi')}
$$

$$
(\mathbf{p2})\,\frac{(x_0,\pi)\checkmark \quad (x_1,\pi')\checkmark}{(x_0\,||\,x_1,\pi'')\checkmark} \qquad (\mathbf{p3})\,\frac{(x_0,\pi)\overset{(\mathtt{J})?a}{\Rightarrow}(y_0,\pi') \quad (x_1,\pi)\overset{(\mathtt{J}')!a}{\Rightarrow}(y_1,\pi'')}{(x_0\,||\,x_1,\pi)\overset{(\mathtt{J}\cup\mathtt{J}')a}{\Rightarrow}(y_0\,||\,y_1,\pi\frown(\mathtt{J}\cup\mathtt{J}')a)}
$$

$$
(=\mathbf{refl})\,\frac{}{\pi\overset{i}{=}\pi} \qquad (=\rho\mathbf{0})\,\frac{\pi\overset{i}{=}\pi' \quad a=b \quad i\in\mathtt{J}\cap\mathtt{J}'}{\pi\frown(\mathtt{J})a\overset{i}{=}\pi'\frown(\mathtt{J}')b}
$$

$$
(=\rho\mathbf{1})\,\frac{\pi\overset{i}{=}\pi' \quad \rho(a)=\rho(b) \quad i\notin\mathtt{J}'\cup\mathtt{J}}{\pi\frown(\mathtt{J})a\overset{i}{=}\pi'\frown(\mathtt{J}')b} \qquad (=\rho\mathbf{2})\,\frac{\pi\overset{i}{=}\pi' \quad a=\rho(b) \quad i\in\mathtt{J}\setminus\mathtt{J}'}{\pi\frown(\mathtt{J})a\overset{i}{=}\pi'\frown(\mathtt{J}')b}
$$

$$
(=\tau\mathbf{0})\,\frac{\pi\overset{i}{=}\pi' \quad i\notin\mathtt{J} \quad \rho(a)=\tau}{\pi\frown(\mathtt{J})a\overset{i}{=}\pi'} \qquad (=\tau\mathbf{2})\quad\frac{\pi\overset{i}{=}\pi'}{\pi\frown(\mathtt{J})\tau\overset{i}{=}\pi'}
$$

$$
(\mathbf{strip})\,\frac{(x,\pi)\overset{(\mathtt{J})a}{\Rightarrow}(y,\pi')}{(x,\pi)\xrightarrow{a}(y,\pi')} \qquad (\mathbf{I})\,\frac{\pi_0\overset{i}{=}\pi_1}{(x_0,\pi_0)\overset{i}{\cdots}(x_1,\pi_1)}
$$

Fig. 1. SOS of *PAi*

annotations that denote when two states are deemed indistinguishable from the viewpoint of a principal, based on the actions taken so far. This is determined by the information that a principal receives in the course of protocol execution, which in turn is determined by the visibility annotations.

Definition 1 (ALTS). *A ALTS 5-* $\langle St, \rightarrow, \checkmark, I, s_0\rangle$, St , $\rightarrow\ \subseteq St\times\mathcal{A}ct\times St$, $\checkmark\subseteq St$, $I\subseteq St\times\mathcal{I}d\times St$ s_0 .

For readability, we denote statements $(s,l,s')\in\ \rightarrow$, $s\in\checkmark$ and $(s,i,s')\in I$ by $s\xrightarrow{l}s'$, $s\checkmark$ and $s\overset{i}{\cdots}s'$, respectively, for each $s,s'\in St$, $l\in\mathcal{A}ct$ and $i\in\mathcal{I}d$. The transition relation $\rightarrow$ has exactly the same role and meaning as in the standard notion of LTS. Formula $s\checkmark$ means that in state s it is possible to terminate. Expression $s_0\overset{i}{\cdots}s_1$ denotes that the principal with identity i cannot distinguish s_0 from s_1 since both s_0 and s_1 are reachable through paths that look identical as far as as principal i can observe and distinguish. It is desirable for $\overset{i}{\cdots}$ to be an equivalence relation for each $i\in\mathcal{I}d$ since this leads to a natural representation of knowledge (i.e., S5 Kripke models in modal logic, see [14]).

In Figure 1, we associate ALTS's to PA processes by means of a semantics in the SOS style of [24]. The operational state of PA is a pair (p, π) where $p \in Proc$ is a PA process and π is a finite sequence of decorated actions recording the perception of the process gathered so far. First we define auxiliary relations $\overset{d}{\Rightarrow} \subseteq St \times St$ and $\overset{i}{=} \subseteq D^* \times D^*$ for each decorated action d and identity i. Transition relation $\overset{d}{\Rightarrow}$ defines transitions among operational states labeled with decorated action d and $\overset{i}{=}$ defines when two traces are deemed indistinguishable by principal i. Note that each process p in the state (p, π) has one past trace π and possibly many futures. That is why, for example, in the deduction (**p3**) both parallel arguments x_0 and x_1 are assumed to start from the same history π, which is the common history of $x_0 \,||\, x_1$. In the deduction rule (**strip**), we strip off the extra information on the labels (concerning the visibility range) and apply encapsulation (leaving out individual send and receive actions) and obtain the transition relation $\rightarrow$. (We could have used an explicit restriction operator but decided not to do so to keep the presentation simple.) Deduction rule (**I**) lifts the concept of indistinguishability from traces to operational states. We omitted symmetric rules (**n1**), (**n3**), (**p1**), (**p4**), ($= \rho$**3**), ($= \tau$**1**), and ($= \tau$**3**). Termination of a process is orthogonal to its past history, so we use different meta-variables for the traces in the premises and the conclusion of rules (**s2**), (**n2**), and (**p2**). The transition relation $\Rightarrow$ and indistinguishability relation $\cdots$ are the sets of all closed statements provable using the deduction rules (plus their symmetric versions) from Figure 1. The semantics of a process p is defined by the ALTS with pairs of processes and decorated traces as states, $\rightarrow$ as transition relation, ✓ as termination relation, $\cdots$ as indistinguishability relation, and $(p, [])$ as the initial state, where $[]$ denotes the empty sequence of decorated actions. The following lemma states that $\overset{i}{\cdots}$ is an equivalence relation.We intentionally did not add deduction rules to enforce symmetry and transitivity of $\overset{i}{=}$ explicitly in order to preserve the inductive structure of our SOS specification.

Lemma 1. R $\overset{i}{\cdots}$.

3 An Epistemic Mu-Calculus

We introduce an epistemic mu-calculus with past ($E\overline{\mu}$) which combines temporal, epistemic, and fixed point constructs. We give our logic an interpretation on the operational model introduced in Section 2.

Syntax. The syntax of $E\overline{\mu}$ is given by the following grammar:

$$\phi ::= \top \mid X \mid \phi \wedge \phi \mid \neg\phi \mid \langle a \rangle \phi \mid \langle \overline{a} \rangle \phi \mid K_i \phi \mid \nu X.\phi(X)$$

(if X occurs only positively in ϕ),

where a ranges over the set of actions ($a \in \mathcal{A}ct$). Then $\langle a \rangle \phi$ stands for "after some execution of a, ϕ holds"; $\langle \overline{a} \rangle \phi$ has the same intuition as $\langle a \rangle \phi$, except that it refers to the , i.e., there is a state in which ϕ holds and from which it is possible to take an a-step to the current state. $K_i \phi$ should be read as "principal

i knows that ϕ holds". The greatest fixed point operator $\nu X.\phi(X)$ is used to define recursive concepts. It intuitively means that the current state is in the largest set X of states that satisfy $\phi(X)$. (Here X is a variable ranging over propositional formulas, which can be identified by the sets of states in which such a formula is true. This is made formal by introducing valuations, but we leave this correspondence informal here.) For convenience, we define and use the following abbreviations for commonly used logical formulae:

$[a]\phi$	i.e., $\neg\langle a\rangle\neg\phi$ and intuitively means that after *all* a-transitions, ϕ holds.
$\mu X.\phi(X)$	(with X occurring positively in ϕ) is the least fixed point operator, which is defined by $\neg\nu X.\neg\phi(\neg X)$ (X also occurs positively in $\neg\phi$). The current state is in the smallest set of states satisfying $\phi(X)$.
$\langle\centerdot\rangle\phi$	(similarly, $\langle\overline{\centerdot}\rangle\phi$) stands for $\bigvee_{a\in\mathcal{A}ct}\langle a\rangle\phi$ ($\bigvee_{a\in\mathcal{A}ct}\langle\overline{a}\rangle\phi$), which is by itself an abbreviation for a finite number of disjunctions. Intuitively, it means that after (before) *some* transition ϕ holds.
$\ulcorner a$	(similarly, $a^{\urcorner}$) is an abbreviation for $\mu X.\langle a\rangle\top\vee\langle x\rangle.X$ (or $\mu X.\langle\overline{a}\rangle\top\vee\langle\overline{x}\rangle.X$). So, it is possible to reach a state in the future where an a-transition is possible (or go back to a state in the past that results from an a-transition).
$[\centerdot^*]\phi$	(similarly, $[\overline{\centerdot^*}]\phi$) is an abbreviation for $\mu X.\phi\vee[\centerdot]X$ (or $\mu X.\phi\vee[\overline{\centerdot}]\phi$). The intuition behind this abbreviation is that all future paths will (paths in the past) lead to a state, in which there is a state satisfying ϕ. ($\langle\centerdot^*\rangle\phi$ and $\langle\overline{\centerdot^*}\rangle\phi$ are defined accordingly.)
$C_J\phi$	stands for $\nu X.(\bigwedge_{i\in J}K_i(X\wedge\phi))$ [14], meaning: "it is common knowledge among the principals in the set J that ϕ holds".

Common knowledge is a very powerful construction, expressing that agents in J not only know that ϕ holds, but also that all agents in J know that ϕ holds, and that all agents in J know that all agents in J know that ϕ holds, and so on. This property has so far not been amenable to specification and verification with standard operational techniques, while it is in fact very interesting, particularly for protocols where trust is an issue. Common knowledge can express, for instance, that participants in a multiparty fair exchange protocol trust each other and the protocol they are running. Let $E\overline{\mu}$-forms denote the set of $E\overline{\mu}$ formulas.

Interpreting $E\overline{\mu}$ Formulas on ALTSs. We now define what it means for a formula $\phi\in E\overline{\mu}$-forms to be satisfied in the ALTS A.

Definition 2 (satisfaction). *L* [illegible] $\mathrm{A}=\langle S,\rightarrow,\checkmark,I,s_0\rangle$ [illegible] *ALTS. T* [illegible] - [illegible] $\models$ [illegible] $\phi\in E\overline{\mu}$- [illegible] :

$$
\begin{array}{lll}
\mathrm{A},s\models\top & \textit{iff} & \textit{true}\\
\mathrm{A},s\models\phi_1\wedge\phi_2 & \textit{iff} & \mathrm{A},s\models\phi_1 \textit{ and } \mathrm{A},s\models\phi_2\\
\mathrm{A},s\models\neg\phi & \textit{iff} & \mathrm{A},s\models\phi \textit{ is not true}\\
\mathrm{A},s\models\langle a\rangle\phi & \textit{iff} & \textit{there is an } s'\in S \textit{ s.t. } s\xrightarrow{a}s' \textit{ and } \mathrm{A},s'\models\phi\\
\mathrm{A},s\models\langle\overline{a}\rangle\phi & \textit{iff} & \textit{there is an } s'\in S \textit{ s.t. } s'\xrightarrow{a}s \textit{ and } \mathrm{A},s'\models\phi\\
\mathrm{A},s\models K_i\phi & \textit{iff} & \textit{for all reachable } s'\in S \textit{ s.t. } s\overset{i}{\cdots}s' :\ \mathrm{A},s'\models\phi\\
\mathrm{A},s\models\nu X.\phi(X) & \textit{iff} & s\in\bigcup\{S'\subseteq S\mid\forall s'\in S'.\mathrm{A},s'\models\phi(X:=S')\}
\end{array}
$$

A [illegible] ϕ, [illegible] $\mathrm{A}\models\phi$, [illegible] $s_0\models\phi$.

The most noticeable of the rules above is the one for $K_i\phi$. It expresses the fact that i knows ϕ if ϕ holds in all states considered possible by i when residing in s, that is in all states belonging to the $\overset{i}{\cdots}$ equivalence class of s. The semantic rules in the previous section constructed this relation based on what i was allowed to observe from the run of the protocol. The intention behind the formula $K_i\phi$ is not to check what i learned in terms of explicit information the principal received (e.g., as contents of some message), but what i learned through observation. Observation (partial observation) of what actually happens, can reduce a principal's uncertainties and thereby 'leak' information. Particularly, if principles are familiar with the protocol, they may derive from certain actions taking place, that the previous action must have been a particular one, even if they did not know it before. This is the case in the example depicted in Figure 2, where principal 2 learns from observation of action d, that the choice made before must have been a. More exactly, sequences of actions which are not properly protected by the visibility restrictions ρ may lead to a refinement of the $\overset{i}{\cdots}$ class which is sufficient for i to distinguish between a state where agent's j secret key is 100 and a state where agent j's secret key is 200, even if i never participated in a direct communication over j's key. This process of learning by the refinement of the indistinguishability relations along the traces is captured in the definition of $\mathrm{A}, s \models K_i\phi$. Our logic satisfies the standard axioms for a logic of knowledge:

Theorem 1. *T - S5 (. [14, .59) $E\overline{\mu}$:*

$$\begin{array}{llll}
\mathbf{K}: K_i\phi \wedge K_i(\phi \rightarrow \psi) \rightarrow K_i\psi & & \mathbf{4}: K_i\phi \rightarrow K_iK_i\phi & \text{(positive introspection)} \\
\mathbf{T}: K_i\phi \rightarrow \phi & \text{(reflexivity)} & \mathbf{5}: \neg K_i\phi \rightarrow K_i\neg K_i\phi & \text{(negative introspection)}
\end{array}$$

The definition of satisfaction provides a model checking algorithm, that will be decidable on the finite trees generated by the semantics of our *PA* . Since the $E\overline{\mu}$ satisfaction relation on ALTSs rests on classically accepted definitions for similar but less expressive models, we expect that it should be possible to reuse and extend existing efficient model checking tools.

An interesting and non-trivial question is to find a behavioral equivalence that is characterized by $E\overline{\mu}$. We expect the answer to be some notion of bisimilarity that considers both $\xrightarrow{a}$ and $\overset{i}{\cdots}$ as transition relations. Due to the presence of past temporal operators, we may have to resort to some notion of bisimilarity that takes backward steps also into account (a notion of forward-backward or history-preserving bisimilarity).

4 Bridging the Gap: Relation to Existing Theories

In this section we show that the framework introduced in this paper is a conservative extension of the traditional process theoretic modeling on the one hand, and epistemic modeling on the other hand. To this end, we prove that the satisfaction relation defined in Section 3 preserves the standard satisfaction relations of $\overline{\mu}$ (μ-calculus with past) formulae on labeled transition systems and of E

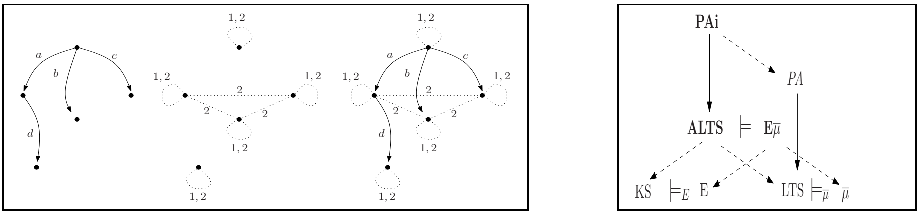


Fig. 2. Left picture: An ALTS A (rightmost), together with its projections: 'the temporal part' *lts*(A) (leftmost) and 'the epistemic part' *em*(A) (center). In *lts*(A), the points are *states*, the arrows are *transitions*. In *em*(A), points are *possible worlds* and lines are *indistinguishability* relations labeled with identities of agents. In (A), the points are states and possible worlds simultaneously. Both temporal and epistemic relations are present. The epistemic valuation in a state is given by the actions executed from the initial state to that state. In the initial state, combined temporal epistemic formulae hold like $\langle a\rangle(K_1 a^{\urcorner} \wedge \neg K_2 a^{\urcorner})$ — expressing that after an a-action, it is known to principal 1 that action a has been executed, but 2 doesn't know that. However, 2 knows that one of the actions a,b,c has been executed ($\langle a\rangle(K_2(a^{\urcorner} \vee b^{\urcorner} \vee c^{\urcorner}))$). More interestingly, after step d is executed, 2 has learned that a must have been the first step: $\langle a\rangle\langle d\rangle K_2 a^{\urcorner}$. Modeling this phenomenon of agents learning facts that were never explicitly told to them is exactly the power of epistemic logic approaches, that we took over in the combined framework. **Right picture:** Projecting into process-theoretic domain and epistemic domain. A dashed arrow $x \dashrightarrow y$ means that x is an extension of y. The arrow $x \rightarrow y$ means y is the semantic model of x. The links between ALTS, LTS, KS, $E\overline{\mu}$, $\overline{\mu}$, E are discussed in this paper. The connection with the process languages *PAi* and *PA* (a pure process theoretic formalism) is explained in [12].

(epistemic logic) formulae on Kripke structures. In Figure 2, the left picture illustrates the three semantic models discussed in this section: the existing LTS and KS, and the newly introduced ALTS. The right picture gives an overview of the connections between the various notions.

Projecting into the Process-Theoretic Domain. A Labeled Transition System (LTS) is a standard semantic domain for process-theoretic formalisms. Formally, an LTS over a set of labels L is a tuple $\langle St, \rightarrow, \checkmark, s_0\rangle$, where St is the set of operational states, $\rightarrow \subseteq St \times L \times St$ is the transition relation, $\checkmark \subseteq St$ is the termination predicate and s_0 is the initial state. It typically represents the behavior of a reactive system in terms of states and transitions. Then requirements formulated in a temporal logic are matched against this behavior in the process of model checking.

A very general logical language to reason about processes is the μ-calculus with past ($\overline{\mu}$) [23], which is obtained by leaving out the knowledge construct $K_i\phi$ from the syntax of our logic presented in Section 3. That a state s in the LTS $T = \langle S, \rightarrow, \checkmark, s_0\rangle$ satisfies a $\overline{\mu}$ formula ϕ (denoted $T, s \models_{\overline{\mu}} \phi$) is defined inductively as follows:

$$
\begin{array}{ll}
T,s \models_{\overline{\mu}} \top & \text{iff true} \\
T,s \models_{\overline{\mu}} \neg\phi & \text{iff } T,s \not\models_{\overline{\mu}} \phi \\
T,s \models_{\overline{\mu}} \phi_1 \wedge \phi_2 & \text{iff } T,s \models_{\overline{\mu}} \phi_1 \text{ and } s \models_{\overline{\mu}} \phi_2 \\
T,s \models_{\overline{\mu}} \langle a\rangle\phi & \text{iff exists } s' \in S, \text{s.t. } s \xrightarrow{a} s' \text{ and } T,s' \models_{\overline{\mu}} \phi \\
T,s \models_{\overline{\mu}} \langle\overline{a}\rangle\phi & \text{iff exists } s' \in S, \text{s.t. } s' \xrightarrow{a} s \text{ and } T,s' \models_{\overline{\mu}} \phi \\
T,s \models_{\overline{\mu}} \nu X.\phi(X) & \text{iff } s \in \bigcup\{S' \subseteq S \mid \forall s' \in S'.T,s' \models_{\overline{\mu}} \phi(X := S')\}
\end{array}
$$

We prove that the ALTS + $E\overline{\mu}$ model checking framework properly extends the LTS + $\overline{\mu}$ model checking framework, in the sense that whatever was possible in the latter, is still possible and has the same meaning in the former. This is witnessed by the fact that LTS + $\overline{\mu}$ can be immediately obtained by simply stripping the ALTS from the I relations and the $E\overline{\mu}$ logic from the epistemic operator K_i. The following theorem formalizes this.

Theorem 2. *C* *PA* p *ALTS* $\mathrm{A} = \langle St, \rightarrow, \checkmark, I, s_0\rangle$ $(p, [])$ *SOS* *F* *1. L* (q, π) A, $(p, [])$ *(. . .* $\rightarrow$ $s_0 = (p, []))$*. L* $lts(\mathrm{A}) = (St, \rightarrow, \checkmark, s_0)$*. T* *,* $\overline{\mu}$ ϕ, $\mathrm{A}, (q, \pi) \models \phi$ *ff* $lts(\mathrm{A}), q \models_{\overline{\mu}} \phi$.

This means that for purely temporal aspects of correctness, one can safely ignore the epistemic aspects of our semantics and our logic.

Projecting into the Epistemic Domain. Epistemic logics are mainly concerned with expressing subtle properties of communication acts, related to the knowledge, beliefs and intentions of communicating parties. In standard epistemic logic (following [17]), epistemic properties are validated in static rich snapshots of communications ( ), that don't express the temporal evolution of the system. The language of epistemic logic with common knowledge defined by:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid K_i\phi \mid C_J\phi$$

Here the p comes from a given set of propositional variables $\mathcal{P}rop$. These propositions represent the atomic facts the agents may know about. The subscript i ranges over a given set of agents $\mathcal{I}$, and J over subsets of $\mathcal{I}$. The standard reading of the epistemic modalities K_i and C_J is the same as ours in the previous section: "i knows that…" and "it is common knowledge among the agents in J that…", respectively.

An *(S5-)* is a Kripke structure $\langle W, \{R_i | i \in \mathcal{I}\}, V\rangle$, where W is a nonempty set of possible worlds, R_i is an equivalence relation on W for each $i \in \mathcal{I}$, and $V : \mathcal{P}rop \rightarrow \mathcal{P}(W)$ is a valuation function assigning to each propositional variable the set of worlds in which it holds. Given an epistemic model M and world $s \in W$, satisfaction ($\models_{\mathrm{E}}$) is defined recursively as follows:

$$
\begin{array}{ll}
M,s \models_{\mathrm{E}} p & \text{iff } s \in V(p) \\
M,s \models_{\mathrm{E}} \neg\phi & \text{iff it is not true that } M,s \models_{\mathrm{E}} \phi \\
M,s \models_{\mathrm{E}} \phi_1 \wedge \phi_2 & \text{iff } M,s \models_{\mathrm{E}} \phi_1 \text{ and } M,s \models_{\mathrm{E}} \phi_2 \\
M,s \models_{\mathrm{E}} K_i\phi & \text{iff for all } M,s' \in W, \text{if } sR_is' \text{ then } M,s' \models_{\mathrm{E}} \phi \\
M,s \models_{\mathrm{E}} C_J\phi & \text{iff for all } M,s' \in W, \text{if } s(\cup_{i\in J}R_i)^*s' \text{ then } M,s' \models_{\mathrm{E}} \phi
\end{array}
$$

To isolate ‘the epistemic part’ of our framework, we make suitable choices for the set of propositions, and the set of agents. In the context of our PA -processes we associate with every action $a \in \mathcal{A}ct$ a proposition $\underline{a}$ (which can be read as “a has been executed sometime before”), and we let $\mathcal{P}rop := \{\underline{a} | a \in \mathcal{A}ct\} \cup \{\top\}$. Furthermore, we let $\mathcal{I}$ be our set of identities $\mathcal{I}d$. We call the resulting logic E.

We can then say that our modeling and verification framework is also conservative when it comes to purely epistemic aspects. Namely, if we restrict the ALTS associated with a PA process to the I relations, we obtain an epistemic model where purely epistemic formulas hold exactly when they hold in the original ALTS, according to the $E\overline{\mu}$ satisfaction relation. Let us define an embedding $\mathcal{E}$: E-forms → $E\overline{\mu}$-forms of formulas into $E\overline{\mu}$ formulas, by taking $\mathcal{E}(\underline{a}) = a^{\urcorner}$ and extending from there:

$$\begin{array}{llll} \mathcal{E}(\top) & = \top & \mathcal{E}(\phi_1 \wedge \phi_2) & = \mathcal{E}(\phi_1) \wedge \mathcal{E}(\phi_2) \\ \mathcal{E}(\underline{a}) & = a^{\urcorner} & \mathcal{E}(K_i\phi) & = K_i\mathcal{E}(\phi) \\ \mathcal{E}(\neg\phi) & = \neg\mathcal{E}(\phi) & \mathcal{E}(C_J\phi) & = \nu X.(\bigwedge_{i \in J} K_i(X \wedge \phi)). \end{array}$$

The following theorem formally expresses the conservativeness of $E\overline{\mu}$ w.r.t. E.

Theorem 3. *C* *PA* p $\mathcal{A}ct$. *L* $\mathrm{A} =$
$\langle St, \rightarrow, \checkmark, I, s_0 \rangle$ *ALTS* $(p, [])$
SOS *F* *1. L* $em(\mathrm{A}) =$
$\langle St, \{\overset{i}{\cdots} | i \in \mathcal{I}d\}, V \rangle$, $\mathcal{P}rop$, $V(\underline{a}) = \{s \in St | \mathrm{A}, s \models \mathcal{E}(\underline{a})\}$
$V(\top) = St$. *T* *E* ϕ $s \in St$,
$\mathrm{A}, s \models \mathcal{E}(\phi)$ *ff* $em(\mathrm{A}), s \models_{\mathtt{E}} \phi$.

5 An Example Protocol: Dining Cryptographers

In order to illustrate the relative advantages of the combined framework compared to using exclusively the operational approach or the epistemic one, we discuss the Dining Cryptographers protocol [10], which has already been independently and extensively analyzed using both operational [26,4] and epistemic approaches [20,16,25]. The story, a metaphor for anonymous broadcast, is about three cryptographers having dinner together. The bill is paid anonymously by one of them, or by the National Security Agency (NSA). They respect each other’s right to anonymity, but they wish to find out whether the payer was NSA or not. To this end, they come up with the following protocol: each neighboring pair of cryptographers generates a shared bit, by flipping a coin; then each cryptographer computes the exclusive or (XOR) of the two bits she sees, then announces the result — or the flipped result, if she was herself the payer. The XOR of the three publicly announced results indicates whether the payer was an insider or NSA.

Model. A model of this protocol in our process language is shown in Figure 3. Inspired by the input construction in the algebraic specification language μCRL, we use $\sum_{x:\{x_1 \ldots x_n\}} P(x)$ as an abbreviation for $P(x_1) + \ldots + P(x_n)$, where $\{x_1 \ldots x_n\}$ is a finite set and $P(x_i)$ denotes the process expression $P(x)$ in which x_i has been substituted for x.

$$
\begin{array}{lcl}
Crypt(\mathrm{i}) & = & \sum_{b:Bool}\ (\ (\texttt{i})?pay(\mathrm{i},b); CryptFlip(\mathrm{i},b)\ ) \\
CryptFlip(\mathrm{i},b) & = & \sum_{c:Bool}\ (\ (\mathrm{i})flip(\mathrm{i},c); CryptShare(\mathrm{i},b,c)\ ) \\
CryptShare(\mathrm{i},b,c) & = & \sum_{d:Bool}\ (\ ((\texttt{i})!share(\mathrm{i} \mod 3+1,c) \,||\, (\mathrm{i})?share(\mathrm{i},d))\,; \\
 & & \qquad CryptBcast(\mathrm{i},b,c,d)\ ) \\
CryptBcast(\mathrm{i},b,c,d) & = & ((\texttt{i})!bcast(\mathrm{i},b\oplus c\oplus d)\,; (\mathrm{i})!bcast(\mathrm{i},b\oplus c\oplus d)) \\
 & & ||\sum_{x,y:Bool}(((\texttt{i})?bcast(\mathrm{i}+1 \mod 3+1,x) \\
 & & \qquad\qquad ||\,(\mathrm{i})?bcast(\mathrm{i} \mod 3+1,y))\,; \\
 & & \qquad\qquad nsa(\mathrm{i},\neg(b\oplus c\oplus d\oplus x\oplus y))) \\
Master & = & \quad (\texttt{M})!pay(1,\top); (\texttt{M})!pay(2,\bot); (\texttt{M})!pay(3,\bot) \\
 & & +\ (\texttt{M})!pay(1,\bot); (\texttt{M})!pay(2,\top); (\texttt{M})!pay(3,\bot) \\
 & & +\ (\texttt{M})!pay(1,\bot); (\texttt{M})!pay(2,\bot); (\texttt{M})!pay(3,\top) \\
 & & +\ (\texttt{M})!pay(1,\bot); (\texttt{M})!pay(2,\bot); (\texttt{M})!pay(3,\bot)
\end{array}
$$

Fig. 3. A *PAi* model of The Dining Cryptographers protocol. $\oplus$ denotes exclusive or.

The model is rather close to the CSP description presented in [26], the only significant difference being that the actions are annotated with identities from the set $\mathcal{I}d = \{1, 2, 3, \mathrm{M}\}$. Note that the parameters used in the basic actions and process definitions are just generic names for the concrete instances resulting from instantiating them. For example, $?pay(\mathrm{i}, b)$ is not defined in our process language but rather it stands for a number of instances such as $?pay(1, \top)$, $?pay(\mathrm{i}, \bot)$ each of which are basic actions (obtained by globally replacing i and b with a member of $\mathcal{I}d$ and $\{\bot, \top\}$ in the process definition each time). The behavior of the ith cryptographer is specified by the process $Crypt(\mathrm{i})$ and the behavior of the whole DC system as a parallel composition of $Crypt(\mathrm{i})$'s and the $Master$ process, $DC_3 = Crypt(1)\,||\,Crypt(2)\,||\,Crypt(3)\,||\,Master$. A cryptographer process executes a series of actions corresponding to the three big steps of the protocol: decide whether to pay or not, flip the coins together with the neighbors, and announce the result of XOR-ing the two coins and her own paying bit. The first step is modeled as a statement $pay(\mathrm{i}, b)$, which is in fact a communication step with the $Master$. The second step is modeled by the processes $CryptFlip(\mathrm{i})$ and $CryptShare(\mathrm{i})$. In other existing models [26,4], the shared coins are represented by separate processes, but in order to keep the specification simple, we merge the behavior of the ith coin with the behavior of the ith cryptographer. Therefore, process C (i) will execute a $flip$ action and then share the result with the right-hand neighbor, by executing an action $!share$ which will synchronize with the $?share$ from the next cryptographer in the ring. $CryptBcast$ models the last phase, announcing the result of one's computation ($!bcast$), receiving the results from all the others ($?bcast$) and concluding for itself that NSA paid or not ($nsa(\mathrm{i}, \top)$, $nsa(\mathrm{i}, \bot)$).

The renaming function ρ specifies how much of a cryptographers' actions is visible for observing parties. For any $\mathrm{i} \in \{1, 2, 3\}$ and $b \in \{\top, \bot\}$, we define $\rho(pay(\mathrm{i}, b)) = pay(\mathrm{i})$, $\rho(bcast(\mathrm{i}, b)) = bcast(\mathrm{i}, b)$, $\rho(share(\mathrm{i}, c)) = share(\mathrm{i})$, $\rho(flip(\mathrm{i}, b)) = flip(\mathrm{i})$ and $\rho(nsa(\mathrm{i}, b)) = nsa(\mathrm{i}, b)$, where $pay(1)$, $bcast(1, \top)$, ... are basic actions.

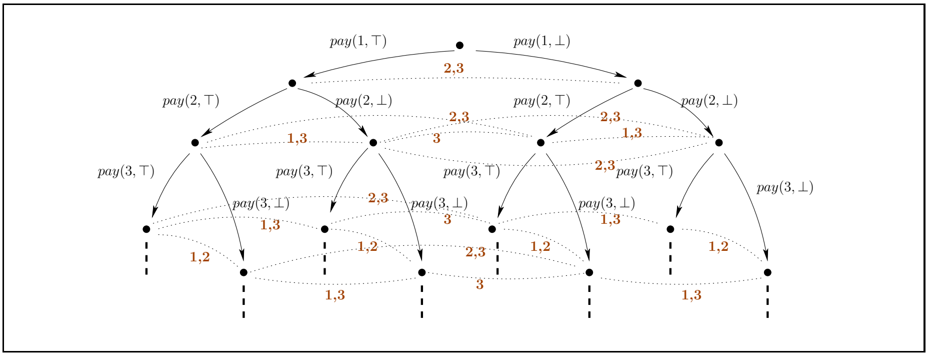


Fig. 4. A small fragment from the ALTS generated for the DC specification. For readability, we omitted some $\overset{i}{\cdots}$ relations generated by reflexivity and transitivity.

Analysis. Figure 4 shows the top part of the ALTS generated by the rules in Figure 1 from the process specification in Figure 3. We check relevant functional and epistemic properties of this protocol by matching $E\overline{\mu}$ formulas against this ALTS, as dictated by the satisfaction relation $\models$ (Definition 2).

First of all, we can check functional correctness, by asking for instance that in all executions where one of the cryptographers paid, the action $nsa(1,\top)$ is eventually observable, meaning that the first cryptographer draws the right conclusion that the payer was an insider. This requirement is a purely temporal formula, for each $i \in \{1,2,3\}$: $[pay(\mathrm{i},\top)]\bigwedge_{j\in\{1,2,3\}}[\centerdot^*]nsa(\mathrm{j},\bot)$.

Better yet, we can also check the powerful epistemic statement that "everybody knows that the payer is an insider" eventually becomes common knowledge among the three cryptographers. This is expressed as: for every $i \in \{1,2,3\}$, it holds that $[pay(\mathrm{i},\top)][\centerdot^*]C_{\{1,2,3\}}(\bigwedge_{j\in\{1,2,3\}} nsa(\mathrm{j},\bot)^{\Lsh})$.

Anonymity, the main goal of the protocol, is not expressible as a purely temporal property, but it is conveniently expressible as a temporal epistemic property. The anonymity of cryptographer i (holding in the initial state of our model) is expressed by the formula $[pay(\mathrm{i},\top)]\bigwedge_{j\in\{1,2,3\}\setminus\{\mathrm{i}\}}\neg\langle\centerdot^*\rangle K_{\mathrm{j}}(pay(\mathrm{i},\top)^{\Lsh})$. All these properties are satisfied by our PA model, according to the satisfaction relation $\models$ defined in Section 3.

Comparison to Other DC Models. PA allows a simple and operational modeling, just as intuitive as any other process language, see also for instance a CSP model [26] and a pi-calculus model [4] of the Dining Cryptographers. All these models are definitely closer to the protocol description than logic models [18,25] and moreover, they are supported by a semantics which formally links the description of a protocol to its actual behavior model.

On the other hand, epistemic logic models allow expressing and checking anonymity as epistemic formulae, which is much more natural than the equivalence checking method employed in the process theoretic approach. More precisely, operational approach to verification of anonymity requires writing down new descriptions for each anonymity property that has to be checked, because these properties are dependent on the point of view of the observer. In the ALTS that our specification generates, all points of view are simultaneously present, thus a direct and natural (epistemic) verification is possible.

6 Conclusion

Motivated by protocols and properties where much importance is given to the participating entities and not only to the actual evolution of the system — like certain security protocols, information flow — we presented a simple process language where the concept of [illegible] is explicitly present. We gave it an operational semantics in terms of an extended form of labeled transition systems and defined a satisfaction relation for properties expressed in a rich logic combining temporal and epistemic operators. The result is a specification and verification framework that combines the best parts of two complementary approaches to protocol analysis: process algebras and epistemic logics.

Our framework is particularly suitable for modeling and verification of protocols on top of authenticated secret channels, ensured for instance by a Public Key Infrastructure. In these protocols, the security threats typically do not come from an external intruder controlling the communication channels, but from the participants themselves. Examples are protocols for fair exchange, voting, auctions, anonymity. In security protocols with cryptography or active attackers, some behavioral choices are determined by the current knowledge of the principals. In particular, a principal can distinguish more traces by gaining access to keys. To properly accommodate this, our framework should be extended, possibly by allowing dynamic update of indistinguishability relation in the course of protocol execution. Note however that the current framework is just as powerful in modeling cryptography aspects as any other (traditional) process algebra. So, for these cases, more research is needed in order to find the best way of integrating the elegance of representing knowledge by indistinguishability relations with the ease of specifying the protocol operationally.

Future Work. First of all, we will build tool support for model checking $E\overline{\mu}$ properties on ALTSs. Ideally, this can be achieved by embedding the new framework in an existing verification tool-set. The starting point will be our already existing Maude prototype [1]. Then we wish to experiment with applying this technique to protocols from the categories mentioned above. On a more theoretical direction, a question is whether it is possible to extend the sequent-based compositional proof system developed for the SOS + Hennessy-Milner Logic [27] in order to cope with $E\overline{\mu}$ formulas, as well. Finally, this framework can support a direct comparison of the operational and epistemic definitions of various properties. For instance, anonymity is defined operationally as (trace) equivalence between certain processes, while epistemically it is simply a negative knowledge formula. The issue of which of these definitions is stronger, if any, is not clear yet and deserves further investigation.

Acknowledgments. We are grateful to Luca Aceto, Dave Clarke, Jan van Eijck, Michael Huth and Michel Reniers for comments on earlier versions of this work.

References

1. A Maude implementation of PAi. `http://www.win.tue.nl/~mousavi/pai.htm`
2. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation 148(1), 1–70 (1999)
3. Baltag, A.: Logics for insecure communication. In: Proc. TARK 2001, pp. 111–121 (2001)
4. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
5. Borgström, J., Kramer, S., Nestmann, U.: Calculus of cryptographic communication. In: Proc. FCS-ARSPA 2006 (2006)
6. Broadfoot, P.J.: Data Independence in the Model Checking of Security Protocols. PhD thesis, Oxford University (2001)
7. Brookes, D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. Journal of the ACM 31(3), 560–599 (1984)
8. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. In: Practical Cryptography for Data Internetworks, IEEE Computer Society Press, Los Alamitos (1996)
9. Caleiroa, C., Viganò, L., Basin, D.: On the semantics of Alice & Bob specifications of security protocols. TCS 367(1-2), 88–122 (2006)
10. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
11. Chothia, T., Orzan, S.M., Pang, J., Dashti, M.T.: A framework for automatically checking anonymity with mCRL. In: Proc. TGC 2006, LNCS (2007)
12. Dechesne, F., Mousavi, M., Orzan, S.M.: Operational and epistemic approaches to protocol analysis: Bridging the gap. Tech. Rep. CS 07-15, TU Eindhoven (2007)
13. van Eijck, J., Orzan, S.M.: Epistemic verification of anonymity. In: Proc. VODCA 2006. ENTCS, vol. 168 (2006)
14. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press, Cambridge (1995)
15. Gerbrandy, J., Groeneveld, W.: Reasoning about information change. Journal of Logic Language and Information 6, 147–169 (1997)
16. Halpern, J.Y., O'Neill, K.R.: Anonymity and information hiding in multiagent systems. Journal of Computer Security, 483–514 (2005)
17. Hintikka, J.: Knowledge and Belief. Cornell University Press (1962)
18. van der Hoek, W., Wooldridge, M.: Model checking knowledge and time. In: Bošnački, D., Leue, S. (eds.) Model Checking Software. LNCS, vol. 2318, pp. 95–111. Springer, Heidelberg (2002)
19. Hommersom, A., Meyer, J.-J., de Vink, E.P.: Update semantics of security protocols. Synthese 142, 229–267 (2004)
20. Hughes, D., Shmatikov, V.: Information hiding, anonymity and privacy: A modular approach. Journal of Computer Security 12(1), 3–36 (2004)
21. Kramer, S.: Logical concepts in cryptography. Cryptology ePrint Archive, Report 2006/262 (2006), `http://eprint.iacr.org/2006/262`

22. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
23. Nielsen, M.: Reasoning about the past. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 117–128. Springer, Heidelberg (1998)
24. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60, 17–139 (2004)
25. Raimondi, F., Lomuscio, A.: Automatic verification of deontic interpreted systems by model checking via OBDD's. Journal of Applied Logic (in Press, 2006)
26. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) ESORICS 1996. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)
27. Simpson, A.K.: Sequent calculi for process verification: Hennessy-Milner logic for an arbitrary GSOS. Journal of Logic and Algebraic Programming, 60–61, 287–322

Protocol Verification Via Rigid/Flexible Resolution

Stépphanie Delaune[2], Hai Lin[1], and Christopher Lynch[1]

[1] Clarkson University, Potsdam, NY 13699-5815, USA
[2] LORIA, CNRS & INRIA project Cassis, Nancy, France

Abstract. We propose a decision procedure, *i.e.* an inference system for clauses containing rigid and flexible variables. Rigid variables are only allowed to have one instantiation, whereas flexible variables are allowed as many instantiations as desired. We assume a set of clauses containing only rigid variables together with a set of clauses containing only flexible variables. When the flexible clauses fall into a particular class, we propose an inference system based on ordered resolution that is sound and complete and for which the inference procedure will halt.

An interest in this form of problem is for cryptographic protocol verification for a bounded number of protocol instances. Our class allows us to obtain a generic decidability result for a large class of cryptographic protocols that may use for instance CBC (Cipher Block Chaining) encryption and blind signature.

1 Introduction

In refutational theorem proving, we must determine if a set of clauses is unsatisfiable. Clauses are implicitly universally quantified, so we allow an unbounded number of renamed copies of each clause. Each copy represents a different instance of the original clause. In proving *rigid unsatisfiability* [1], we ask whether a set of clauses is unsatisfiable, allowing only one instance of each clause. For example, the set of clauses

$$\{I(a),\ I(b),\ \neg I(x) \vee I(f(x)),\ \neg I(f(a)) \vee \neg I(f(b))\}$$

is unsatisfiable but is not rigidly unsatisfiable, because proving unsatisfiability requires two instances of the third clause.

Rigid theorem proving has been used in tableau style theorem provers, but not often in saturation and resolution-based theorem proving. It is generally used to prove unsatisfiability of a set of clauses, by repeatedly solving the rigid satisfiability problem, and continually adding new renamed copies of each clause, because a set of formulas is unsatisfiable if and only if there is a finite number of copies of each clause which make it rigidly unsatisfiable.

Here we give a new use for rigid theorem proving. For example, in cryptographic analysis, we get a reasonable confidence in the protocol security if we

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 242–256, 2007.
© Springer-Verlag Berlin Heidelberg 2007

show that, say after any 2 or 3 sessions of the protocols there is no attack. Since the problem is well-known to be undecidable for an unbounded number of protocol instances, many papers study the security problem under this assumption [4,5,15].

However, the problem is more complicated than this. We need rigid clauses to restrict the number of times an action is performed. But simultaneously, we may also want to model certain properties. For example, in cryptographic protocol analysis, the intruder has the ability to construct and deconstruct messages, to encrypt messages and to decrypt messages if the key is known. It is not realistic to bound the number of times that these properties are applied. Therefore, when describing processes, it makes sense to use rigid variables to model actions which are performed a bounded number of times, and flexible variables to model properties which it makes no sense to bound. Therefore, in this paper we introduce the idea of rigid theorem proving modulo a flexible theory. We will use unary predicates to model the state of the world. Actions can be relatively complicated, whereas properties should be simple, according to the theory we are working in.

Unsatisfiability is an undecidable property for first order logic, but rigid unsatisfiability is Σ_2^p-complete, while for Horn clauses, rigid unsatisfiability is [illegible]-complete [12]. In this paper, we consider Horn clauses, since that is a natural way to model actions. Our intended application is cryptographic protocol analysis, and Horn clauses are sufficient to model many interesting properties.

We show that rigid/flexible Horn clauses unsatisfiability is decidable for a large class of rigid theories (including those that model cryptographic protocols) and a simple class of flexible theories. The flexible theories we consider model the standard Dolev-Yao model [10] for cryptographic protocol analysis, but also more complicated theories, such as the prefix theory and blind signature theory. The prefix theory related to Cipher Block Chaining (CBC) is important since it is a common encryption mode. It allows an attacker to get from an encrypted message the encryption of any of its prefixes. The blind signature scheme is employed in the design of several E-voting protocols ([illegible] [11]). It allows an agent ([illegible] a voter) to have a message ([illegible] his vote) signed blindly by an authority. This scheme offers an intruder new attacks opportunities.

Ordered Resolution is a powerful inference system for first order logic. So, in this paper, we define a modification of Ordered Resolution, called [illegible]. The main result of this paper is the definition of the rigid/flexible unsatisfiability problem, the decidability of a resolution inference system for a useful class of problems, and its application to cryptographic protocol analysis. After the preliminaries (see Section 2), we define how we model protocols in Section 3. Next we define the security problem we want to solve. We give our inference system (Section 5), followed by a termination argument (Section 6) and a completeness argument (Section 7). Then we discuss related and future work. Missing proofs and lemmas can be found at http://people.clarkson.edu/~linh/SHC07.pdf

2 Preliminaries

2.1 Term Algebra

Let $\mathcal{F}$ be a finite set of function symbols with arity and $\mathcal{X}$ be an infinite set of variables. The set of terms on $\mathcal{F}$ and $\mathcal{X}$ is denoted $\mathcal{T}(\mathcal{F},\mathcal{X})$ and $\mathcal{T}(\mathcal{F})$ for ground terms. We note *Var*(t) the set of variables occurring in $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$. We distinguish two kinds of variables, the *rigid* ones denoted by $X, Y, \ldots$, and the *non rigid* ones also called *flexible* that we denote by $x, y, \ldots$. Rigid variables are only allowed to have one instantiation, whereas non-rigid variables are allowed as many instantiations as desired. A *substitution* σ is a mapping from a finite subset of $\mathcal{X}$ called its domain and written *Dom*(σ) to $\mathcal{T}(\mathcal{F},\mathcal{X})$. Substitutions are extended to endomorphisms of $\mathcal{T}(\mathcal{F},\mathcal{X})$ as usual. If t is a term then $t\sigma$ obtained by applying σ to t is defined as usual. A substitution σ is *grounding* for t if $t\sigma$ is ground. If M and N are terms then a unifier of M and N is a substitution σ such that $M\sigma = N\sigma$. It is well-known that unifiable terms have a *most general unifier* (mgu), *i.e.* a substitution σ such that $\sigma \leq \tau$ (there exists ρ such that $\sigma\rho = \tau$) for every other unifier τ of s and t.

2.2 Clauses

Let $\mathcal{P}$ be a finite set of unary predicate symbols. We assume given a well-founded and total precedence ordering on $\mathcal{P}$, denoted by $>$. *Atoms* A are of the form $I(t)$ where $I \in \mathcal{P}$ and t is a term. *Literals* L are either positive literals $+A$ (or simply A) or negative literals $-A$ where A is an atom. A *clause* is a finite set of literals. We denote by *At*(C) the set of atoms that appear in a clause C. If C_1 and C_2 are clauses, $C_1 \vee C_2$ denotes $C_1 \cup C_2$ and we denote by $\Box$ the empty clause. A *Horn clause* is a clause that contains at most one positive literal that is called the *head*. For Horn clauses we use the alternative notation $A_1, \ldots, A_n \Rightarrow A_0$ to denote $-A_1 \vee \ldots \vee -A_n \vee A_0$. A clause with no head is called a *query*. A clause is a *non rigid* or *flexible clause* (resp. rigid) if it only involves flexible (resp. rigid) variables. A *constrained Horn clause*, which is of the form $\Gamma \Rightarrow A_0 \; [\![C]\!]$, is a Horn clause together with a constraint. The constraint C is a set of equations between terms in $\mathcal{T}(\mathcal{F},\mathcal{X})$. The constraint is omitted when C is empty.

A *partial interpretation* $\mathcal{J}$ w.r.t. $\prec$ is a set of ground literals such that $A \in \mathcal{J}$ iff $-A \notin \mathcal{J}$, and if $+A \in \mathcal{J}$ (resp. $-A \in \mathcal{J}$) and $B \prec A$ then $+B \in \mathcal{J}$ or $-B \in \mathcal{J}$. A ground clause C is *false* in $\mathcal{J}$ if, for every literal $\pm A \in C$, the opposite literal $\mp A$ belongs to $\mathcal{J}$. A clause C is [illegible] in the partial interpretation $\mathcal{J}$ if there exists a substitution θ grounding for C such that $C\theta$ is false in $\mathcal{J}$. Let $\mathcal{C}_r$ be a set of rigid Horn clauses and $\mathcal{C}_f$ be a set of flexible Horn clauses which contains only flexible variables. We say that $\mathcal{C}_r$ is [illegible] $\mathcal{C}_f$ if there exists a substitution θ grounding for $\mathcal{C}_r$ such that $\mathcal{C}_r\theta \cup \mathcal{C}_f$ is unsatisfiable.

Example 1. Consider the following set of Horn clauses:

$$\mathcal{C} := \{ \Rightarrow I(a);\ \Rightarrow I(b);\ I(x) \Rightarrow I(f(x));\ I(f(a)), I(f(b)) \Rightarrow \bot \}$$

If the clause $I(x) \Rightarrow I(f(x))$ is flexible then $\mathcal{C}$ is unsatisfiable whereas it is satisfiable if the clause is assumed to be rigid.

2.3 Ordered Resolution

We consider a liftable[1] ordering $\prec$, total on ground atoms. This property is crucial for the completeness of ordered resolution. Let A_1, A_2 and B be three atoms, C_1 and C_2 be Horn clauses and $\sigma = \mathsf{mgu}(A_1, A_2)$.
The [illegible] is defined by:

$$\frac{C_1 \overset{\mathsf{def}}{=} \Gamma_1 \Rightarrow A_1 \qquad \Gamma_2, A_2 \Rightarrow B \overset{\mathsf{def}}{=} C_2}{\Gamma_1\sigma, \Gamma_2\sigma \Rightarrow B\sigma}$$

[illegible] (w.r.t. $\prec$) requires that $A_1\sigma$ is [illegible] in $\Gamma_1\sigma, \Gamma_2\sigma \Rightarrow B\sigma$, [illegible] there is no atom $A \in$ [illegible] $(\Gamma_1\sigma, \Gamma_2\sigma \Rightarrow B\sigma)$ such that $A_1\sigma \prec A$. The clause $\Gamma_1\sigma, \Gamma_2\sigma \Rightarrow B\sigma$ is called the [illegible] of C_1 and C_2. In the remainder of the paper, we will consider the embedding ordering $\preceq_{emb}$.

$t \preceq'_{emb} s$ if one of the following is true:

- $t = s$
- s is of the form $f(s_1, \ldots, s_n)$ and there exists some $1 \leq i \leq n$ s.t. $t \preceq'_{emb} s_i$.
- t is of the form $f(t_1, \ldots, t_n)$, s is of the form $f(s_1, \ldots, s_n)$ and for all $1 \leq i \leq n$: $t_i \preceq'_{emb} s_i$.

For instance, $f(x, z) \preceq_{emb} f(g(x, y), z)$. We assume the embedding ordering is extended to a total ordering. It is extended to atoms as follows: $P(t) \preceq_{emb} P'(t')$ if and only if $t \preceq_{emb} t'$. It can also be extended to clauses by considering clauses as multi-sets of atoms.

A set S of (flexible) clauses is [illegible] by ordered resolution w.r.t. $\prec_{emb}$ if for every resolvent C obtained by ordered resolution from S and for every partial ordered interpretation $\mathcal{J}$, if C is unsatisfiable in $\mathcal{J}$ then S is unsatisfiable in $\mathcal{J}$.

3 Modeling Protocols

The aim of this section is to introduce the class of clauses we consider. We also explain how protocols can be modeled using these clauses.

3.1 Intruder Clauses

The intruder is able to analyze messages that pass over the network. For example, if he sees an encrypted message and he knows the decryption key, he can decrypt it. This can be modeled by $I(\{x\}_y), I(y) \Rightarrow I(x)$. Intuitively, the predicate I represents the knowledge of the intruder and $I(m)$ means that the intruder knows the message m. Other examples of clauses, very useful and classical to model the intruder capabilities are given below:

---

[1] An order $\prec$ is said *liftable* if for any two terms u, v and for any substitutions θ, $u \prec v$ implies $u\theta \prec v\theta$.

$$\mathcal{C}_{\mathsf{DY}} := \left\{\begin{array}{rcll} I(x),\ I(y) & \Rightarrow & I(\{x\}_y) & \text{symmetric encryption} \\ I(\{x\}_y),\ I(y) & \Rightarrow & I(x) & \text{symmetric decryption} \\ I(x),\ I(y) & \Rightarrow & I(\langle x, y\rangle) & \text{pairing} \\ I(\langle x, y\rangle) & \Rightarrow & I(x) & \text{first projection} \\ I(\langle x, y\rangle) & \Rightarrow & I(y) & \text{second projection} \\ I(x),\ I(y) & \Rightarrow & I(\mathsf{sgn}(x, y)) & \text{signing} \\ I(\mathsf{sgn}(x, y)),\ I(y) & \Rightarrow & I(x) & \text{verifying the signature} \end{array}\right.$$

Note that each of these clauses contains at most one function symbol. That is why we introduce the following definition.

Definition 1 (Intruder clause). [illegible] $\mathcal{P}$ [illegible] intruder clause [illegible] $\mathcal{P}$ [illegible] fl[illegible]

$$\pm P_0(f(x_1, \ldots, x_n)) \vee \textstyle\bigvee_{j=1}^{m} \pm P_j(x_{i_j}).$$

[illegible]

- $x_{i_j} \in \{x_1, \ldots, x_n\}$ [illegible] j [illegible] $1 \leq j \leq m$
- $P_j \in \mathcal{P}$ [illegible] j [illegible] $0 \leq j \leq m$

[illegible] $P_0(f(x_1, \cdots, x_n))$ [illegible] constructor clause [illegible] destructor clause

Such kind of clauses do not allow us to model some cryptographic primitives which are useful. This is our main motivation to extend this class by adding some special clauses.

3.2 Extending the Intruder Power

We want to deal with some clauses that do not satisfy the conditions given in Definition 1. The intruder clauses $\mathcal{C}_{\mathsf{DY}}$ given in Section 3.1 represents the classical Dolev-Yao intruder [10], [illegible] assuming [illegible]. In particular, it is assumed that an attacker can not learn anything from an encrypted message $\{m\}_k$ except if he knows the decryption key. This assumption is too strong in some situations. Depending on the implementation of the cryptographic primitives, the attacker may be able to deduce more messages.

[illegible] fi [illegible] ffi [illegible] The prefix property is the ability of an intruder to get from an encrypted message the encryption of any of its prefixes: from a message $\{\langle x, y\rangle\}_z$, he can deduce the message $\{x\}_z$. This can be easily encoded in our formalism by the clause:

$$C_{\mathsf{pre}} := I(\{\langle x_1, x_2\rangle\}_{y_2}) \Rightarrow I(\{x_1\}_{y_2}).$$

This property strongly depends on the encryption algorithm. A relatively good method of encrypting several blocks of data is Cipher Block Chaining (CBC). In such a system, the encryption of message block sequence $P_1P_2\cdots P_n$[2] with the key K is $C_0C_1\cdots C_n$ where $C_0 = I$ (initialization block) and $C_i = \{C_{i-1} \oplus P_i\}_K$. Hence, if $C_0C_1\cdots\cdots C_n = \{P_1P_2\cdots P_n\}_K$ then $C_0C_1C_2\cdots C_i = \{P_1P_2\cdots P_i\}_K$, that is to say an intruder can get $\{x\}_z$ from $\{\langle x, y\rangle\}_z$ if the length of x is a multiple of the block length used by the cryptographic algorithm. This property can be used by an intruder to mount some attacks on several well-known protocols [6] (Denning-Sacco protocol [8], Needham-Schroeder symmetric key protocol [16]). In [14], S. Kremer and M. Ryan notice that one can also reuse any postfix $C_{i+1}\cdots C_n$ of a cipher $C_0C_1C_2\cdots C_n$ as a valid cipher. This can also be modeled by a Horn clause:

$$C_{\mathsf{suf}} := I(\{\langle x_1, x_2\rangle\}_{y_2}) \Rightarrow I(\{x_2\}_{y_2}).$$

Digital signatures are often used in cryptographic protocols. A particular class of signature scheme is the blind signature scheme which is often used in electronic voting protocols (the protocol due to Fujioka [11]). The idea of the protocol is that the voter first sends his vote hidden with a blinding factor r: $\mathsf{bld}(v, r)$. This is used to ensure that the value of his vote will not be revealed to anyone even the administrator who has to sign his vote. Then the administrator signs this message without knowing exactly the message he is signing and he sends back the result, $\mathsf{sgn}(\mathsf{bld}(v, r), ska)$ to the voter. Now the voter can unblind the message to obtain $\mathsf{sgn}(v, ska)$, the signature of his vote. This property of blind signature can be modeled in Horn clauses by considering the following clauses:

$$\mathcal{C}_{\mathsf{bld}} := \begin{cases} I(x),\ I(y) \Rightarrow I(\mathsf{bld}(x, y)) \\ I(\mathsf{bld}(x, y)),\ I(y) \Rightarrow I(x) \end{cases}$$

$$C_{\mathsf{sgn}} := I(\mathsf{sgn}(\mathsf{bld}(x_1, x_2), y_2), I(x_2) \ \Rightarrow\ I(\mathsf{sgn}(x_1, y_2)))$$

Note that the two clauses in $\mathcal{C}_{\mathsf{bld}}$ are intruder clauses whereas the last one is not. This is our main motivation to be able to deal with more complex clauses than the ones we have introduced previously.

Definition 2 (special clause). $\mathcal{P}$ fi special clause $\mathcal{P}$ fl

$$P_0(f_0(g(x_1, \ldots, x_p), y_2, \ldots, y_q)), P_1(x_{i_1}), \ldots, P_m(x_{i_m}) \Rightarrow P_{m+1}(f_0(x_{i_0}, y_2, \ldots, y_q))$$

where

- $f_0 \neq g$
- $x_{i_j} \in \{x_1, \ldots, x_p\}$ j $0 \leq j \leq m$
- $P_j \in \mathcal{P}$ j $0 \leq j \leq m+1$

j-special clause $x_{i_0} = x_j$

The Horn clauses C_{pre} and C_{sgn} are 1-special clauses whereas C_{suf} is a 2-special clause.

---

[2] $P_1P_2P_3\cdots P_n$ should be written $\langle\ldots\langle\langle P_1, P_2\rangle, P_3\rangle, \cdots, P_n\rangle$.

3.3 Protocol Clauses

As a running example we consider the Needham-Schroeder symmetric key protocol [16]. However, note that our definition of a protocol clause (see Definition 3) is general enough to deal with many other protocols. This protocol aims at establishing a fresh shared symmetric key K_{ab} and mutually authenticating the participants: in every session, the value of K_{ab} has to be known only by the participants playing the roles of A, B and S in that session. Messages 1 to 3 perform the distribution of the fresh shared symmetric key K_{ab} and messages 4 and 5 are for mutual authentication of A and B. The fields N_a and N_b are nonces, random numbers, generated by A and B respectively. In the first message, A tells the server that he wants to communicate with B. The server replies with a message (message 2) which contains the nonce N_a, a fresh session K_{ab} and a ciphertext, namely $\{K_{ab}, A\}_{K_{bs}}$, that A has to forward to B. This is done in the third step. Finally, B challenges A by sending him a nonce N_b encrypted with the session key K_{ab} and A answers to this challenge.

$$\begin{array}{rcccl}
1. & A & \rightarrow & S & : A, B, N_a \\
2. & S & \rightarrow & A & : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}} \\
3. & A & \rightarrow & B & : \{K_{ab}, A\}_{K_{bs}} \\
4. & B & \rightarrow & A & : \{N_b\}_{K_{ab}} \\
5. & A & \rightarrow & B & : \{\mathsf{succ}(N_b)\}_{K_{ab}}
\end{array}$$

The operator succ represents the increment operation. Note that the clauses $I(x) \Rightarrow I(\mathsf{succ}(x))$ and $I(\mathsf{succ}(x)) \Rightarrow I(x)$ allow us to model the fact that the attacker is able to increment or decrement an integer. Such clauses are intruder clauses. In our setting, we can model the role of A played by a with the agent b as follows:

$$\text{Role } A := \left\{ \begin{array}{rl}
 & \Rightarrow I_1(\langle\langle a, b\rangle, n_a\rangle) \\
I_2(\{\langle\langle\langle n_a, b\rangle, X_2\rangle, X_3\rangle\}_{K_s(a)}) & \Rightarrow I_3(X_3) \\
I_2(\{\langle\langle\langle n_a, b\rangle, X_2\rangle, X_3\rangle\}_{K_s(a)}), I_4(\{X_5\}_{X_2}) & \Rightarrow I_5(\{\mathsf{succ}(X_5)\}_{X_2})
\end{array} \right.$$

These rigid clauses represent an instance of role A. The role is executed by agent a who wants to communicate with b. Every variable is rigid and different clauses can share rigid variables. In such rules subscripts are used to ensure that messages sent at some step cannot be used to trigger a rule that has to be executed before. Hence, if we consider a single session of the protocol, a message m sent at the jth step of the protocol would be denoted by $I_j(m)$. Moreover, in the third clause, we repeat the atom $I_2(\{\langle\langle\langle n_a, b\rangle, X_2\rangle, X_3\rangle\}_{K_s(a)})$ on the left hand side. This is useless in this particular situation. However, in some protocols, this can be used to ensure condition 1 of Definition 3.

Then to check whether the secrecy of a message m is preserved we add a Horn clause. For instance if we want to check the secrecy of the nonce n_a, we add the goal $I_5(n_a) \Rightarrow$ to the set of rigid Horn clauses modeling the protocol. Moreover, we can give some initial knowledge to the intruder by adding some clauses such as $I_0(a)$, $I_0(b)$, … All these clauses are [illegible].

Definition 3 (Protocol clause). [illegible] $P_1(u_1), \ldots, P_n(u_n) \Rightarrow P_0(u_0)$ [illegible]
[illegible] $(u_0) \subseteq$ [illegible] $(\{u_1, \ldots, u_n\})$
[illegible] $P_0 \geq P_i$ [illegible] i [illegible] $1 \leq i \leq n$

4 Security Problem

Definition 4 (intruder theory). [illegible] I [illegible] $\mathcal{C}_I$ [illegible] I [illegible] C_S [illegible] I [illegible] $\mathcal{C}_I \cup \{C_S\}$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $\mathcal{C}_I \cup \{C_S\}$ [illegible] $\mathcal{P}$ [illegible] $P_1(u_1), \ldots, P_n(u_n) \Rightarrow P_0(u_0)$ [illegible]

- $P_0, \ldots, P_n \in \mathcal{P}$ [illegible] $P_0 \geq P_i$ [illegible] i [illegible] $1 \leq i \leq n$ [illegible]
- $I(u_1), \ldots, I(u_n) \Rightarrow I(u_0) \in \mathcal{C}_I \cup \{C_S, I(x) \Rightarrow I(x)\}$

The intruder knowledge always increases. Hence, if he knows message m at step k, he also knows m at step $k+1$. This is why we add the clause $I_i(x) \Rightarrow I_j(x)$ with $i \leq j$ to our intruder theory. We also assume that the operations available to the attacker are the same at each step.

It is easy to establish the following lemma which states that an intruder theory generated from a finite set of clauses that is saturated w.r.t. $\prec$ is also saturated w.r.t. $\prec$. This will allow us to not consider resolution steps between two flexible clauses in our resolution inference systems presented in Section 5.

Lemma 1. [illegible] $\mathcal{C}_I$ [illegible] I [illegible] C_S [illegible] I [illegible] $\mathcal{C}_I \cup \{C_S\}$ [illegible] $\prec$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $\prec$

[illegible] $\mathcal{C}_{\mathsf{DY}} \cup \{C_{\mathsf{pre}}\}$, $\mathcal{C}_{\mathsf{DY}} \cup \{C_{\mathsf{suf}}\}$ and $\mathcal{C}_{\mathsf{DY}} \cup \mathcal{C}_{\mathsf{bld}} \cup \{C_{\mathsf{sgn}}\}$ are sets of clauses which are saturated by ordered resolution w.r.t. $\prec_{emb}$. Hence, they can be used to define an intruder theory.

In this paper we are interested in the so-called insecurity problem.

Insecurity problem

Input: A finite set $\mathcal{C}_P$ of protocol clauses built on $\mathcal{P}$, a finite set $\mathcal{C}_I$ of intruder clauses built on I and C_S be a special clause built on I such that $\mathcal{C}_I \cup \{C_S\}$ is saturated by ordered resolution. Let $\mathcal{I} = \mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$.

Output: Is $\mathcal{C}_P$ unsatisfiable modulo $\mathcal{I}$? In other words, does there exist a substitution θ grounding for $\mathcal{C}_P$ such that $\mathcal{I} \cup \mathcal{C}_P\theta$ is unsatisfiable?

[illegible] Beyond other existing attacks, O. Pereira and J.-J. Quisquater [17] presented the following flaw, based on the prefix property.

$i.1 \quad a \rightarrow s \quad : a, b, n_a$
$i.2 \quad s \rightarrow a \quad : \{n_a, b, k_{ab}, \{k_{ab}, a\}_{k_{bs}}\}_{k_{as}}$
$ii.3 \quad I(b) \rightarrow a \quad : \{n_a, b\}_{k_{as}}$
$ii.4 \quad a \rightarrow I(b) : \{n'_a\}_{n_a}$
$ii.5 \quad I(b) \rightarrow a \quad : \{\mathsf{succ}(n'_a)\}_{n_a}$

In a first session i, the attacker can listen to $\{n_a, b, k_{ab}, \{k_{ab}, a\}_{k_{bs}}\}_{k_{as}}$ and then, using the prefix property, he can compute $\{n_a, b\}_{k_{as}}$. This message can be sent at step 3 to the agent a who plays the role B in the session ii. Thus a can be fooled into accepting the publicly known nonce n_a as a secret key shared with b.

Such an attack can be retrieved in our formalism by considering one instance of the role B played by the agent a with b. This corresponds to the following two rigid clauses

$$I_3(\{\langle X, b\rangle\}_{K_s(a)}) \;\Rightarrow\; I_4(\{n'_a\}_X) \quad\text{and}\quad I_5(\{\mathsf{succ}(n'_a)\}_X) \Rightarrow I_6(\mathsf{end})$$

where end is a special constant used to model the fact that the agent a has executed his session until the end. We assume that the attacker has listened to a previous session between the two honest participants a and b, and has the following knowledge: $I_0(a),\ I_0(b),\ I_0(n_a),\ I_0(\{n_a, b, k_{ab}, \{k_{ab}, a\}_{K_s(b)}\}_{K_s(a)})$. Since a is playing a session with an honest agent b, if a executes his session until the end with b, the session key he received, represented by X, has to remain secret. Hence, we consider the following protocol clause in order to model the security property: $I_6(\mathsf{end}),\ I_6(X) \Rightarrow \bot$. We can easily show that the set of rigid clauses described above is unsatisfiable modulo $\mathcal{I}_{\mathcal{Q}}(\mathcal{C}_{\mathsf{DY}} \cup \{C_{\mathsf{pre}}, I(x) \Rightarrow I(\mathsf{succ}(x))\})$ where $\mathcal{Q} = \{I_0, \ldots, I_6\}$.

The remainder of the paper is devoted to a proof that the insecurity problem is decidable. As a corollary, the insecurity problem in presence of a bounded number of sessions is decidable for the prefix intruder theory, for the suffix intruder theory and also for the blind signature intruder theory. Note that the conditions on what we have called a protocol clause are not restrictive w.r.t. our application. Hence, we can deal with a large class of cryptographic protocols. We first introduce and motivate our resolution method that is sound (Section 5). Then we establish termination (Section 6) and completeness (Section 7).

5 Inference Systems

First we present an inference system, $\mathcal{I}_{\mathsf{Ror}}$, which is a natural candidate to solve our problem (see Section 5.1). It is clearly sound and complete, but does not terminate (see Section 5.2). In Section 5.3 we provide our inference system, denoted by $\mathcal{I}_{\mathsf{Pr}}$, that allows us to solve the insecurity problem.

5.1 Constrained Rigid Ordered Resolution $\mathcal{I}_{\mathsf{Ror}}$

This inference system contains two kinds of inference rules. One allowing us to perform a resolution step between rigid clauses and two others allowing us to perform resolution steps between a rigid clause and a flexible one. Note that since we have assumed that our intruder theory is saturated w.r.t. $\prec$, we do not have to consider any resolution step between two intruder clauses.

Resolution involving two rigid clauses.

$$\frac{S \cup \{\Gamma_1 \Rightarrow A_1 \;;\; \Gamma_2, A_2 \Rightarrow B\} \;\; [\![C]\!]}{S \cup \{\Gamma_1 \Rightarrow A_1 \;;\; \Gamma_2, A_2 \Rightarrow B \;;\; \Gamma_1, \Gamma_2 \Rightarrow B\} \;\; [\![C, A_1 = A_2]\!]}\;(\mathsf{RR_c})$$

Resolution involving a flexible clause.

$$\frac{S \cup \{\Gamma_2, A_2 \Rightarrow B\} \ \ [\![C]\!]}{S \cup \{\Gamma_2, A_2 \Rightarrow B \ ; \ \Gamma_1, \Gamma_2 \Rightarrow B\} \ \ [\![C, A_1 = A_2]\!]} \ (\mathsf{FR_c})$$

[illegible] $\Gamma_1 \Rightarrow A_1$ is a fresh renaming of a clause in $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$, [illegible] $A_1\sigma$ is strictly maximal in $(\Gamma_1 \Rightarrow A_1)\sigma$ where σ is the mgu of $C, A_1 = A_2$.

$$\frac{S \cup \{\Gamma_1 \Rightarrow A_1\} \ \ [\![C]\!]}{S \cup \{\Gamma_1 \Rightarrow A_1 \ ; \ \Gamma_1, \Gamma_2 \Rightarrow B\} \ \ [\![C, A_1 = A_2]\!]} \ (\mathsf{RF_c})$$

[illegible] $\Gamma_2, A_2 \Rightarrow B$ is a fresh renaming of a clause in $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$, [illegible] $A_2\sigma$ is maximal in $(\Gamma_2, A_2 \Rightarrow B)\sigma$ where σ is the mgu of $C, A_1 = A_2$.

Definition 5 (constrained rigid global derivation). [illegible] constrained rigid global derivation [illegible] $\mathcal{I}$ [illegible] $S_1 \ [\![C_1]\!] \Mapsto \ldots \Mapsto S_n \ [\![C_n]\!]$ [illegible] S_i [illegible] C_i [illegible] $S_{i+1} \ [\![C_{i+1}]\!]$ [illegible] $S_i \ [\![C_i]\!]$ [illegible] $\mathcal{I}_{\mathsf{Ror}}$

5.2 Difficulties with Termination

The first problem with termination is because we can perform a resolution step between a literal $I(X)$ of a rigid clause and an intruder clause such as $I(\langle x_1, x_2 \rangle) \Rightarrow I(x_1)$. We obtain a new rigid literal $I(X_1)$ on which we can apply exactly the same inference rule. Hence a first solution consists in removing the possibility to perform a resolution step between a flexible clause and a rigid one when this allows us to introduce new rigid variables. This can be done by adding some side conditions on the rules (RF) and (FR). We retrieve termination but we lose completeness as illustrated by the example below:

[illegible] Consider the intruder theory made up of the rule $\mathcal{C}_{\mathsf{DY}} \cup \{C_{\mathsf{pre}}\}$ and the following set of rigid clauses:

$$\mathcal{C}_P := \begin{cases} \Rightarrow I(a) & I(X) \Rightarrow I(\{X\}_{\mathsf{h}(\langle a,b \rangle)}) \\ \Rightarrow I(b) & I(\{a\}_{\mathsf{h}(X)}) \Rightarrow \end{cases}$$

This set is unsatisfiable: consider the following substitution $\theta = \{X \mapsto \langle a, b \rangle\}$. However, we are not able to derive the empty clause with the inference system $\mathcal{I}_{\mathsf{Ror}}$ if we forbid resolution steps that introduce new rigid variables ([illegible] the one with $I(X) \Rightarrow I(\{X\}_{\mathsf{h}(\langle a,b \rangle)})$ and C_{pre}). To retrieve completeness we have to introduce new inference rules that we have called [illegible].

5.3 Our Resolution Method: Protocol Resolution

By taking into account all these considerations, we obtain the inference system $\mathcal{I}_{\mathsf{Pr}}$ described below.

Resolution involving two rigid clauses.

$$\frac{S \cup \{\Gamma_1 \Rightarrow A_1 \ ; \ \Gamma_2, A_2 \Rightarrow B\}}{(S \cup \{\Gamma_1 \Rightarrow A_1 \ ; \ \Gamma_2, A_2 \Rightarrow B \ ; \ \Gamma_1, \Gamma_2 \Rightarrow B\})\sigma} \ (\mathsf{RR_p})$$

where $\sigma = \mathsf{mgu}(A_1, A_2)$.

$$\frac{S \cup \{\Gamma_1 \Rightarrow P(f_0(X, t_2, \ldots, t_q)) \ ; \ \Gamma_2, P(f_0(t'_1, \ldots, t'_q)) \Rightarrow B\}}{(S \cup \{\Gamma_1 \Rightarrow P(f_0(X, t'_2, \ldots, t_q)) \ ; \ \Gamma_2, P(f_0(t'_1, \ldots, t'_q)) \Rightarrow B\})\sigma} \ (\mathsf{RR_p} - \mathsf{Inst1})$$

where σ is the mgu of $\{t_i = t'_i \mid 2 \leq i \leq q\}$.

$$\frac{S \cup \{\Gamma_1 \Rightarrow P(f_0(X, t_2, \ldots, t_q)) \ ; \ \Gamma_2, P(t) \Rightarrow B\}}{(S \cup \{\Gamma_1 \Rightarrow P(f_0(X, t_2, \ldots, t_q)) \ ; \ \Gamma_2, P(t) \Rightarrow B\})\sigma} \ (\mathsf{RR_p} - \mathsf{Inst2})$$

where $\sigma = \mathsf{mgu}(t, t')$ and $t' \preceq_{emb} t_i$ for some $i \in \{2, \ldots, q\}$.

Resolution involving a flexible clause.

$$\frac{S \cup \{\Gamma_2, A_2 \Rightarrow B\}}{(S \cup \{\Gamma_2, A_2 \Rightarrow B \ ; \ \Gamma_1, \Gamma_2 \Rightarrow B\})\sigma} \ (\mathsf{FR_p})$$

where [illegible] $\Gamma_1 \Rightarrow A_1$ is a fresh renaming of a clause in $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$, [illegible] $\sigma = \mathsf{mgu}(A_1, A_2)$, [illegible] $A_1\sigma$ is strictly maximal in $(\Gamma_1 \Rightarrow A_1)\sigma$, and [illegible] A_2 is of the form $P(t)$ with $t \notin \mathcal{X}$.

$$\frac{S \cup \{\Gamma_1 \Rightarrow A_1\}}{(S \cup \{\Gamma_1 \Rightarrow A_1 \ ; \ \Gamma_1, \Gamma_2 \Rightarrow B\})\sigma} \ (\mathsf{RF_p})$$

where [illegible] $\Gamma_2, A_2 \Rightarrow B$ is a fresh renaming of a clause in $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$, [illegible] $\sigma = \mathsf{mgu}(A_1, A_2)$, [illegible] $A_2\sigma$ is maximal in $(\Gamma_2, A_2 \Rightarrow B)\sigma$, and [illegible] A_1 is of the form $P(t)$ with $t \notin \mathcal{X}$. Moreover, if $\Gamma_2, A_2 \Rightarrow B$ is a special clause, we require that t is not of the form $f_0(t_1, \ldots, t_q)$ with $t_1 \in \mathcal{X}$.

Definition 6 (rigid global derivation). [illegible] rigid global derivation [illegible] $\mathcal{I}$ [illegible] $S_1 \Mapsto \ldots \Mapsto S_n$ [illegible] S_i [illegible] S_{i+1} [illegible] S_i [illegible] $\mathcal{I}_{\mathsf{Pr}}$

By inspection of each inference rule, it is easy to establish the following result.

Proposition 1 (soundness). [illegible] $\mathcal{C}_P$ fi [illegible] $\mathcal{P}$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] fi [illegible] $\mathcal{C}_I$ [illegible] C_S [illegible] $S_0 = \mathcal{C}_P$ [illegible] $S_0 \Mapsto S_1 \ldots \Mapsto S_n$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $\square \in S_i$ [illegible] $i \leq n$ [illegible] $\mathcal{C}_P$ [illegible] fi [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$

6 Termination

Proposition 2 (termination). [illegible] $\mathcal{C}_P$ [illegible] fi [illegible] $\mathcal{P}$ [illegible] $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$ [illegible] fi [illegible] $\mathcal{C}_I$ [illegible] C_S [illegible] $\mathcal{I}_{\mathsf{Pr}}$ [illegible] $\mathcal{C}_P$ [illegible] fi [illegible]

7 Completeness

In this section, we will show that the Protocol Resolution inference system $\mathcal{I}_{\mathsf{Pr}}$ is complete. We will consider an unsatisfiable set of protocol clauses $\mathcal{C}_P$, meaning that there exists a substitution θ such that $\mathcal{C}_P\theta$ is unsatisfiable modulo a given intruder theory $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$. We will prove that if we cannot derive the empty clause from $\mathcal{C}_P$ using $\mathcal{I}_{\mathsf{Pr}}$ then there must be a [illegible] substitution σ witnessing the fact that $\mathcal{C}_P$ is unsatisfiable modulo $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$. If this notion of smaller is well-founded, this allows us to conclude the completeness of $\mathcal{I}_{\mathsf{Pr}}$.
First of all, we show that our inference system can only produce protocol clauses.

Lemma 2. [illegible] $\mathcal{C}_P$ [illegible] fi [illegible] $\mathcal{P}$ [illegible] $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$ [illegible] fi [illegible] $\mathcal{C}_I$ [illegible] C_S [illegible] $\mathcal{C}'_P$ [illegible] $\mathcal{C}_P \Mapsto \mathcal{C}'_P$ [illegible] $C \in \mathcal{C}'_P$ [illegible] C [illegible]

The standard definition of a set of clauses S to be saturated is that all inferences applied to S are redundant. We define θ [illegible], for some substitution θ, to mean that all inferences [illegible] θ are redundant.

Definition 7. [illegible] S [illegible] C [illegible] S [illegible] $C_1, \ldots, C_n$ [illegible] S [illegible] $C_i \preceq_{emb} C$ [illegible] $i \le n$ [illegible] $C_1, \ldots, C_n \models C$ S [illegible] S' [illegible] $S \Mapsto S'$ [illegible] $C' \in S'$ [illegible] C' [illegible] S

Definition 8. [illegible] S [illegible] θ [illegible] θ-consistent [illegible] $\sigma \le \theta$ [illegible] σ [illegible] *mgu* [illegible] S [illegible] θ-pre-saturated [illegible] S' [illegible] $S \Mapsto S'$ [illegible] θ [illegible] $C' \in S'$ C' [illegible] S [illegible] θ-derivation [illegible] θ [illegible]

If the θ instance of a set of protocol clauses $\mathcal{C}_P$ is unsatisfiable modulo an intruder theory $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$, then the θ instance of a derived set of clauses is also unsatisfiable modulo $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$.

Lemma 3. [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$ [illegible] θ [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{C}_P$ [illegible] fi [illegible] $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $\mathcal{C}_P \Mapsto \mathcal{C}'_P$ θ [illegible] $\mathcal{C}'_P$ [illegible] fi [illegible] $\mathcal{I}_\mathcal{P}(\mathcal{C}_I \cup \{C_S\})$ [illegible] θ [illegible]

For the completeness proof, we will need to define an ordering on variables, given a substitution, and we also define an ordering on substitutions.

Definition 9. [illegible] θ [illegible] θ' [illegible] V [illegible] X [illegible] θ [illegible] Y [illegible] $X\theta \prec Y\theta$ [illegible] $\theta \prec \theta'$ [illegible] x [illegible] V $x\theta \prec x\theta'$ [illegible] $x\theta = x\theta'$ [illegible] y [illegible] V [illegible] $y\theta \prec y\theta'$

Suppose that $\mathcal{C}_P$ is a set of rigid clauses, such that $\mathcal{C}_P$ is unsatisfiable modulo $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$, and $\mathcal{C}_P$ is presaturated under θ. In the full paper we give a series of lemmas to specify in which positions a maximal variable X can appear in a shortest proof of the unsatisfiability of C_P modulo $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$. A maximal variable can appear in a quite restricted number of places. Given that the number of places is restricted, it will be easy to show that the value of $X\theta$ can be changed and unsatisfiability modulo $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ will be preserved. The series of lemmas culminates in the following lemma which shows that, given the above conditions, when the empty clause is not in $\mathcal{C}_P$, it is possible to replace θ by a smaller substitution σ, such that $\mathcal{C}_P\sigma$ is still unsatisfiable. The proof of the lemma is based on the fact that the substitutions in $\mathcal{I}_{\mathsf{Ror}}$ are stored in constraints. If we have a proof of unsatisfiability, then the constraints in the proof can be changed, so that the value of $X\theta$ is smaller, resulting in a new substitution σ, and we now have a proof of unsatisfiability of $\mathcal{C}_P\sigma$. The proof system for the new proof will be slightly different, and one that we might not want to use in practice. But it will be a sound inference system, and that is all we need to prove unsatisfiability.

Lemma 4. [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] θ [illegible] θ [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] σ [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$

The previous lemma is the crucial lemma to prove completeness. We showed that when the empty clause is not generated, then we can find a smaller substitution which preserves unsatisfiability. But since the ordering is well-founded, we cannot continually find smaller substitutions, which means that there must be some substitution for which our inference rules will deduce the empty clause.

Theorem 1 (completeness). [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{P}$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $\mathcal{C}_I$ [illegible] C_S [illegible] $S_0 = \mathcal{C}_P$ [illegible] θ [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{C}_P$ [illegible] $\mathcal{I}_{\mathcal{P}}(\mathcal{C}_I \cup \{C_S\})$ [illegible] $S_0 \Mapsto S_1 \ldots \Mapsto S_n$ [illegible] $\square \in S_n$

8 Conclusion

We have defined the problem of rigid theorem proving modulo a flexible theory. We showed how to use that framework to model cryptographic protocol insecurity problems. Then we gave a resolution-based decision procedure for solving such problems in some interesting protocol theories.

Recently several procedures for deciding trace-based security properties have been proposed for a bounded [4,5,15] or unbounded number of sessions [3,7]. Several years ago, most of the work relied on the so called [illegible]: one cannot learn anything from an encrypted message except if one knows the decryption key. Many papers relax this assumption in order to be closer to the implementation and to model more sophisticated cryptographic primitives. Among the works cited above, the closest to ours are [7,13] since they used a formalism based on Horn clauses. However, they only used flexible clauses. In [7], this allows them to deal with the insecurity problem in presence of an unbounded number of sessions but for a class of protocols more restrictive than ours ([illegible]). In [13] they only use flexible Horn clauses to deal with a bounded number sessions and hence they have to perform some approximation. Our generic result allows us to deal with a class of intruder theories similar to the one described in [2] and also to retrieve the decidability result about the prefix property that has been obtained in [4]. Both of these results [4,2] have been obtained in a completely different formalism.

Future work is to extend our idea to other interesting cryptographic protocol theories. In particular, we would like to add equality to our inference system to handle properties of cryptographic algorithms. We also plan to implement our idea. There are several possible ways to implement it. We could implement it exactly along the lines of this paper, as a nondeterministic procedure. It is also possible to save the constraints with each clause, and to only allow the empty clause if the constraint is satisfiable. The advantage of this method is that it could use a standard deterministic resolution-style inference procedure. Finally, we plan to implement it as an extension of the method given in [9], where rigid theorem proving problems are solved by encoding proofs as SAT problems.

References

1. Andrews, P.: Theorem proving via general matings. Journal of the ACM 28(2), 193–214 (1981)
2. Bernat, V., Comon-Lundh, H.: Normal proofs in intruder theories. In: ASIAN 2006. Proc. of 11th Asian Computing Science Conference, Tokyo, Japan. LNCS, Springer, Heidelberg (2006)
3. Blanchet, B.: An efficient cryptographic protocol verifier based on prolog rules. In: CSFW 2001. Proc. of 14th Computer Security Foundations Workshop, Cape Breton (Canada), pp. 82–96. IEEE Computer Society Press, Los Alamitos (2001)
4. Chevalier, Y., Küsters, R., Rusinowitch, M., Turuani, M.: An NP decision procedure for protocol insecurity with XOR. In: LICS 2003. Proc. of 18th Annual IEEE Symposium on Logic in Computer Science, Ottawa (Canada), pp. 261–270. IEEE Computer Society Press, Los Alamitos (2003)

5. Comon-Lundh, H., Shmatikov, V.: Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In: LICS 2003. Proc. of 18th Annual IEEE Symposium on Logic in Computer Science, Ottawa (Canada), pp. 271–280. IEEE Computer Society Press, Los Alamitos (2003)
6. Cortier, V., Delaune, S., Lafourcade, P.: A survey of algebraic properties used in cryptographic protocols. Journal of Computer Security 14(1), 1–43 (2006)
7. Cortier, V., Rusinowitch, M., Zalinescu, E.: A resolution strategy for verifying cryptographic protocols with cbc encryption and blind signatures. In: PPDP 2005. Proc. of 7th ACM-SIGPLAN International Conference on Principles and Practice of Declarative Programming, Lisboa (Portugal), pp. 12–22. ACM Press, New York (2005)
8. Denning, D.E., Sacco, G.M.: Timestamps in key distribution protocols. Communications of the ACM 24(8), 533–536 (1981)
9. Deshane, T., Hu, W., Jablonski, P., Lin, H., Lynch, C., McGregor, R.E.: Encoding first order proofs in sat. In: Pfenning, F. (ed.) CADE 2007. LNCS, vol. 4603, pp. 476–491. Springer, Heidelberg (2007)
10. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions in Information Theory 2(29), 198–208 (1983)
11. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: Zheng, Y., Seberry, J. (eds.) AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1993)
12. Goubault, J.: The complexity of resource-bounded first-order classical logic. In: Enjalbert, P., Mayr, E.W., Wagner, K.W. (eds.) STACS 94. LNCS, vol. 775, pp. 59–70. Springer, Heidelberg (1994)
13. Jacquemard, F., Rusinowitch, M., Vigneron, L.: Tree automata with equality constraints modulo equational theories. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 557–571. Springer, Heidelberg (2006)
14. Kremer, S., Ryan, M.: Analysing the vulnerability of protocols to produce known-pair and chosen-text attacks. In: SecCo 2004. Proc. 2nd International Workshop on Security Issues in Coordination Models, Languages and Systems, London, UK. ENTCS, Elsevier Science Publishers, Amsterdam (2005)
15. Millen, J., Shmatikov, V.: Symbolic protocol analysis with an abelian group operator or Diffie-Hellman exponentiation. Journal of Computer Security 13(3), 515–564 (2005)
16. Needham, R., Schroeder, M.: Using encryption for authentification in large networks of computers. Communications of the ACM 21(12), 993–999 (1978)
17. Pereira, O., Quisquater, J.-J.: On the perfect encryption assumption. In: WITS 2000. Proc. of 1st Workshop on Issues in the Theory of Security, Geneva (Switzerland), pp. 42–45 (2000)

Preferential Description Logics

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

[1] Dip. di Informatica - Univ. Piemonte O. "A. Avogadro"
`laura@mfn.unipmn.it`
[2] Dip. di Informatica - Università di Torino
`{gliozzi,pozzato}@di.unito.it`
[3] LSIS-UMR CNRS 6168 Univ. "P. Cézanne"
`nicola.olivetti@univ-cezanne.fr`

Abstract. We extend the Description Logic $\mathcal{ALC}$ with a "typicality" operator $\mathbf{T}$ that allows us to reason about the prototypical properties and inheritance with exceptions. The resulting logic is called $\mathcal{ALC} + \mathbf{T}$. The typicality operator is intended to select the "most normal" or "most typical" instances of a concept. In our framework, knowledge bases may then contain, in addition to ordinary ABoxes and TBoxes, subsumption relations of the form "$\mathbf{T}(C)$ is subsumed by P", expressing that typical C-members have the property P. The semantics of a typicality operator is defined by a set of postulates that are strongly related to Kraus-Lehmann-Magidor axioms of preferential logic $\mathbf{P}$. We first show that $\mathbf{T}$ enjoys a simple semantics provided by ordinary structures equipped by a preference relation. This allows us to obtain a modal interpretation of the typicality operator. Using such a modal interpretation, we present a tableau calculus for deciding satisfiability of $\mathcal{ALC} + \mathbf{T}$ knowledge bases. Our calculus gives a nondeterministic-exponential time decision procedure for satisfiability of $\mathcal{ALC} + \mathbf{T}$. We then extend $\mathcal{ALC} + \mathbf{T}$ knowledge bases by a nonmonotonic completion that allows inferring defeasible properties of specific concept instances[1].

1 Introduction

The family of description logics (DLs) is one of the most important formalisms of knowledge representation. DLs are reminiscent of the early semantic networks and of frame-based systems. They offer two key advantages: a well-defined semantics based on first-order logic and a good trade-off between expressivity and complexity. DLs have been successfully implemented by a range of systems and they are at the base of languages for the semantic web such as OWL. A DL knowledge base (KB) comprises two components: (i) the TBox, containing the definition of concepts (and possibly roles), and a specification of inclusions relations among them, and (ii) the ABox containing instances of concepts and

---

[1] This research has been partially supported by "*Progetto Lagrange - Fondazione CRT*" and by the project "*MIUR PRIN05: Specification and verification of agent interaction protocols*".

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 257–272, 2007.
© Springer-Verlag Berlin Heidelberg 2007

roles, in other words, properties and relations of individuals. Since the very objective of the TBox is to build a taxonomy of concepts, the need of representing prototypical properties and of reasoning about defeasible inheritance of such properties easily arises. The traditional approach is to handle defeasible inheritance by integrating some kind of nonmonotonic reasoning mechanism. This has led to study nonmonotonic extensions of DLs [1,2,3,5,6,7,12]. However, finding a suitable nonmonotonic extension for inheritance reasoning with exceptions is far from obvious. Let us put forward some desiderata for such an extension:

1. The (nonmonotonic) extension must have a clear semantics and should be based on the same semantics as the underlying monotonic DL.
2. The extension should allow to specify prototypical properties in a natural and direct way.
3. The extension must be decidable, if so is the underlying monotonic DL and, possibly, computationally effective.

The nonmonotonic extensions proposed in the literature do not seem to fully satisfy all the above desiderata.

[1] proposes the extension of DL with Reiter's default logic. However, the same authors have pointed out that this integration may lead to both semantical and computational difficulties. Indeed, the unsatisfactory treatment of open defaults via Skolemization may lead to an undecidable default consequence relation. For this reason, [1] proposes a restricted semantics for open default theories, in which default rules are only applied to individuals explicitly mentioned in the ABox. Furthermore, Reiter's default logic does not provide a direct way of modeling inheritance with exceptions. This has motivated the study of extensions of DLs with prioritized defaults [12,2].

A more general approach is undertaken in [6], where it is proposed an extension of DL with two epistemic operators. This extension allows one to encode Reiter's default logic as well as to express epistemic concepts and procedural rules. However, this extension has a rather complicated modal semantics, so that the integration with the existing systems requires significant changes to the standard semantics of DLs.

In [3] the authors propose an extension of DL with circumscription. One of motivating applications of circumscription is indeed to express prototypical properties with exceptions, and this is done by introducing "abnormality" predicates, whose extension is minimized. The authors provide decidability and complexity results based on theoretical analysis. However, they do not provide a calculus for their logic. Moreover, the use of circumscription to model inheritance with exceptions is not that straightforward, as we remark below.

In this work, we propose a novel approach to defeasible inheritance reasoning based on the typicality operator $\mathbf{T}$. The intended meaning is that, for any concept C, $\mathbf{T}(C)$ singles out the instances of C that are considered as "typical" or "normal". Thus assertions as "normally students do not pay taxes", or "typically users do not have access to confidential files" [3] are represented by $\mathbf{T}($[illegible]$) \sqsubseteq \neg$ [illegible] and $\mathbf{T}($[illegible]$) \sqsubseteq \neg\exists$[illegible].

Before entering in the technical details, let us sketch how we intend to use the typicality operator and what kind of inferential services we expect to profit. We assume that a KB comprises, in addition to the standard TBox and ABox, a set of assertions of the type $\mathbf{T}(C) \sqsubseteq D$ where D is a concept not mentioning $\mathbf{T}$. The reasoning system should be able to infer or propagate prototypical properties of the concepts specified in the TBox, then to ascribe defeasible properties to individuals. For instance, let the KB contain:

$\mathbf{T}($[illegible]$) \sqsubseteq \neg$[illegible]
$\mathbf{T}($[illegible]$\sqcap$[illegible]$) \sqsubseteq$ [illegible]
$\mathbf{T}($[illegible]$\sqcap Worker \sqcap \exists$[illegible]$.\top) \sqsubseteq \neg$[illegible]
$\mathbf{T}($[illegible]$) \sqsubseteq \neg$[illegible]

corresponding to the assertions: normally a student does not pay taxes, normally a working student pays taxes, but normally a working student having children does not pay taxes (because he is discharged by the government) etc...Observe that, if the same properties were expressed by ordinary inclusions, such as $Student \sqsubseteq \neg$[illegible] etc...we would simply get that there are not working students and so on, thus the KB would collapse. This collapse is avoided as we do not assume that $\mathbf{T}$ is monotonic, that is to say $C \sqsubseteq D$ does not imply $\mathbf{T}(C) \sqsubseteq \mathbf{T}(D)$. To continue with the example, if the TBox contains [illegible] $\equiv$ [illegible] $\sqcup$ [illegible], then the system should be able to infer $\mathbf{T}($[illegible]$) \sqsubseteq \neg$[illegible]. Suppose next that the ABox contains alternatively the following facts about $john$:

1. [illegible]$(john)$
2. [illegible]$(john)$, [illegible]$(john)$
3. [illegible]$(john)$, [illegible]$(john), \exists$[illegible]$.\top(john)$

Then the reasoning system should be able to infer the expected (defeasible) conclusion about $john$ in each case: 1. $\neg$[illegible]$(john)$, 2. [illegible]$(john)$, 3. $\neg$[illegible]$(john)$. Observe that setting up a similar specification of the KB by using default logic or circumscription is not that simple: with default logic [1,2,5,6,7,12], one has to specify a priority on default application (or one has to find a smart encoding of defaults giving priority to more specific information); with circumscription [3], one has to introduce abnormality predicates, and then establish which predicates are minimized, fixed, or variable, and finally for the minimized ones what is their priority wrt minimization (a total or a partial order). As a further step, the system should be able to infer (defeasible) properties also of individuals implicitly introduced by existential restrictions, for instance, if the ABox further contains $\exists$[illegible]$.$[illegible]$(jack)$, it should conclude (defeasibly) $\exists$[illegible]$.\neg$[illegible]$(jack)$.

Given the nonmonotonic character of the $\mathbf{T}$ operator, there is a difficulty with handling irrelevant information, for instance, given the KB as above, one should be able to infer as well:

$\mathbf{T}($[illegible]$\sqcap$[illegible]$) \sqsubseteq \neg$[illegible]
$\mathbf{T}($[illegible]$\sqcap$[illegible]$\sqcap$[illegible]$) \sqsubseteq$ [illegible]

as *S*[illegible] is irrelevant with respect to being a TaxPayer or not. For the same reason, the conclusion about *john* being a TaxPayer or not should not be influenced by the addition of [illegible] (*john*) to the ABox. We refer to this problem as the problem of Irrelevance.

In this paper we lay down the base of an extension of DL with a typicality operator. Our starting point is a monotonic extension of the basic $\mathcal{ALC}$ with the $\mathbf{T}$ operator. The operator is supposed to satisfy a set of postulates that are essentially a reformulation of Kraus, Lehmann, and Magidor (KLM) axioms of preferential logic, namely the assertion $\mathbf{T}(C) \sqsubseteq P$ is equivalent to the conditional assertion $C \mid\!\sim P$ of KLM preferential logic $\mathbf{P}$. It turns out that the semantics of the typicality operator can be equivalently specified by considering a preference relation (a strict partial order) on individuals: the typical members of a concept C are just the most preferred individuals, or "most normal", of C according to the preference relation. The preference relation is the only additional ingredient that we need in our semantics. We assume that "most normal" members of a concept C always exist, whenever the concept C is non-empty. This assumption corresponds to the [illegible] of KLM logics, or the well-known [illegible] in conditional logics. Taking advantage of this semantic setting, we can give a modal interpretation to the typicality operator: the modal operator $\Box$ has intuitively the same properties as in Gödel-Löb modal logic G of arithmetic provability. We then define a tableau system for this extension based on this modal interpretation, thereby obtaining a decision procedure and un upper complexity bound (NEXPTIME). In future research we will further consider whether this bound is optimal or not.

These are the main results of the paper, however the monotonic extension is not enough to perform inheritance reasoning of the kind described above: (i) we need a way of inferring defeasible properties of individuals, (ii) we need a way of handling Irrelevance.

In the paper we deal with (i) by defining a completion of an ABox: the idea is that each individual is assumed to be a typical member of the most specific concept to which it belongs. Such a completion allows to perform inferences as the ones 1.,2.,3. above. The paper outlines how we intend to cope with typicality of all instances and with Irrelevance. In particular, dealing with Irrelevance (ii) requires a nonmonotonic mechanism. The idea is to complete a KB with a set of default rules. The default rules are not used to express defeasible properties of concepts (as in default extension of DLs), but to propagate defeasible properties of a concept to its subsumed concepts, e.g. to infer $\mathbf{T}($[illegible] $\sqcap$ [illegible]$) \sqsubseteq \neg$ [illegible] from $\mathbf{T}($[illegible]$) \sqsubseteq \neg$ [illegible]. Thus in our approach the nonmonotonic mechanism is only needed to handle Irrelevance, whose treatment by means of default rules will be the object of future work.

2 The Logic $\mathcal{ALC}$ + T: The Typicality Operator T

We consider an alphabet of concept names $\mathcal{C}$, of role names $\mathcal{R}$, and of individuals $\mathcal{O}$. The language $\mathcal{L}$ of the logic $\mathcal{ALC} + \mathbf{T}$ is defined by distinguishing [illegible] and [illegible] as follows: (Concepts) $A \in \mathcal{C}$ and $\top$ are [illegible] of $\mathcal{L}$; if $C, D \in \mathcal{L}$ and $R \in \mathcal{R}$, then $C \sqcap D, C \sqcup D, \neg C, \forall R.C, \exists R.C$ are [illegible] of $\mathcal{L}$. (Extended concepts) if C is a concept, then C and $\mathbf{T}(C)$ are [illegible], and all the boolean combinations of extended concepts are extended concepts of $\mathcal{L}$. A knowledge base is a pair (TBox,ABox). TBox contains subsumptions $C \sqsubseteq D$, where $C \in \mathcal{L}$ is either a concept or an extended concept $\mathbf{T}(C')$, and $D \in \mathcal{L}$ is a concept. ABox contains expressions of the form $C(a)$ and aRb where $C \in \mathcal{L}$ is an extended concept, $R \in \mathcal{R}$, and $a, b \in \mathcal{O}$.

In order to provide a semantics to the operator $\mathbf{T}$, we extend the definition of a model used in “standard” terminological logic $\mathcal{ALC}$:

Definition 1 (Semantics of T with selection function). [illegible] $\langle \Delta, I, f_{\mathbf{T}} \rangle$ [illegible] Δ [illegible] I [illegible] C [illegible] $C^I \subseteq \Delta$ [illegible] R [illegible] $R^I \subseteq \Delta^I \times \Delta^I$ I [illegible] fi [illegible] $\mathcal{ALC}$ [illegible] $(\mathbf{T}(C))^I = f_{\mathbf{T}}(C^I)$ $f_{\mathbf{T}} : Pow(\Delta) \rightarrow Pow(\Delta)$ [illegible]

$(f_{\mathbf{T}} - 1)$ $f_{\mathbf{T}}(S) \subseteq S$

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

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

$(f_{\mathbf{T}} - 4)$ $f_{\mathbf{T}}(\bigcup S_i) \subseteq \bigcup f_{\mathbf{T}}(S_i)$

$(f_{\mathbf{T}} - 5)$ $\bigcap f_{\mathbf{T}}(S_i) \subseteq f_{\mathbf{T}}(\bigcup S_i)$

Intuitively, given the extension of some concept C, $f_{\mathbf{T}}$ selects the [illegible] instances of C. $(f_{\mathbf{T}} - 1)$ requests that typical elements of S belong to S. $(f_{\mathbf{T}} - 2)$ requests that if there are elements in S, then there are also [illegible] such elements. The next properties constraint the behavior of $f_{\mathbf{T}}$ wrt $\cap$ and $\cup$ in such a way that they do not entail monotonicity. According to $(f_{\mathbf{T}} - 3)$, if the typical elements of S are in R, then they coincide with the typical elements of $S \cap R$, thus expressing a weak form of monotonicity (namely [illegible]). $(f_{\mathbf{T}} - 4)$ corresponds to one direction of the equivalence $f_{\mathbf{T}}(\bigcup S_i) = \bigcup f_{\mathbf{T}}(S_i)$, so that it does not entail monotonicity. Similar considerations apply to the equation $f_{\mathbf{T}}(\bigcap S_i) = \bigcap f_{\mathbf{T}}(S_i)$, of which only the inclusion $\bigcap f_{\mathbf{T}}(S_i) \subseteq f_{\mathbf{T}}(\bigcap S_i)$ is derivable. $(f_{\mathbf{T}} - 5)$ is a further constraint on the behavior of $f_{\mathbf{T}}$ wrt arbitrary unions and intersections; it would be derivable if $f_{\mathbf{T}}$ were monotonic. We can prove the following proposition:

Proposition 1. $f_{\mathbf{T}}(S \cup R) \cap S \subseteq f_{\mathbf{T}}(S)$

We can give an alternative semantics for $\mathbf{T}$ based on a preference relation. The idea is that there is a global preference relation among individuals and that the typical members of a concept C, i.e. selected by $f_{\mathbf{T}}(C^I)$, are the minimal elements of C wrt this preference relation. Observe that this notion is global, that is to say, it does not compare individuals wrt a specific concept (something like y is more typical than x wrt concept C). In this framework, an object $x \in \Delta$ is a [illegible] of some concept C, if $x \in C^I$ and there is no C-element in Δ [illegible] than x. The typicality preference relation is partial since it is not always possible to establish which object is more typical than which other. The following definition is needed before we provide the Representation Theorem.

Definition 2. [illegible] $<$ [illegible] Δ [illegible] $S \subseteq \Delta$ [illegible] $Min_<(S) = \{x : x \in S$ [illegible] $\nexists y \in S$ [illegible] $y < x\}$ [illegible] $<$ [illegible] Smoothness Condition [illegible] $S \subseteq \Delta$ [illegible] $x \in S$ [illegible] $x \in Min_<(S)$ [illegible] $\exists y \in Min_<(S)$ [illegible] $y < x$

We are now ready to prove the Representation Theorem below, showing that given a model with a selection function, we can define [illegible] a preference relation $<$ such that, for all $S \subseteq \Delta$, $f_{\mathbf{T}}(S) = Min_<(S)$. Notice that, as a difference wrt related results (Theorem 3 in [11]), the relation is defined on the same domain Δ of $f_{\mathbf{T}}$. On the other hand, if $<$ is a preference relation satisfying the Smoothness Condition, then the operator defined as $f_{\mathbf{T}}(S) = Min_<(S)$ satisfies the postulates of Definition 1.

Theorem 1 (Representation Theorem). [illegible] $\langle \Delta, I, f_{\mathbf{T}} \rangle$ $f_{\mathbf{T}}$ [illegible] $(f_{\mathbf{T}} - 1)$ [illegible] $(f_{\mathbf{T}} - 5)$ [illegible] Δ [illegible] $<$ [illegible] $S \subseteq \Delta$ $f_{\mathbf{T}}(S) = Min_<(S)$

[illegible] ("[illegible]" direction) Given $f_{\mathbf{T}}$ satisfying postulates $(f_{\mathbf{T}} - 1)$ to $(f_{\mathbf{T}} - 5)$, we define $<$ as follows: for all $x, y \in \Delta$, we let $x < y$ if $\forall S \subseteq \Delta$, if $y \in f_{\mathbf{T}}(S)$ then $x \notin S$, and $\exists R \subseteq \Delta$ such that $S \subset R$ and $x \in f_{\mathbf{T}}(R)$. We prove that:

1. $<$ is irreflexive. Easily follows by the definition of $<$.
2. $<$ is transitive. Let (a) $x < y$ and (b) $y < z$. Let $z \in f_{\mathbf{T}}(S)$ for some S, then by definition of $<$, $y \notin S$, and $\exists R$ s.t. $S \subset R$ and $y \in f_{\mathbf{T}}(R)$. Furthermore, $x \notin R$ and $\exists Q : R \subset Q$ and $x \in f_{\mathbf{T}}(Q)$. From this we can conclude that $x \notin S$ (otherwise $x \in R$), and $S \subset Q$, hence $x < z$.
3. $f_{\mathbf{T}}(S) \subseteq Min_<(S)$. Let $x \in f_{\mathbf{T}}(S)$. Suppose $x \notin Min_<(S)$, i.e. for some $y \in S$, $y < x$. By definition of $<$, $y \notin S$, contradiction, hence $x \in Min_<(S)$.
4. $Min_<(S) \subseteq f_{\mathbf{T}}(S)$. Let $x \in Min_<(S)$. Then $x \in S$, i.e. $S \neq \emptyset$. By $(f_{\mathbf{T}} - 2)$, $f_{\mathbf{T}}(S) \neq \emptyset$. Suppose $x \notin f_{\mathbf{T}}(S)$. Consider $\bigcup R_i$ for all $R_i \subseteq \Delta$ s.t. $x \in f_{\mathbf{T}}(R_i)$. By $(f_{\mathbf{T}} - 5)$, we have $x \in f_{\mathbf{T}}(\bigcup R_i)$.
   Consider now $f_{\mathbf{T}}(\bigcup R_i \cup S)$. We can easily show that $f_{\mathbf{T}}(\bigcup R_i \cup S) \not\subseteq \bigcup R_i$ (otherwise, by $(f_{\mathbf{T}} - 3)$ $f_{\mathbf{T}}(\bigcup R_i \cup S) = f_{\mathbf{T}}(\bigcup R_i)$, and by Proposition 1, $f_{\mathbf{T}}(\bigcup R_i) \cap S \subseteq f_{\mathbf{T}}(S)$, which contradicts the fact that $x \in f_{\mathbf{T}}(\bigcup R_i)$ but $x \notin f_{\mathbf{T}}(S)$). Consider hence $y \in f_{\mathbf{T}}(\bigcup R_i \cup S)$ s.t. $y \notin \bigcup R_i$. By definition of $<$, $y < x$. Furthermore, by $(f_{\mathbf{T}} - 1)$ $y \in S$ (since $y \in \bigcup R_i \cup S$ and $y \notin \bigcup R_i$). It follows that $x \notin Min_<(S)$, contradiction, hence $Min_<(S) \subseteq f_{\mathbf{T}}(S)$.
5. $<$ satisfies the Smoothness Condition. Let $S \neq \emptyset$ and $x \in S$. If $x \in f_{\mathbf{T}}(S)$ then by point 3 we have $x \in Min_<(S)$. If $x \notin f_{\mathbf{T}}(S)$ we can reason as for point 4 to conclude that there is $y \in f_{\mathbf{T}}(\bigcup R_i \cup S)$ s.t. $y \notin \bigcup R_i$ (hence $y \in S$), and $y < x$. By Proposition 1, we have $y \in f_{\mathbf{T}}(S)$, hence by point 3 we conclude $y \in Min_<(S)$.

The points above allow us to conclude.
("[illegible]" direction) Given a strict partial order $<$ satisfying the Smoothness Condition, we can define $f_{\mathbf{T}} : Pow(\Delta) \rightarrow Pow(\Delta)$ by letting $f_{\mathbf{T}}(S) = Min_<(S)$. It

can be easily shown that $f_{\mathbf{T}}$ satisfies postulates $(f_{\mathbf{T}} - 1)$ to $(f_{\mathbf{T}} - 5)$. The proof is omitted due to space limitations. ■

Having the above Representation System, from now on, we will refer to the following semantics for $\mathcal{ALC} + \mathbf{T}$:

Definition 3 (Semantics of $\mathcal{ALC}$+T). [illegible] $\mathcal{M}$ [illegible] $\langle \Delta, <, I \rangle$ [illegible] Δ [illegible] I [illegible] $<$ [illegible] Δ [illegible] $\mathbf{T}$ [illegible] $(\mathbf{T}(C))^I = Min_<(C^I)$ [illegible] $\mathcal{ALC}$ C^I [illegible]

Definition 4 (Model satisfying a Knowledge Base). [illegible] $\mathcal{M}$ [illegible] I [illegible] a [illegible] $\mathcal{O}$ [illegible] a^I [illegible] Δ [illegible]

- $\mathcal{M}$ [illegible] $C \sqsubseteq D$ [illegible] $x \in \Delta$ [illegible] $x \in C^I$ [illegible] $x \in D^I$
- $\mathcal{M}$ [illegible] $C(a)$ [illegible] $a^I \in C^I$ [illegible] aRb [illegible] $(a^I, b^I) \in R^I$

$\mathcal{M}$ [illegible]

Notice that the meaning of $\mathbf{T}$ can be split into two parts: for any object x of the domain Δ, $x \in (\mathbf{T}(C))^I$ just in case (i) $x \in C^I$, and (ii) there is no $y \in C^I$ such that $y < x$. In order to isolate the second part of the meaning of $\mathbf{T}$ (for the purpose of the calculus that we will present in section 3) we introduce a new modality $\Box$ whose interpretation in $\mathcal{M}$ is defined as follows.

Definition 5. $(\Box C)^I = \{x \in \Delta \mid$ [illegible] $y \in \Delta$ [illegible] $y < x$ [illegible] $y \in C^I\}$

The basic idea is simply to interpret the preference relation $<$ as an accessibility relation. By the Smoothness Condition, it turns out that the modality $\Box$ has the properties of Gödel-Löb modal logic of provability G. The Smoothness Condition ensures that typical elements of C^I exist whenever $C^I \neq \emptyset$, by preventing infinitely descending chains of elements. This condition therefore corresponds to the finite-chain condition on the accessibility relation (as in G). A similar correspondence has been presented in [9,8] to interpret the preference relation in KLM logics. The following relation between $\mathbf{T}$ and $\Box$ holds:

Proposition 2. [illegible] $x \in \Delta$ [illegible] $x \in (\mathbf{T}(C))^I$ [illegible] $x \in C^I$ [illegible] $x \in (\Box \neg C)^I$

Since we only use $\Box$ to capture the meaning of $\mathbf{T}$, in the following we will always use $\Box$ followed by a negated concept, as in $\Box \neg C$.

We can establish the following equation between our typicality operator and the nonmonotonic (conditional) inference operator $\mid\!\sim$ (describing what can be [illegible] derived from a given premise) by letting $C \mid\!\sim D$ iff $\mathbf{T}(C) \sqsubseteq D$. It can be easily shown that there is a correspondence between the properties of $\mathbf{T}$ and the properties of $\mid\!\sim$ in the system $\mathbf{P}$ described in [11].

3 Reasoning

In this section we present a tableau calculus for deciding the satisfiability of a knowledge base. Given a KB (TBox,ABox), any concrete reasoning system should provide the usual reasoning services, namely [illegible], [illegible], and [illegible]. It is well known that the latter three services are reducible to the satisfiability of a KB.

We introduce a labelled tableau calculus for our logic $\mathcal{ALC}+\mathbf{T}$, which enriches the labelled tableau calculus for $\mathcal{ALC}$ presented in [4]. The calculus is called $T^{\mathcal{ALC}+\mathbf{T}}$ and it is based on the notion of [illegible]. We consider a set of [illegible] drawn from a denumerable set $\mathcal{V}$. $T^{\mathcal{ALC}+\mathbf{T}}$ makes use of labels, which are denoted with $x, y, z, \ldots$. Labels represent [illegible]. An object is either a variable or an individual of the ABox, that is to say an element of $\mathcal{O} \cup \mathcal{V}$.

A [illegible] is a syntactic entity of the form $x \xrightarrow{R} y$ or $x : C$, where R is a role and C is either an extended concept or has the form $\Box\neg D$ or $\neg\Box\neg D$, where D is a concept. As we will define in Definition 6, the ABox of an $\mathcal{ALC}+\mathbf{T}$-knowledge base can be translated into a set of constraints by replacing every membership assertion $C(a)$ with the constraint $a : C$ and every role aRb with the constraint $a \xrightarrow{R} b$. A tableau is a tree whose nodes are pairs $\langle S \mid U\rangle$, where:

- S contains constraints (or [illegible] formulas) of the form $x : C$ or $x \xrightarrow{R} y$;
- U contains formulas of the form $C \sqsubseteq D^L$, representing subsumption relations $C \sqsubseteq D$ of the TBox. L is a list of labels. As we will discuss later, this list is used in order to ensure the termination of the tableau calculus.

A node $\langle S \mid U\rangle$ is also called a [illegible]. A branch is a sequence of nodes $\langle S_1 \mid U_1\rangle, \langle S_2 \mid U_2\rangle, \ldots, \langle S_n \mid U_n\rangle \ldots$, where each node $\langle S_i \mid U_i\rangle$ is obtained by its immediate predecessor $\langle S_{i-1} \mid U_{i-1}\rangle$ by applying a rule of $T^{\mathcal{ALC}+\mathbf{T}}$, having $\langle S_{i-1} \mid U_{i-1}\rangle$ as the premise and $\langle S_i \mid U_i\rangle$ as one of its conclusions. A branch is closed if one of its nodes is an instance of (Clash), otherwise it is open. We say that a tableau is closed if all its branches are closed.

Given a KB, we define its [illegible] as follows:

Definition 6 (Corresponding constraint system). [illegible] $\mathcal{ALC}+\mathbf{T}$ [illegible] ([illegible]) [illegible] corresponding constraint system $\langle S \mid U\rangle$ [illegible] $S = \{a : C \mid C(a) \in ABox\} \cup \{a \xrightarrow{R} b \mid aRb \in ABox\}$ [illegible] $U = \{C \sqsubseteq D^{\emptyset} \mid C \sqsubseteq D \in TBox\}$

Definition 7 (Model satisfying a constraint system). [illegible] $\mathcal{M}$ [illegible] α [illegible] $\mathcal{V}$ [illegible] Δ [illegible] $a \in \mathcal{O}$ [illegible] $a^I \in \Delta$ $\mathcal{M}$ [illegible] $x : C$ [illegible] α [illegible] $\alpha(x) \in C^I$ [illegible] $x \xrightarrow{R} y$ [illegible] α [illegible] $(\alpha(x), \alpha(y)) \in R^I$ [illegible] $\langle S \mid U\rangle$ [illegible] $\mathcal{M}$ [illegible] α [illegible] $\mathcal{M}$ [illegible] α [illegible] S [illegible] $C \sqsubseteq D \in U$ [illegible] x [illegible] S [illegible] $\alpha(x) \in C^I$ [illegible] $\alpha(x) \in D^I$

$$\langle S, x : C, x : \neg C \mid U \rangle \quad \text{(Clash)} \qquad\qquad \frac{\langle S, x : \neg\neg C \mid U \rangle}{\langle S, x : C \mid U \rangle}\,(\neg)$$

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

$$\frac{\langle S, x : \forall R.C, x \xrightarrow{R} y \mid U \rangle}{\langle S, x : \forall R.C, x \xrightarrow{R} y, y : C \mid U \rangle}\,(\forall^+)\ \text{if } y : C \notin S \qquad\qquad \frac{\langle S, x : \exists R.C \mid U \rangle}{\langle S, x : \exists R.C, x \xrightarrow{R} y, y : C \mid U \rangle}\,(\exists^+)\ \begin{array}{r} y \text{ new} \\ \text{if } \nexists z \prec x \text{ s.t. } z \equiv_{S,x:\exists R.C} x \text{ and} \\ \nexists u \text{ s.t. } x \xrightarrow{R} u \in S \text{ and } u : C \in S \end{array}$$

$$\frac{\langle S, x : \neg\forall R.C \mid U \rangle}{\langle S, x : \neg\forall R.C, x \xrightarrow{R} y, y : \neg C \mid U \rangle}\,(\forall^-)\ \begin{array}{r} y \text{ new} \\ \text{if } \nexists z \prec x \text{ s.t. } z \equiv_{S,x:\neg\forall R.C} x \text{ and} \\ \nexists u \text{ s.t. } x \xrightarrow{R} u \in S \text{ and } u : \neg C \in S \end{array} \qquad\qquad \frac{\langle S, x : \neg\exists R.C, x \xrightarrow{R} y \mid U \rangle}{\langle S, x : \neg\exists R.C, x \xrightarrow{R} y, y : \neg C \mid U \rangle}\,(\exists^-)\ \text{if } y : \neg C \notin S$$

$$\frac{\langle S, x : \neg\Box\neg C \mid U \rangle}{\langle S, y : C, y : \Box\neg C, S^M_{x \to y} \mid U \rangle}\,(\Box^-)\ y \text{ new} \qquad\qquad \frac{\langle S \mid U, C \sqsubseteq D^L \rangle}{\langle S, x : \neg C \sqcup D \mid U, C \sqsubseteq D^{L,x} \rangle}\,\text{(Unfold)}\ \text{if } x \text{ occurs in } S \text{ and } x \notin L$$

Fig. 1. The calculus $T^{\mathcal{ALC}+\mathbf{T}}$. To save space, we omit the standard rules for $\sqcup$ and $\sqcap$.

Proposition 3. [illegible] $\mathcal{ALC}+\mathbf{T}$ [illegible]

Therefore, in order to check the satisfiability of (TBox,ABox), we build its corresponding constraint system $\langle S \mid U \rangle$, and then we use $T^{\mathcal{ALC}+\mathbf{T}}$ to check the satisfiability of $\langle S \mid U \rangle$. In order to check a constraint system $\langle S \mid U \rangle$ for satisfiability, our calculus $T^{\mathcal{ALC}+\mathbf{T}}$ adopts the usual technique of applying the rules until either a contradiction is generated (Clash) or a model satisfying $\langle S \mid U \rangle$ can be obtained from the resulting constraint system.

In order to take into account the TBox, we use a technique of [illegible], similar to the one described in [4]. Given a node $\langle S \mid U \rangle$, for each subsumption $C \sqsubseteq D^L \in U$ and for each label x that appears in the tableau, we add to S the constraint $x : \neg C \sqcup D$. As mentioned above, each formula $C \sqsubseteq D$ is equipped by the list L of labels in which it has been unfolded in the current branch. This is needed in order to avoid multiple unfolding of the same subsumption by using the same label, generating non-termination in a proof search.

Before introducing the rules of $T^{\mathcal{ALC}+\mathbf{T}}$ we need some more definitions. First, as in [4], we assume that labels are introduced in a tableau according to an ordering $\prec$, that is to say if y is introduced in the tableau, then $x \prec y$ for all labels x that are already in the tableau.

Given a tableau node $\langle S \mid U \rangle$ and an object x, we define $\sigma(\langle S \mid U \rangle, x) = \{C \mid x : C \in S\}$. Furthermore, we say that two labels x and y are S-[illegible], written $x \equiv_S y$, if they label the same set of concepts, i.e. $\sigma(\langle S \mid U \rangle, x) = \sigma(\langle S \mid U \rangle, y)$. Intuitively, S-equivalent labels can represent the same element in the model built by the rules of $T^{\mathcal{ALC}+\mathbf{T}}$. Last, we define $S^M_{x \to y} = \{y : C, y : \Box\neg C \mid x : \Box\neg C \in S\}$.

The rules of $T^{\mathcal{ALC}+\mathbf{T}}$ are presented in Figure 1. Rules $(\exists^+)$, $(\forall^-)$, and $(\Box^-)$ are called [illegible] since they introduce a new variable in their conclusions. The other rules are called [illegible]. We do not need any extra rule for the positive occurrences of the $\Box$ operator, since these are taken into account by the computation of $S^M_{x\rightarrow y}$. The side conditions on the rules ($\exists^+$) and ($\forall^-$) are introduced in order to ensure a terminating proof search, by implementing the standard [illegible] technique described below. The rules of $T^{\mathcal{ALC}+\mathbf{T}}$ are applied with the following [illegible]: 1. apply a rule to a variable $x \in \mathcal{V}$ only if no rule is applicable to a variable $y \in \mathcal{V}$ such that $y \prec x$; 2. apply dynamic rules (($\Box^-$) first) only if no static rule is applicable. This strategy ensures that the variables are considered one at a time according to the ordering $\prec$. Consider an application of a dynamic rule to a variable x of a constraint system $\langle S \mid U\rangle$. For all $\langle S' \mid U'\rangle$ obtained from $\langle S \mid U\rangle$ by a sequence of rule applications, it can be easily shown that (i) no rule can be applied in $\langle S' \mid U'\rangle$ to a variable y s.t. $y \prec x$ and (ii) $\sigma(\langle S \mid U\rangle, x) = \sigma(\langle S' \mid U'\rangle, x)$. The calculus so obtained is sound and complete wrt to the semantics described in Definition 7 (we omit the proof to save space).

Theorem 2 (Soundness and Completeness of $T^{\mathcal{ALC}+\mathbf{T}}$). [illegible] $\langle S \mid U\rangle$ [illegible]

Let us conclude this section by analyzing termination and complexity of $T^{\mathcal{ALC}+\mathbf{T}}$. In general, non-termination in labelled tableau calculi can be caused by two different reasons: 1. some rules copy their principal formula in the conclusion(s), and can thus be reapplied over the same formula without any control; 2. dynamic rules may generate infinitely-many labels, creating infinite branches.

Concerning the first source of non-termination (point 1), the only rules copying their principal formulas in their conclusions are ($\forall^+$), ($\exists^-$), (Unfold), ($\forall^-$), and ($\exists^+$). However, the side conditions on these rules avoid multiple applications on the same formula. Indeed, (Unfold) can be applied to a constraint system $\langle S \mid U, C \sqsubseteq D^L\rangle$ by using the label x only if it has not yet been applied to x in the current branch (i.e. x does not belong to L). Concerning ($\forall^+$), the rule can be applied to $\langle S, x : \forall R.C, x \xrightarrow{R} y \mid U\rangle$ only if $y : C$ does not belong to S. When $y : C$ is introduced in the branch, the rule will not further apply to $x : \forall R.C$. The same for ($\exists^-$), ($\exists^+$), and ($\forall^-$).

Concerning the second source of non-termination (point 2), we can prove that we only need to adopt the standard loop-checking machinery known as [illegible], which ensures that the rules ($\exists^+$) and ($\forall^-$) do not introduce infinitely-many labels on a branch. Thanks to the properties of $\Box$, no other additional machinery is required to ensure termination. Indeed, we can show that the interplay between rules ($\mathbf{T}^-$) and ($\Box^-$) does not generate branches containing infinitely-many labels. Let us discuss the termination in more detail.

Without the side conditions on the rules ($\exists^+$) and ($\forall^-$), the calculus $T^{\mathcal{ALC}+\mathbf{T}}$ does not ensure a terminating proof search. Indeed, given a constraint system $\langle S \mid U\rangle$, it could be the case that ($\exists^+$) is applied to a constraint $x : \exists R.C \in S$, introducing a new label y and the constraints $x \xrightarrow{R} y$ and $y : C$. If an inclusion $\mathbf{T}(\exists R.C) \sqsubseteq D$ belongs to U, then (Unfold) can be applied by using y, thus generating a branch containing $y : \neg\mathbf{T}(\exists R.C)$, to which ($\mathbf{T}^-$) can be applied introducing $y : \neg\Box\neg(\exists R.C)$. An application of ($\Box^-$) introduces a new variable z and the constraint $z : \exists R.C$, to which ($\exists^+$) can be applied generating a new label u. (Unfold) can then be re-applied on $\mathbf{T}(\exists R.C) \sqsubseteq D$ by using u, incurring a loop. In order to prevent this source of non termination, we adopt the standard technique of [illegible]: the side condition of the $(\exists^+)$ rule says that this rule can be applied to a node $\langle S, x : \exists R.C \mid U\rangle$ only if there is no z occurring in S such that $z \prec x$ and $z \equiv_{S,x:\exists R.C} x$. In other words, if there is an "older" label z which is equivalent to x wrt $S, x : \exists R.C$, then $(\exists^+)$ is not applicable, since the condition and the strategy imply that the $(\exists^+)$ rule has already been applied to z. In this case, we say that x is [illegible] by z. The same for $(\forall^-)$.

As mentioned, another possible source of infinite branches could be determined by the interplay between rules $(\mathbf{T}^-)$ and $(\Box^-)$. This cannot occur, i.e. the interplay between these two rules does not generate branches containing infinitely-many labels. Intuitively, the application of $(\Box^-)$ to $x : \neg\Box\neg C$ adds $y : \Box\neg C$ to the conclusion, so that $(\mathbf{T}^-)$ can no longer consistently introduce $y : \neg\Box\neg C$. This is due to the properties of $\Box$ (no infinite descending chains of $<$ are allowed). More in detail, if (Unfold) is applied to $\mathbf{T}(C) \sqsubseteq D$ by using x, an application of $(\mathbf{T}^-)$ introduces a branch containing $x : \neg\Box\neg C$; when a new label y is generated by an application of $(\Box^-)$ on $x : \neg\Box\neg C$, we have that $y : \Box\neg C$ is added to the current constraint system. If (Unfold) and $(\mathbf{T}^-)$ are also applied to $\mathbf{T}(C) \sqsubseteq D$ on the new label y, then the conclusion where $y : \neg\Box\neg C$ is introduced is closed, by the presence of $y : \Box\neg C$. By this fact, we do not need to introduce any loop-checking machinery on the application of $(\Box^-)$.

Theorem 3 (Termination of $T^{\mathcal{ALC}+\mathbf{T}}$). [illegible] $\langle S \mid U\rangle$ [illegible] $T^{\mathcal{ALC}+\mathbf{T}}$ [illegible]

Since the calculus $T^{\mathcal{ALC}+\mathbf{T}}$ is sound and complete (Theorem 2), and since an $\mathcal{ALC} + \mathbf{T}$-knowledge base is satisfiable iff its corresponding constraint system is satisfiable (Proposition 3), from Theorem 3 above it immediately follows that:

Theorem 4 (Decidability). [illegible] $\mathcal{ALC} + \mathbf{T}$ [illegible]

Let us conclude this section with a complexity analysis of the calculus $T^{\mathcal{ALC}+\mathbf{T}}$:

Theorem 5 (Complexity). [illegible] $\mathcal{ALC} + \mathbf{T}$ [illegible]

[illegible] We first show that the number of labels generated on a branch is at most exponential in the size of KB. Let n be the size of a KB. Given a constraint system $\langle S \mid U\rangle$, the number of extended concepts appearing in $\langle S \mid U\rangle$, including also all the ones appearing as a subformula of other concepts, is $O(n)$. As there are at most $O(n)$ concepts, there are at most $O(2^n)$ variables labelling distinct sets of concepts. Hence, there are $O(2^n)$ non-blocked variables in S.

Let m be the maximum number of direct successors of each variable $x \in S$, obtained by applying dynamic rules. m is bound by the number of $\exists R.C$ concepts $(O(n))$ plus the number of $\neg\forall R.C$ concepts $(O(n))$ plus the number of $\neg\Box\neg C$ concepts $(O(n))$. Therefore, there are at most $O(2^n \times m)$ [illegible] in S, where

$m \leq 3n$. The number of [illegible] in the ABox is bound by n too, and each individual has at most m direct successors. The number of [illegible] in S is then bound by $O((2^n + n) \times m)$, and hence by $O(2^{2n})$.

For a given label x, the concepts labelled by x introduced in the branch (namely, all the possible subconcepts of the initial constraint system, as well as all boxed subconcepts) are $O(n)$. According to the standard strategy, after all static rules have been applied to a label x in phase 1, no other concepts labelled by x can be introduced later on a branch. Hence, the labelled concepts introduced on the branch is $O(n)$ for each label, and the number of all labelled concepts on the branch is $O(n \times 2^{2n})$. Therefore, a branch can contain at most an exponential number of applications of tableau rules.

The satisfiability of a KB can thus be solved by defining a procedure which nondeterministically generates an open branch of exponential size (in the size of KB). The problem is in NEXPTIME. ■

4 Reasoning About Typicality

Logic $\mathcal{ALC} + \mathbf{T}$ allows one to reason monotonically about typicality. In $\mathcal{ALC} + \mathbf{T}$ we can consistently express, for instance, the fact that three different concepts, as [illegible] and [illegible], have a different status as taxpayers.

What about the typical properties of an individual [illegible] that we know being a working student, and having children? Of course, if we know that [illegible] is a typical instance of the concept [illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top$, i.e. if the ABox contains the assertion (∗) $\mathbf{T}($[illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top)(john)$, then, in $\mathcal{ALC} + \mathbf{T}$, we can conclude that $\neg$ [illegible]([illegible]). However, in absence of (*), we cannot derive $\neg$ [illegible]([illegible]).

In general, we would like to infer that individuals have the properties which are typical of the most specific concept to which they belong. To this purpose, we define a completion of the knowledge base which adds to the ABox, for each individual a occurring in the ABox, the assertion that a is a typical instance of the most specific concept C to which it belongs. Although in general ABoxes can contain typicality assertions about individuals, in practice we assume that typicality assertions are automatically generated by the system by means of the completion, and are not inserted by the user. From now on, we therefore assume that the initial ABox of a KB does not contain any typicality assertion.

Definition 8 (Completion of a Knowledge Base). [illegible] a [illegible] $\mathbf{T}(C_1 \sqcap \ldots \sqcap C_j)(a)$ [illegible] $C_1, \ldots, C_j$ [illegible] C_i [illegible] C_i [illegible] C_i [illegible] $\mathbf{T}$ [illegible] a [illegible] C_i [illegible] $C_i(a)$ [illegible] $\mathcal{ALC}$ [illegible]

For instance, assuming that [illegible]([illegible]), [illegible]([illegible]) and $\exists$ [illegible]$.\top$([illegible]) are the only assertions concerning [illegible] derivable from the KB, the completion above would add $\mathbf{T}($[illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top)(john)$ to the ABox, as [illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top$ is the most specific concept of which [illegible] is an instance. From this, we can conclude in $\mathcal{ALC} + \mathbf{T}$ that [illegible] does not pay taxes.

The completion adds $\mathbf{T}(C_1 \sqcap \ldots \sqcap C_j)(a)$ by considering each $C_i(a)$ [illegible] in $\mathcal{ALC}$ from the KB, rather than considering only $C_i(a)$ in the ABox. This is needed, for instance, to infer that [illegible] does not pay taxes from the KB containing [illegible] $\sqsubseteq \forall$ [illegible], [illegible]([illegible]), and [illegible]([illegible]).

As a matter of fact, if we had in the ABox the information that [illegible] is a [illegible], this would not cause an inconsistent completion of the KB. Indeed, in such a case, [illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top \sqcap$ [illegible] would be the most specific concept of which [illegible] is an instance, so that the assertion $\mathbf{T}($[illegible] $\sqcap$ [illegible] $\sqcap \exists$ [illegible]$.\top \sqcap$ [illegible]$)($[illegible]$)$ would be added in the completion of the KB. This does not allow to infer that $\neg$ [illegible]([illegible]). Hence, no inconsistency arises. However, it could be the case that the KB obtained by the completion is inconsistent, even if the initial KB is consistent. For instance, the KB containing $\mathbf{T}(C) \sqsubseteq \forall R.E$, $\mathbf{T}(D) \sqsubseteq \forall R.\neg E$, $C(a)$, $D(b)$, $R(a,c)$, and $R(b,c)$ is consistent, whereas its completion, including also $\mathbf{T}(C)(a)$ and $\mathbf{T}(D)(b)$, is not. In this case, we keep the initial KB unaltered, instead of the one obtained by the completion.

Notice that the completion of the ABox only introduces $O(n)$ new formulas $a : \mathbf{T}(C_1 \sqcap \ldots \sqcap C_j)$, one for each named individual a in the ABox. Furthermore, the size of each formula $\mathbf{T}(C_1 \sqcap \ldots \sqcap C_j)$ is $O(n^2)$ as $C_1, \ldots, C_j$ are all distinct subformulas of the initial formula ($O(n)$), and each C_i has size $O(n)$. Hence, after the completion construction, the size of the KB is polynomial in n. Moreover, for each individual a ($O(n)$) and for each concept C ($O(n)$), we have to check whether $C(a)$ is derivable in $\mathcal{ALC}$ from the KB, which is a problem in EXPTIME. Hence, the completion construction requires exponential time and produces a KB of size polynomial in the size of the original one:

Theorem 6. [illegible] NEXPTIME [illegible]

As mentioned, given a consistent KB, its completion could be inconsistent. In this case, we choose to keep the original KB. As an alternative, we could consider all maximal consistent KBs (extensions) that can be generated by adding, for all individuals, the relative most-specific concept assumptions. We could then perform either a skeptical or a credulous reasoning with respect to such extensions.

Preferential logic allows to deal with some forms of inheritance among concepts, by the property of cautious monotonicity (which comes from the semantic property $(f_{\mathbf{T}} - 3)$): if $\mathbf{T}(C) \sqsubseteq D$ and $\mathbf{T}(C) \sqsubseteq E$, then $\mathbf{T}(C \sqcap D) \sqsubseteq E$. Coming back to the example above, if we knew that all students typically have a teacher, i.e. $\mathbf{T}($[illegible]$) \sqsubseteq \exists$ [illegible]$.\top$, and that *john* is a student and has a teacher ([illegible]([illegible]) and $\exists$ [illegible]$.\top(john)$ are in the ABox) then, by the completion construction above, we would get $\mathbf{T}($[illegible] $\sqcap \exists$ [illegible]$.\top)($[illegible]$)$, and, by cautious monotonicity, we would conclude that *john* does not pay taxes.

5 Extensions

We consider this work as a first step. We plan to extend our approach in the following directions.

[illegible] Once the completion of a KB has been defined as above, the problem of inferring the typical properties of an individual is reduced to the problem of inferring the properties of the most specific concept to which it belongs. This can be done by reasoning on the typical properties of concepts in the TBox. Although Preferential Description Logic allows to capture - through cautious monotonicity - some form of inheritance of typical properties among concepts, there are cases in which cautious monotonicity is not strong enough to derive the intended conclusions. For instance, if we know that [illegible] is a student who is a sport lover, we cannot conclude that [illegible] is not a tax payer, as we do not have the property that typical students (or all students) are sport lovers, and hence cautious monotonicity is not applicable. Here we are faced with the problem of [illegible]. Since the property of being a sport lover is irrelevant with respect to the property of paying taxes, we would like to infer that also $\mathbf{T}($[illegible] $\sqcap$ [illegible]$) \sqsubseteq \neg$[illegible], and therefore that [illegible] is not a tax payer. In order to allow this form of inheritance among concepts, we can introduce a default rule of the following type:

$$\frac{\mathbf{T}(\mathit{Student}) \sqsubseteq \neg \mathit{TaxPayer} \qquad\qquad : \mathbf{T}(\mathit{Student} \sqcap \mathit{SportLover}) \not\sqsubseteq \mathit{TaxPayer}}{\mathbf{T}(\mathit{Student} \sqcap \mathit{SportLover}) \sqsubseteq \neg \mathit{TaxPayer}} \; (\mathit{IRR})$$

By the default rule above, if typical students are not tax payers, and [illegible] to assume that typical students who are sport lovers are not tax payers, then we could conclude that typical sport lover students are not tax payers. With this rule, the typical properties of a more general concept C are considered one by one, and are inherited by a more specific concept $(C \sqcap D)$ if it is consistent to do so. In order to deal with default rules like this one, we need to integrate our calculus with a standard mechanism to reason about defaults.

[illegible] The completion of a knowledge base, as defined above, only applies to individuals explicitly named in the ABox. However, we would like to reason on the typical properties of all individuals. Assume, for instance, that the ABox contains the assertions: $\exists$ [illegible](bill) and $\forall$ [illegible](bill). Thus, *bill* has a child who is a student and is working. We want to be able to infer that *bill* has a child who is a tax payer. To this purpose, we need to assume that the *bill*'s child is a typical working student.

To reason about the typicality of all individuals, we would need to assume that all individuals generated during the tableau construction are [illegible] instances of the most specific concept to which they belong. To this purpose, we could think of applying the completion construction of Definition 8 to all generated individuals. The completion construction would be applied only when all relevant formulas $y : C_1, \ldots, y : C_j$ with label y have already been introduced in the branch. According to the strategy described in section 3, the completion should be performed only after the application of the rules to all x s.t. $x \prec y$.

[illegible] We want to study the extension of our approach to more expressive description logics. For instance, we plan to extend $\mathcal{ALCNR}$ considered in [4] with our typicality operator $\mathbf{T}$, and consider which is the complexity corresponding to this extension. Finally, we want to study the extension of the language of concepts by allowing arbitrary occurrences of the operator $\mathbf{T}$.

6 Conclusions

We have proposed an extension of $\mathcal{ALC}$ for reasoning about typicality in Description Logic framework. The resulting logic is called $\mathcal{ALC} + \mathbf{T}$. We have proposed a calculus for deciding the satisfiability of a general knowledge base in $\mathcal{ALC} + \mathbf{T}$. The calculus, called $T^{\mathcal{ALC}+\mathbf{T}}$, is analytic, terminating, and allows us to decide the satisfiability of a knowledge base in $\mathcal{ALC} + \mathbf{T}$ in nondeterministic exponential time. The calculus is reminiscent of the tableaux calculi for KLM logics presented in [9,8]. We have then shown how to complete the ABox by means of typicality assumptions, in order to infer prototypical properties of the individuals explicitly mentioned in the ABox. We have argued how to apply a similar completion also to individuals implicitly mentioned in the ABox, in order to infer their properties. Finally, we have sketched how to reason about the inheritance of typical properties from more general to more specific concepts handling with irrelevant information, by using appropriate default rules.

KLM logics are related to probabilistic reasoning. A probabilistic extension of DLs has been proposed in [10]. In particular, the notion of conditional constraint in [10] allows typicality assertions to be expressed (with a specified probability). We plan to compare in details this probabilistic approach to ours elsewhere.

References

1. Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. Autom. Reasoning 14(1), 149–180 (1995)
2. Baader, F., Hollunder, B.: Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. J. Autom. Reasoning 15(1), 41–68 (1995)
3. Bonatti, P.A., Lutz, C., Wolter, F.: Description logics with circumscription. In: Proc. of KR, pp. 400–410 (2006)
4. Buchheit, M., Donini, F.M., Schaerf, A.: Decidable reasoning in terminological knowledge representation systems. J. Artif. Int. Research (JAIR) 1, 109–138 (1993)
5. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W., Schaerf, A.: An epistemic operator for description logics. Artif. Intell. 100(1-2), 225–274 (1998)
6. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Log. 3(2), 177–225 (2002)
7. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: Proc. of KR, pp. 141–151 (2004)
8. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux Calculi for KLM Rational Logic R. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 190–202. Springer, Heidelberg (2006)

9. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: Analytic Tableaux for KLM Preferential and Cumulative Logics. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 666–681. Springer, Heidelberg (2005)
10. Giugno, R., Lukasiewicz, T.: P-$\mathcal{SHOQ}$(D): A Probabilistic Extension of $\mathcal{SHOQ}$(D) for Probabilistic Ontologies in the Semantic Web. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 86–97. Springer, Heidelberg (2002)
11. Kraus, S., Lehmann, D., Magidor, M.: Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence 44(1-2), 167–207 (1990)
12. Straccia, U.: Default inheritance reasoning in hybrid kl-one-style logics. In: Proc. of IJCAI, pp. 676–681 (1993)

On Two Extensions of Abstract Categorial Grammars

Philippe de Groote[1], Sarah Maarek[2], and Ryo Yoshinaka[1]

[1] LORIA & INRIA-Lorraine
`Philippe.de.Groote@loria.fr`,
`Ryo.Yoshinaka@loria.fr`
[2] LORIA & Université Nancy 2
`Sarah.Maarek@loria.fr`

1 Introduction

The abstract categorial grammars (ACGs, for short) are a type-theoretic grammatical formalism intended for the description of natural languages [1]. It is based on the implicative fragment of multiplicative linear logic, which results in a rather simple framework.

From a language-theoretic standpoint, however, this simplicity is not synonymous of a weak expressive power [2,4]. In particular, the string languages generated by the second-order ACGs, whose parsing is known to be polynomial, corresponds to the class of mildly context sensitive languages [7,11]. Nevertheless, in [5], we have argued that it would be interesting to increase the intentional expressive power of the formalism by providing high level constructs.

From a formal point of view, to provide ACGs with new constructs consists in extending the type system of the formalism. In the present paper, we study two such type-theoretic extensions of the ACGs. They consist in providing the ACG type system with Cartesian product and dependent product, respectively. We prove that both extensions result in Turing-complete formalisms that allow any recursively enumerable language to be specified.

The paper is organized as follows. In the next section, we remind the reader of the definition of an abstract categorial grammar. In section 3, we study ACGs with Cartesian product. In section 4, we study ACGs with dependent product.

2 Abstract Categorial Grammars

Let A be a set of atomic types. The set $\mathscr{T}_A$ of [illegible] built upon A is inductively defined by the following rules:

$$\mathscr{T}_A \ ::= \ A \ | \ (\mathscr{T}_A \multimap \mathscr{T}_A)$$

A [illegible] is defined to be a triple $\Sigma = \langle A, C, \tau \rangle$, where:

1. A is a finite set of atomic types;
2. C is a finite set of constants;

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 273–287, 2007.
© Springer-Verlag Berlin Heidelberg 2007

3. τ is a mapping from C to $\mathscr{T}_A$.

A higher-order linear signature will also be called a . In the sequel, we will write A_Σ, C_Σ, and τ_Σ to designate the three components of a signature Σ, and we will write $\mathscr{T}_\Sigma$ for $\mathscr{T}_{A_\Sigma}$.

The set of untyped λ-terms is defined as usual, and one takes the relation of $\beta\eta$-conversion as the notion of equality between λ-terms. Then, given a signature Σ, the set of well-typed linear λ-terms Λ_Σ is the set of λ-terms that may be assigned a linear implicative types by the following typing rules.

$$\vdash_\Sigma c : \tau_\Sigma(c) \quad \text{(cons)}$$

$$x : \alpha \vdash_\Sigma x : \alpha \quad \text{(var)}$$

$$\frac{\Gamma, x : \alpha \vdash_\Sigma t : \beta}{\Gamma \vdash_\Sigma (\lambda x.\, t) : (\alpha \multimap \beta)} \; x \notin \mathrm{dom}(\Gamma) \quad \text{(abs)}$$

$$\frac{\Gamma \vdash_\Sigma t : (\alpha \multimap \beta) \quad \Delta \vdash_\Sigma u : \alpha}{\Gamma, \Delta \vdash_\Sigma (t\, u) : \beta} \; \mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \varnothing \quad \text{(app)}$$

In the above rules, as usual, Γ and Δ range over typing environments, i.e., finite sets of declarations of the form '$x : \alpha$' such that each variable is declared at most once. 'Γ, Δ' stands for $\Gamma \cup \Delta$, and '$\Gamma, x : \alpha$' for $\Gamma \cup \{x : \alpha\}$. Finally, $\mathrm{dom}(\Gamma)$ denotes the set of variable declared in Γ.

Given two signatures Σ and Ξ, a $\mathscr{L}$ from Σ to Ξ (in notation, $\mathscr{L} : \Sigma \to \Xi$) is defined to be a pair $\mathscr{L} = \langle \eta, \theta \rangle$ such that:

1. η is a mapping from A_Σ into $\mathscr{T}_\Xi$;
2. θ is a mapping from C_Σ into Λ_Ξ;
3. for every $c \in C_\Sigma$, the following typing judgement is derivable:

$$\vdash_\Xi \theta(c) : \hat{\eta}(\tau_\Sigma(c)),$$

where $\hat{\eta} : \mathscr{T}_\Sigma \to \mathscr{T}_\Xi$ is the unique homomorphic extension of η.[1]

As stated in Condition 3 of the above definition, there exists a unique type homomorphism $\hat{\eta} : \mathscr{T}_\Sigma \to \mathscr{T}_\Xi$ that extends η. Similarly, there exists a unique λ-term homomorphism $\hat{\theta} : \Lambda_\Sigma \to \Lambda_\Xi$ that extends θ.[2] In the sequel, $\mathscr{L}$ will denote both $\hat{\eta}$ and $\hat{\theta}$, the intended meaning being clear from the context. In addition, when Γ denotes a typing environment '$x_1 : \alpha_1, \ldots, x_n : \alpha_n$', we will write $\mathscr{L}(\Gamma)$ for '$x_1 : \mathscr{L}(\alpha_1), \ldots, x_n : \mathscr{L}(\alpha_n)$'. Using these notations, we have that Condition 3 induces the following property:

$$\text{if} \quad \Gamma \vdash_\Sigma t : \alpha \quad \text{then} \quad \mathscr{L}(\Gamma) \vdash_\Xi \mathscr{L}(t) : \mathscr{L}(\alpha).$$

We now give the main definition of this section. An abstract categorial grammar is a quadruple $\mathscr{G} = \langle \Sigma, \Xi, \mathscr{L}, s \rangle$ where:

---

[1] That is $\hat{\eta}(a) = \eta(a)$ and $\hat{\eta}(\alpha \multimap \beta) = \hat{\eta}(\alpha) \multimap \hat{\eta}(\beta)$.

[2] That is $\hat{\theta}(c) = \theta(c)$, $\hat{\theta}(x) = x$, $\hat{\theta}(\lambda x.\, t) = \lambda x.\, \hat{\theta}(t)$, and $\hat{\theta}(t\, u) = \hat{\theta}(t)\, \hat{\theta}(u)$.

1. Σ and Ξ are two higher-order linear signatures, which are called the *abstract vocabulary* and the *object vocabulary*, respectively;
2. $\mathscr{L} : \Sigma \rightarrow \Xi$ is a lexicon from the abstract vocabulary to the object vocabulary;
3. $s \in \mathscr{T}_\Sigma$ is a type of the abstract vocabulary, which is called the *distinguished type* of the grammar.

The intuition behind this definition is that the abstract vocabulary is used to express the parse structures of the grammar while the object vocabulary corresponds somehow to the terminal symbols of the grammar. This explains that an ACG generates two languages: an *abstract language* and an *object language*. The abstract language is the set of closed linear λ-terms that are built on the abstract vocabulary, and whose type is the distinguished type:

$$\mathcal{A}(\mathscr{G}) = \{t \in \Lambda_\Sigma \mid \ \vdash_\Sigma t \colon s \text{ is derivable}\}$$

On the other hand, the object language is defined to be the image of the abstract language by the lexicon:

$$\mathcal{O}(\mathscr{G}) = \{t \in \Lambda_\Xi \mid \exists u \in \mathcal{A}(\mathscr{G}).\ t = \mathscr{L}(u)\}$$

Then, given some term $t \in \Lambda_\Xi$, the membership problem for $\mathscr{G}$ consists in deciding whether t belongs to $\mathcal{O}(\mathscr{G})$, i.e., whether there exists $u \in \mathcal{A}(\mathscr{G})$ such that $\mathscr{L}(u) = t$. The decidability of this problem is open. What is known, is that it is equivalent to the decidability of the multiplicative exponential fragment of linear logic [4,12]. It is also known that, for second-order ACGs (i.e., ACGs whose abstract constants are at most second-order) membership is decidable in polynomial time [11,7].

3 Abstract Categorial Grammars with Cartesian Product

Feature structures, which are akin to records, are one of the main primitives of unification based grammatical formalisms such as HPSG. Records themselves are intensively used in Ranta's GF [10]. This explains our motivation in defining an extension of the ACGs where a notion of record would be available. From a theoretical point of view this amounts to extend the ACG typing system with a Cartesian product.

3.1 Definition

From the perspective of linear logic, the Cartesian product corresponds to the additive conjunction. Consequently, the set of types must be extended as follows:

$$\mathscr{T}_A \ ::= \ A \ \mid \ (\mathscr{T}_A \multimap \mathscr{T}_A) \ \mid \ (\mathscr{T}_A \mathbin{\&} \mathscr{T}_A)$$

Then the set of untyped λ-terms must be extended with a pair constructor together with its two projection operators:

$$T \ ::= \ c \ \mid \ x \ \mid \ \lambda x.T \ \mid \ (T\,T) \ \mid \ \langle T, T\rangle \ \mid \ (\pi_1 T) \ \mid \ (\pi_2 T)$$

The notion of equality between λ-terms must be adapted accordingly by taking into account the following additional reduction rules:

$$\begin{array}{rl} \pi_1\,\langle t, u\rangle \rightarrow t & (\textit{left projection}) \\ \pi_2\,\langle t, u\rangle \rightarrow u & (\textit{right projection}) \\ \langle \pi_1\,t, \pi_2\,t\rangle \rightarrow t & (\textit{surjective pairing}) \end{array}$$

Finally, the three following rules are added to the typing system:

$$\frac{\Gamma \vdash_\Sigma t : \alpha \quad \Gamma \vdash_\Sigma u : \beta}{\Gamma \vdash_\Sigma \langle t, u\rangle : \alpha \,\&\, \beta} \quad \text{(pair)}$$

$$\frac{\Gamma \vdash_\Sigma t : \alpha \,\&\, \beta}{\Gamma \vdash_\Sigma \pi_1\,t : \alpha} \quad \text{(left proj.)} \qquad \frac{\Gamma \vdash_\Sigma t : \alpha \,\&\, \beta}{\Gamma \vdash_\Sigma \pi_2\,t : \beta} \quad \text{(right proj.)}$$

Then, the very definitions of a lexicon and of an abstract categorial grammar may be kept unchanged.

3.2 Turing Completeness

As we already mentioned, membership for purely implicative ACGs is equivalent to provability in multiplicative exponential linear logic (the decidability of which is still open). Analogously, one may expect that membership for ACGs with Cartesian product is equivalent to provability in multiplicative additive exponential linear logic (which is known to be undecidable [9]). This is indeed the case, as shown in this section.

Given any recursively enumerable set of integers, we will construct an ACG whose object language is this given set. As well known, k-counter machines compute arbitrary recursive functions [8]. A *k-counter machine* is a quadruple $M = \langle Q, \delta, q_0, q_f\rangle$ where Q is a finite set of *states*, δ is a finite set of *transition rules*, $q_0 \in Q$ is the *initial state* and $q_f \in Q$ is the *final state*. A machine has k counters in each of which one natural number is stored. Each transition rule in δ has one of the following forms:

$$\langle q \ \mathbf{Inc}\ i\ r\rangle, \quad \langle q\ \mathbf{Dec}\ i\ r\rangle, \quad \langle q\ \mathbf{Zero?}\ i\ r\rangle$$

for some $i \in \{1, \ldots, k\}$ and $q, r \in Q$. An *instantaneous description* (*ID*) is an element of $Q \times \mathbb{N}^k$, where $\mathbb{N}$ is the set of natural numbers $0, 1, 2, \ldots$. When the machine is in the state q and the increment rule $\langle q\ \mathbf{Inc}\ i\ r\rangle$ is applied, then the machine increments the ith counter by 1 and it enters the state r. The decrement rule $\langle q\ \mathbf{Dec}\ i\ r\rangle$ is applied to the machine only when it is in the state q and moreover the entry of the ith counter is not zero. In that case,the machine decrements the ith counter by 1 and it goes to the state r. The zero-testrule $\langle q\ \mathbf{Zero?}\ i\ r\rangle$

is applied to the machine only when it is in the state q and moreover the entry of the ith counter is exactly zero. In that case, the machine moves to the state r. For two IDs $X, Y \in Q \times \mathbb{N}^k$, we write $X \to Y$ when the transition from X to Y is possible. We say that M [illegible] $\boldsymbol{m} \in \mathbb{N}^k$ if and only if $\langle q_0, \boldsymbol{m} \rangle \xrightarrow{*} \langle q_f, 0^k \rangle$, where $\xrightarrow{*}$ is the reflexive transitive closure of $\to$ and 0^k is short for the sequence of 0s of length k. The following theorem is an alternative presentation of Lambek's result [8].

Theorem 1. [illegible] n [illegible] ϕ [illegible] k [illegible] M [illegible] $k > n$ [illegible]

$$\phi(m_1, \ldots, m_n) = m_0 \text{ iff } M \text{ [illegible] } \langle m_0, m_1, \ldots, m_n, 0^{k-n-1} \rangle.$$

Our encoding of k-counter machines by ACGs is given in a way similar to Lincoln et al.'s technique for showing the undecidability of the multiplicative additive exponential linear logic [9]. They have introduced a variant of two-counter machines, which they call [illegible] (2-ACMs, for short), and shown that 2-ACMs simulate standard two-counter machines. An [illegible] k [illegible] (k-ACM, for short) is also a quadruple $M = \langle Q, \delta, q_0, q_f \rangle$, where δ has no zero-test rules. Instead, it has [illegible] of the form $\langle q \textbf{ Fork } r_1\ r_2 \rangle$ with $q, r_1, r_2 \in Q$, which allow us to simulate zero-test rules. An ID of a k-ACM is a finite sequence of elements of $Q \times \mathbb{N}^k$. The fork rule $\langle q \textbf{ Fork } r_1\ r_2 \rangle$ allows the machine to move from $X_1 \langle q, \boldsymbol{m} \rangle X_2$ to $X_1 \langle r_1, \boldsymbol{m} \rangle \langle r_2, \boldsymbol{m} \rangle X_2$, where $\boldsymbol{m} \in \mathbb{N}^k$ and $X_1, X_2 \in (Q \times \mathbb{N}^k)^*$. Only fork rules increase the number of elements of an ID of the machine. The transition by an increment or decrement rule is defined in the same way as standard k-counter machines. A sequence of the form $\langle q_f, 0^k \rangle \ldots \langle q_f, 0^k \rangle$ is called an [illegible]. One says that M [illegible] X if and only if $X \xrightarrow{*} \langle q_f, 0^k \rangle \ldots \langle q_f, 0^k \rangle$. One also says that M [illegible] $\boldsymbol{m} \in \mathbb{N}^k$ if and only if M accepts from $\langle q_0, \boldsymbol{m} \rangle$.

Lincoln et al.'s proof for that a 2-ACM simulates an arbitrary standard two-counter machine is also applied to k-counter machines. It is easy to modify a k-counter machine so that it has no transition rule going out from the final state while keeping the acceptable k-tuples of natural numbers. Then a zero-test rule $\langle q \textbf{ Zero? } i\ r \rangle$ is simulated by the following rules of a k-ACM:

$$\langle q \textbf{ Fork } r\ s_i \rangle, \quad \langle s_i \textbf{ Dec } j\ s_i \rangle \text{ for all } j \neq i, \quad \langle s_i \textbf{ Fork } q_f\ q_f \rangle$$

where s_i is a new state not in the original set of states.

Lemma 1. [illegible] k [illegible] k [illegible]

Now, to any k-ACM $M = \langle Q, \delta, q_0, q_f \rangle$, we associate the following ACG $\mathscr{G}_M = \langle \Sigma_M, \Sigma_{\mathbb{N}}, \mathscr{L}, s \rangle$:

$$\Sigma_M \begin{cases} s : \text{type}, & \\ a_i : \text{type} & \text{for all } i \in \{1, \dots, k\}, \\ q : \text{type} & \text{for all } q \in Q, \\ c_f : q_f, & \\ c_\rho : \alpha_\rho & \text{for all } \rho \in \delta, \\ \quad \text{where } \alpha_\rho = \begin{cases} (a_i \multimap r) \multimap q & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ r \multimap (a_i \multimap q) & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ (r_1 \mathbin{\&} r_2) \multimap q & \text{for } \rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle, \end{cases} & \\ d_0 : q_0 \multimap s, & \\ d_i : (a_i \multimap s) \multimap s. & \end{cases}$$

$$\Sigma_{\mathbb{N}} \begin{cases} o : \text{type}, \\ 0 : o, \\ S : o \multimap o, \end{cases}$$

$$\mathscr{L} \begin{cases} s := (o^k \multimap o) \multimap o, & \\ a_i := o \multimap o & \text{for all } i \in \{1, \dots, k\}, \\ q := o \multimap o & \text{for all } q \in Q, \\ c_f := \lambda z.\, z, & \\ c_\rho := \begin{cases} \lambda x.\, x\,(\lambda z.\, z) & \text{for } \rho = \langle q\ \mathbf{Inc}\ i\ r\rangle, \\ \lambda xyz.\, x\,(y\,z) & \text{for } \rho = \langle q\ \mathbf{Dec}\ i\ r\rangle, \\ \lambda x.\, \pi_1\, x & \text{for } \rho = \langle p\ \mathbf{Fork}\ q\ r\rangle, \end{cases} & \\ d_0 := \lambda xy.\, x\,(y\, 0^k), & \\ d_i := \lambda xy.\, x\,(\lambda z.\, z)\,(\lambda z_1 \dots z_k.\, y\, z_1 \dots z_{i-1} (S\, z_i) z_{i+1} \dots z_k). & \end{cases}$$

We now prove that $\lambda y.\, y\,(S^{m_1}\, 0) \dots (S^{m_k}\, 0)$ belongs to the object language of $\mathscr{G}_M$ if and only if M accepts $\langle m_1, \dots, m_k\rangle$, for any k natural numbers $m_1, \dots, m_k$. The proof consists of four technical lemmas, the first two of which establish the if part of the property. For notational convenience, to each $\boldsymbol{m} = \langle m_1, \dots, m_k\rangle \in \mathbb{N}^k$, we assign the typing environment

$$\Gamma_{\boldsymbol{m}} = x_{1,1} : a_1, \dots, x_{1,m_1} : a_1, \dots, x_{k,1} : a_k, \dots, x_{k,m_k} : a_k.$$

Lemma 2. [illegible] M [illegible] X [illegible] $\langle q, \boldsymbol{m}\rangle$ [illegible] X [illegible] t [illegible] $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t : q$ [illegible] t [illegible] d_i [illegible] i

[illegible] By induction on the length of the transition from X to an accepting ID $\langle q_f, 0^k\rangle \dots \langle q_f, 0^k\rangle$. The constant c_f of type q_f satisfies the lemma for the zero step transition.

Case 1. Suppose that $\rho = \langle q\ \mathbf{Inc}\ i\ r\rangle \in \delta$ induces the first step of the transition as

$$X_1\, \langle q, \boldsymbol{m}\rangle\, X_2 \to X_1\, \langle r, \boldsymbol{m}'\rangle\, X_2 \xrightarrow{*} \langle q_f, 0^k\rangle \dots \langle q_f, 0^k\rangle$$

where $\boldsymbol{m}'$ is obtained by incrementing the ith element of $\boldsymbol{m} = m_1, \dots, m_k$ by 1. We have $\Gamma_{\boldsymbol{m}'} = \Gamma_{\boldsymbol{m}}, x_{i,m_i+1} : a_i$ and $\Gamma_{\boldsymbol{m}}, x_{i,m_i+1} : a_i \vdash_{\Sigma_M} t : r$ for some t

by induction hypothesis. Σ_M contains the constant c_ρ of type $(a_i \multimap r) \multimap q$. We have

$$\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} c_\rho\,(\lambda x_{i,m_i+1}.\, t) : q.$$

Case 2. Suppose that $\rho = \langle q\ \mathbf{Dec}\ i\ r\rangle \in \delta$ induces the first step of the transition as

$$X_1\,\langle q, \boldsymbol{m}\rangle\,X_2 \rightarrow X_1\,\langle r, \boldsymbol{m}'\rangle\,X_2 \xrightarrow{*} \langle q_f, 0^k\rangle \ldots \langle q_f, 0^k\rangle$$

where $\boldsymbol{m}'$ is obtained by decrementing the ith element of $\boldsymbol{m} = m_1, \ldots, m_k$ by 1. That is, $m_i > 0$. We have $\Gamma_{\boldsymbol{m}} = \Gamma_{\boldsymbol{m}'}, x_{i,m_i} : a_i$ and $\Gamma_{\boldsymbol{m}'} \vdash_{\Sigma_M} t : r$ for some t by induction hypothesis. Σ_M contains the constant c_ρ of type $r \multimap (a_i \multimap q)$. We have

$$\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} c_\rho\, t\, x_{i,m_i} : q.$$

Case 3. Suppose that $\rho = \langle q\ \mathbf{Fork}\ r_1\ r_2\rangle \in \delta$ induces the first step of the transition as

$$X_1\,\langle q, \boldsymbol{m}\rangle\,X_2 \rightarrow X_1\,\langle r_1, \boldsymbol{m}\rangle\,\langle r_2, \boldsymbol{m}\rangle\,X_2 \xrightarrow{*} \langle q_f, 0^k\rangle \ldots \langle q_f, 0^k\rangle.$$

By induction hypothesis, there are t_i for $i = 1, 2$ such that $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t_i : r_i$. Σ_M contains the constant c_ρ of type $(r_1 \,\&\, r_2) \multimap q$. We have

$$\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} c_\rho\,\langle t_1, t_2\rangle : q. \qquad \square$$

Lemma 3. [illegible] M [illegible] $\langle m_1, \ldots, m_k\rangle$ [illegible] t [illegible] $\vdash_{\Sigma_M} t : s$ [illegible] $\mathscr{L}(t) = \lambda y.\, y\,(S^{m_1}\,0)\,\ldots\,(S^{m_k}\,0)$

[illegible] By Lemma 2, there exists t' such that $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t' : q_0$ where $\boldsymbol{m} = \langle m_1, \ldots, m_k\rangle$. Let

$$t = d_k\,(\lambda x_{k,m_k}.\, \ldots d_k\,(\lambda x_{k,1}.\, \ldots d_1\,(\lambda x_{1,m_1}.\, \ldots d_1\,(\lambda x_{1,1}.\, d_0\, t')\ldots)\ldots)\ldots).$$

Then $\vdash_{\Sigma_M} t : s$. It is easy to check that for any subterm t'' of t of the form

$$t'' = d_i\,(\lambda x_{i,j}.\, \ldots d_1\,(\lambda x_{1,1}.\, d_0\, t')\ldots)$$

where $1 \leq i \leq k$ and $1 \leq j \leq m_i$, we have

$$\mathscr{L}(t'') = \lambda y.\, u[z{:=}y\,(S^{m_1}\,0)\ldots(S^{m_{i-1}}\,0)\,(S^{j}\,0)\,0^{k-i}]$$

for some u that contains no constants. The fact $\vdash_{\Sigma_{\mathbb{N}}} \mathscr{L}(t) : (o^k \multimap o) \multimap o$ implies that $\mathscr{L}(t)$ β-reduces to $\lambda y.\, y\,(S^{m_1}\,0)\,\ldots\,(S^{m_k}\,0)$. $\square$

Lemma 4. [illegible] $\boldsymbol{m} \in \mathbb{N}^k$ [illegible] $q \in Q$ [illegible] $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t : q$ [illegible] t [illegible] M [illegible] $\langle q, \boldsymbol{m}\rangle$ [illegible] t [illegible] d_i [illegible] i

[illegible] Suppose that $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t : q$. We assume that t is β-normal. We prove this lemma by induction on the number of occurrences of constants in t.

Case 0. $t = c_f$ and $q = q_f$. Then the typing environment $\Gamma_{\boldsymbol{m}}$ must be empty, i.e., $\boldsymbol{m} = 0^k$. Indeed M accepts from $\langle q_f, 0^k\rangle$.

Case 1. $t = c_\rho t'$ with $\rho = \langle q \ \mathbf{Inc} \ i \ r\rangle \in \delta$ and t' such that $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t' : a_i \multimap r$. Then $\Gamma_{\boldsymbol{m}}, x_{i,m_i+1} : a_i \vdash_{\Sigma_M} t'\, x_{i,m+1} : r$. Hence, by induction hypothesis, M accepts from $\langle r, \boldsymbol{m}'\rangle$ where $\boldsymbol{m}'$ is obtained by incrementing the ith element m_i of $\boldsymbol{m}$ by 1. Since $\langle q, \boldsymbol{m}\rangle \to \langle r, \boldsymbol{m}'\rangle$, M also accepts from $\langle q, \boldsymbol{m}\rangle$.

Case 2. $t = c_\rho t'\, t''$ with $\rho = \langle q \ \mathbf{Dec} \ i \ r\rangle \in \delta$, $\Gamma_1 \vdash_{\Sigma_M} t' : r$ and $\Gamma_2 \vdash_{\Sigma_M} t'' : a_i$ for some partition Γ_1 and Γ_2 of the typing environment $\Gamma_{\boldsymbol{m}}$. Now, the only possibility for Γ_2 and t'' is $\Gamma_2 = x_{i,j} : a_i$ and $t'' = x_{i,j}$ for some $j \in \{1, \ldots, m_i\}$. Hence, $m_i \geq 1$ and $\Gamma_1 = \Gamma_{\boldsymbol{m}} - \{x_{i,j} : a_i\}$. By applying induction hypothesis to $\Gamma_1 \vdash_{\Sigma_M} t' : r$, we get that M accepts from $\langle r, \boldsymbol{m}'\rangle$ where $\boldsymbol{m}'$ is obtained by decrementing the ith element m_i of $\boldsymbol{m}$ by 1. Since $\langle q, \boldsymbol{m}\rangle \to \langle r, \boldsymbol{m}'\rangle$, M also accepts from $\langle q, \boldsymbol{m}\rangle$.

Case 3. $t = c_\rho t'$ with $\rho = \langle q \ \mathbf{Fork} \ r_1 \ r_2\rangle \in \delta$. The type of c_ρ is $(r_1 \,\&\, r_2) \multimap q$. Consequently, $t' = \langle t_1, t_2\rangle$ for some t_1 and t_2 such that $\Gamma_{\boldsymbol{m}} \vdash_{\Sigma_M} t_i : r_i$ for each $i = 1, 2$. By induction hypothesis, M accepts from both $\langle r_1, \boldsymbol{m}\rangle$ and $\langle r_2, \boldsymbol{m}\rangle$, which implies that M also accepts from $\langle q, \boldsymbol{m}\rangle$, because $\langle q, \boldsymbol{m}\rangle \to \langle r_1, \boldsymbol{m}\rangle\, \langle r_2, \boldsymbol{m}\rangle$. □

Lemma 5. [illegible] $t \in \mathcal{A}(\mathscr{G}_M)$ [illegible] $\mathscr{L}(t) = \lambda y.\, y\, (S^{m_1}\, 0) \ldots (S^{m_k}\, 0)$ [illegible] $m_1, \ldots, m_k \in \mathbb{N}$ [illegible] M [illegible] $\langle m_1, \ldots, m_k\rangle$

[illegible] By $\vdash_{\Sigma_M} t : s$, t has the form

$$t = d_{i_1}\, (\lambda x'_1. \ldots d_{i_n}\, (\lambda x'_n.\, d_0\, t') \ldots)$$

for some $i_1, \ldots, i_n \in \{1, \ldots, k\}$ and t' such that $\Gamma \vdash_{\Sigma_M} t' : q_0$ for $\Gamma = \{\, x'_j : a_{i_j} \mid 1 \leq j \leq n\,\}$. Let m_i be the number of occurrences of d_i in t and $\boldsymbol{m} = \langle m_1, \ldots, m_k\rangle$. By renaming variables, we can assume $\Gamma = \Gamma_{\boldsymbol{m}}$. By Lemma 4, t' does not contain any d_i. It is not hard to see that $\mathscr{L}(t) = \lambda y.\, y\, (S^{m_1}\, 0) \ldots (S^{m_k}\, 0)$. Moreover Lemma 4 implies that M accepts $\boldsymbol{m}$. □

Finally, we obtain the expected property as a direct consequence of Lemmas 1, 3 and 5.

Proposition 1. [illegible] k [illegible] M [illegible] $\mathscr{G}_M$ [illegible]

$$M \ [illegible]\ \langle m_1, \ldots, m_k\rangle \ [illegible]\ \lambda y.\, y\, (S^{m_1}\, 0) \ldots (S^{m_k}\, 0) \in \mathcal{O}(\mathscr{G}_M).$$

Corollary 1. [illegible] ϕ [illegible] $\mathscr{G}_\phi$ [illegible]

$$\phi(a_1, \ldots, a_n) = b \ [illegible]\ \lambda y.\, y\, (S^b\, 0)\, (S^{a_1}\, 0) \ldots (S^{a_n}\, 0) \in \mathcal{O}(\mathscr{G}_\phi).$$

[illegible] By Theorem 1 and Proposition 1. It is easy to modify the definition of $\mathscr{G}_M$ so that counters not used for representing the arguments and values of the function ϕ are completely suppressed in the object language. □

4 Abstract Categorial Grammars with Dependent Product

Dependent product allows ones to specify types that depend upon terms. In a grammatical setting (where types corresponds to syntactic categories), dependent products are useful in defining generic syntactic categories (for instance, [illegible] for [illegible]) that can be instantiated according to the value of some feature (for instance, ([illegible]) for [illegible], ([illegible]) for [illegible], etc.)

4.1 Definition

In the presence of dependent products, types may depend upon terms. Consequently, it is no longer the case that the notion of well-formed types may be specified only by means of context-free rules. In the same way terms are assigned types, types will be assigned kinds. Consequently, we first introduce the raw syntax of three forms of expressions, namely, the [illegible] ($\mathscr{K}$), the [illegible] ($\mathscr{T}$), and the [illegible] (T).

$$
\begin{aligned}
\mathscr{K} &::= \mathrm{type} \mid (\mathscr{T})\mathscr{K} \\
\mathscr{T} &::= a \mid (\lambda x.\, \mathscr{T}) \mid (\mathscr{T}\, T) \mid (\mathscr{T} \multimap \mathscr{T}) \mid (\Pi x : \mathscr{T})\, \mathscr{T} \\
T &::= c \mid x \mid (\lambda^{\circ} x.\, T) \mid (\lambda x.\, T) \mid (T\, T)
\end{aligned}
$$

where a ranges over atomic types, and c over constants. In addition to atomic types, linear functional types, and dependent products, we have two other type constructs: the abstraction of a λ-variable over a type, and the application of a type to a λ-term. At the level of the λ-terms, we now distinguish between two forms of λ-abstractions: a linear λ-abstraction ($\lambda^{\circ} x.\, T$), and a non-linear one ($\lambda x.\, T$).

Let a range over atomic types, c over constants, A over kinds, and α over types. A raw signature is then defined as a sequence of declarations either of the form 'a:A' or of the form 'c:α'. Let Σ be such a raw signature, we define two partial functions. The first one, κ_{Σ}, assigns kinds to atomic types. It is inductively defined as follows:

$$
\begin{aligned}
&\kappa_{()}(a) \text{ is undefined} \\
&\kappa_{\Sigma;\, a_1:A}(a) = \begin{cases} A & \text{if } a = a_1 \\ \kappa_{\Sigma}(a) & \text{otherwise} \end{cases} \\
&\kappa_{\Sigma;\, c:\alpha}(a) = \kappa_{\Sigma}(a)
\end{aligned}
$$

Similarly, τ_{Σ}, assigns types to λ-term constants:

$$
\begin{aligned}
&\tau_{()}(c) \text{ is undefined} \\
&\tau_{\Sigma;\, a:A}(c) = \tau_{\Sigma}(c) \\
&\tau_{\Sigma;\, c_1:\alpha}(c) = \begin{cases} \alpha & \text{if } c = c_1 \\ \tau_{\Sigma}(c) & \text{otherwise} \end{cases}
\end{aligned}
$$

We now give the type system of the calculus. It relies on four forms of judgements:

$$\mathsf{sig}\,(\Sigma) \qquad \vdash_\Sigma A : \mathrm{kind} \qquad \Gamma \vdash_\Sigma \alpha : A \qquad \Gamma; \Delta \vdash_\Sigma t : \alpha$$

where A, α, and t range over kinds, types, and λ-terms, respectively. Σ is a given signature. Γ and Δ range over typing environments, which are now defined to be sequences of declarations of the form '$x : \alpha$'.

These four forms of judgements may be paraphrased as follows:

1. Σ is a well-formed signature.
2. Given the signature Σ, A is a well-formed kind.
3. Given the signature Σ, α is a type of kind A according to the non-linear typing environment Γ.
4. Given the signature Σ, t is a term of type α according to the non-linear typing environment Γ and the linear typing environment Δ.

Finally, the rules of the typing system are as follows.

WELL-FORMED SIGNATURES:

$$\mathsf{sig}\,(\,)$$

$$\frac{\mathsf{sig}\,(\Sigma) \qquad \vdash_\Sigma A : \mathrm{kind}}{\mathsf{sig}\,(\Sigma;\, a : A)}$$

$$\frac{\mathsf{sig}\,(\Sigma) \qquad \vdash_\Sigma \alpha : \mathrm{type}}{\mathsf{sig}\,(\Sigma;\, c : \alpha)}$$

In the above rules, the introduced symbols (a and c) must be fresh with respect to Σ.

WELL-FORMED KINDS:

$$\vdash_\Sigma \mathrm{type} : \mathrm{kind}$$

$$\frac{\vdash_\Sigma \alpha : \mathrm{type} \qquad \vdash_\Sigma A : \mathrm{kind}}{\vdash_\Sigma (\alpha)\, A : \mathrm{kind}}$$

WELL-KINDED TYPES:

$$\vdash_\Sigma a : \kappa_\Sigma(a) \quad \text{(type const.)}$$

$$\frac{\vdash_\Sigma \alpha : \mathrm{type} \quad \Gamma \vdash_\Sigma \beta : A}{\Gamma, x : \alpha \vdash_\Sigma \beta : A} \quad \text{(type weak.)}$$

$$\frac{\Gamma, x : \alpha \vdash_{\Sigma} \beta : A}{\Gamma \vdash_{\Sigma} \lambda x.\, \beta : (\alpha)\, A} \quad \text{(type abs.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : (\beta)\, A \quad \Gamma;\ \vdash_{\Sigma} t : \beta}{\Gamma \vdash_{\Sigma} \alpha\, t : A} \quad \text{(type app.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : \text{type} \quad \Gamma \vdash_{\Sigma} \beta : \text{type}}{\Gamma \vdash_{\Sigma} \alpha \multimap \beta : \text{type}} \quad \text{(lin. fun.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : \text{type} \quad \Gamma, x : \alpha \vdash_{\Sigma} \beta : \text{type}}{\Gamma \vdash_{\Sigma} (\Pi x : \alpha)\, \beta : \text{type}} \quad \text{(dep. prod.)}$$

In Rule (type weak.), x must be fresh with respect to Γ.

WELL-TYPED TERMS:

$$;\ \vdash_{\Sigma} c : \tau_{\Sigma}(c) \quad \text{(const.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : \text{type}}{\Gamma;\ x : \alpha \vdash_{\Sigma} x : \alpha} \quad \text{(lin. var.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : \text{type}}{\Gamma,\ x : \alpha;\ \vdash_{\Sigma} x : \alpha} \quad \text{(var.)}$$

$$\frac{\Gamma \vdash_{\Sigma} \alpha : \text{type} \quad \Gamma;\ \Delta \vdash_{\Sigma} t : \beta}{\Gamma, x : \alpha;\ \Delta \vdash_{\Sigma} t : \beta} \quad \text{(weak.)}$$

$$\frac{\Gamma;\ \Delta_1, x : \alpha, y : \beta, \Delta_2 \vdash_{\Sigma} t : \gamma}{\Gamma;\ \Delta_1, y : \beta, x : \alpha, \Delta_2 \vdash_{\Sigma} t : \gamma} \quad \text{(perm.)}$$

$$\frac{\Gamma;\ \Delta, x : \alpha \vdash_{\Sigma} t : \beta}{\Gamma;\ \Delta \vdash_{\Sigma} \lambda^{\circ} x.\, t : \alpha \multimap \beta} \quad \text{(lin. abs.)}$$

$$\frac{\Gamma;\ \Delta_1 \vdash_{\Sigma} t : \alpha \multimap \beta \quad \Gamma;\ \Delta_2 \vdash_{\Sigma} u : \alpha}{\Gamma;\ \Delta_1, \Delta_2 \vdash_{\Sigma} t\, u : \beta} \quad \text{(lin. app.)}$$

$$\frac{\Gamma, x : \alpha;\ \Delta \vdash_{\Sigma} t : \beta}{\Gamma;\ \Delta \vdash_{\Sigma} \lambda x.\, t : (\Pi x : \alpha)\, \beta} \quad \text{(abs.)}$$

$$\frac{\Gamma;\ \Delta \vdash_{\Sigma} t : (\Pi x : \alpha)\beta \quad \Gamma;\ \vdash_{\Sigma} u : \alpha}{\Gamma;\ \Delta \vdash_{\Sigma} t\, u : \beta[x{:=}u]} \quad \text{(app.)}$$

$$\frac{\Gamma;\ \Delta \vdash_{\Sigma} t : \alpha \quad \Gamma \vdash_{\Sigma} \beta : \text{type} \quad \alpha =_{\beta\eta} \beta}{\Gamma;\ \Delta \vdash_{\Sigma} t : \beta} \quad \text{(type conv.)}$$

In Rules (lin. var.) and (var.), x must be fresh with respect to Γ. In Rule (weak.), x must be fresh with respect to both Γ and Δ. Moreover, t must be either a λ-variable, or a constant. In Rule (abs.), x cannot occur free in Δ.

Let Σ be a signature. We will write A_Σ for the set of atomic types declared in Σ. Similarly, we will write C_Σ for the set of λ-term constants declared in Σ. Finally, given a well-formed signature Σ, we will write $\mathscr{K}_\Sigma$, $\mathscr{T}_\Sigma$, and Λ_Σ for the corresponding sets of well-formed kinds, well-kinded types, and well-typed terms, respectively.

In order to define a notion of ACG with dependent product, it remains to adapt the definition of a lexicon. Let Σ and Ξ be two well-formed signatures. A lexicon $\mathscr{L}$ from Σ to Ξ is a pair $\langle \eta, \theta \rangle$ such that:

1. η is a mapping from A_Σ into $\mathscr{T}_\Xi$;
2. θ is a mapping form C_Σ into Λ_Ξ;
3. for every $c \in C_\Sigma$, the following typing judgement is derivable:

$$\vdash_\Xi \theta(c) : \hat{\eta}(\tau_\Sigma(c)),$$

   where $\hat{\eta} : \mathscr{T}_\Sigma \rightarrow \mathscr{T}_\Xi$ is the unique homomorphic extension of η;
4. for every $a \in A_\Sigma$, the following kinding judgement is derivable:

$$\vdash_\Xi \eta(a) : \tilde{\eta}(\kappa_\Sigma(a)),$$

   where $\tilde{\eta} : \mathscr{K}_\Sigma \rightarrow \mathscr{K}_\Xi$ is defined by $\tilde{\eta}(\text{type}) = \text{type}$ and $\tilde{\eta}((\alpha)A) = (\hat{\eta}(\alpha))\tilde{\eta}(A)$.

4.2 Turing Completeness

The λ-calculus we have defined in the previous section contains the Edinburgh logical framework [6] as a subsystem.[3] We may therefore expect ACGs with dependent product to be Turing-complete. In order to show it is indeed the case, we explain how to encode any general phrase structure grammar as an ACG with dependent product.

We first remind the reader of some basic definitions. A [illegible] is a quadruple $G = \langle V, T, R, S \rangle$, where V is a finite set of [illegible], $T \subseteq V$ is a finite set of [illegible], $S \in V$ is the [illegible] and R is a finite set of [illegible] of the form $\alpha \rightarrow \beta$ for $\alpha, \beta \in V^*$. One writes $\alpha \Rightarrow \beta$ if $\alpha = \gamma_1 \alpha' \gamma_2$, $\beta = \gamma_1 \beta' \gamma_2$ and $\alpha' \rightarrow \beta' \in R$ for some $\gamma_1, \gamma_2 \in V^*$. As usual, $\overset{*}{\Rightarrow}$ is the reflexive, transitive closure of $\Rightarrow$. The language generated by G is defined as $L(G) = \{\alpha \in T^* \mid S \overset{*}{\Rightarrow} \alpha\}$.

To any alphabet T, we associate a signature Σ_T. This signature has one atomic type o and its set of constants is T, the elements of which are assigned the type $o \multimap o$. Then, every string $a_1 \ldots a_n \in T^*$ may be encoded as $\lambda^\circ z.\, a_1 (\ldots (a_n\, z) \ldots)$. Let us write $/a_1 \ldots a_n/$ to denote this last term. We have, in particular, that the empty string ε is represented by the linear identity function, i.e., $/\varepsilon/ = \lambda^\circ z.\, z$. We also have that concatenation is represented by functional composition, and we will write $t + u$ for $\lambda^\circ z.\, t\,(u\, z)$.

---

[3] Actually, the notion of dependent type we use is slightly weaker.

Now, to any phrase structure grammar $G = \langle V, T, R, S\rangle$, we associate the ACG $\mathscr{G}_G = \langle \Sigma_G, \Sigma_T, \mathscr{L}, s\rangle$ that is defined as follows.

$$\Sigma_G \left\{ \begin{array}{rll} o, s & : \mathrm{type}, & \\ \sigma & : (o \multimap o)\mathrm{type}, & \\ \tau & : (o \multimap o)\mathrm{type}, & \\ A & : o \multimap o & \text{for all } A \in V, \\ c_S & : \sigma(/S/), & \\ c_{\alpha\to\beta} & : (\Pi x, y \in o \multimap o)(\sigma(x + /\alpha/ + y) \multimap \sigma(x + /\beta/ + y)) & \\ & & \text{for all } \alpha \to \beta \in R, \\ d_\varepsilon & : \tau(/\varepsilon/), & \\ d_a & : (\Pi x \in o \multimap o)(\tau(x) \multimap \tau(/a/ + x)) & \text{for all } a \in T, \\ e & : (\Pi x \in o \multimap o)(\sigma(x) \multimap \tau(x) \multimap s), & \end{array} \right.$$

$$\Sigma_T \left\{ \begin{array}{l} o : \mathrm{type}, \\ a : o \multimap o \ \ \text{for all } a \in T, \end{array} \right.$$

$$\mathscr{L} \left\{ \begin{array}{rll} o & := o, & \\ s & := o \multimap o, & \\ \tau & := \lambda x.\, o \multimap o, & \\ \sigma & := \lambda x.\, o \multimap o, & \\ A & := \lambda^\circ z.\, z & \text{for all } A \in V, \\ c_S & := /\varepsilon/, & \\ c_{\alpha\to\beta} & := \lambda xy.\, \lambda^\circ z.\, z & \text{for all } \alpha \to \beta \in R, \\ d_\varepsilon & := /\varepsilon/, & \\ d_a & := \lambda x.\, \lambda^\circ y.\, /a/ + y & \text{for all } a \in T, \\ e & := \lambda x.\, \lambda^\circ yz.\, y + z. & \end{array} \right.$$

The signature Σ_G consists of two independent parts that are connected through the constant e. We will establish the two following properties:

1. for every $\alpha \in V^*$, $S \overset{*}{\Rightarrow} \alpha$ if and only if there exists u_α such that $;\vdash_{\Sigma_G} u_\alpha : \sigma(/\alpha/)$;
2. for every $\alpha \in V^*$, $\alpha \in T^*$ if and only if there exists t_α such that $;\vdash_{\Sigma_G} t_\alpha : \tau(/\alpha/)$.

This implies that $\alpha \in L(G)$ if and only if $;\vdash_{\Sigma_G} u_\alpha : \sigma(/\alpha/)$ and $;\vdash_{\Sigma_G} t_\alpha : \tau(/\alpha/)$ for some u_α and t_α.

Lemma 6. $\alpha \in V^*$ $S \overset{*}{\Rightarrow} \alpha$ t $;\vdash_{\Sigma_G} t : \sigma(/\alpha/)$ $\mathscr{L}(t) \twoheadrightarrow_\beta /\varepsilon/$

Induction on the length of the derivation. For $\alpha = S$, $t = c_S$ satisfies the lemma. Suppose that $S \overset{*}{\Rightarrow} \gamma_1\alpha\gamma_2 \Rightarrow \gamma_1\beta\gamma_2$ and $\alpha \to \beta \in R$. By induction hypothesis, we have t' such that $;\vdash_{\Sigma_G} t' : \sigma(/\gamma_1\alpha\gamma_2/)$ and $\mathscr{L}(t') \twoheadrightarrow_\beta /\varepsilon/$. Let $t = c_{\alpha\to\beta}/\gamma_1//\gamma_2/t'$. Then we have $;\vdash_{\Sigma_G} t : \sigma(/\gamma_1\beta\gamma_2/)$ and $\mathscr{L}(t) \twoheadrightarrow_\beta \mathscr{L}(t') \twoheadrightarrow_\beta /\varepsilon/$. $\square$

Lemma 7. $\alpha \in T^*$ t $;\vdash_{\Sigma_G} t : \tau(/\alpha/)$ $\mathscr{L}(t) \twoheadrightarrow_\beta /\alpha/$

Induction on the length of α. For $\alpha = \varepsilon$, $t = d_\varepsilon$ satisfies the lemma. For $\alpha \neq \varepsilon$, let $\alpha = a\alpha'$ with $a \in T$ and $\alpha' \in T^*$. By induction hypothesis, we have t' such that $; \vdash_{\Sigma_G} t' : \tau(/\alpha'/)$ and $\mathscr{L}(t') \twoheadrightarrow_\beta /\alpha'/$. Let $t = d_a/\alpha'/t'$. Then we have $; \vdash_{\Sigma_G} t : \tau(/\alpha/)$ and $\mathscr{L}(t) \twoheadrightarrow_\beta /a/ + \mathscr{L}(t') \twoheadrightarrow_\beta /\alpha/$. □

Lemma 8. $\alpha \in L(G)$ $/\alpha/ \in \mathcal{O}(\mathscr{G}_G)$

For any $\alpha \in L(G)$, we have t_1 and t_2 such that $; \vdash_{\Sigma_G} t_1 : \sigma(/\alpha/)$, $\mathscr{L}(t_1) \twoheadrightarrow_\beta /\varepsilon/$, $; \vdash_{\Sigma_G} t_2 : \tau(/\alpha/)$ and $\mathscr{L}(t_2) \twoheadrightarrow_\beta /\alpha/$ by Lemmas 6 and 7. Thus we have $; \vdash_{\Sigma_G} e/\alpha/t_1 t_2 : s$ and $\mathscr{L}(e/\alpha/t_1 t_2) \twoheadrightarrow_\beta /\alpha/$. □

Lemma 9. $\beta \in V^*$ $; \vdash_{\Sigma_G} t : \sigma(/\beta/)$ $S \overset{*}{\Rightarrow} \beta$ $\mathscr{L}(t) \twoheadrightarrow_\beta /\varepsilon/$

Suppose that $; \vdash_{\Sigma_G} t : \sigma(/\beta/)$. We assume that t is β-normal. We prove this lemma by induction on t. If $t = c_S$, then $\beta = S$ and the lemma holds clearly. Otherwise, t must have the form $t = c_{\alpha' \to \beta'}/\gamma_1//\gamma_2/t'$ with $\beta = \gamma_1\beta'\gamma_2$ for some t' such that $; \vdash_{\Sigma_G} t' : \sigma(/\gamma_1\alpha'\gamma_2/)$. By induction hypothesis, we have $S \overset{*}{\Rightarrow} \gamma_1\alpha'\gamma_2$ and $\mathscr{L}(t') \twoheadrightarrow_\beta /\varepsilon/$. The fact that Σ_G has the constant $c_{\alpha' \to \beta'}$ implies that $\alpha' \to \beta' \in R$. We have $S \overset{*}{\Rightarrow} \gamma_1\alpha'\gamma_2 \Rightarrow \gamma_1\beta'\gamma_2 = \beta$ and $\mathscr{L}(t) \twoheadrightarrow_\beta \mathscr{L}(t') \twoheadrightarrow_\beta /\varepsilon/$. □

Lemma 10. $\beta \in V^*$ $; \vdash_{\Sigma_G} t : \tau(/\beta/)$ $\beta \in T^*$ $\mathscr{L}(t) \twoheadrightarrow_\beta /\beta/$

Suppose that $; \vdash_{\Sigma_G} t : \tau(/\beta/)$. We assume that t is β-normal. We prove this lemma by induction on t. If $t = d_\varepsilon$, then $\beta = \varepsilon$ and the lemma holds clearly. Otherwise, t must have the form $t = d_a/\alpha/t'$ for some $a \in T$, $\alpha \in V^*$ and t' such that $\beta = a\alpha$ and $; \vdash_{\Sigma_G} t' : \tau(/\alpha/)$. By induction hypothesis, we have $\mathscr{L}(t') \twoheadrightarrow_\beta /\alpha/$, $\alpha \in T^*$ and thus $a\alpha \in T^*$. Besides $\mathscr{L}(t) \twoheadrightarrow_\beta /a/ + \mathscr{L}(t') \twoheadrightarrow_\beta /a\alpha/$. □

Lemma 11. $/\alpha/ \in \mathcal{O}(\mathscr{G}_G)$ $\alpha \in L(G)$

Suppose that $t \in \mathcal{A}(\mathscr{G}_G)$. t must have the form $t = e/\alpha/t_1 t_2$ with $; \vdash_{\Sigma_G} t_1 : \sigma(/\alpha/)$ and $; \vdash_{\Sigma_G} t_2 : \tau(/\alpha/)$ for some $\alpha \in V^*$. We have $S \overset{*}{\Rightarrow} \alpha$ and $\mathscr{L}(t_1) \twoheadrightarrow_\beta /\varepsilon/$ by Lemma 9. $\alpha \in T^*$ and $\mathscr{L}(t_2) \twoheadrightarrow_\beta /\alpha/$ by Lemma 10. Therefore, $\alpha \in L(G)$ and $\mathscr{L}(t) \twoheadrightarrow_\beta /\alpha/ \in \mathcal{O}(\mathscr{G})$. □

As a consequence of Lemmas 8 and 11 we obtain the main result of this section.

Proposition 2.

5 Conclusions

This work shows that quite simple extensions of the abstract categorial grammars immediately result in undecidable formalisms. This is not quite surprising as it is not even known whether membership is decidable or not for the original ACGs. On the other hand, from a practical point of view, there is a need for powerful constructs such as feature structures, records, or generic types. Consequently, future work must consist in trying to identify fragments of the extended formalism proposed in [5] that offer a good compromise between intentional expressive power and tractability.

References

1. de Groote, Ph.: Towards abstract categorial grammars. In: Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, pp. 148–155 (2001)
2. de Groote, Ph.: Tree-Adjoining Grammars as Abstract Categorial Grammars. In: TAG+6, Proceedings of the sixth International Workshop on Tree Adjoining Grammars and Related Frameworks, pp. 145–150 (2001)
3. de Groote, Ph., Guillaume, B., Salvati, S.: Vector Addition Tree Automata. In: LICS 2004. 19th IEEE Symposium on Logic in Computer Science, pp. 64–73. IEEE Computer Society, Los Alamitos (2004)
4. de Groote, Ph., Pogodalla, S.: On the Expressive Power of Abstract Categorial Grammars: Representing Context-Free Formalisms. Journal of Logic, Language and Information 13(4), 421–438 (2004)
5. de Groote, P., Maarek, S.: Type-theoretic Extensions of Abstract Categorial Grammars. In: Proceedings of the Workshop on New Directions in Type-theoretic Grammars. ESSLLI (2007)
6. Harper, R., Honsel, F., Plotkin, G.: A framework for defining logics. In: Proceedings of the second annual IEEE symposium on logic in computer science, pp. 194–204. IEEE Computer Society Press, Los Alamitos (1987)
7. Kanazawa, M.: Parsing and Generation as Datalog Queries. In: Association for Computational Linguistics, 45th Annual Meeting, Proceedings of the Conference (to appear, 2007)
8. Lambek, J.: How to program an infinite abacus. Canadian Mathematical Bulletin 4, 279–293 (1961)
9. Lincoln, P., Mitchell, J.C., Scedrov, A., Shankar, N.: Decision Problems for Propositional Linear Logic. Annals of Pure and Applied Logic 56(1-3), 239–311 (1992)
10. Ranta, A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming 14(2), 145–189 (2004)
11. Salvati, S.: Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites. Thèse de Doctorat. Institut National Polytechnique de Lorraine (2005)
12. Yoshinaka, R., Kanazawa, M.: The Complexity and Generative Capacity of Lexicalized Abstract Categorial Grammars. In: Blache, P., Stabler, E.P., Busquets, J.V., Moot, R. (eds.) LACL 2005. LNCS (LNAI), vol. 3492, pp. 330–346. Springer, Heidelberg (2005)

Why Would You Trust *B*?

Éric Jaeger[1,2] and Catherine Dubois[3]

[1] LIP6, Université Paris 6, 4 place Jussieu, 75252 Paris Cedex 05, France
[2] LTI, Direction centrale de la sécurité des systèmes d'information, 51 boulevard de La Tour-Maubourg, 75700 Paris 07 SP, France
[3] CEDRIC, École nationale supérieure d'informatique pour l'industrie et l'entreprise, 18 allée Jean Rostand, 91025 Evry Cedex, France

Abstract. The use of formal methods provides confidence in the correctness of developments. Yet one may argue about the actual level of confidence obtained when the method itself – or its implementation – is not formally checked. We address this question for the *B*, a widely used formal method that allows for the derivation of correct programs from specifications. Through a deep embedding of the *B* logic in *Coq*, we check the *B* theory but also implement *B* tools. Both aspects are illustrated by the description of a proved prover for the *B* logic.

Keywords: Confidence, Formal Methods, Prover, Deep embedding.

A clear benefit of formal methods is to increase the confidence in the correctness of developments. However, one may argue about the actual level of confidence obtained, when the method or its implementation are not themselves formally checked. This question is legitimate for safety, as one may accidentally derive invalid results. It is even more relevant when security is a concern, as any flaw can be deliberately exploited by a malicious developer to obfuscate undesirable behaviours of a system while still getting a certification.

B [1] is a popular formal method that allows for the derivation of correct programs from specifications. Several industrial implementations are available (e.g. *Atelier B*, *B-Toolkit*), and it is widely used in the industry for projects where safety or security is mandatory. So the *B* is a good candidate for addressing our concern: when the prover says that a development is right, who says that the prover is right? To answer this question, one has to check the theory as well as the prover w.r.t. this theory (or, alternatively, to provide a proof checker). Those are the objectives of *BiCoq*, a deep embedding of the *B* logic in *Coq* [2].

BiCoq benefits from the support of *Coq* to study the theory of *B*, and to check the validity of standard definitions and results. *BiCoq* also allows us, through an implementation strategy, to develop formally checked *B* tools. This strategy is illustrated in this paper by the development of a prover engine for the *B* logic, that can be extracted and used independently of *Coq*. *BiCoq* is therefore our notary public, witnessing the validity of the results associated to the *B* theory, as well as the correctness of tools implementing those results – ultimately increasing confidence in *B* developments. The approach, combining a deep embedding and

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 288–302, 2007.
© Springer-Verlag Berlin Heidelberg 2007

an implementation technique, can be extended to address further elements of the [illegible], beyond its logic, or to safely enrich it, as illustrated in this paper.

This paper is divided into 9 sections. Sections 1, 2 and 3 briefly introduce [illegible] and the notion of embedding. The [illegible] logic and its formalisation in [illegible] are presented in Sec. 4. Section 5 describes various results proved using [illegible]. Section 6 focuses on the implementation strategy, and presents its application to the development of a set of extractible proof tactics for a [illegible] prover. Section 7 discusses further uses of [illegible], and mentions some existing extensions. Finally, Sect. 8 concludes and identifies further activities.

1 A Short Introduction to *B*

In a nutshell, the [illegible] method defines a first-order predicate logic completed with elements of set theory, a [illegible] ([illegible]) and a methodology of development. An abstract [illegible] is a module combining a state, properties and operations (described as substitutions) to read or alter the state.

The logic is used to express preconditions, invariants, etc. and to conduct proofs. The [illegible] allows for definitions of substitutions that can be abstract, declarative and non-deterministic (that is, specifications) as well as concrete, imperative and deterministic (that is, programs). The following example uses the non-deterministic substitution **ANY** (a "magic" operator finding a value which satisfies a property) to specify the square root of a natural number n:

[illegible] **ANY** x **WHERE** $x{*}x \leq n < (x+1){*}(x+1)$ **THEN** $\sqrt{}(n) := x$ **END**

Regarding the methodology, a machine M_C [illegible] an abstract machine M_A if one cannot distinguish M_C from M_A by valid operation calls – this notion being independent of the internal representations, as illustrated by the following example of a system returning the maximum of a set of stored values:

[illegible] The state of M_A is a (non implementable) set of natural numbers; the state of M_C is a natural number. Yet M_C, having the expected behaviour, refines M_A.

MACHINE M_A	**REFINEMENT** M_C
VARIABLES S	**VARIABLES** s
INVARIANT $S \subseteq \mathbb{N}$	**INVARIANT** $s = max(S \cup \{0\})$
INITIALISATION $S := \emptyset$	**INITIALISATION** $s := 0$
OPERATIONS	**OPERATIONS**
$store(n) \triangleq$	$store(n) \triangleq$
PRE $n \in \mathbb{N}$ **THEN** $S := S \cup \{n\}$ **END**	**IF** $s < n$ **THEN** $s := n$ **END**
$m \leftarrow get \triangleq$	$m \leftarrow get \triangleq$
PRE $S \neq \emptyset$ **THEN** $m := max(S)$ **END**	**BEGIN** $m := s$ **END**
END	**END**

Refinement being transitive, it is possible to go progressively from the specification to the implementation. By discharging at each step the [illegible] defined by the [illegible] methodology, a program can be proved to be a correct and complete implementation of a specification. This methodology, combined with the numerous native notions provided by the set theory and the existence of toolkits, make the [illegible] a popular formal method, widely used in the industry.

Note that the [illegible] logic is not genuinely typed and allows for manipulation of free variables. A special mechanism, called [illegible] (but thereafter referred to as [illegible]), filters ill-formed (potentially paradoxal) terms; it is only mentioned in this paper, deserving a dedicated analysis.

The rest of the paper only deals with the [illegible] logic (its inference rules).

2 A Short Introduction to *Coq*

[illegible] is a proof assistant based on a type theory. It offers a higher-order logical framework that allows for the construction and verification of proofs, as well as the development and analysis of functional programs in an [illegible]-like language with pattern-matching. It is possible in [illegible] to define values and types, including dependent types (that is, types that explicitly depend on values); types of sort **Set** represent sets of computational values, while types of sort **Prop** represent logical propositions. When defining an inductive type (that is, a least fixpoint), associated structural induction principles are automatically generated.

For the intent of this paper, it is sufficient to see [illegible] as allowing for the manipulation of inductive sets of terms. For example, let's consider the standard representation of natural numbers:

[illegible] $\textbf{Inductive}\,\mathbb{N}:\textbf{Set}:=0:\mathbb{N}\mid S:\mathbb{N}\rightarrow\mathbb{N}$

It defines a type $\mathbb{N}$ which is the smallest set of terms stable by application of the constructors 0 and S. $\mathbb{N}$ is exactly made of the terms 0 and $S^n(0)$ for any finite n; being well-founded, structural induction on $\mathbb{N}$ is possible.

Coq also allows for the declaration of inductive logical properties, e.g.:

[illegible] $\textbf{Inductive}\,ev:\mathbb{N}\rightarrow\textbf{Prop}:=ev_0:ev\,0\mid ev_2:\forall(n:\mathbb{N}),\;ev\,n\rightarrow ev\,(S(S\,n))$

It defines a family of [illegible]: $ev\,0$ is a type inhabited by the term (ev_0), $ev\,2$ is another type inhabited by $(ev_2\,0\,ev_0)$, and $ev\,1$ is an empty type. The standard interpretation is that ev_0 is a proof of the proposition $ev\,0$ and that there is no proof of $ev\,1$, that is we have $\neg(ev\,1)$.

An intuitive interpretation of our two examples is that $\mathbb{N}$ is a set of terms, and ev a predicate marking some of them, defining a subset of $\mathbb{N}$.

3 Deep Embedding and Related Works

[illegible] in a proof assistant consists in mechanizing a [illegible] logic by encoding its syntax and semantic into a [illegible] logic ([3,4,5]). In a [illegible] embedding, the encoding is partially based on a direct translation of the guest logic into constructs of the host logic. In a [illegible] embedding the syntax and the semantic are formalised as datatypes. At a fundamental level, taking the view presented in Sec. 2, the deep embedding of a logic is simply a definition of the set of all sequents (the terms) and a predicate marking those that are (the inference rules of the guest logic being encoded as constructors of this predicate).

Shallow embeddings of in higher-order logics have been proposed in several papers (cf. [6,7]) formalising the in or . Such embeddings are not dealing with the logic, and by using directly the host logic to express notions, they introduce a form of . If the objective is to have an accurate formalisation of the guest system, the definition of a valid interpretation is difficult – e.g. functions are relations, possibly partial or undecidable, and translating accurately this concept in is a tricky exercise. aims at such an accurate formalisation, to pinpoint any problem of the theory with the objective to increase confidence in the developments when safety or security is a concern; in addition, we also have an implementation objective. In such cases, a deep embedding is fully justified – see for example the development of a sound and complete theorem prover for first-order logic verified in proposed in [8].

A deep embedding of the logic in is described in [9] (using notations with names), to validate the used by the prover of – yet not checking standard results, and without implementation goal. As far as the implementation of a trusted prover is concerned, we can also mention the encoding of the logic as a rewriting system proposed in [10].

Deep embeddings have also the advantage to clearly separate the host and the guest logics: in , excluded middle, provable in , is not promoted to . This improves readibility, and allows one to study meta-theoretical questions such as consistency. Furthermore, the host logic consistency is not endangered.

4 Formalising the *B* Logic in *Coq*

In this section, we present our embedding of the logic in the system; the embedding uses a representation that avoids ambiguities and constitutes an efficient solution w.r.t. the implementation objective (see [11,12]). Deviations between and its formalisation are described and justified.

Notation. B *definitions use upper case letters with standard notations.* BiCoq *uses lower case letters, and mixes* B *and* Coq *notations; standard notations are used for* Coq *(e.g.* $\forall$ *is the universal quantification) while dotted notations are used for the embedded* B *(e.g.* $\dot{\forall}$ *is the universal quantification constructor).*

Notation. $[T]$ *denotes the type of the lists whose elements have type* T*.*

4.1 Syntax

Given a set of identifiers (I), the logic syntax defines predicates (P), expressions (E), sets (S) and variables (V) as follows:

$$\begin{array}{lcllllll}
P & := & P \wedge P & \mid P \Rightarrow P & \mid \neg P & \mid \forall V \cdot P & \mid E = E & \mid E \in E \mid [V{:=}E]P \\
E & := & V & \mid S & \mid E \mapsto E & \mid \downarrow S & \mid [V{:=}E]E & \\
S & := & \mathbf{BIG} & \mid \uparrow S & \mid S \times S & \mid \{V \mid P\} & & \\
V & := & I & \mid V, V & & & &
\end{array}$$

In this syntax, $[V := E]T$ represents the (elementary) substitution, V_1, V_2 a list of variables, $E_1 \mapsto E_2$ a pair of expressions, $\downarrow$ and $\uparrow$ the and operators, and **BIG** a constant set. The comprehension set operator, while syntactically defined by $\{V|P\}$, is rejected at if not of the form $\{V|V \in S \wedge P\}$, with V a variable not free in S

Definition. *Other connectors are defined from the previous ones, $P \Leftrightarrow Q$ is defined as $P \Rightarrow Q \wedge Q \Rightarrow P$, $P \vee Q$ as $\neg P \Rightarrow Q$, and $\exists V \cdot P$ as $\neg \forall V \cdot \neg P$.*

The first design choice of is to use a pure nameless notation (see [11,13]), where variables are represented by indexes giving the position of their binder – here the universal quantifier and the comprehension set. When an index exceeds the number of parent binders, it is said to be and represents a , whose name is provided by a scope (left implicit in this paper), so that any syntactically correct term is semantically valid, and there is no need for well-formedness condition[1]. In this representation, proofs of side conditions related to name clashing are replaced by computations on indexes, but the index representing a variable is not constant in a term.

The syntax is formalised in by two mutually inductive types with the following constructors, $\mathbb{I}$ being the set of indexes (that is, $\mathbb{N} \backslash \{0\}$) and $\mathbb{J}$ an infinite set of names with a decidable equality:

$$\begin{array}{lllllllll}
\mathbb{P} & := & \mathbb{P} \dot{\wedge} \mathbb{P} & \mid \mathbb{P} \dot{\Rightarrow} \mathbb{P} & \mid \dot{\neg} \mathbb{P} & \mid \dot{\forall}\, \mathbb{P} & \mid \mathbb{E} \dot{=} \mathbb{E} & \mid \mathbb{E} \dot{\in} \mathbb{E} & \\
\mathbb{E} & := & \dot{\chi} \mathbb{I} & \mid \mathbb{E} \dot{\mapsto} \mathbb{E} & \mid \dot{\downarrow} \mathbb{E} & \mid \dot{\Omega} & \mid \dot{\uparrow} \mathbb{E} & \mid \mathbb{E} \dot{\times} \mathbb{E} & \mid \{\mathbb{E} \dot{|} \mathbb{P}\} \mid \dot{\omega} \mathbb{J}
\end{array}$$

$\mathbb{P}$ represents predicates, while $\mathbb{E}$ merges expressions, sets and variables.

Using a representation, binders $\dot{\forall}$ and $\{\dot{|}\}$ have no attached names and only bind (implicitly) a single variable. Binding over list of variables can be eliminated without loss of expressivity, as illustrated by the following example:

$\{V \mid V \in S_1 \times S_2 \wedge \exists V_1 \cdot (V_1 \in S_1 \wedge \exists V_2 \cdot (V_2 \in S_2 \wedge V_1 \mapsto V_2 = V \wedge P))\}$ represents $\{V_1, V_2 \mid V_1, V_2 \in S_1 \times S_2 \wedge P\}$ [2]

The constructor $\{\dot{|}\}$ is further modified to be parameterised by an expression, to keep in the syntax definition only wf-checkable terms. Indeed, only comprehension sets of the form $\{V \mid V \in E \wedge P\}$, with V not free in E, are valid. The representation of this set is $\{e \dot{|} p\}$; to reflect the non-freeness condition, $\{e \dot{|} p\}$ only binds variables in its predicate parameter p. By these design choices, we bridge the gap between syntactically correct terms and wf-checkable ones, while being conservative.

$\dot{\Omega}$ represents the constant set **BIG**, $\dot{\chi}$ unary ( ) variables. The constructor $\dot{\omega}$ is without equivalent, and provides elements of $\dot{\Omega}$ (cf. Par. 4.3).

Notation. *$\dot{\chi}_i$ denotes the application of constructor $\dot{\chi}$ to $i : \mathbb{I}$ and $\dot{\omega}_j$ of constructor $\dot{\omega}$ to $j : \mathbb{J}$. By abuse of notation the variable $\dot{\chi}_i$ is also denoted simply by i.*

---

[1] An alternative approach to avoid well-formedness conditions is described in [14].

[2] This second representation, while standard in B, appears to be an illegal binding over the expression $x \mapsto y$ rather than over the variable x, y, but the same notations are used for both in [1] and such confusions are frequent.

Finally, the elementary substitution is not considered in as a syntactical construct but is replaced by functions on terms – substitution being introduced earlier in only to be used in the description of inference rules. Note however that the full of can still be formalised by additional terms constructors (the approach, see [15,16]).

Notation. *$p_1 \dot{\Leftrightarrow} p_2$ is defined as $p_1 \dot{\Rightarrow} p_2 \dot{\wedge} p_2 \dot{\Rightarrow} p_1$, $p_1 \dot{\vee} p_2$ as $\dot{\neg} p_1 \dot{\Rightarrow} p_2$, and $\dot{\exists} p$ as $\dot{\neg} \dot{\forall} \dot{\neg} p$.*

Notation. *$\mathbb{T}$ denotes the type of terms, that is the union of $\mathbb{P}$ and $\mathbb{E}$.*

4.2 Dealing with the *De Bruijn* Notation

notation is an elegant solution to avoid complex name management, and it has numerous merits. But it also has a big drawback, being an unusual representation for human readers:

If $x \in y$ is the interpretation of the term $1 \dot{\in} 2$, the interpretation of the term $\dot{\forall}(1 \dot{\in} 2)$ is $\forall t \cdot t \in x$; because of the binder, the scope has shifted (so 2 now represents x), and (likely) the semantic has been distorted.

In this paragraph, we illustrate some of the consequences of using a notation, as well as how to mask such consequences from the users.

Induction. When defining type $\mathbb{T}$, automatically generates the associated structural induction principle. As illustrated in Ex. 6, it is however not semantically adequate, because it does not reflect indexes scoping. A more interesting principle is derived in by using the syntactical depth function $\mathcal{D}$ of a term as a well-founded measure:

$$\forall\,(P \colon \mathbb{T} \rightarrow Prop), (\forall\,(t \colon \mathbb{T}), (\forall\,(t' \colon \mathbb{T}), \mathcal{D}(t') < \mathcal{D}(t) \rightarrow P\,t') \rightarrow P\,t) \rightarrow \forall\,(t \colon \mathbb{T}),\ P\,t$$

With this principle, for the term $\dot{\forall}(1 \dot{\in} 3)$ (that is, $\forall t \cdot t \in y$) we can choose to use an induction hypothesis on $1 \dot{\in} 2$ (that is, $x \in y$) instead of $1 \dot{\in} 3$ (that is, $x \in z$).

Non-Freeness. The notation $V \backslash T$ means that the variable V does not appear free in T. Non-freeness is defined in as a type $\dot{\setminus} \colon \mathbb{I} \rightarrow \mathbb{T} \rightarrow Prop$ (a relation between $\mathbb{I}$, representing the variables, and $\mathbb{T}$), with the following rules[3]:

$$\frac{}{i \dot{\setminus} \dot{\Omega}} \qquad \frac{}{i \dot{\setminus} \dot{\omega}_k} \qquad \frac{i_1 \neq i_2}{i_1 \dot{\setminus} i_2} \qquad \frac{(i{+}1) \dot{\setminus} p}{i \dot{\setminus} \dot{\forall}\, p} \qquad \frac{i \dot{\setminus} e \quad (i{+}1) \dot{\setminus} p}{i \dot{\setminus} \{e \dot{|} p\}}$$

The two first rules are axioms, the associated constructors are atomic and do not interact with variables. The rules for $\dot{\forall}$ and $\{\dot{|}\}$ reflect the fact that the associated constructors are binders and therefore shift the scope.

---

[3] The rules for the other constructors are trivial and can be obtained by straightforward extension, e.g. here $i \dot{\setminus} p$ and $i \dot{\setminus} q$ allow to derive $i \dot{\setminus} p \dot{\Rightarrow} q$.

Binding, Instantiation and Substitution. It is possible to define functions to simulate binding (that is the use of ∀ or {}, representing λ-abstraction). These functions constitute a built-in user interface to produce terms while using the usual representation, making indexes and their management invisible to the user (see also [17] for a similar approach):

$$\begin{tikzcd}
\text{Usual rep.} & & \forall V_1 \cdot V_1 \in \{V_2 \mid V_2 \in E \wedge V_1 = V_2\} \arrow[dll, bend right] \\
\text{Functional rep.} & \upharpoonleft_{\forall}(i_1 \cdot i_1 \dot{\in} \uparrow_{\{\}}(i_2 : e \cdot i_1 \dot{=} i_2)) \arrow[dr, bend right, "Computation"'] & \\
\text{Internal rep.} & & \dot{\forall}\,(1 \,\dot{\in} \{e \dot{|} 2 \dot{=} 1\}) \arrow[uu, bend right, "\text{Pretty-printing}"']
\end{tikzcd}$$

The binding functions are defined by:

$$\upharpoonleft_{\forall}(i \cdot p) := \dot{\forall}\,\mathbf{Bind}\; i\; 1\; p \qquad \uparrow_{\{\}}(i : e \cdot p) := \{e \dot{|} \mathbf{Bind}\; i\; 1\; p\} \qquad \upharpoonleft_{\exists}(i \cdot p) := \dot{\exists}\mathbf{Bind}\; i\; 1\; p$$

$$\begin{array}{lll}
\multicolumn{3}{l}{\mathbf{Bind}(i_1\; i_2 : \mathbb{I})(t : \mathbb{T}) : \mathbb{T} := \mathit{match}\, t\, \mathit{with}} \\
\mid & \dot{\Omega} \;\mid\; \dot{\omega}_{j'} & \Rightarrow t \\
\mid & \dot{\chi}_{i'} & \Rightarrow t \text{ if } i' < i_2 \text{, or else } \dot{\chi}_{i_2} \text{ if } i' = i_1 \text{, or else } \dot{\chi}_{i'+1} \\
\mid & \dot{\forall}\, p' & \Rightarrow \dot{\forall}\,(\mathbf{Bind}\; (i_1+1)\; (i_2+1)\; p') \\
\mid & \{e' \dot{|} p'\} & \Rightarrow \{\mathbf{Bind}\; i_1\; i_2\; e' \dot{|} \mathbf{Bind}\; (i_1+1)\; (i_2+1)\; p'\} \\
\mid & \ldots & \Rightarrow \ldots \text{ (straightforward extension)}
\end{array}$$

On the same principles, the definition of instantiation functions (for elimination of ∀ or {}, representing β-reduction and denoted by $\downarrow_{\forall}(p \leftarrow e) : \mathbb{P} \rightarrow \mathbb{E} \rightarrow \mathbb{P}$ and $\downarrow_{\{\}}(e_1 \leftarrow e_2) : \mathbb{E} \rightarrow \mathbb{E} \rightarrow \mathbb{P}$) is straightforward – being partial, these functions just require in an additional proof parameter (omitted in this paper) that the term is of the expected form. Finally, it is also possible to define a substitution function[4]:

$$\begin{array}{lll}
\multicolumn{3}{l}{\langle i := e \rangle t : \mathbb{I} \rightarrow \mathbb{E} \rightarrow \mathbb{T} \rightarrow \mathbb{T} := \mathit{match}\, t\, \mathit{with}} \\
\mid & \dot{\Omega} \;\mid\; \dot{\omega}_{j'} & \Rightarrow t \\
\mid & \dot{\chi}_{i'} & \Rightarrow \mathit{if}\; i' = i\; \mathit{then}\; e\; \mathit{else}\; t \\
\mid & \dot{\forall}\, p' & \Rightarrow \dot{\forall}\,\langle i+1 := \mathbf{Lift}(e) \rangle p' \\
\mid & \{e' \dot{|} p'\} & \Rightarrow \{\langle i := e \rangle e' \dot{|} \langle i+1 := \mathbf{Lift}(e) \rangle p'\} \\
\mid & \ldots & \Rightarrow \ldots \text{ (straightforward extension)}
\end{array}$$

where **Lift**, not detailed in this paper, increments dangling indexes. Remember that substitution is introduced early in as a syntactical construct, but only to be used in inference rules. We consider that such rules are better represented using the resulting term (that is, the reduction of the application of the substitution).

Once these functions are defined, numerous lemmas are proved, such as the (in)famous ones describing all possible interactions between lifting, binding, instantiation and substitution. The following results are then derived, proving the irrelevance of α-renaming or describing relationships between instantiation, binding and substitution (with = the term structural equality):

---

[4] Substitution and instantiation may seem similar in usual notation, but their differences are emphasised when using *De Bruijn* notation.

$$i_2 \dot{\setminus} p \rightarrow \Uparrow_\forall(i_1 \cdot p) = \Uparrow_\forall(i_2 \cdot \langle i_1 := i_2 \rangle p) \qquad i_2 \dot{\setminus} p \rightarrow \uparrow_{\{\}}(i_1 : e \cdot p) = \uparrow_{\{\}}(i_2 : e \cdot \langle i_1 := i_2 \rangle p)$$

$$\downarrow_\forall(\Uparrow_\forall(i \cdot p) \leftarrow i) = p \qquad \downarrow_{\{\}}(\uparrow_{\{\}}(i : e \cdot p) \leftarrow i) = i \dot{\in} e \mathbin{\dot{\wedge}} p$$

$$\downarrow_\forall(\Uparrow_\forall(i \cdot p) \leftarrow e) = \langle i := e \rangle p$$

4.3 Inference Rules

Having formalised the syntax and defined some functions and properties on terms, the next step is to encode the inference rules. Thanks to the use of the functional representation described in the previous paragraph, rules look very much like the standard rules. The translation is therefore straightforward, merely a syntactical one, and the risk of error is very limited.

In our formalisation sets of hypothesis are represented by lists, with membership ($\in$) and inclusion ($\subseteq$) as well as the pointwise extension of non-freeness ($\dot{\setminus}$). The inference rules are formalised as constructors of an inductive type $\vdash\, : [\mathbb{P}] \rightarrow \mathbb{P} \rightarrow Prop$, that is $g \vdash p$ is the type of all proofs of p under the assumptions g. Such a type may be inhabited (i.e. p is provable assuming g) or empty (i.e. there is no proof of p under the assumptions of g).

The rules and their encoding as constructors are detailed in Tab. 1, universal quantifications being omitted (the types are $g, g_1, g_2 : [\mathbb{P}]$; $p, p_1, p_2 : \mathbb{P}$; $e, e_1, e_2, e_3, e_4 : \mathbb{E}$, $i, i_1, i_2 : \mathbb{I}$ and $j, j_1, j_2 : \mathbb{J}$). For most of them, translation is straightforward, only taking care to use functional substitution and binding where appropriate. On the other hand, the use of the functional representation imposes to keep the syntactical side conditions, except for the comprehension set rule, where such condition is embedded in the syntax; new rules have to be derived to benefit of the internal representation.

Only the last two inference rules deserve discussion. The first one of these indicates that the constant set **BIG** is infinite, using the **infinite** predicate defined by a fixpoint; unfolding this definition to produce a translation is possible, but not practical. Therefore, this rule is replaced in by two different rules allowing to exhibit an infinity of elements of **BIG**, $\mathbb{J}$ being itself infinite.

The last rule, defining the semantics of pairs and products, is more interesting. A straightforward translation of this rule indeed leads to the impossibility to prove, in , the following theorems from [1]:

$$\vdash (E \mapsto F) = (E' \mapsto F') \Rightarrow E = E' \wedge F = F'$$
$$\vdash S \in\uparrow U \wedge T \in\uparrow V \Rightarrow (S \times T) \in\uparrow (U \times V)$$

The proof of the first result provided in [1] is flawed, due to a confusion between pairs of expressions and lists of variables (as pointed out in [18]), both using the same notation – and cannot be corrected in the absence of a form of destructor for pairs. On the other hand, the proof of the monotonicity of cartesian product w.r.t. inclusion is not detailed in [1], being considered trivial. However, using the listed rules, one may derive predicates of the form $V \in S \times T$ but without being able to constraint V to be a pair to apply the last rule (a classical problem of

Table 1. Encoding of the B inference rules

B inference rules	*BiCoq* formalisation
$\dfrac{}{P \vdash P}$	None, derived from $[\in]$
$\dfrac{P \text{ appears in } \Gamma}{\Gamma \vdash P}$	$p \in g \rightarrow g \dot{\vdash} p \; [\in]$
$\dfrac{\Gamma' \text{ includes } \Gamma \qquad \Gamma \vdash P}{\Gamma' \vdash P}$	$g_1 \dot{\vdash} p \rightarrow g_1 \subseteq g_2 \rightarrow g_2 \dot{\vdash} p \; [\subseteq]$
$\dfrac{\Gamma \vdash P \quad \Gamma, P \vdash Q}{\Gamma \vdash Q}$	None, derived from $[\neg_n]\ [\neg_p]\ [\subseteq]\ [\in]$
$\dfrac{\Gamma \vdash P \Rightarrow Q}{\Gamma, P \vdash Q} \qquad \dfrac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q}$	$g \dot{\vdash} p_1 \dot{\Rightarrow} p_2 \rightarrow g, p_1 \dot{\vdash} p_2$ $g, p_1 \dot{\vdash} p_2 \rightarrow g \dot{\vdash} p_1 \dot{\Rightarrow} p_2$
$\dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}$	$g \dot{\vdash} p_1 \rightarrow g \dot{\vdash} p_2 \rightarrow g \dot{\vdash} p_1 \dot{\wedge} p_2 \; [\wedge_i]$
$\dfrac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \qquad \dfrac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q}$	$g \dot{\vdash} p_1 \dot{\wedge} p_2 \rightarrow g \dot{\vdash} p_1$ $g \dot{\vdash} p_1 \dot{\wedge} p_2 \rightarrow g \dot{\vdash} p_2$
$\dfrac{\Gamma, Q \vdash P \quad \Gamma, Q \vdash \neg P}{\Gamma \vdash \neg Q}$	$g, p_2 \dot{\vdash} p_1 \rightarrow g, p_2 \dot{\vdash} \dot{\neg} p_1 \rightarrow g \dot{\vdash} \dot{\neg} p_2 \; [\neg_p]$
$\dfrac{\Gamma, \neg Q \vdash P \quad \Gamma, \neg Q \vdash \neg P}{\Gamma \vdash Q}$	$g, \dot{\neg} p_2 \dot{\vdash} p_1 \rightarrow g, \dot{\neg} p_2 \dot{\vdash} \dot{\neg} p_1 \rightarrow g \dot{\vdash} p_2 \; [\neg_n]$
$\dfrac{}{\Gamma \vdash E = E}$	$g \dot{\vdash} e \doteq e$
$\dfrac{\Gamma \vdash P \quad V \backslash \Gamma}{\Gamma \vdash \forall V \cdot P}$	$i \dot{\setminus} g \rightarrow g \dot{\vdash} p \rightarrow g \dot{\vdash} \upharpoonleft_{\forall}(i \cdot p) \; [\forall_i]$
$\dfrac{\Gamma \vdash \forall V \cdot P}{\Gamma \vdash [V := E] P}$	$g \dot{\vdash} \upharpoonleft_{\forall}(i \cdot p) \rightarrow g \dot{\vdash} \langle i := e \rangle p$
$\dfrac{V \backslash S}{\vdash E \in \{V \mid V \in S \wedge P\} \Leftrightarrow E \in S \wedge [V := E] P}$	$\dot{\vdash} e_1 \dot{\in} \upharpoonleft_{\{\}}(i : e_2 \cdot p) \dot{\Leftrightarrow} e_1 \dot{\in} e_2 \dot{\wedge} \langle i := e_1 \rangle p$
$\dfrac{\Gamma \vdash E = F \quad \Gamma \vdash [V := E] P}{\Gamma \vdash [V := F] P}$	$g \dot{\vdash} e_1 \doteq e_2 \rightarrow g \dot{\vdash} \langle i := e_1 \rangle p \rightarrow g \dot{\vdash} \langle i := e_2 \rangle p$
$\dfrac{V \backslash S}{\vdash \exists V \cdot (V \in S) \Rightarrow \downarrow S \in S}$	$i \dot{\setminus} e \rightarrow g \dot{\vdash} \upharpoonleft_{\exists}(i \cdot i \dot{\in} e) \dot{\Rightarrow} \dot{\downarrow} e \dot{\in} e$
$\dfrac{V \backslash S, T}{\vdash S \in \uparrow T \Leftrightarrow \forall V \cdot (V \in S \Rightarrow V \in T)}$	$i \dot{\setminus} e_1 \rightarrow i \dot{\setminus} e_2 \rightarrow g \dot{\vdash} e_1 \dot{\in} \dot{\uparrow} e_2 \dot{\Leftrightarrow} \upharpoonleft_{\forall}(i \cdot i \dot{\in} e_1 \dot{\Rightarrow} i \dot{\in} e_2)$
$\dfrac{V \backslash S, T}{\vdash \begin{pmatrix} \forall V \cdot (V \in S \Rightarrow V \in T) \\ \wedge \forall V \cdot (V \in T \Rightarrow V \in S) \end{pmatrix} \Leftrightarrow S = T}$	$g \dot{\vdash} e_1 \dot{\in} \dot{\uparrow} e_2 \rightarrow g \dot{\vdash} e_2 \dot{\in} \dot{\uparrow} e_1 \rightarrow g \dot{\vdash} e_1 \doteq e_2$
$\dfrac{}{\vdash \mathbf{infinite}(BIG)}$	$g \dot{\vdash} \dot{\omega}_j \dot{\in} \dot{\Omega}$ $j_1 \neq j_2 \rightarrow g \dot{\vdash} \dot{\neg}(\dot{\omega}_{j_1} \doteq \dot{\omega}_{j_2})$
$\dfrac{}{\vdash (E \mapsto F) \in (S \times T) \Leftrightarrow (E \in S) \wedge (F \in T)}$	$g \dot{\vdash} e_1 \dot{\mapsto} e_2 \doteq e_3 \dot{\mapsto} e_4 \rightarrow g \dot{\vdash} e_1 \doteq e_3$ $g \dot{\vdash} e_1 \dot{\mapsto} e_2 \doteq e_3 \dot{\mapsto} e_4 \rightarrow g \dot{\vdash} e_2 \doteq e_4$ $i_1 \dot{\setminus} e \dot{\in} (e_1 \dot{\times} e_2) \rightarrow i_2 \dot{\setminus} e \dot{\in} (e_1 \dot{\times} e_2) \rightarrow i_1 \neq i_2 \rightarrow$ $g \dot{\vdash} \upharpoonleft_{\exists}(i_1 \cdot i_1 \dot{\in} e_1 \dot{\wedge} \upharpoonleft_{\exists}(i_2 \cdot i_2 \dot{\in} e_2 \dot{\wedge} e \doteq i_1 \dot{\mapsto} i_2)) \dot{\Leftrightarrow} e \dot{\in} (e_1 \dot{\times} e_2)$

the untyped λ-calculus). Basically, injectivity and surjectivity rules are lacking; these observations, probably well known of the gurus but not documented to our knowledge, have led us to replace this rule by three new rules in order to be able to prove the expected theorems. Again, this process illustrates our conservative approach.

5 Proofs in *BiCoq*

5.1 Standard *B* Proofs

Using the definition of $\dot{\vdash}$, we formally prove in all propositional calculus and predicate calculus results of [1], using the functional representation and following the proposed proof structure, e.g.:

$$i_1 \dot{\diagdown} g \rightarrow i_1 \dot{\diagdown} p \rightarrow g \dot{\vdash} \langle i_2 := i_1 \rangle p \rightarrow g \dot{\vdash} \Uparrow_{\forall}(i_2 \cdot p), \text{ that is } \frac{\Gamma \vdash [V_2 := V_1]P \quad V_1 \backslash \Gamma, P}{\Gamma \vdash \forall V_2 \cdot P}$$

To assist the proof construction provides tactics written in the tactic language [19]. For example, the propositional calculus procedure described in [1], proposing a strategy based on propositional calculus theorems, is provided as a tactic. More technical tactics are also available in , e.g. to obtain proved fresh variables.

An alternative form of theorems is also derived, using the internal representation; e.g. the $\forall$-introduction rule (to be compared with $[\forall_i]$) is:

$$i \dot{\diagdown} g \rightarrow i \dot{\diagdown} \dot{\forall}\, p \rightarrow g \dot{\vdash} \textbf{Inst}\ i\ 1\ p \rightarrow g \dot{\vdash} \dot{\forall}\, p$$

These last results are of course rather technical, not benefiting from the functional representation. Yet they have some interest, for technical lemmas or as derived rules in which only semantical side conditions remain (computations over indexes dealing with the syntactical ones).

5.2 Mixing *BiCoq* and *Coq* Logics

As it is standard in such a deep embedding (e.g. see [9]), provides also results expressing relations between host and guest logics:

$$(g \dot{\vdash} p \vee g \dot{\vdash} q) \rightarrow g \dot{\vdash} p \dot{\vee} q \qquad g \dot{\vdash} p \dot{\Rightarrow} g \rightarrow (g \dot{\vdash} p \rightarrow g \dot{\vdash} p)$$

$$(g \dot{\vdash} p \wedge g \dot{\vdash} q) \leftrightarrow g \dot{\vdash} p \dot{\wedge} q \qquad (\forall\, (y : \mathbb{I}),\ g \dot{\vdash} \langle x := y \rangle p) \leftrightarrow g \dot{\vdash} \Uparrow_{\forall}(x \cdot p)$$

Asymmetrical results mark the differences between the classical logic and the constructive logic – e.g. a reciprocal of the first rule, combined with the excluded middle, would prove that for any predicate p either $\dot{\vdash} p$ or $\dot{\vdash} \dot{\neg} p$, which of course is not the case. This emphasises the fact that both logics are well separated, the logic being embedded has an external theory.

By providing the best of both worlds, these results constitute efficient proof tactics. For example, the last theorem does not reflect non-freeness side conditions from to the logic (taking care of such conditions automatically).

6 Developing a Proved *B* Toolkit

In this section, we detail how is used as a framework for the development of formally checked toolkits. offers mechanisms to extract programs from constructive proofs (i.e. software from logical definitions and theorems), but a different approach is chosen here. Indeed, includes code (in the form of functions using the -like internal language of) which is proved correct. This code is extractible by a pure syntactical process, e.g. in , using the extraction mechanism of . By doing so, we obtain proved tools whose code is small, readable and efficient – and independent of .

Notation. *$\mathbb{B}$ represents the booleans, $\top$ being true and $\bot$ being false.*

Notation. *Hat notations are used for boolean functions (e.g. $\widehat{\wedge}$ is the boolean and).*

6.1 Implementing Decidable Properties

For P and f respectively a predicate and a boolean function over a type S, we note $(P \rightsquigarrow f)$ when f decides P, i.e. when the following property is proved:

$$\forall(s:S), (f(s)=\top \rightarrow P(s)) \wedge (f(s)=\bot \rightarrow \neg P(s))$$

By defining folding as the extension of predicates and functions to lists, we prove that if f decides P, then the folding of f decides the folding of P:

$$\mathbf{Fold_p}(P) := fun(L:[S]) \Rightarrow \forall(s:S), s \in L \rightarrow P(s)$$
$$\mathbf{Fold_f}(f) := fun(L:[S]) \Rightarrow if\ \mathbf{empty}(L)\ then\ \top\ else\ f(\mathbf{head}(L))\widehat{\wedge}\mathbf{Fold_f}(f)(\mathbf{tail}(L))$$
$$(P \rightsquigarrow f) \rightarrow (\mathbf{Fold_p}(P) \rightsquigarrow \mathbf{Fold_f}(f))$$

Non-freeness is defined in B as a logical proposition and represented by the inductive type $\dot{\setminus}$ in *BiCoq*. Our implementation strategy consists in developing a program $\widehat{\setminus} : \mathbb{I} \rightarrow \mathbb{T} \rightarrow \mathbb{B}$ and to prove that $(\dot{\setminus} \rightsquigarrow \widehat{\setminus})$. Hence $\widehat{\setminus}$ and its extension (checking that a variable does not occur free in a list of hypotheses) are proved correct and can be extracted.

In this approach is systematic; all typed equalities are implemented and proved correct (e.g. term equality), as well as non-freeness, list membership, inclusion, etc. to constitute our formally checked toolkit.

6.2 A Proved Prover for the B Logic

In this paragraph we focus on the definition of an extractible prover to conduct first-order proofs for standard developments.

includes programs, named in the following, to simulate the application of inference rules or theorems. By providing such a dedicated piece of code for each of the inference rules listed in Tab. 1, and by proving them correct, we got a correct and complete prover (that is, any standard result can be derived using this prover).

To this end, a type for is defined as the product $[\mathbb{P}]\times\mathbb{P}$; for $g:[\mathbb{P}]$ and $p:\mathbb{P}$ we denote $g \Vdash p$ the associated pair. While $g \vdash p$ is the type of proofs of p under the assumptions g, that can be inhabited or not, $g \Vdash p$ is a syntactical construct extending $\mathbb{T}$. To interpret a sequent, we use the translation $\mathbf{Trans}_{\vdash}$ that for a pair $g \Vdash p$ returns the type $g \vdash p$ (and its extension derived by $\mathbf{Fold_p}$).

A tactic is a function $T_B :\Vdash \rightarrow [\Vdash]$ that, provided a goal $g \Vdash p$, returns a list of subgoals $[g_1 \Vdash p_1, \ldots, g_n \Vdash p_n]$ which together are sufficient to prove

$g \Vdash p$; if a tactic concludes (proves the goal) this list is empty. The following (elementary) examples give the definition of the tactics associated respectively to the inference rules $[\in]$ and $[\wedge_i]$:

$\mathbf{T}_{\in}(s) := let\ (g \Vdash p := s)\ in\ if\ p \widehat{\in} g\ then\ []\ else\ [s]$

$\mathbf{T}_{\wedge_i}(s) := let\ (g \Vdash p := s)\ in\ match\ p\ with\ p_1 \dot{\wedge} p_2 \Rightarrow [g \Vdash p_1, g \Vdash p_2] \mid \_ \Rightarrow [s]$

The implementation strategy described in Par. 6.1 is now particularly relevant, as $\mathbf{T}_{\in}$ uses the boolean function $\widehat{\in}$ instead of the logical proposition $\in$.

Following the same principles, numerous (much more complex) tactics are provided in , implementing theorems or strategies, such as the decision procedure for propositional calculus described in [1]. For each tactic T_B, the correctness is ensured by a proof of the following property:

$$\forall\,(s : \Vdash), \mathbf{Trans}_{\vdash}(T_B(s)) \rightarrow \mathbf{Trans}_{\vdash}(s), \text{ that is } \frac{g_1 \vdash p_1 \quad \dots \quad g_n \vdash p_n}{g \vdash p}$$

Thanks to the functions defined in Par. 4.2, management of the indexes can be hidden from the users of the tactics. With the programs already provided in (such as non-freeness, binding, etc.), these tactics constitute the core of a proved prover. This prover still lacks automation and , and should be coupled with other tools, for example a parser using the platform [20].

7 Higher-Order Considerations and Extensions

While the logic is first-order, various definitions and proofs in [1] are conducted in a higher-order meta-logic: results in propositional calculus are proved by induction over terms, and refinement is defined by quantification over predicates before being transformed into an equivalent first-order definition. Using the higher-order framework provided by , can clearly be extended to integrate and to formally check such concepts.

New results can also be derived; for example, using the proof depth function $\mathcal{D}_{\vdash} : \vdash \dot{\rightarrow} \mathbb{I}$, we obtain a depth induction principle on proof trees e.g. for results about proof rewriting. Other results, proved in higher-order logic, are applicable in first-order logic, and implemented as tactics for standard proofs. This is the case for the following congruence results.

Predicate Substitution. We extend the logic syntax with a new constructor $\dot{\pi}\mathbb{K} : \mathbb{P}$ ($\mathbb{K}$ being an infinite set of names with a decidable equality), without adding any inference rules in order not to enrich the logic[5]. Only limited modifications of are required to deal with this new constructor, e.g. non-freeness with the additional rule $\forall\,(i : \mathbb{I})(k : \mathbb{K}),\ i \dot{\setminus} \dot{\pi}_k$.

Predicate variables play a role similar to the one of the variables – they are placeholders that can be replaced by a predicate using the substitution function

---

[5] However, some new (propositional) sequents became provable, such as $\dot{\pi}_k \dot{\vdash} \dot{\pi}_k$.

$\langle k :\equiv p_1\rangle p_2 : \mathbb{K} \rightarrow \mathbb{P} \rightarrow \mathbb{P} \rightarrow \mathbb{P}$, not detailed in this paper, that mimicks the expression substitution function (see Par. 4.2). Thanks to this extension, we can prove the following congruence rules for $\dot{\Leftrightarrow}$ and implement associated tactics that can be used e.g. to unfold a definition in a term, even under binders:

$$g \dot{\vdash} p_1 \dot{\Leftrightarrow} p_2 \rightarrow g \dot{\vdash} \langle j :\equiv p_1\rangle p \dot{\Leftrightarrow} \langle j :\equiv p_2\rangle p \qquad g \dot{\vdash} p_1 \dot{\Leftrightarrow} p_2 \rightarrow g \dot{\vdash} \langle j :\equiv p_1\rangle e \doteq \langle j :\equiv p_2\rangle e$$

$x \doteq 0, y \dot{\in} \mathbb{N} \dot{\vdash} y \leq x \dot{\Leftrightarrow} y \doteq 0$, therefore we immediately derive (in one step) $x \doteq 0, y \dot{\in} \mathbb{N} \dot{\vdash} \Uparrow_{\forall}(v \cdot v \dot{\in} \uparrow_{\{\}}(t : \mathbb{N} \cdot t \leq y \dot{\wedge} y \leq x)) \dot{\Leftrightarrow} \Uparrow_{\forall}(v \cdot v \dot{\in} \uparrow_{\{\}}(t : \mathbb{N} \cdot t \leq y \dot{\wedge} y \doteq 0))$

Note that predicate substitution and expression substitution mechanically forbid the capture of variables in the substituted subterm, by lifting dangling indexes when crossing a binder. That is, in Ex. 10, if v or t appear free in the substituted subterm, they escape capture during substitution.

Predicate Grafting. Other congruence results can be derived for of predicates, a modified substitution (not lifting the substituted subterm) allowing for the capture of variables:

$$\begin{array}{llll}
\multicolumn{4}{l}{\langle k \lhd p\rangle t : \mathbb{K} \rightarrow \mathbb{P} \rightarrow \mathbb{T} \rightarrow \mathbb{T} := match\, t\, with} \\
\quad | & \dot{\Omega} \quad | \quad \dot{\omega}_{j'} \quad | \quad \dot{\chi}_{i'} & \Rightarrow & t \\
\quad | & \dot{\pi}_{k'} & \Rightarrow & if\ k' = k\ then\ p\ else\ t \\
\quad | & \dot{\forall}\, p' & \Rightarrow & \dot{\forall}\, \langle k \lhd p\rangle p' \\
\quad | & \{e' \dot{|} p'\} & \Rightarrow & \{\langle k \lhd p\rangle e' \dot{|} \langle k \lhd p\rangle p'\} \\
\quad | & \ldots & \Rightarrow & \ldots \text{ (straightforward extension)}
\end{array}$$

The associated congruence results and proofs are technical, and not detailed in this paper. We just provide for illustration a simplified version of these results:

$$\dot{\vdash} p_1 \dot{\Leftrightarrow} p_2 \rightarrow g \dot{\vdash} \langle j \lhd p_1\rangle p \dot{\Leftrightarrow} \langle j \lhd p_2\rangle p \qquad \dot{\vdash} p_1 \dot{\Leftrightarrow} p_2 \rightarrow g \dot{\vdash} \langle j \lhd p_1\rangle e \doteq \langle j \lhd p_2\rangle e$$

$g \dot{\vdash} \langle k \lhd \dot{\neg}\dot{\neg} p\rangle q \dot{\Leftrightarrow} \langle j \lhd p\rangle q$, that is the elimination of double negations in a subterm (even if dangling *De Bruijn* indexes of p are bound in q)

Remark. Results such as the ones in Exs. 10 or 11 are provable in , on a case-by-case basis, with a first-order proof depending on the structure of the term in which substitution or grafting is done. It is therefore conceivable to develop a specific (and likely complex) tactic automatically building for such goals a proof using the inference rules. On the contrary, the proposed extensions provide a new approach through results derived from a higher-order proof; the associated tactics are therefore simpler, and produce generic (and shorter) proofs by using not only the inference rules but also induction on $\mathbb{T}$.

8 Conclusion

Through an accurate deep embedding of the logic in , we identify shortfalls or confusions in [1] and propose amendments in order to be able to validate standard results – improving the confidence in the method and in the developments conducted with it. We describe a strategy to further benefit from this deep embedding by implementing verified tools, extractible to be used independently of . The approach is illustrated by the development of tactics that constitute a complete and correct prover – usable to conduct proofs (provided further automation), or to check proofs produced by other tools. The objective, again, is to have better confidence in the developments conducted in .

We also explain how, benefiting from the higher-order features of , new results for can be derived, and present an extension to derive congruence theorems related to equivalence, implemented in our prover.

All the results presented in this paper are mechanically checked; currently represents about 550 definitions (i.e. types, properties, functions), 750 theorems and proofs in – and about 6 man.months of development. It has now to be extended with the following definitions and results:

- Generation by the prover of proof terms checkable by .
- Use of a locally nameless representation with named free variables to derive unified congruence results (merging substitution and grafting).
- Fixpoint constructs, with application to the definition of natural numbers in the style; on the innovative side, we expect to derive inductive tactics, not available in current implementations.
- definition – either through a shallow embedding (an approach similar to the one presented in [6], but in) or through a deep embedding (with higher-order and first-order refinement definitions, and proof of equivalence).

We would like to emphasise the simplicity and the efficiency of the deep embedding approach, when having both validation and implementation objectives. In a relatively short amount of time, it was possible to describe the logic, to check its standard results, and to implement a proved prover for this logic.

Acknowledgements. We thank Pr. Hardin for reviewing earlier versions of this paper.

References

1. Abrial, J.R.: The B-Book - Assigning Programs to Meanings. Cambridge University Press, Cambridge (1996)
2. The Coq development team: The Coq proof assistant reference manual. LogiCal Project (2004)
3. Gordon, M.J.C.: Mechanizing programming logics in higher-order logic. In: Birtwistle, G.M., Subrahmanyam, P.A. (eds.) Current Trends in Hardware Verification and Automatic Theorem Proving (Proceedings of the Workshop on Hardware Verification), Banff, Canada, pp. 387–439. Springer, Berlin (1988)
4. Boulton, R.J., Gordon, A., Gordon, M.J.C., Harrison, J., Herbert, J., Tassel, J.V.: Experience with embedding hardware description languages in hol. In: Stavridou, V., Melham, T.F., Boute, R.T. (eds.) TPCD. IFIP Transactions, North-Holland, vol. A-10, pp. 129–156 (1992)

5. Azurat, A., Prasetya, I.: A survey on embedding programming logics in a theorem prover. Technical Report UU-CS-2002-007, Institute of Information and Computing Sciences, Utrecht University (2002)
6. Bodeveix, J.P., Filali, M., Muñoz, C.: A formalization of the B-method in Coq and PVS. In: Woodcock, J.C.P., Davies, J., Wing, J.M. (eds.) FM 1999. LNCS, vol. 1709, pp. 33–49. Springer, Heidelberg (1999)
7. Chartier, P.: Formalisation of B in Isabelle/HOL. In: Bert, D. (ed.) B 1998. LNCS, vol. 1393, pp. 66–82. Springer, Heidelberg (1998)
8. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 294–309. Springer, Heidelberg (2005)
9. Berkani, K., Dubois, C., Faivre, A., Falampin, J.: Validation des règles de base de l'Atelier B. Technique et Science Informatiques 23(7), 855–878 (2004)
10. Cirstea, H., Kirchner, C.: Using rewriting and strategies for describing the B predicate prover. In: Kirchner, C., Kirchner, H. (eds.) CADE-15. LNCS (LNAI), vol. 1421, pp. 25–36. Springer, Heidelberg (1998)
11. de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. Indagationes Mathematicae (Proceedings), pp. 381–392 (1972)
12. Liang, C., Nadathur, G.: Tradeoffs in the intensional representation of lambda terms. In: Tison, S. (ed.) RTA 2002. LNCS, vol. 2378, pp. 192–206. Springer, Heidelberg (2002)
13. Aydemir, B., Charguéraud, A., Pierce, B.C., Weirich, S.: Engineering aspects of formal metatheory, Manuscript (2007)
14. Bird, R., Paterson, R.: De Bruijn notation as a nested datatype. Journal of Functional Programming 9, 77–91 (1999)
15. Abadi, M., Cardelli, L., Curien, P.L., Lévy, J.J.: Explicit substitutions. Journal of Functional Programming 1, 375–416 (1991)
16. Curien, P.L., Hardin, T., Lévy, J.J.: Confluence properties of weak and strong calculi of explicit substitutions. Journal of the ACM 43, 362–397 (1996)
17. Gordon, A.D.: A mechanisation of name-carrying syntax up to alpha-conversion. In: Joyce, J.J., Seger, C.-J.H. (eds.) HUG 1993. LNCS, vol. 780, pp. 413–425. Springer, Heidelberg (1994)
18. Mussat, L.: Private Communication (2005)
19. Delahaye, D.: A tactic language for the system Coq. In: Parigot, M., Voronkov, A. (eds.) LPAR 2000. LNCS (LNAI), vol. 1955, pp. 85–95. Springer, Heidelberg (2000)
20. Colin, S., Petit, D., Rocheteau, J., Marcano, R., Mariano, G., Poirriez, V.: BRILLANT: An open source and XML-based platform for rigourous software development. In: SEFM (Software Engineering and Formal Methods), Koblenz, Germany, AGKI (Artificial Intelligence Research Koblenz), IEEE Computer Society Press, Los Alamitos (2005) selectivity : 40/120

How Many Legs Do I Have?
Non-Simple Roles in Number Restrictions Revisited

Yevgeny Kazakov, Ulrike Sattler, and Evgeny Zolin

School of Computer Science, The University of Manchester, UK
`{ykazakov,sattler,ezolin}@manchester.ac.uk`

Abstract. The Description Logics underpinning OWL impose a well-known syntactic restriction in order to preserve decidability: they do not allow to use non-simple roles—that is, transitive roles or their super-roles—in number restrictions. When modeling composite objects, for example in bio-medical ontologies, this restriction can pose problems.X

Therefore, we take a closer look at the problem of counting over non-simple roles. On the one hand, we sharpen the known undecidability results and demonstrate that: (i) for DLs with inverse roles, counting over non-simple roles leads to undecidability even when there is only one role in the language; (ii) for DLs without inverses, two transitive and an arbitrary role are sufficient for undecidability. On the other hand, we demonstrate that counting over non-simple roles does not compromise decidability in the absence of inverse roles provided that certain restrictions on role inclusion axioms are satisfied.

1 Introduction

Recently, Description Logics (DLs) [1] have attracted increasing attention, partially due to their usage as logical underpinning of ontology languages such as OIL, DAML+OIL, and OWL[1] [5]. All these languages are based on DLs of the $\mathcal{SHQ}$ family, which are decidable fragments of first order logic and close relatives of modal logics. In DLs, unary predicates/propositional variables are usually called concepts, binary predicates/modal parameters are called roles and, in a nutshell, $\mathcal{SHQ}$ extends $\mathcal{ALC}$ (a notational variant of multi-modal K) with transitivity and role inclusion axioms and with *number restrictions*: these are concepts of the form $(\leqslant n\, R.C)$ for n a non-negative integer, R a role, and C a possibly complex concept. Number restrictions are heavily used to define concepts, e.g., the following expression makes use of standard DL notation to define the concept `Human` as featherless bipeds:

$$\mathtt{Human} = \mathtt{Mammal} \sqcap \forall \mathtt{hasPart}.\neg\mathtt{Feather} \sqcap (\geqslant 2\, \mathtt{hasPart.Leg}) \sqcap (\leqslant 2\, \mathtt{hasPart.Leg})$$

We find numerous more convincing yet less readable such applications of number restrictions in bio-informatics and medical applications, e.g., they are used to restrict the number of certain components of proteins [8].

---

[1] OWL comes in three flavours, OWL Lite, OWL DL, and OWL Full. Here, we are only concerned with the first two.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 303–317, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Other heavily used features are the above mentioned *transitivity* and *role inclusion* axioms. They allow to express, e.g., that `hasPart` must be interpreted as a transitive relation (which is closely related to the modal logic K4) and that `hasComponent` implies `hasPart`.

Now ontology design and maintenance is a non-trivial task, especially since ontologies can be quite large: e.g., SNOMED and the National Cancer Institute ontology have over 300,000 resp. 17,000 defined concepts. In order to check for consistency and compute the (implicit) concept hierarchy w.r.t. the subsumption relationship, ontology editors make use of DL reasoners[2] which implement decision procedures for concept satisfiability and subsumption w.r.t. DL axioms. For this to be possible, i.e., for these reasoning problems to be decidable for $\mathcal{SHQ}$, their designer had to impose a syntactic restriction: in number restrictions, one can neither use transitive roles nor super-roles of transitive roles, i.e., number restrictions can only be used on *simple roles*. For example, if we want to make use of our definition of `Human`, we have to either refrain from making `hasPart` a transitive role or use, e.g., a (non-transitive) subrole such as `hasComp` of `hasPart` in its number restrictions. Both options are sub-optimal since they result in the loss of other, useful consequences. For the first option, e.g., we could add the following definition of `HumanBird` without causing a (useful) inconsistency:

$$\mathtt{HumanBird} = \mathtt{Human} \sqcap \exists \mathtt{hasPart}.(\mathtt{Wing} \sqcap \exists \mathtt{hasPart}.\mathtt{Feather}).$$

For the second option, e.g., we could add the following definition of `3LHuman` without causing an inconsistency (please note that here we use twice the sub-role `hasComp` of `hasPart` and only once `hasPart`):

$$\begin{aligned}\mathtt{3LHuman} = \mathtt{Human} &\sqcap \exists \mathtt{hasComp}.(\mathtt{Leg} \sqcap \mathtt{Left}) \sqcap \exists \mathtt{hasComp}.(\mathtt{Leg} \sqcap \mathtt{Right} \sqcap \neg \mathtt{Left})\\ &\sqcap \exists \mathtt{hasPart}.(\mathtt{Leg} \sqcap \neg \mathtt{Right} \sqcap \neg \mathtt{Left}).\end{aligned}$$

In [6], it is shown that satisfiability of concepts in $\mathcal{SHQ}$ (even in its sublogic $\mathcal{SHN}$) is undecidable if non-simple roles (i.e., transitive roles or their super-roles) are used in number restrictions. In this paper, we explore this area more thoroughly with the goal of finding a more expressive but still decidable DL where we can use non-simple roles in number restrictions. Our contributions are two-fold: on the one hand, we sharpen the above undecidability result and demonstrate that: (i) for DLs such as $\mathcal{SHIN}$ (which extends $\mathcal{SHN}$ with inverse roles), counting over non-simple roles leads to undecidability even with only one role in the language; (ii) for DLs without inverses such as $\mathcal{SHN}$, two transitive and a third role are sufficient for undecidability. On the other hand, we demonstrate that, in the absence of inverse roles, counting over non-simple roles does not compromise decidability provided that they satisfy certain other restrictions regarding role inclusion axioms. Roughly speaking, as long as any two transitive roles are either completely unrelated w.r.t. inclusion or one of them implies the other, we can use them in number restrictions without losing decidability. We believe that the latter result will turn out to be useful in practice since it allows, for example, to capture a transitive role `hasPart` alongside other, possibly transitive roles such as `hasComp` or `hasSegment` *and* to use them all in number restrictions—as long as any two of these transitive roles are related by a (bi)-implication.

---

[2] See `http://www.cs.man.ac.uk/~sattler/reasoners.html` for a list.

2 Preliminaries and Known Results

The vocabulary of a DL consists of disjoint infinite sets of *concept names* CN, *role names* RN, and *individual names* IN. A *role* is an expression of the form r or r^-, where r is a role name. For convenience, we introduce a syntactic operator defined on roles: $\mathsf{Inv}(R) := r^-$, if R is a role name r; and $\mathsf{Inv}(R) := r$, if $R = r^-$ for some role name r. Finally, we use $\mathsf{Card}(M)$ for the cardinality of a set M.

Definition 1 (RBox). An *RBox* $\mathcal{R}$ is a finite collection of *transitivity axioms* of the form $\mathsf{Tr}(R)$ and *role inclusion axioms* of the form $R \sqsubseteq S$, where R, S are roles.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, its *domain*, and an interpretation function $\cdot^{\mathcal{I}}$ that maps each role name $r \in \mathsf{RN}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$; $\mathcal{I}$ is *finite* if the domain of $\mathcal{I}$ is finite. We define $(r^-)^{\mathcal{I}} := \{\langle x, y\rangle \mid \langle y, x\rangle \in r^{\mathcal{I}}\}$. We define whether $\mathcal{I}$ *satisfies* an axiom α, written $\mathcal{I} \models \alpha$ as follows: $\mathcal{I} \models \mathsf{Tr}(R)$ iff $R^{\mathcal{I}}$ is transitive, and $\mathcal{I} \models R \sqsubseteq S$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$. An interpretation satisfying all axioms in $\mathcal{R}$ is called a *model* of $\mathcal{R}$. An RBox $\mathcal{R}$ *entails* an axiom α, written $\mathcal{R} \models \alpha$, if all models of $\mathcal{R}$ satisfy α.

The deductive *closure* $[\mathcal{R}]$ of $\mathcal{R}$ is the minimal set that contains $\mathcal{R}$ and axioms $R \sqsubseteq R$, for all roles R in $\mathcal{R}$, and that is closed under the following rules:

$$\frac{R \sqsubseteq S \quad S \sqsubseteq T}{R \sqsubseteq T} \qquad \frac{R \sqsubseteq S}{\mathsf{Inv}(R) \sqsubseteq \mathsf{Inv}(S)} \qquad \frac{T \sqsubseteq S \sqsubseteq T \quad \mathsf{Tr}(T)}{\mathsf{Tr}(S)} \qquad \frac{\mathsf{Tr}(T)}{\mathsf{Tr}(\mathsf{Inv}(T))}$$

We write $\mathcal{R} \vdash \alpha$ as an alternative notation for $\alpha \in [\mathcal{R}]$, where α is an RBox axiom.

Definition 2. The set of concepts in DL $\mathcal{ALCIQ}$ is defined by the grammar:

$$\mathbf{C} ::= \ \bot \mid A \mid \neg C \mid C \sqcap D \mid \exists R.C \mid \leqslant n\, S.C,$$

where $A \in \mathsf{CN}$, $C, D \in \mathbf{C}$, R and S are roles, and n is a non-negative integer.

The interpretation function $\cdot^{\mathcal{I}}$ maps, additionally, each concept name $C \in \mathsf{CN}$ to a subset $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and $\cdot^{\mathcal{I}}$ is extended to complex concepts inductively as follows:

$$\begin{aligned} \bot^{\mathcal{I}} = \varnothing,\ \ (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}},\ \ (C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\ (\exists R.C)^{\mathcal{I}} = \{\, e \in \Delta^{\mathcal{I}} \mid \text{ there exists } d \in C^{\mathcal{I}} \text{ such that } \langle e, d\rangle \in R^{\mathcal{I}}\}, \\ (\leqslant n\, S.C)^{\mathcal{I}} = \{\, e \in \Delta^{\mathcal{I}} \mid \mathsf{Card}(\{d \in C^{\mathcal{I}} \mid \langle e, d\rangle \in S^{\mathcal{I}}\}) \leqslant n\}. \end{aligned}$$

For C and D $\mathcal{ALCIQ}$ concepts, $C \sqsubseteq D$ is a *general concept inclusion* (GCI), and a finite set of GCIs is called a *TBox*. An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. An interpretation is a *model* of a TBox if it satisfies all its axioms. If a interpretation $\mathcal{I}$ is a model of an RBox $\mathcal{R}$ and a TBox $\mathcal{T}$, then we say that $\mathcal{I}$ is a *model* of $\langle \mathcal{R}, \mathcal{T}\rangle$, or $\langle \mathcal{R}, \mathcal{T}\rangle$ is *satisfiable*. A concept C is *satisfiable w.r.t.* $\langle \mathcal{R}, \mathcal{T}\rangle$ if there exists a model $\mathcal{I}$ of $\langle \mathcal{R}, \mathcal{T}\rangle$ such that $C^{\mathcal{I}} \neq \varnothing$.

As usual, the concept expressions $\top$, $C_1 \sqcup C_2$, $\forall R.C$ and $\geqslant n\, S.C$ are assumed to be abbreviations for $\neg\bot$, $\neg(\neg C_1 \sqcap \neg C_2)$, $\neg(\exists R.\neg C)$ and $\neg(\leqslant (n-1)\, S.\neg C)$ respectively. Concepts of $\mathcal{ALCIQ}$ that do not use number restrictions $(\leqslant n\, R.C)$, or inverse roles, or both, will be called $\mathcal{ALCI}$-, $\mathcal{ALCQ}$-, and $\mathcal{ALC}$ concepts, resp. The letter $\mathcal{N}$ in the name of a DL indicates that this DL supports only number restrictions of the form $(\leqslant n\, R.\top)$.

Please note that, so far, we have introduced RBoxes and $\mathcal{ALCIQ}$ TBoxes separately, i.e., we did not put them into a single logic, which is slightly unusual. Recall that in [6] a role S is called *simple w.r.t.* $\mathcal{R}$ if there is no transitive subrole of S in $\mathcal{R}$. Traditionally, the DL that allows for

- an RBox without inverse roles and an $\mathcal{ALCQ}$ TBox where all roles in number restrictions are simple is called $\mathcal{SHQ}$, and
- an RBox and an $\mathcal{ALCIQ}$ TBox where all roles in number restrictions are simple is called $\mathcal{SHIQ}$.

For $\mathcal{SHIQ}$ and related DLs, roles in number restrictions are restricted to simple ones to ensure decidability of concept satisfiability w.r.t. a TBox and an RBox: in $\mathcal{SHN}$ (and hence $\mathcal{SHIQ}$), non-simple roles in number restrictions lead to the undecidability of the satisfiability problem [6]. Our aim is to find conditions under which we can relax or even get rid of this restriction to simple roles in number restrictions while preserving decidability. This aim can be achieved by extending the notion of a simple role in such a way that it covers, besides roles that are usually called simple, also some transitive roles or their super-roles. In this paper, we focus on a sub-problem, namely, we are looking for conditions on an RBox under which one can use all its roles in number restrictions and still have a decidable logic. Therefore, we introduce the following notion.

Definition 3. Let $\mathcal{L}$ be a logic between $\mathcal{ALC}$ and $\mathcal{ALCIQ}$ and $\mathcal{R}$ an RBox. The problem of $\mathcal{L}(\mathcal{R})$-satisfiability is to determine, given an $\mathcal{L}$-concept C and an $\mathcal{L}$-TBox $\mathcal{T}$, whether C is satisfiable w.r.t. $\langle \mathcal{R}, \mathcal{T} \rangle$. We say that an RBox $\mathcal{R}$ is $\mathcal{L}$-*safe* (or *safe for* $\mathcal{L}$) if $\mathcal{L}(\mathcal{R})$-satisfiability is decidable, and $\mathcal{L}$-*unsafe* otherwise.

Any RBox is $\mathcal{ALCI}$-safe because (i) neither $\mathcal{ALCI}$ nor $\mathcal{SHI}$ support number restrictions, and (ii) since a concept C and a TBox $\mathcal{T}$ are $\mathcal{ALCI}(\mathcal{R})$-satisfiable iff C is satisfiable w.r.t. $\langle \mathcal{R}, \mathcal{T} \rangle$, we have that $\mathcal{ALCI}(\mathcal{R})$ satisfiability can be viewed as the standard $\mathcal{SHI}$ satisfiability problem which is known to be decidable [6]. With a similar argument, any RBox $\mathcal{R}$ without transitivity axioms is $\mathcal{ALCIQ}$-safe because (i) all roles are simple in this case, and (ii) $\mathcal{ALCIQ}(\mathcal{R})$-satisfiability can be viewed as the standard $\mathcal{SHIQ}$ satisfiability problem which is known to be decidable [6]. There are numerous other restrictions on the syntax that could possibly lead to decidability, for example to use only number restrictions of the form $(\leqslant 1\, R)$.

At the same time, we know from [6] that the following RBox Star_4 (with eight roles, of which four are transitive) is $\mathcal{ALCN}$-unsafe:

$$\mathsf{Star}_4 = \{\, s_i \sqsubseteq t_{ij},\ r_j \sqsubseteq t_{ij},\ \mathsf{Tr}(t_{ij}) \ \mid\ 0 \leqslant i, j \leqslant 1 \,\}.$$

In what follows, we show that

- there is a large class of RBoxes involving role inclusions and transitivity axioms that are $\mathcal{ALCQ}$-safe (Theorem 4),
- there exists an $\mathcal{ALCIN}$-unsafe RBox with only one transitive role (Theorem 2),
- there exist $\mathcal{ALCN}$-unsafe RBoxes involving only three roles (Theorem 1).

Many proofs in this paper are rather sketchy or completely omitted; all the details however can be found in the accompanying technical report [7].

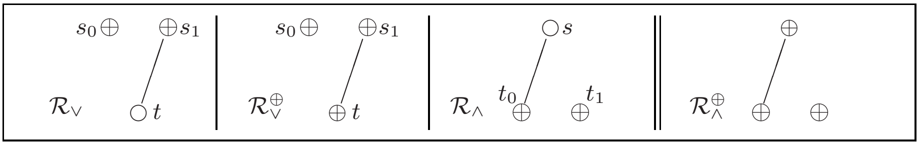


Fig. 1. The first three RBoxes are unsafe (Theorem 1) and for the last one the problem is open

3 Undecidability Results

Here we show that three roles are sufficient for building an unsafe RBox for $\mathcal{ALCQ}$, whereas for $\mathcal{ALCIQ}$, even one role is sufficient for that. In order to provide a geometric intuition for our results, we depict RBoxes (without inverse roles) as directed graphs whose nodes are non-transitive ($\bigcirc$) and transitive ($\oplus$) roles and arrows represent implications between roles. Our plan is as follows. First, in Theorem 1 we show that the RBoxes $\mathcal{R}_\vee$, $\mathcal{R}_\vee^\oplus$, and $\mathcal{R}_\wedge$ shown in Fig. 1 are $\mathcal{ALCQ}$-unsafe; we give only a sketch of the proof to illustrate the idea. Our conjecture is that the fourth RBox $\mathcal{R}_\wedge^\oplus$ in Fig. 1 is $\mathcal{ALCQ}$-safe. Next, we formulate a more general result (Theorem 2) that the RBoxes depictured in Fig. 3 are also $\mathcal{ALCQ}$-unsafe (its fully detailed proof is given in the accompanying technical report [7]). Finally, in Theorem 3 we demonstrate that the RBox $\{\mathsf{Tr}(r)\}$ is $\mathcal{ALCIQ}$-unsafe. We obtain the undecidability results by reduction from the undecidable *domino problem* (see, e.g., [3]).

Definition 4 (Domino). A *domino system* is a triple $\mathcal{D} = \langle D, H, V\rangle$, where $D = \{d_1, \dots, d_n\}$ is a finite set of *tile types* and $H, V \subseteq D \times D$ are horizontal and vertical *matching* relations. We say that $\mathcal{D}$ *tiles* $\mathbb{N}\times\mathbb{N}$ if there exists a $\mathcal{D}$*-tiling*, i.e., a mapping $\tau\colon \mathbb{N}\times\mathbb{N} \to D$ such that, for all $i, j \in \mathbb{N}$, the following *compatibility* conditions hold: $\langle \tau(i,j), \tau(i{+}1,j)\rangle \in H$ and $\langle \tau(i,j), \tau(i,j{+}1)\rangle \in V$. The *domino problem* is to check, given a domino system $\mathcal{D}$, whether $\mathcal{D}$ tiles $\mathbb{N}\times\mathbb{N}$.

Our proofs follow the usual pattern: in order to show $\mathcal{L}$-unsafety of some RBox $\mathcal{R}$, we first build an $\mathcal{L}$-TBox $\mathcal{T}_{\mathsf{grid}}$ that, together with $\mathcal{R}$ "encodes", the $\mathbb{N}\times\mathbb{N}$ grid. Then, given a domino system $\mathcal{D}$, we build (efficiently) an $\mathcal{ALC}$-TBox $\mathcal{T}_\mathcal{D}$ that "tiles" the grid and "ensures" the compatibility conditions. Finally, we prove that $\mathcal{D}$ tiles $\mathbb{N}\times\mathbb{N}$ iff some concept (usually a concept name) C is satisfiable w.r.t. $\mathcal{R} \cup \mathcal{T}_{\mathsf{grid}} \cup \mathcal{T}_\mathcal{D}$. We give fine-grained formulations of results by indicating, as a subscript to the name of a logic, the maximal number n occurring in number restrictions $(\leqslant n\, R.C)$ in the proof.

Theorem 1. *The RBoxes $\mathcal{R}_\wedge$, $\mathcal{R}_\vee$, and $\mathcal{R}_\vee^\oplus$ shown in Fig. 1 are unsafe for $\mathcal{ALCN}$; more precisely, they are unsafe for $\mathcal{ALCN}_9$ and $\mathcal{ALCQ}_1$.*

Proof (sketch for $\mathcal{R}_\wedge$). We use 16 concept names A_{ij}, $0 \leqslant i, j \leqslant 3$, place them on an $\mathbb{N}\times\mathbb{N}$ grid (by repeating a $[0,3]\times[0,3]$ pattern periodically) and link them with t_0- and t_1-edges as shown in Fig. 2a. We will refer to edges in this grid as $\langle A, r, B\rangle$, where $A, B \in \{A_{ij} \mid 0 \leqslant i, j \leqslant 3\}$ and $r \in \{t_0, t_1\}$. Having this picture in mind, we add the following axioms (a)–(c), (d) to an $\mathcal{ALCQ}$-TBox $\mathcal{T}_{\mathsf{grid}}^{\mathcal{Q}}$ and axioms (a)–(c), (d') to an $\mathcal{ALCN}$-TBox $\mathcal{T}_{\mathsf{grid}}^{\mathcal{N}}$, where $i, j, k, \ell \in \{0, 1, 2, 3\}$ and we denote $i \oplus j := (i{+}j) \bmod 4$.

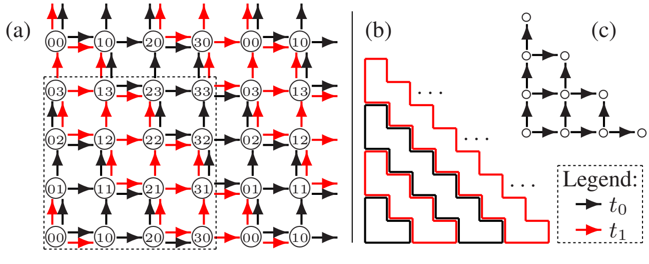


Fig. 2. A grid for Theorem 1: (a) A detailed view of the grid. (b) A grid at a glance. (c) Accessibility relation for R (and similarly for S). (d) A pre-grid for $\mathcal{ALCN}$.

(a) $A_{ij} \sqcap A_{k\ell} \sqsubseteq \bot$, for all $\langle i,j\rangle \neq \langle k,\ell\rangle$, i.e., concepts A_{ij} are pairwise disjoint;
(b) $A \sqsubseteq \exists r.B$ for each edge $\langle A,r,B\rangle$, where $r \in \{t_0,t_1\}$;
(c) $A \sqsubseteq\, \leqslant 1\, s.B$ for each double edge $\langle A,t_0,B\rangle$ and $\langle A,t_1,B\rangle$;
(d) $A_{ij} \sqsubseteq\, \leqslant 1\, s.A_{i\oplus 1,j\oplus 1}$ for all $0 \leqslant i,j \leqslant 3$;
(d') $A_{ij} \sqsubseteq\, \leqslant 9\, s$ for all $0 \leqslant i,j \leqslant 3$ such that $i{+}j$ is even.

For instance, we have axioms $A_{10} \sqsubseteq \exists t_0.A_{11}$ from (b), $A_{11} \sqsubseteq\, \leqslant 1\, s.A_{12}$ from (c), $A_{32} \sqsubseteq\, \leqslant 1\, s.A_{03}$ from (d) in $\mathcal{T}^{\mathcal{Q}}_{\mathsf{grid}}$, and $A_{13} \sqsubseteq\, \leqslant 9\, s$ from (d') in $\mathcal{T}^{\mathcal{N}}_{\mathsf{grid}}$.

Next, given a domino system $\mathcal{D} = \langle D,H,V\rangle$ with $D = \{d_1,\ldots,d_n\}$, we introduce fresh concept names $D_1,\ldots,D_n$ and add the following $\mathcal{ALC}$-axioms to a TBox $\mathcal{T}_{\mathcal{D}}$:

(e) $\top \sqsubseteq D_1 \sqcup \ldots \sqcup D_n$;
(f) $D_k \sqcap D_\ell \sqsubseteq \bot$, for all $1 \leqslant k < \ell \leqslant n$;
(g) $A \sqcap D_k \sqsubseteq \forall r.\big(B \rightarrow \bigsqcup_{\ell:\,\langle d_k,d_\ell\rangle\in H} D_\ell\big)$ for each horizontal edge $\langle A,r,B\rangle$;
(h) $A \sqcap D_k \sqsubseteq \forall r.\big(B \rightarrow \bigsqcup_{\ell:\,\langle d_k,d_\ell\rangle\in V} D_\ell\big)$ for each vertical edge $\langle A,r,B\rangle$.

Now, for $\mathcal{X} \in \{\mathcal{Q},\mathcal{N}\}$, we set $\mathcal{K}^{\mathcal{X}}_{\mathcal{D}} := \mathcal{R}_\wedge \cup \mathcal{T}^{\mathcal{X}}_{\mathsf{grid}} \cup \mathcal{T}_{\mathcal{D}}$ and prove the following lemma.

Lemma 1.1 (For $\mathcal{R}_\wedge$). *The concept A_{00} is satisfiable w.r.t. $\mathcal{K}^{\mathcal{X}}_{\mathcal{D}}$ iff $\mathcal{D}$ tiles $\mathbb{N}\times\mathbb{N}$.*

($\Leftarrow$) Given a tiling $\tau\colon \mathbb{N}\times\mathbb{N} \to D$, we build a model $\mathcal{I}$ as follows: set $\Delta^{\mathcal{I}} := \mathbb{N}\times\mathbb{N}$, interpret concepts A_{ij} exactly as in Fig. 2a; roles t_0, t_1 as the transitive closures of the relations depicted by arrows in Fig. 2a; let $s^{\mathcal{I}} := t_0^{\mathcal{I}} \cup t_1^{\mathcal{I}}$; and set $\langle i,j\rangle \in D_k^{\mathcal{I}}$ iff $\tau(i,j) = d_k$. Then $A_{00} \neq \varnothing$, as $\langle 0,0\rangle \in A_{00}^{\mathcal{I}}$.

It remains to check that $\mathcal{I} \models \mathcal{K}^{\mathcal{X}}_{\mathcal{D}}$. Clearly, $\mathcal{I} \models \mathcal{R}_\wedge$; here it is important that $\mathcal{R}_\wedge$ has no transitivity axiom for s, as the relation $s^{\mathcal{I}}$ is not transitive. To show that $\mathcal{I} \models \mathcal{T}^{\mathcal{X}}_{\mathsf{grid}}$, one needs the following observation: any element in $\mathcal{I}$ has at most 9 s-successors, which belong to 9 pairwise disjoint sets $A_{ij}^{\mathcal{I}}$. Finally, it is straightforward to show that $\mathcal{I} \models \mathcal{T}_{\mathcal{D}}$.

($\Rightarrow$) Suppose that $\mathcal{I} \models \mathcal{K}^{\mathcal{X}}_{\mathcal{D}}$ and $A_{00}^{\mathcal{I}}$ is nonempty, say $a_{00} \in A_{00}^{\mathcal{I}}$. Then we show that there exist (not necessarily distinct) elements $a_{ij} \in \Delta^{\mathcal{I}}$, for all $i,j \in \mathbb{N}$, that are linked with t_0- and t_1-edges as in Fig. 2a. After that, we "read-off" a $\mathcal{D}$-tiling of $\mathbb{N}\times\mathbb{N}$.

Since $\mathcal{I}$ satisfies axiom (b), we can find in $\mathcal{I}$ a "pre-grid" with the root a_{00}, i.e., elements a_{ij}, b_{ij}, c_{ij} linked with t_0- and t_1-edges as shown in Fig. 2d. Axioms (c) allow to "glue" double edges, i.e., entail that $b_{10} = a_{10}$ and $b_{01} = a_{01}$. Now, with the help of

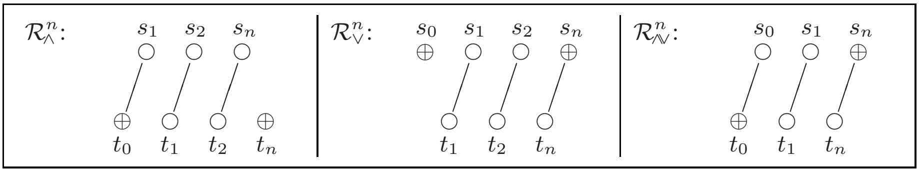


Fig. 3. The chain RBoxes $\mathcal{R}^n_\wedge$, $\mathcal{R}^n_\vee$, $\mathcal{R}^n_{\wedge\!\!\!\vee}$, for $n \geqslant 1$, are $\mathcal{ALCN}$-unsafe

axioms (d), we "glue" cells, i.e., infer that $c_{11} = a_{11}$, $c_{12} = a_{12}$, and $c_{21} = a_{21}$. Thus, using axioms from $\mathcal{T}^{\mathcal{Q}}_{\mathsf{grid}}$, we have found in $\mathcal{I}$ a structure depicted in Fig. 2c. In $\mathcal{T}^{\mathcal{N}}_{\mathsf{grid}}$, axiom (d') applied to the element a_{00}, together with the fact that all $A^{\mathcal{I}}_{ij}$ are disjoint, entail all the above equalities and hence ensure the existence of the same structure Fig. 2c in $\mathcal{I}$. After that, we repeat the same argument, starting at the root a_{ij} with $i + j = 2$, then with $j + i = 4$ and so on.

Once we have built all the elements a_{ij}, for $i, j \in \mathbb{N}$ we define $\tau\colon \mathbb{N}\times\mathbb{N} \to D$ by setting $\tau(i, j) := d_k$ iff $a_{ij} \in D^{\mathcal{I}}_k$. By (e) and (f), τ is well-defined and total; and the compatibility conditions easily follow from (g) and (h). Thus τ is indeed a $\mathcal{D}$-tiling of $\mathbb{N}\times\mathbb{N}$. This completes the proof of Lemma 1.1 and hence of Theorem 1. ⊣

In order to generalise Theorem 1, we introduce the following RBoxes, depicted in Fig. 3, where $n \geqslant 1$ (observe that $\mathcal{R}^1_\wedge$ and $\mathcal{R}^1_\vee$ correspond to $\mathcal{R}_\wedge$ and $\mathcal{R}_\vee$, resp.):

$$\begin{aligned}
\mathcal{R}^n_\wedge &:= \{\mathsf{Tr}(t_0), \mathsf{Tr}(t_n)\} \cup \{t_{k-1} \sqsubseteq s_k,\ s_k \sqsupseteq t_k \mid 1 \leqslant k \leqslant n\},\\
\mathcal{R}^n_\vee &:= \{\mathsf{Tr}(s_0), \mathsf{Tr}(s_n)\} \cup \{s_{k-1} \sqsupseteq t_k, t_k \sqsubseteq s_k \mid 1 \leqslant k \leqslant n\},\\
\mathcal{R}^n_{\wedge\!\!\!\vee} &:= \{\mathsf{Tr}(t_0), \mathsf{Tr}(s_n)\} \cup \{s_{k-1} \sqsupseteq t_k,\ t_\ell \sqsubseteq s_\ell \mid 1 \leqslant k \leqslant n,\ 0 \leqslant \ell \leqslant n\}.
\end{aligned}$$

Theorem 2. *The RBoxes $\mathcal{R}^n_\wedge$, $\mathcal{R}^n_\vee$, $\mathcal{R}^n_{\wedge\!\!\!\vee}$, with $n \geqslant 1$, are $\mathcal{ALCN}$-unsafe.*

Theorem 3. *The RBox $\mathcal{R} := \{\mathsf{Tr}(r)\}$ is unsafe for $\mathcal{ALCIN}$ (more precisely, for $\mathcal{ALCIN}_8$ and $\mathcal{ALCIQ}_1$), even for TBoxes that involve a single role name r.*

Proof. By reduction from the undecidable domino problem for $\mathbb{Z}\times\mathbb{Z}$, which is formulated analogously to Def. 4. Take 16 concept names A_{ij}, $0 \leqslant i, j \leqslant 3$. Place them on the $\mathbb{Z}\times\mathbb{Z}$ grid (by repeating a $[0, 3]\times[0, 3]$ pattern periodically) and link them with r-edges in accordance with Fig. 4a. Now, having this picture in mind (we refer to its edges as $\langle A, r, B\rangle$, where A, B are concept names), we add the following axioms (a)–(c) to an $\mathcal{ALCIQ}$-TBox $\mathcal{T}^{\mathcal{Q}}_{\mathsf{grid}}$ and axioms (a) and (d) to an $\mathcal{ALCIN}$-TBox $\mathcal{T}^{\mathcal{N}}_{\mathsf{grid}}$:

(a) All 16 concept names A_{ij}, $0 \leqslant i, j \leqslant 3$, are pairwise disjoint;
(b) $A \sqsubseteq \exists r.B$ and $B \sqsubseteq \exists r^-.A$, for each edge $\langle A, r, B\rangle$;
(c) $A_{ij} \sqsubseteq\, \leqslant 1\, r.A_{k\ell}$ and $A_{k\ell} \sqsubseteq\, \leqslant 1\, r^-.A_{ij}$, for all even i, j and odd k, ℓ;
(d) $A_{ij} \sqsubseteq\, \leqslant 8\, r$ and $A_{k\ell} \sqsubseteq\, \leqslant 8\, r^-$, for all even i, j and odd k, ℓ.

Given a domino system $\mathcal{D}$, we build a $\mathcal{ALC}$-TBox $\mathcal{T}_{\mathcal{D}}$: axioms (e) and (f) are the same as in the proof of Theorem 1, whereas (g) is the following (and (h) is analogous):

(g) $A \sqcap D_k \sqsubseteq \forall r.\big(B \to \bigsqcup_{\ell\colon \langle d_k, d_\ell\rangle \in H} D_\ell\big)$ for each right-going edge $\langle A, r, B\rangle$;
$A \sqcap D_\ell \sqsubseteq \forall r.\big(B \to \bigsqcup_{k\colon \langle d_k, d_\ell\rangle \in H} D_k\big)$ for each left-going edge $\langle A, r, B\rangle$.

Finally, for each $\mathcal{X} \in \{\mathcal{Q}, \mathcal{N}\}$, we set $\mathcal{K}^{\mathcal{X}}_{\mathcal{D}} := \mathcal{R} + \mathcal{T}^{\mathcal{X}}_{\mathsf{grid}} + \mathcal{T}_{\mathcal{D}}$. It remains to prove.

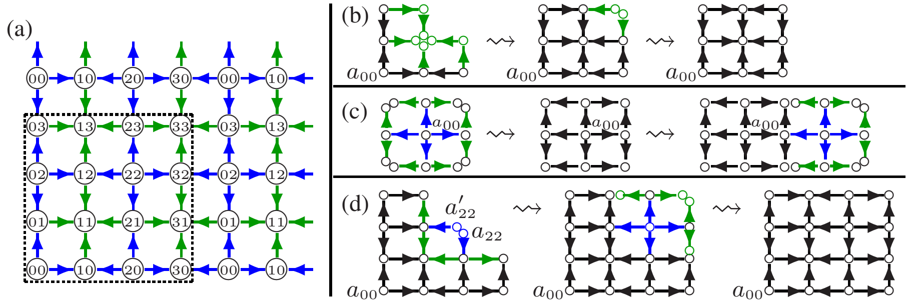


Fig. 4. (a) A grid for Theorem 3 (coloured for decoration only, as we have only 1 role). (b) A pre-grid for $\mathcal{ALCIQ}$. (c) Building a horisontal axis in $\mathcal{ALCIN}$. (d) Building new cells in $\mathcal{ALCIN}$.

Lemma 3.1. *The concept A_0 is satisfiable w.r.t. $\mathcal{K}^{\mathcal{X}}_{\mathcal{D}}$ iff $\mathcal{D}$ tiles $\mathbb{Z}\times\mathbb{Z}$.*

The '$\Leftarrow$' part is proved as in Theorem 1. To prove '$\Rightarrow$', suppose that $\mathcal{I} \models \mathcal{K}^{\mathcal{X}}_{\mathcal{D}}$ and $a_{00} \in A^{\mathcal{I}}_{00}$. Then we show that in $\mathcal{I}$ there are (not necessarily distinct) elements a_{ij}, $i, j \in \mathbb{Z}$, linked via r-edges as shown in Fig. 4a, and then build a $\mathcal{D}$-tiling of $\mathbb{Z}\times\mathbb{Z}$. The steps of building elements a_{ij} are illustrated in Fig. 4(b–d); we omit the details, which can be found in the technical report [7]. ⊣

4 Internalization of RBoxes in TBoxes Using Extended Roles

In order to study safety of RBoxes for different DLs, it is somewhat inconvenient to work separately with RBoxes and TBoxes. Therefore, in this section, we demonstrate how RBoxes can be internalized into TBoxes, provided additional role constructors—role unions and transitive closure operator—can be used. We also demonstrate that it is sufficient to focus only on TBoxes of some simple form. The results of this section can be applied to any logic $\mathcal{L}$ between $\mathcal{ALC}$ and $\mathcal{ALCIQ}$.

Definition 5. We say that an $\mathcal{L}$-TBox $\mathcal{T}$ is in a *simple form* if all axioms in $\mathcal{T}$ have the following forms, where $A_{(i)}$, $B_{(j)}$ are concept names, m, n integers, and S a role:

$$\textstyle\bigsqcap A_i \sqcap \bigsqcap \neg B_j \sqsubseteq \bot \tag{1}$$

$$A \sqsubseteq \geqslant n\, S.B \tag{2}$$

$$A \sqsubseteq \leqslant m\, S.B \tag{3}$$

Lemma 1 (Simplification of $\mathcal{L}$-TBoxes). *Given an $\mathcal{L}$-TBox $\mathcal{T}$, one can construct in polynomial time an $\mathcal{L}$-TBox $\mathcal{T}_{sf}$ in simple form such that, for every RBox $\mathcal{R}$, $\langle\mathcal{T}, \mathcal{R}\rangle$ is (finitely) satisfiable iff $\langle\mathcal{R}, \mathcal{T}_{sf}\rangle$ is (finitely) satisfiable.*

Definition 6. The set of *extended roles* $\mathbf{R}^{\sqcup,+}$ is defined by the following grammar:

$$\mathbf{R}^{\sqcup,+} ::= \; R \mid \rho_1 \sqcup \rho_2 \mid \rho^+, \quad \text{where } R \text{ is a role and } \rho_{(i)} \in \mathbf{R}^{\sqcup,+}.$$

The additional role constructors are interpreted as follows: $(\rho_1 \sqcup \rho_2)^{\mathcal{I}} = \rho_1^{\mathcal{I}} \cup \rho_2^{\mathcal{I}}$, $(\rho^+)^{\mathcal{I}} = (\rho^{\mathcal{I}})^+$, where $(\cdot) \cup (\cdot)$ and $(\cdot)^+$ are usual operators of union and transitive closure on binary relations. Concepts of $\mathcal{L}(\sqcup,+)$ are defined as for $\mathcal{L}$ except that extended roles can be used in place of roles.

Our goal is to demonstrate that every RBox can be internalized in a simple $\mathcal{L}$-TBox producing an $\mathcal{L}(\sqcup,+)$-TBox of a certain simple form:

Definition 7 (Simple $\mathcal{L}(\sqcup,+)$-TBox). We say that an $\mathcal{L}(\sqcup,+)$-TBox $\mathcal{T}$ is *simple* if every axiom from $\mathcal{T}$ is either of the form (1), (2), or:

$$A \sqsubseteq \leqslant m\,(\textstyle\bigsqcup u_i^+ \sqcup v).B \tag{4}$$

where $A_{(i)}$, $B_{(j)}$ are concept names, m, n integers, and u_i and v are disjunctions of roles: $u_i, v = \bigsqcup R_i$. For a simple TBox $\mathcal{T}$, we denote by $K(\mathcal{T})$ the number of axioms of type (4) in $\mathcal{T}$, by $N(\mathcal{T})$ and $M(\mathcal{T})$ the sum of all numbers n, resp. m, over all axioms of type (2), resp. (4), by $C(\mathcal{T})$ the number of concept names in $\mathcal{T}$.

Definition 8 ($\mathcal{R}$-extension). Given an RBox $\mathcal{R}$, an *extension* of a role S in $\mathcal{R}$ (or the *$\mathcal{R}$-extension* of S, for short) is an extended role $\mathcal{R}(S) \in \mathbf{R}^{\sqcup,+}$ defined as follows:

- If S is transitive in $\mathcal{R}$ then $\mathcal{R}(S) := (\bigsqcup S_i)^+$, where $\{S_i\}$ is the set of all subroles of S in $\mathcal{R}$ (including S itself);
- If S is not transitive, then $\mathcal{R}(S) := \bigsqcup \mathcal{R}(T_i) \sqcup \bigsqcup S_j$, where $\{T_i\}$ is exactly the set of all maximal transitive subroles of S, and $\{S_j\}$ is the set of all subroles of S.

Definition 9 (Internalization of an RBox in an $\mathcal{L}$-TBox).
Let $\mathcal{R}$ be an RBox and $\mathcal{T}$ be a simple $\mathcal{L}$-TBox. The internalization of $\mathcal{R}$ in $\mathcal{T}$ is a simple $\mathcal{L}(\sqcup,+)$-TBox $\mathcal{R}(\mathcal{T}) := \{\mathcal{R}(\alpha) \mid \alpha \in \mathcal{T}\}$, where:

- $\mathcal{R}(\alpha) := \alpha$ if α is of the form (1) or (2), and
- $\mathcal{R}(\alpha) := A \sqsubseteq \leqslant m\,(\mathcal{R}(S)).B$ if $\alpha = A \sqsubseteq \leqslant m\,S.B$ is of the form (3).

Lemma 2. *Let $\mathcal{R}$ be an RBox and $\mathcal{T}$ a simple $\mathcal{L}$-TBox. Then $\langle\mathcal{R},\mathcal{T}\rangle$ is satisfiable iff $\mathcal{R}(\mathcal{T})$ is satisfiable.*

5 Decidability Results

As we have demonstrated in Theorem 3, an RBox consisting of just one transitivity axiom is already unsafe for $\mathcal{ALCIN}$. Hence, there is a little room left for non-trivial safe RBoxes for $\mathcal{ALCIN}$. In contrast, the undecidability results in Section 3 for $\mathcal{ALCN}$ require a certain interaction between several transitive roles. This poses a question about safety of those RBoxes that do not fit such a pattern. In this section, we investigate this question and define a relatively large class of so-called admissible RBoxes that, as we will prove, are safe for $\mathcal{ALCQ}$. Since we focus on $\mathcal{ALCQ}$, within this section we assume that there are no inverse roles in RBoxes.

Definition 10. For a TBox $\mathcal{T}$, RBox $\mathcal{R}$, or an axiom α, let $\mathsf{RN}(\mathcal{T})$, $\mathsf{RN}(\mathcal{T})$, $\mathsf{RN}(\alpha)$ denote the set of role names that occur in $\mathcal{T}$, $\mathcal{R}$, α, respectively.

An RBox $\mathcal{R}$ is *strongly admissible* if, for every two transitive roles $T_1, T_2 \in \mathsf{RN}(\mathcal{R})$, we have $\mathcal{R} \vdash T_1 \sqsubseteq T_2$ or $\mathcal{R} \vdash T_2 \sqsubseteq T_1$. An RBox $\mathcal{R}$ is *admissible* if $\mathcal{R} = \bigcup \mathcal{R}_i$ where (1) each $\mathcal{R}_i$ is strongly admissible and (2) $\mathsf{RN}(\mathcal{R}_i) \cap \mathsf{RN}(\mathcal{R}_j) = \varnothing$ for all $i \neq j$.

In the remainder of this section, we prove the following Theorem:

Theorem 4. *Every admissible RBox is $\mathcal{ALCQ}$-safe.*

Note 1. For $\mathcal{R} = \{\mathsf{Tr}(r)\}$, this result corresponds to the decidability of the graded variant of the modal logic $\mathbf{K4}$ (called $\mathbf{GrK4}$), which has already been addressed in [4]. However, we found that the proof in that paper is incorrect (see [7] for details). Therefore, here we re-establish decidability of $\mathbf{GrK4}$ as a special case of Theorem 4.

First of all, we demonstrate that, for the purpose of proving safety, it is sufficient to focus only on strongly admissible RBoxes.

Lemma 3 (Modularity). *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be RBoxes with $\mathsf{RN}(\mathcal{R}_1) \cap \mathsf{RN}(\mathcal{R}_2) = \varnothing$ and $\mathcal{L}$ is between $\mathcal{ALC}$ and $\mathcal{ALCIQ}$. Then $\mathcal{R}_1 \cup \mathcal{R}_2$ is $\mathcal{L}$-safe iff $\mathcal{R}_1$ and $\mathcal{R}_2$ are $\mathcal{L}$-safe.*

Proof. The '⇒' part of the lemma is obvious. The '⇐' part of the lemma follows from the results about fusions of DLs from [2]. See [7] for details. ⊣

Corollary 1. *Let $\mathcal{L}$ be a logic between $\mathcal{ALC}$ and $\mathcal{ALCIQ}$. Then every admissible RBox is $\mathcal{L}$-safe provided every strongly admissible RBox is $\mathcal{L}$-safe.*

In order to prove that every strongly admissible RBox $\mathcal{R}$ is safe, it is sufficient to show that the problem of satisfiability of a pair $\langle \mathcal{R}, \mathcal{T} \rangle$, with $\mathcal{T}$ an $\mathcal{L}$-TBox, is decidable. Indeed, a concept C is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff the pair $\langle \mathcal{R}, \mathcal{T} \cup \{\top \sqsubseteq \exists R.C\} \rangle$ is satisfiable, where R is a fresh role. To this end, we first simplify the TBox $\mathcal{T}$ using Proposition 1 and then internalize RBox $\mathcal{R}$ using Definition 9, which will result in some $\mathcal{L}(\sqcup, +)$-TBox of a restricted form, which we call admissible. We then demonstrate that satisfiability of admissible $\mathcal{L}(\sqcup, +)$-TBoxes is decidable.

In what follows, for convenience, we often identify an extended role $u = \bigsqcup R_i$ with the set $\bigcup\{R_i\}$. Using this convention, we can write $r \in u$ or $u \subseteq u'$ for disjunction of roles u and u', as well as $u^{\mathcal{I}}$ for sets of roles u.

Definition 11. A simple $\mathcal{L}(\sqcup, +)$ TBox $\mathcal{T}$ is *admissible* if (i) all axioms of form (4) are of the forms (5) and (6) below, and (ii) for every two axioms $A_1 \sqsubseteq \leqslant m_1\, (u_1^+ \sqcup v_1).B_1$ and $A_2 \sqsubseteq \leqslant m_2\, (u_2^+ \sqcup v_2).B_2$ of form (6), we have that either $u_1 \subseteq u_2$, or $u_2 \subseteq u_1$.

$$A \sqsubseteq \leqslant m\, (v).B \tag{5}$$

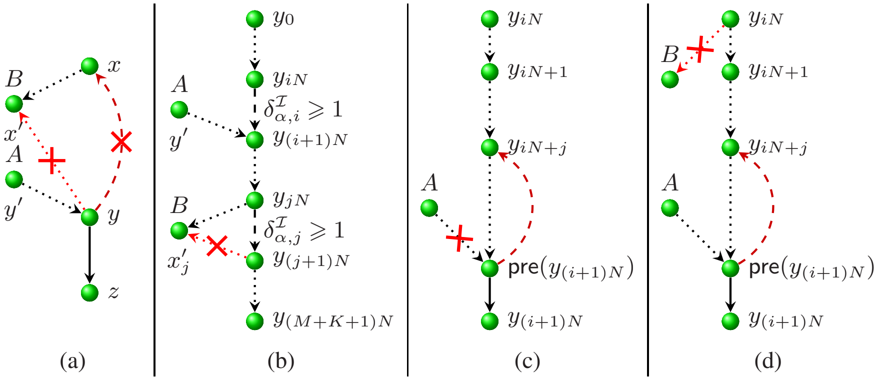
$$A \sqsubseteq \leqslant m\, (u^+ \sqcup v).B \tag{6}$$

In other words, a simple $\mathcal{L}(\sqcup, +)$-TBox is admissible if in every axiom of form (4) there is at most one occurrence of a transitively closed disjunction of roles.

Lemma 4. *Let $\mathcal{T}$ be a simple $\mathcal{L}$-TBox and $\mathcal{R}$ a strongly admissible RBox. Then $\mathcal{R}(\mathcal{T})$ is a simple admissible $\mathcal{L}(\sqcup, +)$-TBox.*

The condition (ii) from Definition 11 can be alternatively formulated as follows:

Proposition 1. *Let $\mathcal{T}$ be a simple admissible $\mathcal{L}(\sqcup, +)$-TBox. Then all roles in $\mathcal{T}$ can be ordered as $r_1, \ldots, r_n$ in such a way that for every axiom $A \sqsubseteq \leqslant m\, (u^+ \sqcup v).B$ of type* (6) *and every $1 \leqslant i \leqslant j \leqslant n$, we have that $r_j \in u$ implies $r_i \in u$.*



(a) (b) (c) (d)

Fig. 5. Looping long chains in a model back

We prove that satisfiability of simple admissible $\mathcal{L}(\sqcup, +)$-TBoxes is decidable by demonstrating the finite model property (FMP) for such TBoxes. The key property that will guarantee FMP is that, in every model of a simple admissible TBox, it is possible to "loop back" every sufficiently long chain of elements connected via roles. This idea is reminiscent to blocking conditions in tableau decision procedures for modal and description logics [6]. The next lemma states that every model of a simple $\mathcal{L}(\sqcup, +)$ TBox can be reduced to a model with bounded branching degree by removing edges that are not "required" by axioms of type (2).

Definition 12. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an $\mathcal{L}$-interpretation. A *branching degree* of an element $x \in \Delta^{\mathcal{I}}$ in $\mathcal{I}$ is $\mathsf{deg}(\mathcal{I}, x) = \mathsf{Card}\{y \mid \langle x, y\rangle \in r^{\mathcal{I}} \text{ for some } r\}$. A branching degree of $\mathcal{I}$ is $\mathsf{deg}(\mathcal{I}) = \max\{\mathsf{deg}(\mathcal{I}, x) \mid x \in \Delta^{\mathcal{I}}\}$.

Lemma 5. *Any satisfiable simple $\mathcal{L}(\sqcup, +)$-TBox $\mathcal{T}$ has a model $\mathcal{I}$ with* $\mathsf{deg}(\mathcal{I}) \leqslant N(\mathcal{T})$.

Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. For each axiom α of type (6) in $\mathcal{T}$, we introduce a function $\delta^{\mathcal{I}}_{\alpha}(x, y)$ defined on elements of $\Delta^{\mathcal{I}}$ as follows:

$$\delta^{\mathcal{I}}_{\alpha}(x,y) = \begin{cases} \mathsf{Card}\{x' \mid x' \in B^{\mathcal{I}},\ \langle x, x'\rangle \in (u^{+})^{\mathcal{I}},\ \langle y, x'\rangle \notin (u^{+})^{\mathcal{I}}\} & \\ & \text{if there exists } y' \in A^{\mathcal{I}} \text{ with } \langle y', y\rangle \in (u^{+})^{\mathcal{I}} \\ 0 & \text{otherwise} \end{cases}$$

In other words, if y has a u^{+} predecessor in which A holds, $\delta^{\mathcal{I}}_{\alpha}(x, y)$ equals to the number of elements in which B holds and that are reachable via u^{+} from x but not from y (see Fig. 5a). The value of $\delta^{\mathcal{I}}_{\alpha}(x, y)$ intuitively indicates the number of new u^{+} successors of y that might appear and potentially violate the axiom α (at the points, where A holds), if x becomes reachable from y via u^{+}.

Definition 13. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation. For an element $x \in \Delta^{\mathcal{I}}$, let $\mathsf{CN}^{\mathcal{I}}(x) := \{A \in \mathsf{CN} \mid x \in A^{\mathcal{I}}\}$ denote the set of concept names that hold at x in $\mathcal{I}$.

Given a simple admissible $\mathcal{L}(\sqcup, +)$-TBox $\mathcal{T}$, an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and $x, y, z \in \Delta^{\mathcal{I}}$, we say that x *can foster* z *for* y *in* $\mathcal{I}$ *(w.r.t.* $\mathcal{T}$*)* if (i) $\mathsf{CN}^{\mathcal{I}}(z) = \mathsf{CN}^{\mathcal{I}}(x)$,

(ii) $\langle y, x\rangle \in r^{\mathcal{I}}$ for no atomic role r, and (iii) for every axiom α of type (6) in $\mathcal{T}$, if $\langle y, z\rangle \in r^{\mathcal{I}}$ for some role $r \in u_\alpha = u$, then $\delta^{\mathcal{I}}_\alpha(x, y) = 0$.

Lemma 6 (Model Transformation). *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a model of a simple admissible $\mathcal{L}(\sqcup, +)$-TBox $\mathcal{T}$ and x, y, z elements of $\Delta^{\mathcal{I}}$ such that x can foster z for y in $\mathcal{I}$ w.r.t. $\mathcal{T}$. Let $\mathcal{J} = \mathsf{swap}(\mathcal{I}, x, y, z)$ be obtained from $\mathcal{I}$ by setting $A^{\mathcal{J}} := A^{\mathcal{I}}$ and, $r^{\mathcal{J}} := r^{\mathcal{I}} \setminus \{\langle y, z\rangle\} \cup \{\langle y, x\rangle\}$ if $\langle y, z\rangle \in r^{\mathcal{I}}$, and $r^{\mathcal{J}} := r^{\mathcal{I}}$ otherwise, for every concept name A and role name r. Then $\mathcal{J}$ is a model of $\mathcal{T}$.*

Our main lemma states that, in every model of simple admissible $\mathcal{L}(\sqcup, +)$-TBox $\mathcal{T}$, every sufficiently long chain $x_0, \ldots, x_p$ of elements connected with roles contains two elements x_i and x_j with $i < j$ such that x_i can foster x_j for the predecessor x_{j-1} of x_j w.r.t. $\mathcal{T}$. Thus, every sufficiently long chain can be "looped back" using the transformation described in Lemma 6.

Lemma 7 (Main Lemma). *Let $\mathcal{T}$ be a simple admissible $\mathcal{L}(\sqcup, +)$-TBox and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ a model for $\mathcal{T}$ with $\mathsf{deg}(\mathcal{I}) \leqslant N$. Let $r_1, \ldots, r_n$ be all the role names in $\mathcal{T}$ enumerated according to Proposition 1, k an integer with $1 \leqslant k \leqslant n$, and $x_0, \ldots, x_p$ a sequence of distinct elements in $\Delta^{\mathcal{I}}$ such that, for every $i \geqslant 1$, there exists $\ell \leqslant k$ such that $\langle x_{i-1}, x_i\rangle \in r^{\mathcal{I}}_\ell$. Then there exist i and j with $1 \leqslant i < j \leqslant p$ such that x_i can foster x_j for x_{j-1}, provided that $p \geqslant p_k := ((M + K + 1)N \cdot 2^C + 1)^k$, where $M = M(\mathcal{T})$, $K = K(\mathcal{T})$, and $C = C(\mathcal{T})$ as defined in Definition 7.*

Before proving Lemma 7, we demonstrate the following auxiliary property. For convenience, if x is an element of the sequence $x_0, \ldots, x_p$, i.e., $x = x_i$ for some i, then its predecessor *in this sequence* will be denoted by $\mathsf{pre}(x) := x_{i-1}$.

Lemma 8 (Auxiliary Lemma). *Let a TBox $\mathcal{T}$, a model $\mathcal{I}$, and a sequence $x_0, \ldots, x_p$ be as in Lemma 7. Let $y_0, \ldots, y_q$ be a sub-sequence in $x_1, \ldots, x_p$ such that (i) $q \geqslant (M + K + 1)N$, (ii) $\mathsf{CN}^{\mathcal{I}}(y_0) = \cdots = \mathsf{CN}^{\mathcal{I}}(y_q)$, (iii) $\langle \mathsf{pre}(y_i), y_i\rangle \notin r^{\mathcal{I}}_\ell$ for all $0 \leqslant i \leqslant q$ and $\ell < k$. Then for some $0 \leqslant i < j \leqslant q$, y_i can foster y_j for $\mathsf{pre}(y_j)$.*

Proof. Let U_k be the set of axioms $\alpha = (A \sqsubseteq \leqslant m\,(u^+ \sqcup v).B) \in \mathcal{T}$ of type (6) such that $r_k \in u$. Take any axiom $\alpha \in U_k$ and consider a sequence of values $\delta^{\mathcal{I}}_{\alpha,i} := \delta^{\mathcal{I}}_\alpha(y_{iN}, y_{(i+1)N})$ for $0 \leqslant i \leqslant M + K$ (see Fig. 5b). We claim that at most $m + 1$ of values $\delta^{\mathcal{I}}_{\alpha,i}$ are positive.

Indeed, for the first i with $\delta^{\mathcal{I}}_{\alpha,i} \geqslant 1$, by definition of $\delta^{\mathcal{I}}_\alpha(x, y)$, there exists $y' \in A^{\mathcal{I}}$ with $\langle y', y_{(i+1)N}\rangle \in (u^+)^{\mathcal{I}}$. For all subsequent $j > i$ with $d^j_\alpha \geqslant 1$, there exists an element x'_j such that $\langle y_{jN}, x'_j\rangle \in (u^+)^{\mathcal{I}}$, but $\langle y_{(j+1)N}, x'_j\rangle \notin (u^+)^{\mathcal{I}}$. In particular, all such x_j are distinct for different j. Note that since $r_k \in u$, by Proposition 1, $r_\ell \in u$ for all $\ell \leqslant k$. Hence $\langle y', x'_j\rangle \in (u^+)^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of α, the number of such different j can be at most m.

Hence, the number of different i such that, for some $\alpha \in U_k$, $\delta^{\mathcal{I}}_{\alpha,i} \geqslant 1$, is at most $\sum_{\alpha \in U_k} (m_\alpha + 1) \leqslant M + K$. Since $q \geqslant (M + K + 1)N$, there exists at least one i such that $\delta^{\mathcal{I}}_{\alpha,i} = 0$ for all $\alpha \in U_k$. For every $\alpha \in U_k$, there are two cases possible: either (1) there exists no $y' \in A^{\mathcal{I}}$ such that $\langle y', y_{(i+1)N}\rangle \in (u^+)^{\mathcal{I}}$ (see Fig. 5c), or (2) such a

y' exists, but there exists no $x' \in B^{\mathcal{I}}$ with $\langle y_{iN}, x'\rangle \in (u^+)^{\mathcal{I}}$ (see Fig. 5d). Hence, in particular, $\delta^{\mathcal{I}}_{\alpha}(y_{iN+j}, \mathsf{pre}(y_{(i+1)N})) = 0$ for all $j < N$ and all $\alpha \in U_k$.

Since $\mathsf{deg}(\mathcal{I}) \leqslant N$ and $\langle \mathsf{pre}(y_{(i+1)N}), y_{(i+1)N}\rangle \in r^{\mathcal{I}}_{\ell}$ for $\ell \leqslant k$, there exists $j < N$ such that $\langle \mathsf{pre}(y_{(i+1)N}), y_{iN+j}\rangle \notin r^{\mathcal{I}}$, for every r. Since, by condition (iii), we have $\langle \mathsf{pre}(y_{(i+1)N}), y_{(i+1)N}\rangle \notin r^{\mathcal{I}}_{l}$ for each $\ell < k$, we have $\delta^{\mathcal{I}}_{\alpha}(y_{iN+j}, \mathsf{pre}(y_{(i+1)N})) = 0$ for each axiom α of type (6) such that $\langle \mathsf{pre}(y_{(i+1)N}), y_{(i+1)N}\rangle \in r^{\mathcal{I}}_{\ell}$ and $r_\ell \in u$. Indeed, those are exactly $\alpha \in U_k$, because $r_\ell \in u$ implies $r_k \in u$ for every $\ell \geqslant k$ by Proposition 1. Hence, by Definition 13, y_{iN+j} can foster $y_{(i+1)N}$ for $\mathsf{pre}(y_{(i+1)N})$. ⊣

Proof (of Lemma 7). We prove the lemma by induction on k, using Lemma 8 both in induction base and induction step. Denote $L := (M + K + 1)N$ for short.

Induction base: For $k = 1$, we have a sequence of elements $x_0, \ldots, x_p \in \Delta^{\mathcal{I}}$ with $p \geqslant p_1 := L \cdot 2^C + 1$ such that $\langle x_{i-1}, x_i\rangle \in r^{\mathcal{I}}_1$, for all $1 \leqslant i \leqslant p$. We claim that there exists a subsequence $y_0, \ldots, y_q$ in $x_1, \ldots, x_p$ with $q \geqslant L$ such that $\mathsf{CN}^{\mathcal{I}}(y_0) = \cdots = \mathsf{CN}^{\mathcal{I}}(y_q)$. Indeed, otherwise, since the number of different values of $\mathsf{CN}^{\mathcal{I}}(x)$ is bounded by 2^C, and the number of elements x in $x_1, \ldots, x_p$ with the same value of $\mathsf{CN}^{\mathcal{I}}(x)$ is at most L, the total number of elements in $x_1, \ldots, x_p$ cannot exceed $L \cdot 2^C$, which contradicts to $p \geqslant p_1$. Now, Lemma 8 can be applied to the sequence $y_0, \ldots, y_q$, since there are no roles r_ℓ with $\ell < k = 1$. By Lemma 8 there exist elements y_i and y_j in this sequence with $0 \leqslant i < j \leqslant q$, such that y_i can foster y_j for $\mathsf{pre}(y_j)$.

Induction Step: Assume that the lemma holds for $k - 1$. Two cases are possible:

(A) There exists a sub-sequence of consecutive elements $x_i, x_{i+1}, \ldots, x_{i+p_{k-1}}$ with $p_{k-1} = (L \cdot 2^C + 1)^{k-1}$ and for each j with $1 \leqslant j \leqslant p_{k-1}$, there exists $\ell \leqslant k - 1$ such that $\langle x_{i+j-1}, x_{i+j}\rangle \in r^{\mathcal{I}}_{\ell}$. In this case the lemma holds by the induction hypothesis.

(B) Otherwise, in every sequence $x_{ip_{k-1}}, x_{ip_{k-1}+1} \ldots, x_{(i+1)p_{k-1}}$ of consecutive elements with $0 \leqslant i \leqslant p_1 - 1 = L \cdot 2^C$, there exists an element $x'_i = x_{ip_{k-1}+j}$, with $1 \leqslant j \leqslant p_{k-1}$, such that $\langle \mathsf{pre}(x'_i), x'_i\rangle \notin r^{\mathcal{I}}_{\ell}$ for all $\ell \leqslant k - 1$. By applying a combinatorial argument as in the induction base, from the sequence $x'_0, \ldots, x'_{p_1-1}$ of $p_1 = L \cdot 2^C$ distinct elements one can select a subsequence $y_0, \ldots, y_q$ with $q \geqslant L$ such that $\mathsf{CN}^{\mathcal{I}}(y_0) = \cdots = \mathsf{CN}^{\mathcal{I}}(y_q)$. Hence the claim of the lemma follows from Lemma 8 applied to the sequence $y_0, \ldots, y_q$. ⊣

Theorem 5. *An admissible $\mathcal{ALCQ}(\sqcup, +)$-TBox $\mathcal{T}$ is satisfiable iff $\mathcal{T}$ has a finite model.*

Proof. The "if" direction is trivial. To prove the "only if" part, we use the following argument: given any model $\mathcal{I}$ of $\mathcal{T}$, we build a finite model $\mathcal{J}$ of $\mathcal{T}$ by "looping back" all sufficiently long paths from some element x_0 using Lemma 6 and Lemma 7 until none of them left, and then removing elements that became disconnected from x_0 after this transformation. For a detailed proof, see the technical report [7]. ⊣

Now it is time to harvest our decidability results (see [7] for all proofs).

Theorem 6. *Let $\mathcal{T}$ be an $\mathcal{ALCQ}$-TBox and $\mathcal{R}$ a strongly admissible RBox. Then $\langle \mathcal{R}, \mathcal{T}\rangle$ is satisfiable iff it has a finite model.*

Corollary 2 (Theorem 4). *Every admissible RBox is safe for $\mathcal{ALCQ}$.*

Corollary 3. *Every satisfiable* **GrK4***-formula has a finite model.*

6 Extending RBoxes

In Section 5 we have described a rather large class of $\mathcal{ALCQ}$-safe RBoxes. However, so far, only few RBoxes were shown to be unsafe for $\mathcal{ALCN}$ and $\mathcal{ALCIN}$ in Section 3. In this section we are concerned with a question whether every RBox "containing" any of the patterns described in Section 3 is necessarily unsafe? Or, in general, what happens to the (un)safety of an RBox when the RBox are extended?

It is clear that adding axioms may turn a safe RBox into unsafe and vice versa: an $\mathcal{ALCN}$-safe RBox $\{\mathsf{Tr}(r)\}$ can be extended to an $\mathcal{ALCN}$-unsafe RBox $\mathcal{R}_\wedge$ from Theorem 1; adding to $\mathcal{R}_\wedge$ an inclusion between its incomparable transitive roles yields an $\mathcal{ALCN}$-safe RBox by Theorem 4. So it is not sufficient for an RBox $\mathcal{R}'$ to be unsafe if it contains some unsafe RBox $\mathcal{R}$. The question now is: what additional property an extension $\mathcal{R}'$ of $\mathcal{R}$ should fulfill so that unsafety of $\mathcal{R}$ can be transferred to $\mathcal{R}'$. In this section we demonstrate that it is sufficient to require that $\mathcal{R}'$ is *semantically conservative* over $\mathcal{R}$.

Definition 14. Let $\mathcal{R}$ and $\mathcal{R}'$ be two RBoxes. We say that $\mathcal{R}'$ *is semantically conservative over* $\mathcal{R}$ (notation: $\mathcal{R} \trianglerighteq \mathcal{R}'$), if every model $\mathcal{I}$ of $\mathcal{R}$ can be expanded to a model $\mathcal{I}'$ of $\mathcal{R}'$ by interpreting new role names from $\mathsf{RN}(\mathcal{R}') \setminus \mathsf{RN}(\mathcal{R})$. If, additionally, we have $[\mathcal{R}] \subseteq [\mathcal{R}']$, then $\mathcal{R}'$ is called a *semantic conservative extension of* $\mathcal{R}$.

Example 1. Consider RBoxes depicted in Fig. 6. We have $\mathcal{R}_c \trianglerighteq \mathcal{R}'_c$. Indeed, given any model $\mathcal{I} \models \mathcal{R}_c$, we define $\mathcal{I}'$ by setting $R^{\mathcal{I}'} := R^{\mathcal{I}}$, $S^{\mathcal{I}'} := S^{\mathcal{I}}$, and $T^{\mathcal{I}'} := R^{\mathcal{I}} \cap S^{\mathcal{I}}$. Then $\mathcal{I}' \models \mathcal{R}'_c$; in particular, $T^{\mathcal{I}'}$ is transitive as the intersection of transitive relations.

At the same time, $\mathcal{R}_a \ntrianglerighteq \mathcal{R}'_a$, since one can easily construct two non-transitive relations on some set that have no transitive relations between them. Furthermore, $\mathcal{R}_b \ntrianglerighteq \mathcal{R}'_b$; indeed, take $\Delta^{\mathcal{I}} = \{0, 1, 2\}$ and set $R^{\mathcal{I}} := \{\langle 0, 1\rangle\}$, $S^{\mathcal{I}} := \{\langle 1, 2\rangle\}$, and $Q^{\mathcal{I}} := R^{\mathcal{I}} \cup S^{\mathcal{I}}$. Then we cannot interpret the transitive role T to satisfy $\mathcal{R}'_b$.

Theorem 7 (Preservation of Unsafety under Conservative Extensions of RBoxes). *If $\mathcal{R}'$ is a conservative extension of $\mathcal{R}$ and $\mathcal{R}$ is $\mathcal{L}$-unsafe, then $\mathcal{R}'$ is $\mathcal{L}$-unsafe.*

Let $\mathcal{F} := \{\mathcal{R}_\vee^\oplus\} \cup \{\mathcal{R}_\wedge^n, \mathcal{R}_\vee^n, \mathcal{R}_{\wedge\!\!\!\vee}^n \mid n \geqslant 1\}$ be the family of RBoxes (see. Fig. 3) that we have shown to be $\mathcal{ALCQ}$-unsafe in Sect. 3. As a consequence of Theorems 2, 3, and 7, we obtain the following result:

Corollary 4. *(1) Any $\mathcal{R}'$ that is a conservative extension of $\{\mathsf{Tr}(r)\}$ is $\mathcal{ALCIQ}$-unsafe; (2) Any $\mathcal{R}'$ that is a conservative extension of some RBox $\mathcal{R} \in \mathcal{F}$ is $\mathcal{ALCQ}$-unsafe.*

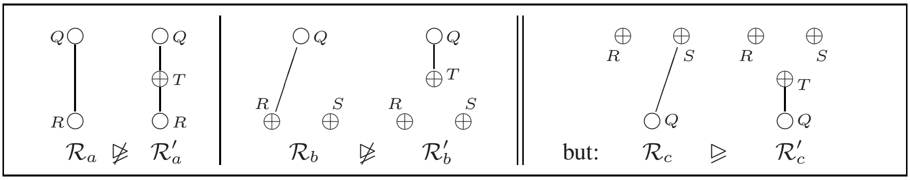

Q
R
Q
T
R
$\mathcal{R}_a \ntrianglerighteq \mathcal{R}'_a$
Q
R
S
Q
T
R
S
$\mathcal{R}_b \ntrianglerighteq \mathcal{R}'_b$
R
S
Q
R
S
T
Q
but: $\mathcal{R}_c \trianglerighteq \mathcal{R}'_c$

Fig. 6. For $i \in \{a, b\}$, we have $\mathcal{R}_i \sqsubseteq \mathcal{R}'_i$ and $\mathcal{R}_i \ntrianglerighteq \mathcal{R}'_i$; whereas $\mathcal{R}_c \trianglerighteq \mathcal{R}'_c$

It turns out, surprisingly, that properties (1) and (2) in Corollary 4 can be checked in polynomial in the size of $\mathcal{R}'$ (see [7] for details). We conjecture that Corollary 4 describes *all* RBoxes (modulo role renaming) that are unsafe for $\mathcal{ALCIQ}$ and $\mathcal{ALCQ}$.

7 Conclusions and Future Work

Driven by applications, we have looked more closely at the effect of non-simple roles in number restrictions on the decidability of standard DL reasoning problems. We have shown that, in the absence of inverse roles, the restriction imposed by $\mathcal{SHQ}$ to non-simple roles in number restrictions can be relaxed substantially and that, in the presence of inverse roles, this restriction turns out to be crucial for decidability.

These results raise numerous further questions. Firstly, given a DL $\mathcal{L}$, can we formulate necessary and sufficient conditions for an RBox to be $\mathcal{L}$-safe? Secondly, for an interesting class of $\mathcal{L}$-safe RBoxes $\mathcal{R}$, what is the computational complexity of deciding $\mathcal{L}(\mathcal{R})$-satisfiability? And can these decision procedures be implemented and used in practice? Thirdly, in the approach taken here, we allow all roles to occur in number restrictions. Given an $\mathcal{L}$-unsafe RBox $\mathcal{R}$, can we extend the notion of simple roles to regain decidability of $\mathcal{L}(\mathcal{R})$? And how applicable would this be in practice? Finally, in the presence of inverse roles, can we restrict the usage of inverse roles in TBoxes so as to re-gain decidability? For example, would disallowing number restrictions on inverse roles whilst allowing number restrictions on transitive role names help? For the list of other interesting open problems, see the accompanying technical report [7].

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Baader, F., Lutz, C., Sturm, H., Wolter, F.: Fusions of Description Logics and Abstract Description Systems. Journal of Artificial Intelligence Research (JAIR) 16, 1–58 (2002)
3. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Perspectives in Mathematical Logic. Springer, Heidelberg (1997)
4. Cerrato, C.: Decidability by Filtration for Graded Modal Logics (Graded Modalities V). Studia Logica 53, 61–74 (1994)
5. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
6. Horrocks, I., Sattler, U., Tobies, S.: Practical reasoning for very expressive description logics. Logic Journal of the IGPL 8(3), 239–263 (2000)
7. Kazakov, Y., Sattler, U., Zolin, E.: Is Your RBox Safe? Technical report. The University of Manchester (2007), available at `http://www.cs.man.ac.uk/~ezolin/pub/`
8. Wolstencroft, K., Brass, A., Horrocks, I., Lord, P., Sattler, U., Turi, D., Stevens, R.: A Little Semantic Web goes a long way in biology. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 786–800. Springer, Heidelberg (2005)

On Finite Satisfiability of the Guarded Fragment with Equivalence or Transitive Guards*

Emanuel Kieroński[1] and Lidia Tendera[2]

[1] Institute of Computer Science, University of Wrocław
[2] Institute of Mathematics and Informatics, Opole University

Abstract. The guarded fragment of first-order logic, GF, enjoys the finite model property, so the satisfiability and the finite satisfiability problems coincide.

We are concerned with two extensions of the two-variable guarded fragment that do not possess the finite model property, namely, GF^2 with equivalence and GF^2 with transitive guards. We prove that in both cases every finitely satisfiable formula has a model of at most double exponential size w.r.t. its length.

To obtain the result we invent a strategy of building finite models that are formed from a number of multidimensional grids placed over a cylindrical surface. The construction yields a 2NEXPTIME-upper bound on the complexity of the finite satisfiability problem for these fragments. For the case with equivalence guards we improve the bound to 2EXPTIME.

1 Introduction

In this paper we assume that a first-order signature contains only relational symbols and, possibly, equality.

The *guarded fragment*, GF, of first-order logic is defined as the least set of formulas such that: (i) every atomic formula belongs to GF; (ii) GF is closed under logical connectives $\neg, \vee, \wedge, \rightarrow$; and (iii) quantifiers are appropriately relativized by atoms, i.e. if $\varphi(\mathbf{x}, \mathbf{y})$ is a formula of GF and $\alpha(\mathbf{x}, \mathbf{y})$ is an atomic formula containing all the free variables of φ, then the formulas

$$\forall \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \rightarrow \varphi(\mathbf{x}, \mathbf{y})) \quad \text{and} \quad \exists \mathbf{y}(\alpha(\mathbf{x}, \mathbf{y}) \wedge \varphi(\mathbf{x}, \mathbf{y}))$$

belong to GF. The atom $\alpha(\mathbf{x}, \mathbf{y})$ is called the *guard* of the quantifier. Equalities are also allowed as guards.

The guarded fragment was introduced by Andréka et al., [1], who showed that modal logic can be embedded in GF and they also argued convincingly that GF inherits the nice properties of modal logic. The nice behavior of GF was confirmed by Grädel, [4], who proved that GF enjoys the finite model property and that the satisfiability problem for GF is complete for double exponential time and complete for exponential time, when the number of variables is bounded.

GF was later generalized to the *loosely guarded fragment*, [19], and to the *clique guarded fragment*, [5], where all quantifiers were relativized by more general formulas,

---

* Supported by Polish Ministry of Science and Higher Education research project nr N206 022 31/3660 2006/2009.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 318–332, 2007.
© Springer-Verlag Berlin Heidelberg 2007

preserving the idea of quantification only over elements that were close together in the model. Most of the properties of GF can be generalized to those fragments. In particular, I. Hodkinson [8] showed that they all enjoy the finite model property (see also [9] for a simpler and nicer proof of the result).

For modal or description logics, a restriction of the underlying class of structures is very natural. We are particularly interested in classes with some transitive or equivalence relations that correspond e.g. to Kripke structures for multi-modal variants of the logics K4 and S5. The results for these cases can be phrased either for satisfiability of extensions of GF, or of GF itself over restricted classes of structures.

In this paper we are concerned with two extensions of GF^2, i.e. the two-variable guarded fragment with equivalence guards, $\mathrm{GF}^2+\mathrm{EG}$, and the two-variable guarded fragment with transitive guards, $\mathrm{GF}^2+\mathrm{TG}$. In $\mathrm{GF}^2+\mathrm{EG}$ ($\mathrm{GF}^2+\mathrm{TG}$) some binary relation symbols are required to be interpreted as equivalences (transitive relations), but these *special* symbols are allowed to appear only in guards. These fragments capture many expressive modal and description logics. Both are decidable for satisfiability: $\mathrm{GF}^2+\mathrm{EG}$ is NEXPTIME-complete, [12], $\mathrm{GF}^2+\mathrm{TG}$ – 2EXPTIME-complete, [16], and both contain infinity axioms, i.e. formulas that possess only infinite models.

The lack of the finite model property for the above mentioned fragments naturally leads to the question, whether their finite satisfiability problems are decidable. This question is particularly important, if one would like to use these formalism for automatic reasoners in practical applications, where the structures investigated are essentially finite.

In [18] it is shown that the finite satisfiability problem for GF with one transitive relation is decidable in double exponential time, thus it is of the same complexity as the (unrestricted) satisfiability problem. For the case of $\mathrm{GF}^2+\mathrm{EG}$, a NEXPTIME lower bound is implied by GF with one equivalence relation, that enjoys the finite (exponential) model property and is NEXPTIME-complete for satisfiability, [12].

In this paper we show that every finitely satisfiable $\mathrm{GF}^2+\mathrm{EG}$-formula and every finitely satisfiable $\mathrm{GF}^2+\mathrm{TG}$-formula has a model of at most double exponential size. In the case of $\mathrm{GF}^2+\mathrm{EG}$ our small models are constructed as multidimensional grids, in which the number of dimensions equals the number of equivalence relations. Models for $\mathrm{GF}^2+\mathrm{TG}$ are built from a number of such grids placed over a cylindrical surface and connected in a special way.

The results give also a trivial, double exponential, bound on the size of a single equivalence class in a finite model of a $\mathrm{GF}^2+\mathrm{EG}$-formula. We argue that this bound is essentially optimal, which contrasts with the general case, where it can be shown that every satisfiable $\mathrm{GF}^2+\mathrm{EG}$-sentence has a (possibly infinite) model with at most exponential equivalence classes.

The constructions yield 2NEXPTIME-upper bounds on the complexity of the finite satisfiability problem for these fragments. For the case with equivalence guards we improve the bound to 2EXPTIME. We also note that in $\mathrm{GF}^2+\mathrm{TG}$ transitive relations can be enforced in a simple way to be symmetric and reflexive, see Lemma 2 in [12]. Thus, the upper bound for $\mathrm{GF}^2+\mathrm{TG}$ gives also the upper bound for a combined variant $\mathrm{GF}^2+\mathrm{EG}+\mathrm{TG}$, in which some symbols appearing only in guards have to be interpreted as equivalences and some other as transitive relations.

It is perhaps worth mentioning that there is another interesting extension of the guarded fragment, namely GF with fixpoints, that has been shown decidable for satisfiability, [7], and of the same complexity as pure GF. However, decidability of the finite satisfiability problem for this fragment is only partially answered by Bojańczyk in [2], where decidability of the finite satisfiability problem for the modal μ-calculus with backwards modalities is shown.

We also remark that recently, [15], tight exponential complexity bounds for GF^2 with counting quantifiers for both satisfiability and finite satisfiability have been established.

2 Preliminaries

For a fragment $\mathcal{L}$ of first-order logic we denote by $\mathcal{L}^k$ the restriction of the logic to k variables. Since in this paper we are concerned with extensions of GF^2, we assume without loss of generality that signatures contain only unary and binary relation symbols.

We fix a signature σ. An *atomic k-type* is a maximal consistent set of atomic or negated atomic formulas over σ in k variables. Let $\boldsymbol{\alpha}$ be the set of all atomic 1-types, and $\boldsymbol{\beta}$ the set of all atomic 2-types over σ.

Let $\mathfrak{A}$ be a σ-structure with universe A, and let $a, b \in A$, $a \neq b$. We denote by $\alpha(a)$ the unique atomic 1-type realized in $\mathfrak{A}$ by the element a, and by $\beta(a, b)$ – the unique atomic 2-type realized in $\mathfrak{A}$ by the pair (a, b). We note that the notation $\alpha(a)$ and $\beta(a, b)$ does not explicitly show from which structure the elements a and b are taken, but, in this paper, the structure will always be clear from the context.

2.1 Simple Observations on FO^2 Models

Since equivalence classes in models of $\mathrm{GF}^2 + \mathrm{EG}$ formulas behave similarly to whole models of FO^2 formulas, we start with some simple definitions and lemmas for FO^2.

Let $M = 3|\boldsymbol{\beta}|^3$ be fixed for the rest of this paper. Note that the value of M is exponential in $|\sigma|$.

Definition 1 (Scott normal form). *We say that an* FO^2*-formula φ is in* normal form *if $\varphi = \forall x \forall y \psi_0(x, y) \wedge \bigwedge_i \forall x \exists y \psi_i(x, y)$, where ψ_i are quantifier-free.*

Definition 2. (i) *An* exact counting type *is a function $\theta : \boldsymbol{\alpha} \to \mathbb{N}$. We say that a structure $\mathfrak{A}$ has an exact counting type θ if for every $t \in \boldsymbol{\alpha}$, $\theta(t)$ is the number of realizations of t in $\mathfrak{A}$.*

(ii) *Let $n > 0$. An n-*counting type *is a function $\theta : \boldsymbol{\alpha} \to \{0, 1, \ldots, n\}$. We say that a structure $\mathfrak{A}$ has an n-counting type θ if for every $t \in \boldsymbol{\alpha}$, $\theta(t)$ is the number of realizations of t in $\mathfrak{A}$, if this number is less than n, and $\theta(t) = n$ otherwise.*

We say that a counting type θ' *safely extends* a counting type θ if for every $t \in \boldsymbol{\alpha}$, $\theta'(t) = \theta(t)$ if $\theta(t) = 0$ or $\theta(t) = 1$, and $\theta'(t) \geq \theta(t)$ otherwise.

The following lemma establishes two useful properties of counting types and the relevance of M. Part (ii) is a simple application of the "small substructures lemma" from [13]. The proof is omitted due to space limits.

Lemma 1. (i) *Let φ be an* FO^2*-formula in normal form, $\mathfrak{A} \models \varphi$ and let θ be the exact counting type of $\mathfrak{A}$. Then for every exact counting type θ' safely extending θ, there exists a model $\mathfrak{A}' \models \varphi$ whose exact counting type is θ'.*

(ii) *Let φ be an* FO^2*-formula in normal form, $\mathfrak{A} \models \varphi$, θ be the M-counting type of $\mathfrak{A}$. Then, there exists a model $\mathfrak{A}' \models \varphi$ whose exact counting type is θ.*

2.2 Normal Forms

Let $S_1, S_2, \ldots, S_k$ denote the *special* binary relation symbols of σ, i.e. those that are required to be interpreted as equivalence or transitive relations, and $R_1, R_2, \ldots$ – the remaining binary symbols in σ. A σ-structure is *special*, if all the special relation symbols are interpreted in the required way.

When the special relation symbols are allowed to appear in guards only, it is useful to consider the following normal form for GF^2-formulas.

Definition 3. *We say that a* GF^2*-formula φ with special guards is in normal form if it is a conjunction of formulas of the following form:*

(i) $\exists x(p(x) \wedge \psi(x))$,
(ii) $\forall x \forall y(S_i xy \rightarrow \psi(x,y))$,
(iii) $\forall x \forall y(q(x,y) \rightarrow \psi(x,y))$,
(iv) $\forall x(p(x) \rightarrow \exists y(q(x,y) \wedge \psi(x,y)))$,
(v) $\forall x(p(x) \rightarrow \exists y(S_i xy \wedge S_i yx \wedge \psi(x,y)))$,
(vi) $\forall x(p(x) \rightarrow \exists y(S_i xy \wedge \neg S_i yx \wedge \psi(x,y)))$,
(vii) $\forall x(p(x) \rightarrow \exists y(S_i yx \wedge \neg S_i xy \wedge \psi(x,y)))$,

where each ψ is quantifier-free and does not contain special relation symbols, $p(x)$ is atomic, $q(x,y)$ is an atom $R_i(x,y)$ or $R_i(y,x)$. We denote by φ_{sym} the fragment of φ consisting of all the conjuncts of types (i), (ii), (iii), (iv) and (v), and by φ_{eq} the fragment of φ consisting of all the conjuncts of types (i), (ii), (iii) and (v). For a given i, we denote by $\varphi_{eq}^{S_i}$ the fragment of φ_{eq} consisting of those conjuncts of type (ii) and (v), which have S_i in their guards, and all conjuncts of type (iii). Additionally, let φ_{univ} be the conjunction of all implications $q(x,y) \rightarrow \psi(x,y)$ from formulas of type (iii).

The (finite) satisfiability problem for GF^2-formulas with special guards over special structures can be reduced to the (finite) satisfiability problem for disjunctions of exponential number of linear size GF^2-formulas in normal form over special structures. See for example [18] for a proof of a similar result.

We note that if the special relation symbols are required to be interpreted as equivalence relations, then in the normal form we do not need to consider conjuncts of the forms (vi) and (vii) and the conjuncts of the form (v) may be simplified to $\forall x(p(x) \rightarrow \exists y(S_i xy \wedge \psi(x,y)))$.

We remark that, although in conjuncts of the forms (v)-(vii) the special symbols appear not only as simple guards, the normal form is more useful than the standard one where, instead of conjuncts of the forms (v)-(vii), we have just the form: $\forall x(p(x) \rightarrow \exists y(S_i xy \wedge \psi(x,y)))$.

In next sections we use E_is to denote the special relations symbols that are to be interpreted as equivalence relations, and T_is to denote the transitive relation symbols.

3 Equivalence Guards

In this section we show that every finitely satisfiable $\mathrm{GF}^2+\mathrm{EG}$-formula has a model of a bounded size.

If we restrict the number of equivalence symbols to one, then the finite satisfiability coincides with satisfiability, since, [13], even whole FO^2 with one equivalence relation has the finite (exponential) model property. However, in the presence of two equivalence symbols, there are satisfiable $\mathrm{GF}^2+\mathrm{EG}$ formulas whose all models are infinite [13].

3.1 Example

Let us now observe, that finite models for $\mathrm{GF}^2+\mathrm{EG}$ have different properties from infinite models. In the following example we show how to construct a family of finitely satisfiable formulas $\{\varphi_n\}$ with only two equivalence relation symbols, such that every finite model of φ_n contains at least one equivalence class of size at least doubly exponential in n, and $|\varphi_n|$ is polynomial in n. This is in contrast to the (unrestricted) satisfiability: in [12] it was observed that every satisfiable $\mathrm{GF}^2+\mathrm{EG}$-formula φ has a model, in which all equivalence classes have size at most exponential in $|\varphi|$. The following example is a refinement of the example from [18], where several transitive relations were used.

Example 1. Let us assume that E_1 and E_2 have to be interpreted as equivalence relations. We construct a finitely satisfiable formula φ_n, such that its every model contains some number of full binary trees of depth 2^n, whose every leaf requires a root in its E_2-class. Trees will have to be disjoint, so there will always be 2^{2^n} leaves per one root, which will guarantee the existence of at least one large E_2-class.

Except E_1 and E_2, we use only a number of unary relation symbols. Symbols $P_0, \ldots, P_{n-1}$ encode in each element a number from $\{0, \ldots 2^n - 1\}$ which is the depth of the element in the tree (the number of the level to which the element belongs). Let us denote by $\mathcal{L}_i$ the i-th level, i.e. the set of elements a, with the encoded number i. Symbol R indicates roots, symbol L is used to distinguish left sons from right sons. The formula φ_n consists of conjuncts stating:

- There exists an element satisfying R.
- Every element satisfying R belongs to $\mathcal{L}_0$.
- For each element a from $\mathcal{L}_{2i}$ there exist two elements in $\mathcal{L}_{2i+1}$ connected to a by E_1. One of them satisfies L, the other $\neg L$.
- For each element a from $\mathcal{L}_{2i+1}$ (for $2i+1 < 2^n - 1$) there exist two elements in $\mathcal{L}_{2i+2}$ connected to a by E_2. One of them satisfies L, the other $\neg L$.
- If two distinct elements belong to a level $\mathcal{L}_{2i}$ for some i, then they are not connected by E_1.
- If two distinct elements belong to a level $\mathcal{L}_{2i+1}$ (for $2i+1 < 2^n - 1$), then they are not connected by E_2.
- Every element in $\mathcal{L}_{2^n-1}$ is connected by E_2 to an element satisfying R.

It is not difficult to formulate the above sentences in $\mathrm{GF}^2+\mathrm{EG}$, and to see, that each $\mathfrak{A} \models \varphi_n$ has a desired large E_2-class.

3.2 System of Linear Inequalities

Let φ be a $\mathrm{GF}^2+\mathrm{EG}$-formula in normal form over a signature σ, where $E_1, E_2, \ldots, E_k$ are the equivalence relation symbols.

We say that an n-counting type θ is *admissible for* φ and E_i, if there exists a structure $\mathfrak{C}$ whose exact counting type is θ and

$$\mathfrak{C} \models \varphi_{eq}^{E_i} \wedge \forall xy E_i xy \wedge \bigwedge_{j \neq i} \forall xy (x \neq y \rightarrow \neg E_j xy).$$

Let $\Theta = (\Theta^{E_1}, \Theta^{E_2}, \ldots, \Theta^{E_k})$ be a tuple of sets of M-counting types. We say that an atomic type $t \in \boldsymbol{\alpha}$ *appears* in Θ^{E_i} if there exists $\theta \in \Theta^{E_i}$ such that $\theta(t) > 0$. We say that t appears in Θ if t appears in Θ^{E_i} for some i. We say that $t \in \boldsymbol{\alpha}$ is *royal* for E_i-classes, if t appears in Θ^{E_i} and, for every $\theta \in \Theta^{E_i}$, we have $\theta(t) \leq 1$.

Let $\boldsymbol{\alpha}_0 \subseteq \boldsymbol{\alpha}$ and let $\overline{\rho} = \rho_1, \rho_2, \ldots, \rho_k$ be a sequence of subsets of $\boldsymbol{\alpha}_0$. Let $\Theta(\varphi, \boldsymbol{\alpha}_0, \overline{\rho}) = (\Theta^{E_1}, \Theta^{E_2}, \ldots, \Theta^{E_k})$ be the tuple of maximal sets of M-counting types such that: (i) $\boldsymbol{\alpha}_0$ is the set of one-types appearing in $\Theta(\varphi, \boldsymbol{\alpha}_0, \overline{\rho})$, (ii) ρ_i is the set of royal types for E_i-classes, and (iii) every element of Θ^{E_i} is admissible for φ and E_i.

The set $\boldsymbol{\alpha}_0$ corresponds to the set of one-types realized in a structure in which elements of ρ_i are royal for E_i-classes. Elements of Θ^{E_i} correspond to M-counting types of restrictions of the structure to equivalence classes of E_i.

With $\Theta(\varphi, \boldsymbol{\alpha}_0, \overline{\rho})$ we associate a system of linear inequalities Γ_Θ. For each i and each $\theta \in \Theta^{E_i}$ we use a variable $X_\theta^{E_i}$, whose purpose is to count the number of E_i-classes with type θ, and, for each $t \in \boldsymbol{\alpha}_0$, we use a variable $Y_t^{E_i}$ to count the lower bound of the number of elements of type t in E_i-classes.

For every $t \in \boldsymbol{\alpha}_0$ and every i, we put to Γ_Θ the following (in)equalities:

-
$$Y_t^{E_i} = \sum_{\theta \in \Theta^{E_i}} \theta(t) \cdot X_\theta^{E_i}, \tag{E0}$$
- if $t \in \rho_i$, then we write:
$$Y_t^{E_i} \geq 1, \tag{E1a}$$
- if $t \notin \rho_i$, then we write:
$$\sum_{\theta \in \Theta^{E_i}, \theta(t) > 1} X_\theta^{E_i} \geq 1, \tag{E1b}$$
- if $t \in \rho_i$, then, for every $j \neq i$, we write:
$$Y_t^{E_i} \geq Y_t^{E_j}. \tag{E2}$$

Please note that if t is royal for E_i-classes then the number $Y_t^{E_i}$ is exact, and if t is royal for both E_i-classes and E_j-classes ($i \neq j$) then we write two inequalities of the form (E2) that give $Y_t^{E_i} = Y_t^{E_j}$.

Moreover, we add to Γ_Θ the following inequalities:

- for every conjunct δ of the form (i), $\delta = \exists x(p(x) \wedge \psi(x))$,we write

$$\sum_{t \in \boldsymbol{\alpha}_0, t \models p(x) \wedge \psi(x)} Y_t^{E_1} > 0, \tag{E3}$$

- for every conjunct δ of the form (iv), $\delta = \forall x(p(x) \rightarrow \exists y(q(x,y) \wedge \psi(x,y)))$, for every $t \in \boldsymbol{\alpha}_0$ such that $t \models p(x)$, we write:

$$\sum_{t' \in \boldsymbol{\alpha}_0, t \rightarrow \delta t'} Y_{t'}^{E_1} > 0, \tag{E4}$$

  where $t \rightarrow \delta t'$, if for some $s(x,y) \in \boldsymbol{\beta}$ such that $t(x) \subseteq s(x,y)$, $t'(y) \subseteq s(x,y)$, we have $\neg E_i xy \in s(x,y)$, for all i, and $s(x,y) \models \varphi_{univ} \wedge q(x,y) \wedge \psi(x,y)$.

Observe that the number of inequalities is at most exponential and the number of variables is at most doubly exponential in the size of the signature.

3.3 Model Construction

Lemma 2. *Let φ be a $\mathrm{GF}^2+\mathrm{EG}$-formula φ in normal form. Then, φ is finitely satisfiable if and only if there exist $\boldsymbol{\alpha}_0$, $\overline{\rho} = \rho_1, \ldots, \rho_k$, with $\rho_i \subseteq \boldsymbol{\alpha}_0$, such that the system Γ_Θ constructed for $\Theta(\varphi, \boldsymbol{\alpha}_0, \overline{\rho})$ has a nonnegative integer solution.*

Proof. ($\Rightarrow$) Let $\mathfrak{A}$ be a finite model of φ. Let $\boldsymbol{\alpha}_0$ be the set of 1-types realized in $\mathfrak{A}$. Let A/E_i be the set of equivalence classes of E_i in $\mathfrak{A}$. Let Θ^{E_i} be the set of all M-counting types of the structures $\mathfrak{A} \restriction K$, for $K \in A/E_i$. Let ρ_i be the set of all royal types for E_i-classes.

Let $\theta \in \Theta^{E_i}$ be the M-counting type of $\mathfrak{A} \restriction K$ for some K. Observe that $\mathfrak{A} \restriction K$ is a model of the FO^2-formula $\varphi_{eq}^{E_i} \wedge \forall xy E_i xy$. It can be simply modified to a model of $\varphi_{eq}^{E_i} \wedge \forall xy E_i xy \wedge \bigwedge_{j \neq i} \forall xy(x \neq y \rightarrow \neg E_j xy)$, by replacing in every atomic 2-type atoms $E_j xy$ by $\neg E_j xy$ and atoms $E_j yx$ by $\neg E_j yx$. The last formula can be easily rewritten to FO^2 normal form. Now we apply part (ii) of Lemma 1, obtaining the desired exact counting type of the model. So, θ is admissible for φ and E_i.

It is not difficult to see that $X_\theta^{E_i} = r_\theta^{E_i}$, where $r_\theta^{E_i}$ is the number of those E_i-classes in $\mathfrak{A}$, whose M-counting types equal θ, and $Y_t^{E_i} = \sum_{\theta \in \Theta^{E_i}} \theta(t) \cdot r_\theta^{E_i}$ is a nonnegative integer solution of Γ_Θ.

($\Leftarrow$) Let the system Γ_Θ constructed for $\Theta(\varphi, \boldsymbol{\alpha}_0, \overline{\rho}) = (\Theta^{E_1}, \ldots \Theta^{E_k})$ with appropriate $\boldsymbol{\alpha}_0$ and $\overline{\rho}$ has a nonnegative integer solution: $X_\theta^{E_i} = r_\theta^{E_i}$ and $Y_t^{E_i} = r_t^{E_i}$.

First, we show how to build a model $\mathfrak{A} \models \varphi_{eq}$. Then, a model $\mathfrak{A}' \models \varphi$, will be obtained by taking an appropriate number of copies of $\mathfrak{A}$ and setting the 2-types in a simple way.

For every $t \in \boldsymbol{\alpha}_0$, let $n_t = \max_i\{r_t^{E_i}\}$. Observe that by (E0) and (E1a) or (E1b), $n_t > 0$. The structure $\mathfrak{A}$ is constructed from some number of copies of the *base set* of elements. The base set consists of exactly n_t realizations of type t, for every $t \in \boldsymbol{\alpha}_0$. Let $\{a_0, \ldots a_{l-1}\}$ be the elements of the base set. We put copies of elements from the base set into the nodes of a k-dimensional grid (recall that k is the number of equivalence relation symbols): at location $(x_1, \ldots, x_k)$, for $0 \leq x_i < l$, we put an element whose 1-type equals the 1-type of a_s, where $s = (x_1 + \ldots + x_k) \bmod l$. Note that the set of elements located on each line of the grid is exactly a copy of the base set. In Fig. 1 we picture the desired arrangement of elements for $l = 5$ and $k = 2$.

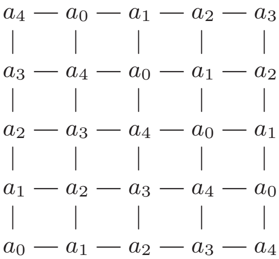

a_4 — a_0 — a_1 — a_2 — a_3
a_3 — a_4 — a_0 — a_1 — a_2
a_2 — a_3 — a_4 — a_0 — a_1
a_1 — a_2 — a_3 — a_4 — a_0
a_0 — a_1 — a_2 — a_3 — a_4

Fig. 1. Arrangement of elements on the two-dimensional grid

To settle properly the 2-types of elements in $\mathfrak{A}$ we first observe that for every i, the base set can be divided into $\sum_{\theta \in \Theta^{E_i}} r_\theta^{E_i}$ disjoint parts, so that for each part P there exists an exact counting type θ', safely extending some $\theta \in \Theta^{E_i}$, such that the number of elements of every atomic 1-type t in P equals $\theta'(t)$. The desired division is obtained as follows. We create $r_\theta^{E_i}$ parts for every $\theta \in \Theta^{E_i}$. To each of this parts we put exactly $\theta(t)$ elements of type t, for every $t \in \boldsymbol{\alpha}_0$. Note, that, because of the choice of the numbers n_t, we have enough copies of elements of every type t. After this step we may have some elements remaining. Observe, that due to inequalities of the form (E2), none of the types of the remaining elements is royal for E_i. All the remaining elements of type t are joined to a part which contains at least two elements of t. Note that such a part exists due to inequalities of the form (E1b).

Now, horizontal lines of the grid are divided as above to form equivalence classes of E_1, vertical lines – to form classes of E_2, lines going in the third dimension – to form classes of E_3, and so on. To define the structure on each of the classes we use the fact that each variable $X_\theta^{E_i}$ in the system corresponds to an M-counting type θ^{E_i} admissible for φ and E_i and, (if necessary) part (i) of Lemma 1. We complete the definition of $\mathfrak{A}$ by setting all 2-types for pairs of elements not belonging to the same class to *binary-free types*, i.e. 2-types containing $\neg Rxy$ and $\neg Ryx$ for every binary R (including equivalence symbols).

To construct $\mathfrak{A}'$ we take three sets of copies of $\mathfrak{A}$, each consisting of h elements, where h is the number of conjuncts of φ of the form (iv); we can suppose $h > 0$. This step is reminiscent of the construction of the small model for FO^2 [6].

Let $A' = A \times \{0, 1, \ldots, h-1\} \times \{0, 1, 2\}$. We set $\alpha((a, i, j)) = \alpha(a)$ and $\beta((a, i, j), (b, i, j)) = \beta(a, b)$. Now we have to provide witnesses for all elements in A' for conjuncts of φ of the form (iv). This is done in a circular way: elements from $A \times \{0, 1, \ldots, h-1\} \times \{j\}$ find witnesses in $A \times \{0, 1, \ldots, h-1\} \times \{j'\}$, where $j' = j + 1 \bmod 3$. Let us take for example an element (a, i, j). Let $\delta_0, \ldots, \delta_{h-1}$ be all conjuncts of φ of the form (iv), $\delta_m = \forall x (p(x) \rightarrow \exists y (q(x, y) \wedge \psi(x, y)))$. Let $t = \alpha(a)$.

For $0 \leq m < h$ we choose $t'_m \in \boldsymbol{\alpha}_0$, and $s_m \in \boldsymbol{\beta}$, whose existence is postulated by inequality (E4). By an appropriate inequality of the form $(E1a)$ or $(E1b)$, t'_m is realized in $\mathfrak{A}$, say, by an element b_m. We set $\beta((a,i,j),(b_m,m,j+1 \bmod 3)) := s_m$. This circular scheme guarantees no conflict in setting 2-types, since a 2-type for a pair of elements in $\mathfrak{A}'$ is defined at most once. We complete the structure by setting 2-types for all remaining pairs of elements. Let $((a,i,j),(a',i',j'))$ be such a pair. Let $t = \alpha(a)$, $t' = \alpha(b)$. Let s be the unique 2-type containing $t(x)$, $t(y)$ and negations of all binary atoms. We set $\beta((a,i,j),(a',i',j')) := s$. □

3.4 Size of Models and Complexity of Finite Satisfiability

We use the following lemma from [3], built on the classical result [14], to obtain an upper bound on the size of the model constructed in part $\Leftarrow$ of the proof of Lemma 2.

Lemma 3 (Calvanese). *Let Γ be a system of m linear inequalities in n unknowns, and let the coefficients and constants that appear in the inequalities be in $\{-a,\dots,a-1,a\}$. If Γ admits a nonnegative integer solution, then it also admits one in which the values assigned to the unknows are all bounded by $(n+m)\cdot(m\cdot a)^{2m+1}$.*

In our system Γ_Θ the parameters a and m are at most exponential, and n – at most doubly exponential in the size of the signature. For a given φ in normal form we may consider the signature consisting only of relation symbols appearing in φ. Henceforth, by Lemma 3, if Γ_Θ has a solution, it has also one with values at most doubly exponential in $|\varphi|$. Thus, in the proof of Lemma 2 we may choose a base set consisting of a double exponential number of elements, obtaining finally a double exponential bound on the size of the whole model. In the consequence, we can state the following theorem.

Theorem 1. *Every finitely satisfiable* $\mathrm{GF}^2{+}\mathrm{EG}$*-formula φ has a model of size at most doubly exponential in $|\varphi|$.*

The obtained bound on the size of finite models is essentially optimal – recall the formula from Example 1, which has only models of at least double exponential size.

Theorem 1 immediately gives a 2NEXPTIME decision procedure for the finite satisfiability problem for $\mathrm{GF}^2{+}\mathrm{EG}$: guess a structure of at most double exponential size and check, if it is a model for the input sentence. We show that we can do better.

Theorem 2. *The finite satisfiability problem for* $\mathrm{GF}^2{+}\mathrm{EG}$ *is decidable in deterministic double exponential time.*

Proof. By Lemma 2, to check if a normal form $\mathrm{GF}^2{+}\mathrm{EG}$-sentence φ has a finite model it suffices to check if there exists $\Theta(\varphi,\boldsymbol{\alpha}_0,\overline{\rho})$ such that Γ_Θ has a nonnegative integer solution.

To do this we enumerate all possible $\boldsymbol{\alpha}_0$ and $\overline{\rho}$, write the system Γ_Θ for $\Theta = \Theta(\varphi,\boldsymbol{\alpha}_0,\overline{\rho})$, check admissibility of each M-counting type in Θ and check if Γ_Θ has a nonnegative integer solution.

Observe that Γ_Θ consists of inequalities of the form $\sum_i c_i x_i \geq b$, with $b \geq 0$. So Γ_Θ has a nonnegative integer solution if and only if it has a nonnegative rational solution.

(To get an integer solution, it suffices to take the rational solution and multiply it by the product of all the denominators.)

Since linear programming is in PTIME, [10], we can check the existence of a non-negative rational solution of Γ_Θ in time doubly exponential w.r.t. $|\varphi|$.

All these gives a deterministic procedure working in double exponential time. $\square$

We note that the best lower bound for the finite satisfiability problem for $\mathrm{GF}^2{+}\mathrm{EG}$ is NEXPTIME. This bound is obtained for the fragment with one equivalence relation that enjoys the finite model property [12].

4 Transitive Guards

In this section we show that every finitely satisfiable $\mathrm{GF}^2{+}\mathrm{TG}$ formula φ in normal form has a model of size at most doubly exponential in $|\varphi|$. In the proof we essentially use Theorem 1 and the model construction from the previous section for solutions of systems of linear inequalities for appropriate tuples of M-counting types.

4.1 Basic Notions and Outline of the Construction

Let $\mathfrak{A}$ be a finite model of φ. We construct a small model $\mathfrak{B} \models \varphi$. To follow the construction it is not harmful to think that in $\mathfrak{A}$ every pair of distinct elements is a member of at most one transitive relation. The small model $\mathfrak{B}$ will also have this property.

To construct $\mathfrak{B}$ we first distinguish in $\mathfrak{A}$ classes of elements forming T-*cliques*, for every transitive T. If we remove from $\mathfrak{A}$ nonsymmetric transitive connections then, in the obtained structure $\mathfrak{A}'$, transitive relations behave like equivalence relations and transitive cliques behave like equivalence classes. Obviously, $\mathfrak{A}' \models \varphi_{sym}$. Then, for every M-counting type θ of a T-class in $\mathfrak{A}'$, we apply the construction from the proof of part $\Leftarrow$ of Lemma 2 to $\mathfrak{A}'$ and produce a bounded model of φ_{sym} containing at least a T-class of M-counting type θ. This way, we obtain building blocks for the construction of $\mathfrak{B}$.

To be able to provide witnesses for conjuncts of the forms (vi) and (vii), we enrich the counting types of classes of $\mathfrak{A}$ with two subsets of 1-types, $\mathcal{A}$ and $\mathcal{B}$, corresponding to 1-types of elements located in $\mathfrak{A}$ *above*, respectively *below*, elements of the class. Additionally, we get out off $\mathfrak{A}$ a finite bounded pattern for providing nonsymmetric witnesses for each enriched counting type, called *a graph of witnesses*.

Finally, the model $\mathfrak{B}$ is formed out of a number of the building blocks located on a cylindrical surface and the nonsymmetric witnesses are provided in a regular manner using the graphs of witnesses.

A T-clique in a structure $\mathfrak{C}$ is a maximal set D of elements, such that $\forall x, y \in D.\ \mathfrak{C} \models x \neq y \rightarrow (Txy \wedge Tyx)$. Let $C/\hat{T}$ be the set of all T-cliques in $\mathfrak{C}$. For a pair of T-cliques D, D' in $\mathfrak{C}$ we write $D <_T^{\mathfrak{C}} D'$ if for all $a \in D$, $b \in D'$, $\mathfrak{C} \models Tab \wedge \neg Tba$.

Definition 4. *We say that a T-clique $D \in A/\hat{T}$ has an enriched n-counting type $\hat{\theta} = (\theta, \mathcal{A}, \mathcal{B})$ if:*

(i) $\mathfrak{A}\restriction D$ *has the n-counting type θ,*
(ii) $\mathcal{A} = \{t \in \boldsymbol{\alpha} : \exists a \in D, a' \in A.\ \alpha(a') = t \wedge \mathfrak{A} \models T(a, a') \wedge \neg T(a', a)\}$
(iii) $\mathcal{B} = \{t \in \boldsymbol{\alpha} : \exists a \in D, a' \in A.\ \alpha(a') = t \wedge \mathfrak{A} \models T(a', a) \wedge \neg T(a, a')\}$

Similarly to the previous section we are interested in enriched M-counting types for $M = 3|\boldsymbol{\beta}|^3$. Let $\hat{\Theta}^T$ be the set of enriched M-counting types, and Θ^T the set of M-counting types, realized in $\mathfrak{A}$ by T-cliques. In our new model $\mathfrak{B} \models \varphi$, for every enriched M-counting type $\hat{\theta} = (\theta, \mathcal{A}, \mathcal{B}) \in \hat{\Theta}^T$ we will put a T-clique of type $\hat{\theta} = (\theta, \mathcal{A}', \mathcal{B}')$, such that $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{B}' \subseteq \mathcal{B}$.

4.2 Structures for Symmetric Requirements

Lemma 4. *For every transitive symbol T, for every $\hat{\theta} = (\theta, \mathcal{A}, \mathcal{B}) \in \hat{\Theta}^T$ there exists $\mathfrak{C} \models \varphi_{sym}$, such that*

(i) *$\mathfrak{C}$ contains a distinguished T-clique of M-counting type θ,*
(ii) *$|C|$ is at most doubly exponential in $|\varphi|$,*
(iii) *all transitive symbols are interpreted as transitive and symmetric relations in $\mathfrak{C}$,*
(iv) *for every transitive T', and every M-counting type θ' of a T'-clique in $\mathfrak{C}$ there exists an enriched M-counting type $\hat{\theta}' = (\theta', \mathcal{A}, \mathcal{B}) \in \hat{\Theta}^{T'}$, for some $\mathcal{A}, \mathcal{B}$.*

Proof. Note, that a model meeting requirements (i), (iii), (iv) exists: an example is $\hat{\mathfrak{A}}$ – a modification of $\mathfrak{A}$ in which, in all 2-types, we substitute $Txy \wedge \neg Tyx$ or $Tyx \wedge \neg Txy$ with $\neg Txy \wedge \neg Tyx$, for all transitive T.

Let $\boldsymbol{\alpha}_0$ be the set of 1-types realized in $\hat{\mathfrak{A}}$, Θ^T be the set of M-counting types realized in $\hat{\mathfrak{A}}$ by T-classes, ρ_i be the set of royal types for T_i-classes in $\hat{\mathfrak{A}}$, and $\overline{\rho} = \rho_1, \rho_2, \ldots, \rho_k$. Let Γ_Θ be the system of inequalities associated with $\Theta(\varphi_{sym}, \boldsymbol{\alpha}_0, \overline{\rho})$. We extend Γ_Θ to Γ'_Θ by adding the inequality:

$$X^T_\theta > 0 \tag{E5}$$

and the equality:

$$\sum_{(T,\theta):\theta \notin \Theta^T} X^T_\theta = 0. \tag{E6}$$

Note, that Γ'_Θ has a nonnegative solution: the one corresponding to the model $\hat{\mathfrak{A}}$. Now, using Lemma 3 and the construction from the proof of part $\Leftarrow$ of Lemma 2, we obtain $\mathfrak{C}$ as desired. ☐

We apply Lemma 4 to every transitive T and $\hat{\theta} \in \hat{\Theta}^T$, obtaining structures $\mathfrak{C}^T_{\hat{\theta}}$. Additionally, we define a function enr on the cliques in these structures. If D is the distinguished T-clique in $\mathfrak{C}^T_{\hat{\theta}}$, then $enr(D) = \hat{\theta}$. In the other case let us assume that D is a T'-clique and let θ' be the M-counting type of D. Choose any $\hat{\theta}' = (\theta', \mathcal{A}, \mathcal{B}) \in \hat{\Theta}^{T'}$ as the value of $enr(D)$. The purpose of $enr(D)$ is to say, elements of which 1-types are allowed to be connected to D by transitive relations from above and from below.

4.3 Graphs of Witnesses

In this subsection we define labeled multidigraphs which contain patterns for providing nonsymmetric transitive witnesses. Nodes of these graphs are enriched M-types realized in $\mathfrak{A}$. An edge from $\hat{\theta}$ to $\hat{\theta}'$, labeled with a triple of atomic types (t, t', s), says that each element of 1-type t from a clique of type $\hat{\theta}$ should look for a witness of 1-type t' in a clique of type $\hat{\theta}'$, and that the element and its witness should be connected by the 2-type s.

We construct independently two graphs for each transitive symbol T. The first of them says where to look for witnesses for formulas with guards of the form (vii) $\forall x(p(x) \rightarrow \exists y(Tyx \wedge \neg Txy \wedge \psi(x,y)))$, i.e. *lower witnesses*. The second – for formulas of the form (vi) $\forall x(p(x) \rightarrow \exists y(Txy \wedge \neg Tyx \wedge \psi(x,y)))$, i.e. *upper witnesses*.

We show how to build a *graph of lower witnesses* for relation T, $G^T_<$. *A graph of upper witnesses*, $G^T_>$, can be constructed similarly. Formally, $G^T_< = (\hat{\Theta}^T, E, lab)$, where lab, is a labeling of edges $lab : E \rightarrow \boldsymbol{\alpha} \times \boldsymbol{\alpha} \times \boldsymbol{\beta}$. We will construct $G^T_<$ in steps, starting with $E = \emptyset$. During the construction we define a partial function $f : \hat{\Theta}^T \rightarrow A/\hat{T}$, associating with each enriched M-counting type a T-clique in $\mathfrak{A}$. When the whole construction is completed, f will be a total function. Additionally we use a marking of vertices. Initially all vertices are unmarked.

1. Choose any unmarked vertex $\hat{\theta}$ from $\hat{\Theta}^T$. Find a T-clique D in $\mathfrak{A}$ such that the enriched M-counting type of D is $\hat{\theta}$ and there is no T-clique D' of type $\hat{\theta}$ in $\mathfrak{A}$, such that $D' <^{\mathfrak{A}}_T D$. The existence of the desired clique D is implied by finiteness of $\mathfrak{A}$. Set $f(\hat{\theta}) = D$.
2. Repeat the following steps as long as possible:
   (a) Choose an unmarked $\hat{\theta} = (\theta, \mathcal{A}, \mathcal{B}) \in \hat{\Theta}^T$ for which f is defined, let $D = f(\hat{\theta})$.
   (b) For every conjunct of the form (vii): $\forall x(p(x) \rightarrow \exists y(Tyx \wedge \neg Txy \wedge \psi(x,y)))$, and every 1-type t appearing in θ, $t = \alpha(a)$ for some $a \in D$, such that $t \models p(x)$:
      i. Find $b \in A$, such that $\mathfrak{A} \models Tba \wedge \neg Tab \wedge \psi(a,b)$.
      ii. Find $c \in A$, such that $\alpha(b) = \alpha(c)$, $\mathfrak{A} \models Tca \wedge \neg Tac$ and there is no d such that $\alpha(d) = \alpha(c)$ and $\mathfrak{A} \models Tdc \wedge \neg Tcd$. Again, such a c exists since $\mathfrak{A}$ is finite. Please note, that it is not necessary that $\mathfrak{A} \models \psi(a,c)$.
      iii. Let D' be the T-clique of c, and $\hat{\theta}'$ be its enriched M-counting type. Set $f(\hat{\theta}') = D'$. Add to E an edge e from $\hat{\theta}$ to $\hat{\theta}'$, and set $lab(e) = (\alpha(a), \alpha(b), \beta(a,b))$.
   (c) Mark $\hat{\theta}$.
3. If there is an unmarked vertex in $G^T_<$ then go to 1.

We note two properties of graphs of lower witnesses, shared also by graphs of upper witnesses.

Fact 1. $G^T_<$ `is an acyclic graph.`

Proof. Assume to the contrary, that there exists a cycle $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_l, \hat{\theta}_1$. But then $f(\hat{\theta}_2) <^{\mathfrak{A}}_T f(\hat{\theta}_1)$, $f(\hat{\theta}_3) <^{\mathfrak{A}}_T f(\hat{\theta}_2)$, $f(\hat{\theta}_1) <^{\mathfrak{A}}_T f(\hat{\theta}_l)$. Thus, from transitivity of T, $f(\hat{\theta}_1) <^{\mathfrak{A}}_T f(\hat{\theta}_1)$, which is impossible. □

Fact 2. `Every path in` $G^T_<$ `has length at most` $|\boldsymbol{\alpha}|$.

Proof. Let $\hat{\theta}_1, \hat{\theta}_2, \ldots, \hat{\theta}_l$ form a path in $G^T_<$. Assume that $\hat{\theta}_i = (\theta_i, \mathcal{A}_i, \mathcal{B}_i)$. Our construction implies that $\mathcal{B}_1 \supseteq \mathcal{B}_2 \supseteq \ldots \supseteq \mathcal{B}_l$. In fact, the choice of c in step 2(b)ii., as a farthest element of the required 1-type, guarantees, that the above set containments are strict. Since $\mathcal{B}_1 \subseteq \boldsymbol{\alpha}$ we have the desired bound on l. $\square$

4.4 Model Construction

Universe. Let $K = 2|\boldsymbol{\alpha}| + 1$. Let m be the maximal degree of a vertex in all graphs of lower and upper witnesses. Note that K and m are exponential in $|\sigma|$. We define the universe of $\mathfrak{B}$ to be

$$B = \{0, \ldots, K-1\} \times \{0, \ldots, 3\} \times \{1, \ldots, m\} \times \dot{\bigcup_{T \in \sigma, \hat{\theta} \in \hat{\Theta}^T}} C^T_{\hat{\theta}}.$$

Thus, if we assume that σ consists only of symbols appearing in φ, $|B|$ is at most doubly exponential in $|\varphi|$. We can think that B consists of K rows and 4 columns, and that the intersection of each row and each column contains m copies of each $\mathfrak{C}^T_{\hat{\theta}}$. In fact, we glue row $K-1$ to row 0 obtaining thus a cylindrical surface.

Initially, we impose appropriate structures on copies of $\mathfrak{C}^T_{\hat{\theta}}$. For $a \in C^T_{\hat{\theta}}$ let us set $\alpha(i, j, l, a) := \alpha(a)$, and for $a, b \in C^T_{\hat{\theta}}$, set $\beta((i, j, l, a), (i, j, l, b)) := \beta(a, b)$.

General scheme of providing witnesses. Now we provide, for all elements, witnesses for all formulas with nonsymmetric transitive guards, i.e. formulas of the form (vi) and (vii). This is done in a regular manner. Elements from row i find their lower witnesses in the row $i+1 (\mathrm{mod}\ K)$, and their upper witnesses in the row $i-1 (\mathrm{mod}\ K)$. Elements from column j look for their lower witnesses in column $l(j)$ and for their upper witnesses in column $u(j)$, where $l, u : \{0, 1, 2, 3\} \to \{0, 1, 2, 3\}$ are defined as follows: $l(0) = l(2) = 0$, $l(1) = l(3) = 1$, $u(0) = u(3) = 3$, $u(1) = u(2) = 2$. This strategy guarantees two important properties. First, there will be no pair of elements a, b such that a use b as a lower witness and b use a as an upper witness (since both ud and du have no fixed points). Second, none of the cliques will be used as a source for both a lower witness and an upper witness (since u and d have disjoint images).

Example of providing witnesses. Let us consider explicitly the case of, say, providing lower T-witnesses for element $(5, 1, 3, a)$, where $a \in C^{T'}_{\hat{\theta}'}$. Let D be the T-clique of a in $\mathfrak{C}^{T'}_{\hat{\theta}'}$. Let $\hat{\theta} = enr(D)$. The outlined strategy says that lower witnesses should be looked for in row 6, column 1. Let $F_w = \{F_1, F_2, \ldots F_l\}$ be the set of edges in $G^T_<$ outgoing from the vertex $\hat{\theta}$. Note that $l \leq m$. Let F_i goes to $\hat{\theta}_i$, let $lab(F_i) = (\alpha_i, \alpha'_i, \beta)$. For each $i \leq m$, such that $\alpha_i = \alpha(a)$, choose an element a' in $\{6\} \times \{1\} \times \{i\} \times C^T_{\hat{\theta}_i}$ of type α'_i. Set $\beta(a, a') := \beta$.

In a similar way we provide lower and upper witnesses for all elements in $\mathfrak{B}$.

Transitive closure. During the step of providing witnesses we defined 2-types containing nonsymmetric transitive connections between some pairs of elements. We say that there exists a T-path from T-clique D' to a T-clique D in $\mathfrak{B}$ if there exists a sequence of T-cliques $D = D_0, D_1, \ldots D_{l-1}, D_l = D'$ such that for every pair D_i, D_{i+1} we defined a lower T-witness for an element from D_i in D_{i+1} or an upper T-witness for an element from D_{i+1} in D_i.

If there exists a T-path, for some transitive T, from D to D', then we make $D' <_T^{\mathfrak{B}} D$. For every $a' \in D'$, $a \in D$, if the 2-type for a' and a is not defined, then let t' be the 1-type of a' and t – the 1-type of a. We choose any 2-type $s(x, y)$ realized in $\mathfrak{A}$, such that $s(x, y) \models Txy \wedge \neg Tyx \wedge t'(x) \wedge t(y)$, and set $\beta(a', a) := s$.

The following claim shows that nothing bad can happen during this step. Part (i) implies, that finding a required $s(x, y)$ in $\mathfrak{A}$ is always possible. Part (ii) says, that we have not created a T-path from a T-clique D to the same clique D, i.e. we have not formed a T-cycle. Part (iii) says, that setting 2-types can be done without conflicts.

Claim (OK).

(i) Assume that, after the step of providing nonsymmetric witnesses, there is a T-path from D to D' in $\mathfrak{B}$. Let $a' \in D', a \in D, t' = \alpha(a'), t = \alpha(a)$. Then there exist elements $c', c \in A$ whose 1-types are t' and t, respectively, such that $\mathfrak{A} \models Tc'c \wedge \neg Tcc'$.

(ii) Let $D_0, D_1, \ldots, D_l$ be a T-path from D_0 to D_l in $\mathfrak{B}$. Then $l \leq 2|\boldsymbol{\alpha}|$, i.e. $l < K$.

(iii) If an element $a \in B$ belongs to a T-clique D, $b \in B$ belongs to a T-clique D' and there is a T-path from D to D', then there is no pair C, C' of T'-cliques with $T \neq T'$, such that $a \in C$, $b \in C'$ and there is a T'-path from C to C', or a T'-path from C' to C.

We skip the proof of this claim. We note only that in the proof of part (iii) we use the fact that both $u^i d^j$ and $d^i u^j$ have no fixed points (for $i, j > 0$).

Remaining 2-types. For every pair of elements $a, a' \in B$ for which we have not defined a 2-type yet, we set it to be the unique binary free type containing 1-types of a and a'. This step completes the construction of the structure $\mathfrak{B}$.

4.5 Complexity

The construction from the previous section allows us to state the following theorems.

Theorem 3. *Every finitely satisfiable* $\mathrm{GF}^2+\mathrm{TG}$ *formula* φ *has a model of size at most doubly exponential in* $|\varphi|$.

Theorem 4. *The finite satisfiability problem for* $\mathrm{GF}^2+\mathrm{TG}$ *is decidable in* 2NEXPTIME.

The best lower complexity bound we know is 2EXPTIME and can be obtained in the presence of just one transitive symbol, exactly as in [11], where (unrestricted) satisfiability problem for $\mathrm{GF}^2+\mathrm{TG}$ was concerned.

References

1. Andréka, H., van Benthem, J., Németi, I.: Modal Languages and Bounded Fragments of Predicate Logic, ILLC Research Report, 1996. Journal ver.: J. Philos. Logic 27(3), 217–274 (1998)
2. Bojańczyk, M.: Two–way alternating automata and finite models. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 833–844. Springer, Heidelberg (2002)

3. Calvanese, D.: Unrestricted and finite model reasoning in class-based representation formalisms, Ph.D. thesis, Dipartimento di Informatica e Sistemistica, Universitaa di Roma ”La Sapienza” (1996)
4. Grädel, E.: On the Restraining Power of Guards. Journal of Symbolic Logic 64, 1719–1742 (1999)
5. Grädel, E.: Decision procedures for guarded logics. In: Ohsuga, S., Raś, Z.W. (eds.) ISMIS 2000. LNCS (LNAI), vol. 1932, pp. 31–51. Springer, Heidelberg (2000)
6. Grädel, E., Kolaitis, P., Vardi, M.: On the Decision Problem for Two-Variable First Order Logic. Bulletin of Symbolic Logic 3(1), 53–96 (1997)
7. Grädel, E., Walukiewicz, I.: Guarded fixed point logic. In: Fourteenth Annual IEEE Symposium on Logic in Computer Science, pp. 45–54. IEEE Computer Society Press, Los Alamitos (1999)
8. Hodkinson, I.: Loosely guarded fragment of first-order logic has the finite model property. Studia Logica 70, 205–240 (2002)
9. Hodkinson, I., Otto, M.: Finite conformal hypergraph covers, with two applications. Bull. Symbolic Logic 9, 387–405 (2003)
10. Khachiyan, L.G.: A polynomial algorithm in linear programming. Soviet Mathematics Doklady 20, 191–194 (1979)
11. Kieroński, E.: The Two-Variable Guarded Fragment with Transitive Guards is 2EXPTIME-Hard. In: Gordon, A.D. (ed.) ETAPS 2003 and FOSSACS 2003. LNCS, vol. 2620, pp. 299–312. Springer, Heidelberg (2003)
12. Kieroński, E.: Results on the Guarded Fragment with Equivalence or Transitive Relations. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 309–324. Springer, Heidelberg (2005)
13. Kieroński, E., Otto, M.: Small Substructures and Decidability Issues for First-Order Logic with Two Variables. In: Proc. of 20-th IEEE Symp. on Logic in Computer Science (LICS), pp. 448–457. IEEE Computer Society Press, Los Alamitos (2005)
14. Papadimitriou, Ch.: On the Complexity of Integer Programming. Journal of ACM 28(4), 765–786 (1981)
15. Pratt-Hartmann, I.: Complexity of the guarded two-variable fragment with counting quantifiers. Journal of Logic and Computation 17(1), 133–155 (2007)
16. Szwast, W., Tendera, L.: On the decision problem for the guarded fragment with transitivity. In: Proc. 16th IEEE Symposium on Logic in Ccomputer Science, pp. 147–156. IEEE Computer Society Press, Los Alamitos (2001)
17. Szwast, W., Tendera, L.: The Guarded Fragment with Transitive Guards. Annals of Pure and Applied Logic 128, 227–276 (2004)
18. Szwast, W., Tendera, L.: On the Finite Satisfiability Problem For the Guarded Fragment with Transitivity. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 307–321. Springer, Heidelberg (2005)
19. van Benthem, J.: Dynamics bits and pieces. ILLC Research Report LP-97-01, University of Amsterdam (1997)

Data Complexity in the $\mathcal{EL}$ Family of Description Logics

Adila Krisnadhi[1] and Carsten Lutz[2]

[1] Faculty of Computer Science, University of Indonesia
`adila@cs.ui.ac.id`
[2] Institute for Theoretical Computer Science, TU Dresden, Germany
`lutz@tcs.inf.tu-dresden.de`

Abstract. We study the data complexity of instance checking and conjunctive query answering in the $\mathcal{EL}$ family of description logics, with a particular emphasis on the boundary of tractability. We identify a large number of intractable extensions of $\mathcal{EL}$, but also show that in $\mathcal{ELI}^f$, the extension of $\mathcal{EL}$ with inverse roles and global functionality, conjunctive query answering is tractable regarding data complexity. In contrast, already instance checking in $\mathcal{EL}$ extended with only inverse roles or global functionality is ExpTime-complete regarding combined complexity.

1 Introduction

In recent years, lightweight description logics (DLs) have experienced increased interest because they admit highly efficient reasoning on large-scale ontologies. Most prominently, this is witnessed by the ongoing research on the DL-Lite and $\mathcal{EL}$ families of DLs (see also [12,16] for other examples). The main application of $\mathcal{EL}$ and its relatives is as an ontology language [6,2,4]. In particular, the DL $\mathcal{EL}^{++}$ proposed in [2] admits tractable reasoning while still providing sufficient expressive power to represent, for example, life-science ontologies. In contrast, the DL-Lite family of DLs is specifically tailored towards applications with a massive amount of instance data [9,7,8,1]. In such applications, instance checking and conjunctive query answering are the most relevant reasoning services and should thus be computationally cheap, preferably tractable. When determining the computational complexity of these tasks for a given DL, it is often realistic to consider *data complexity*, where the size of the input is measured only in terms of the ABox (which represents instance data), but not in terms of the TBox (which corresponds to the schema) and the query, as the latter both tend to be small compared to the former. This is in contrast to *combined complexity*, where also the size of the TBox and query are taken into account.

The aim of this paper is to study the $\mathcal{EL}$ family of DLs in the light of data intensive applications. To this end, we analyze the data complexity of instance checking and conjunctive query answering in extensions of $\mathcal{EL}$. For the DL-Lite family, such an investigation has been carried out e.g. in [8,1], with complexities ranging from LogSpace-complete to coNP-complete. It follows from the results in [8] that we cannot expect the data complexity to be below PTime for members of the $\mathcal{EL}$ family (at least in the presence of so-called general TBoxes, i.e., sets of GCIs). The reason is that, in a crucial aspect, DL-Lite is even more lightweight than $\mathcal{EL}$: in contrast to $\mathcal{EL}$, DL-Lite does not

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 333–347, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Table 1. Syntax and semantics of relevant DL constructors

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
sink restriction	$\forall r.\bot$	$\{x \mid \neg\exists y : (x,y) \in r^{\mathcal{I}}\}$
value restriction	$\forall r.C$	$\{x \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
at-least restriction	$(\geqslant k\ r)$	$\{x \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \geq k\}$
at-most restriction	$(\leqslant k\ r)$	$\{x \mid \#\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in r^{\mathcal{I}}\} \leq k\}$
inverse roles	$\exists r^{-}.C$	$\{x \mid \exists y : (y,x) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
role negation	$\exists \neg r.C$	$\{x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \notin r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
role union	$\exists r \cup s.C$	$\{x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \cup s^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
transitive closure	$\exists r^{+}.C$	$\{x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in (r^{\mathcal{I}})^{+} \wedge y \in C^{\mathcal{I}}\}$

allow for qualified existential (neither universal) restrictions, and thus the interaction between different domain elements is very limited. When analyzing the data complexity of instance checking and conjunctive query answering in $\mathcal{EL}$ and its extensions, we therefore concentrate on mapping out the boundary of tractability.

We consider a wide range of extensions of $\mathcal{EL}$, and analyze the data complexity of the mentioned tasks with acyclic TBoxes and with general TBoxes. When selecting extensions of $\mathcal{EL}$, we focus on DLs for which instance checking has been proved *intractable* regarding combined complexity in [2]. We show that, in most of these extensions, instance checking is also intractable regarding data complexity. The notable exceptions are $\mathcal{EL}$ extended with globally functional roles and $\mathcal{EL}$ extended with inverse roles. It is shown in [3] that instance checking in these DLs is ExpTime-complete regarding combined complexity. On the other hand, it follows from results in [12] that instance checking is tractable regarding data complexity in $\mathcal{ELI}^f$, the extension of $\mathcal{EL}$ with both globally functional and inverse roles. In this paper, we extend this result to conjunctive query answering in $\mathcal{ELI}^f$, and show that this problem is still tractable regarding data complexity.

2 Preliminaries

In DLs, *concepts* are inductively defined with the help of a set of *constructors*, starting with a set $\mathsf{N_C}$ of *concept names* and a set $\mathsf{N_R}$ of *role names*. In $\mathcal{EL}$, concepts are formed using the three topmost constructors in Table 1. There and in general, we use r and s to denote role names, A and B to denote concept names, and C, D to denote concepts. The additional constructors shown in Table 1 give rise to extensions of $\mathcal{EL}$. We use canonical names to refer to such extensions, writing e.g. $\mathcal{EL}^{\forall r.\bot}$ for $\mathcal{EL}$ extended with sink restrictions and $\mathcal{EL}^{C \sqcup D}$ for $\mathcal{EL}$ extended with disjunction. Since we perform a very fine grained analysis, $\mathcal{EL}^{(\leq kr)}$ means the extension of $\mathcal{EL}$ with $(\leq k\, r)$ for some *fixed* $k \geq 0$ (but not for some fixed r).

In DLs, TBoxes are used to represent general knowledge about an application domain, and thus play the role of an ontology. We introduce two different forms of TBoxes. An *acyclic TBox* $\mathcal{T}$ is a finite set of concept equations $A \doteq C$ such that the left-hand sides are unique and there are no cycles, i.e., if $\{A_0 \doteq C_0, \ldots, A_{n-1} \doteq C_{n-1}\} \subseteq \mathcal{T}$ then for some $i \leq n$, A_i does not occur in C_{i+1} where $A_n := A_0$ and $C_n := C_0$. A *general TBox* is a finite set of concept inclusions $C \sqsubseteq D$ (often called *GCIs*). Every concept equation $A \doteq C$ can be written as two inclusions $A \sqsubseteq C$ and $C \sqsubseteq A$, and thus general TBoxes subsume acyclic ones. ABoxes are used to represent instance data. Let $\mathsf{N_I}$ be a set of *individual names*. An *ABox* is a finite set of expressions $A(a)$ and $r(a, b)$, where a and b are from $\mathsf{N_I}$ (here and in what follows). Observe that we disallow complex concepts in the ABox, as usual when studying data complexity.

The semantics of $\mathcal{EL}$ and its extensions is defined in terms of *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set and the *interpretation function* $\cdot^{\mathcal{I}}$ maps each concept name $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name $r \in \mathsf{N_R}$ to a binary relation $r^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, and each individual name $a \in \mathsf{N_I}$ to a domain element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to complex concepts is inductively defined as shown in the third column of Table 1, where $\#S$ denotes the cardinality of the set S. An interpretation $\mathcal{I}$ *satisfies* an equation $A \doteq C$ iff $A^{\mathcal{I}} = C^{\mathcal{I}}$, an inclusion $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, an assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and an assertion $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. It is a *model* of a TBox $\mathcal{T}$ (ABox $\mathcal{A}$) if it satsfies all equations/inclusions in $\mathcal{T}$ (assertions in $\mathcal{A}$).

We will also consider $\mathcal{EL}^{kf}$, the extension of $\mathcal{EL}$ with k-functional roles, i.e., roles for which every domain element can have at most k successors. In $\mathcal{EL}^{kf}$, there are no additional concept constructors that may be used to build up complex concepts. Instead, a new kind of expression $\top \sqsubseteq (\leq k\ r)$ is allowed in the TBox. These expressions can be understood as *global* at-most restrictions, in contrast to the local at-most restrictions shown in Table 1. An interpretation $\mathcal{I}$ satisfies $\top \sqsubseteq (\leq k\ r)$ if $|\{e \mid (d, e) \in r^{\mathcal{I}}\}| \leq k$ for all $d \in \Delta^{\mathcal{I}}$. Instead of 1-functional roles, we will speak of functional roles as usual.

The two main inference problems considered in this paper are instance checking and conjunctive query entailment. An individual name a is an *instance* of a concept C w.r.t. an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ (written $\mathcal{A}, \mathcal{T} \models C(a)$) iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ in all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$. The instance problem is to decide, given a, C, $\mathcal{A}$ and $\mathcal{T}$, whether $\mathcal{A}, \mathcal{T} \models C(a)$.

Conjunctive query entailment is the decision problem corresponding to conjunctive query answering, which is a search problem. A *conjunctive query* is a set q of atoms $C(v)$ and $r(u, v)$, where u, v are variables. We use $\mathsf{Var}(q)$ to denote the variables used in q. If $\mathcal{I}$ is an interpretation and π is a mapping from $\mathsf{Var}(q)$ to $\Delta^{\mathcal{I}}$, we write $\mathcal{I} \models^{\pi} C(v)$ if $\pi(v) \in C^{\mathcal{I}}$, $\mathcal{I} \models^{\pi} r(u, v)$ if $(\pi(u), \pi(v)) \in r^{\mathcal{I}}$, $\mathcal{I} \models^{\pi} q$ if $\mathcal{I} \models^{\pi} \alpha$ for all $\alpha \in q$, and $\mathcal{I} \models q$ if $\mathcal{I} \models^{\pi} q$ for some π. Finally, $\mathcal{A}, \mathcal{T} \models q$ means that for all models $\mathcal{I}$ of the ABox $\mathcal{A}$ and the TBox $\mathcal{T}$, we have $\mathcal{I} \models q$. Now, *conjunctive query entailment* is to decide given $\mathcal{A}$, $\mathcal{T}$, and q, whether $\mathcal{A}, \mathcal{T} \models q$.

It is not hard to see that, in $\mathcal{EL}$, instance checking is a special case of conjunctive query entailment, as every $\mathcal{EL}$-concept C can be converted into a tree-shaped query. Note that we do not partition the variables in a conjunctive query into answer variables and existentially quantified variables as usual. Since we are dealing with query entailment instead of query answering, this distinction is meaningless. Also observe that we do not allow individual names in conjunctive queries in place of variables. It is well-known that individual names in the query can be simulated by concept names with only a linear blowup of the input, see for example [10] for details.

The last preliminary worth mentioning is the *unique name assumption (UNA)*, which requires that for all $a, b \in \mathsf{N_I}$ with $a \neq b$, we have $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Most of our results do not depend on the UNA. Whenever they do, we will state explicitly whether the UNA is adopted or not.

3 Lower Bounds

We show that, in almost all extensions of $\mathcal{EL}$ introduced in Section 2, instance checking is co-NP-hard regarding data complexity. All our lower bounds assume only acyclic TBoxes.

For the sake of completeness, we note that the case where there is no TBox is not very interesting: because only concept *names* are admitted in the ABox, the additional concept constructors can then only occur in the query (which is a concept in the case of instance checking and a conjunctive query otherwise). In most cases (such as $\mathcal{EL}^{(\neg)}$ and $\mathcal{EL}^{\forall r.C}$), this means that no query which contains the additional constructor is entailed by any ABox. Thus, there is a trivial reduction to query answering in basic $\mathcal{EL}$. In other cases such as $\mathcal{EL}^{C \sqcup D}$, it is easily shown that conjunctive query containment is tractable regarding data complexity. A notable exception is $\mathcal{EL}^{kf}$, $k \geq 2$, for which instance checking is coNP-complete already without TBoxes (as is proved below).

3.1 Basic Cases

In [19], Schaerf proves that instance checking in $\mathcal{EL}^{\neg A}$ is co-NP-hard regarding data complexity. He uses a reduction from a variant of SAT that he calls 2+2-SAT. Our lower bounds for extensions of $\mathcal{EL}$ are obtained by variations of Schaerf's reduction. For this reason, we start with repeating the original reduction of Schaerf. Before we go into detail, a remark on $\mathcal{EL}^{\neg A}$ is in order. In this extension of $\mathcal{EL}$, the application of negation is restricted to concept names. However, full negation can easily be recovered using acyclic TBoxes: instead of writing $\neg C$, we may write $\neg A$ and add a concept equation $A \doteq C$, with A a fresh concept name. Thus, we restrict the use of negation even further, namely to concept names that do not occur on the left-hand side of any concept equation in the (acyclic) TBox. As we shall see shortly, the TBoxes required for our lower bound are actually of very simple form.

A *2+2 clause* is of the form $(p_1 \vee p_2 \vee \neg n_1 \vee \neg n_2)$, where each of p_1, p_2, n_1, n_2 is a propositional letter or a truth constant 1, 0. A *2+2 formula* is a finite conjunction of 2+2 clauses. Now, 2+2-SAT is the problem of deciding whether a given 2+2 formula is satisfiable. It is shown in [19] that 2+2-SAT is NP-complete.

Let $\varphi = c_0 \wedge \cdots \wedge c_{n-1}$ be a 2+2-formula in m propositional letters $q_0, \ldots, q_{m-1}$. Let $c_i = p_{i,1} \vee p_{i,2} \vee \neg n_{i,1} \vee \neg n_{i,2}$ for all $i < n$. We use f, the propositional letters $q_0, \ldots, q_{m-1}$, the truth constants $1, 0$, and the clauses $c_0, \ldots, c_{n-1}$ as individual names.

Define the TBox $\mathcal{T}$ as $\{\overline{A} \doteq \neg A\}$ and the ABox $\mathcal{A}_\varphi$ as follows, where c, p_1, p_2, n_1, and n_2 are role names:

$$\begin{aligned}\mathcal{A}_\varphi := &\{A(1), \overline{A}(0)\} \cup \\ &\{c(f, c_0), \ldots, c(f, c_{n-1})\} \cup \\ &\textstyle\bigcup_{i<n}\{p_1(c_i, p_{i,1}), p_2(c_i, p_{i,2}), n_1(c_i, n_{i,1}), n_2(c_i, n_{i,2})\}\end{aligned}$$

It should be obvious that $\mathcal{A}_\varphi$ is a straightforward representation of φ. Models of $\mathcal{A}_\varphi$ and $\mathcal{T}$ represent truth assignments for φ by way of setting q_i to true if $q_i \in A^\mathcal{I}$ and to false if $q_i \in \overline{A}^\mathcal{I}$. Since $\mathcal{I}$ is a model of $\mathcal{T}$, this truth assignment is well-defined. Set $C := \exists c.(\exists p_1.\overline{A} \sqcap \exists p_2.\overline{A} \sqcap \exists n_1.A \sqcap \exists n_2.A)$. Intuitively, C expresses that φ is not satisfied, i.e., there is a clause in which the two positive literals and the two negative literals are all false. It is not hard to show the following.

Lemma 1 (Schaerf). *$\mathcal{A}_\varphi, \mathcal{T} \not\models C(f)$ iff φ is satisfiable.*

Thus, instance checking in $\mathcal{EL}^{\neg A}$ w.r.t. acyclic TBoxes is co-NP-hard regarding data complexity.

This reduction can easily be adapted to $\mathcal{EL}^{\forall r.\bot}$. In all interpretations $\mathcal{I}$, $\exists r.\top$ and $\forall r.\bot$ partition the domain $\Delta^\mathcal{I}$ and can thus be used to simulate the concept name A and its negation $\neg A$ in the original reduction. We can thus simply replace the TBox $\mathcal{T}$ with $\mathcal{T}' := \{A \doteq \exists r.\top, \overline{A} \doteq \forall r.\bot\}$.

In some extensions of $\mathcal{EL}$, we only find concepts that cover the domain, but not necessarily partition it. An example is $\mathcal{EL}^{(\leq kr)}$, $k \geq 1$, in which $\exists r.\top$ and $(\leq k\, r)$ provide a covering (for $k = 0$, observe that $(\leq k\, r)$ is equivalent to $\forall r.\bot$). Interestingly, this does not pose a problem for the reduction. In the case of $\mathcal{EL}^{(\leq kr)}$, we use the TBox $\mathcal{T} := \{A \doteq \exists r.\top, \overline{A} \doteq (\leq k\, r)\}$, and the ABox $\mathcal{A}_\varphi$ as well as the query concept C remain unchanged. Let us show that

Lemma 2. *$\mathcal{A}_\varphi, \mathcal{T} \not\models C(f)$ iff φ is satisfiable.*

Proof. "if". This direction is as in the proof of Lemma 1. Let t be a truth assignment satisfying φ. Define an interpretation $\mathcal{I}$ as follows:

$$\begin{aligned}\Delta^\mathcal{I} &:= \{f, c_0, \ldots, c_{n-1}, q_0, \ldots, q_{m-1}, 0, 1, d\} \\ c^\mathcal{I} &:= \{(f, c_0), \ldots, (f, c_{n-1})\} \\ p_j^\mathcal{I} &:= \{(c_0, p_{0,j}), \ldots, (c_{n-1}, p_{n-1,j})\} \\ n_j^\mathcal{I} &:= \{(c_0, n_{0,j}), \ldots, (c_{n-1}, n_{n-1,j})\} \\ A^\mathcal{I} &:= \{1\} \cup \{q_i \mid i < m \text{ and } t(q_i) = 1\} \\ \overline{A}^\mathcal{I} &:= \Delta^\mathcal{I} \setminus A^\mathcal{I} \\ r^\mathcal{I} &:= \{(e, d) \mid e \in A^\mathcal{I}\}\end{aligned}$$

All individual names are interpreted as themselves. It is not hard to verify that $\mathcal{I}$ is a model of $\mathcal{A}_\varphi$ and $\mathcal{T}$, and that $f \notin C^\mathcal{I}$.

"only if". Here we need to deal with the non-disjointness of $\exists r.\top$ and $(\leq k\, r)$. Let $\mathcal{I}$ be a model of $\mathcal{A}_\varphi$ and $\mathcal{T}$ such that $f \notin C^\mathcal{I}$. Define a truth assignment t by choosing for each propositional letter q_i, a truth value $t(q_i)$ such that $t(q_i) = 1$ implies $q_i^\mathcal{I} \in A$

and $t(q_i) = 0$ implies $q_i^{\mathcal{I}} \in \overline{A}$. Such a truth assignment exists since A and $\overline{A}$ cover the domain. However, it is not necessarily unique since A and $\overline{A}$ need not be disjoint. To show that t satisfies φ, assume that it does not. Then there is a clause $c_i = (p_{i,1} \vee p_{i,2} \vee \neg n_{i,1} \vee \neg n_{i,2})$ that is not satisfied by t. By definition of t, $p_{i,1}, p_{i,2} \in \overline{A}^{\mathcal{I}}$ and $n_{i,1}, n_{i,2} \in A^{\mathcal{I}}$. Thus $c_i^{\mathcal{I}} \in (\exists p_1.\overline{A} \sqcap \exists p_2.\overline{A} \sqcap \exists n_1.A \sqcap \exists n_2.A)^{\mathcal{I}}$ and we get $f \in C^{\mathcal{I}}$, which is a contradiction. ❑

The cases $\mathcal{EL}^{\forall r.C}$ and $\mathcal{EL}^{\exists\neg r.C}$ can be treated similarly because a covering of the domain can be achieved by choosing the concepts $\exists r.\top$ and $\forall r.X$ in the case of $\mathcal{EL}^{\forall r.C}$, and $\exists r.\top$ and $\exists\neg r.\top$ in the case of $\mathcal{EL}^{\exists\neg r.C}$. In the case, $\mathcal{EL}^{C\sqcup D}$, we use a TBox $\mathcal{T}' := \{V \doteq X \sqcup Y\}$. In all models of $\mathcal{T}'$, the extension of V is covered by the concepts X and Y. Thus, we can use the above ABox $\mathcal{A}_\varphi$, add $V(q_i)$ for all $i < m$, and use the TBox $\mathcal{T} := \mathcal{T}' \cup \{A \doteq X, \overline{A} \doteq Y\}$ and the same query concept C as above. The case $\mathcal{EL}^{\exists r^+.C}$ is quite similar. In all models of the TBox $\mathcal{T}' := \{V \doteq \exists r^+.C\}$, the extension of V is covered by the concepts $\exists r.C$ and $\exists r.\exists r^+.C$. Thus, we can use the same ABox and query concept as for $\mathcal{EL}^{C\sqcup D}$, together with the TBox $\mathcal{T} := \mathcal{T}' \cup \{A \doteq \exists r.C, \overline{A} \doteq \exists r.\exists r^+.C\}$.

Theorem 1. *For the following, instance checking w.r.t. acyclic TBoxes is co-*NP*-hard regarding data complexity:* $\mathcal{EL}^{\neg A}$, $\mathcal{EL}^{\forall r.\bot}$, $\mathcal{EL}^{\forall r.C}$, $\mathcal{EL}^{\exists\neg r.C}$, $\mathcal{EL}^{C\sqcup D}$, $\mathcal{EL}^{\exists r^+.C}$, *and* $\mathcal{EL}^{(\leq k r)}$ *for all* $k \geq 0$.

For $\mathcal{EL}^{\forall r.\bot}$, $\mathcal{EL}^{\forall r.C}$, and $\mathcal{EL}^{C\sqcup D}$, co-NP-hardness of conjunctive query containment w.r.t. general TBoxes has been established in [8]. It seems likely that the proofs (which are not given in detail) actually apply to instance checking and acyclic TBoxes.

3.2 Cases that Depend on the UNA

The results in the previous subsection are independent of whether or not the UNA is adopted. In the following, we consider some cases that depend on the (non-)UNA, starting with $\mathcal{EL}^{(\geq k\, r)}$.

In $\mathcal{EL}^{(\geq k\, r)}$, $k \geq 2$, it does not seem possible to find two concepts that a priori cover the domain and can be used to represent truth values in truth assignments. However, if we add slightly more structure to the ABox, such concepts can be found. We treat only the case $k = 3$ explicitly, but it easily generalizes to other values of k as long as $k \geq 2$. Consider the following auxiliary ABox, also shown in Figure 1.

$$\mathcal{A} = \{r(a, b_1), r(a, b_2), r(a, b_3), r(b_1, b_2), r(b_2, b_3), r(b_1, b_3)\}.$$

Without the UNA, there are two cases for models of $\mathcal{A}$: either two of b_1, b_2, b_3 identify the same domain element or they do not. In the first case, a satisfies $\exists r^4.\top$, where $\exists r^4$ denotes the four-fold nesting of $\exists r$. In the second case, a satisfies $(\geq 3\, r)$. It follows that we can reduce satisfiability of 2+2 formulas using a reduction very similar to the one for $\mathcal{EL}^{(\leq k\, r)}$. The main differences are that (i) a copy of $\mathcal{A}$ is plugged in for each q_i, with a replaced by q_i and (ii) we use the TBox $\mathcal{T} := \{A \doteq \exists r^4.\top, \overline{A} \doteq (\geq 3\, r)\}$.

Unlike the previous results, this lower bound clearly depends on the fact that the UNA is not adopted. If the UNA is adopted, we can prove the same result using a

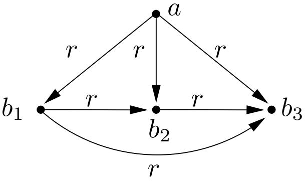


Fig. 1. Auxiliary ABox $\mathcal{A}$ for $\mathcal{EL}^{(\geq 3\,r)}$ without UNA

different auxiliary ABox. Again, we only treat the case $k = 3$, which easily generalizes. Let

$$\mathcal{A}' = \{r(a, b_1), r(a, b_2), V(a), A(b_1), A(b_2)\}$$

and consider the TBox $\mathcal{T}' = \{V \doteq \exists r.B\}$. In every model $\mathcal{I}$ of $\mathcal{A}'$ and $\mathcal{T}'$, there is a $d \in B^{\mathcal{I}}$ such that $(a^{\mathcal{I}}, d) \in r^{\mathcal{I}}$. We can distinguish two cases: if $d = b_i$ for some $i \in \{1, 2\}$, then a satisfies $\exists r.(A \sqcap B)$. Otherwise, a satisfies $(\geq 3\ r)$. We can now continue the reduction as in the previous cases. Start with the ABox $\mathcal{A}_\varphi$ from the reduction for $\mathcal{EL}^{\neg A}$, add $V(q_i)$ for all $i < m$ and a copy of $\mathcal{A}'$ for each q_i, with a replaced by q_i. Then use the TBox $\mathcal{T} = \mathcal{T}' \cup \{A \doteq \exists r.(A \sqcap B), \overline{A} \doteq (\geq 3\, r)\}$ and the original query concept C. Observe that this reduction does not work without the UNA.

Theorem 2. *For $\mathcal{EL}^{(\geq k\, r)}$ with $k \geq 2$, instance checking w.r.t. acyclic TBoxes is co-*NP*-hard regarding data complexity, both with and without the UNA.*

Another case that depends on the (non-)UNA is $\mathcal{EL}^{kf}$ with $k \geq 2$. We start with proving coNP-hardness provided that the UNA is not adopted. For the case $\mathcal{EL}^{1f}$, we will prove in Section 4 that instance checking (and even conjunctive query entailment) is tractable regarding data complexity, with or without the UNA. For simplicity, we only treat the case $\mathcal{EL}^{2f}$ explicitly. It is easy to generalize our argument to larger values of k. Like in $\mathcal{EL}^{(\geq 3r)}$, in $\mathcal{EL}^{2f}$ it does not seem possible to find two concepts that cover the domain without providing additional structure via an ABox. Set

$$\mathcal{A}'' = \{r(a, b_1), r(a, b_2), r(a, b_3), r(b_1, b_2), A(b_1), A(b_2), B(b_3)\}\}.$$

where r is 2-functional and thus at least two of b_1, b_2, b_3 have to identify the same domain element. A graphical representation is given in Figure 2. Regarding models of $\mathcal{A}''$, we can distinguish two cases: either b_3 is identified with b_1 or b_2, then a satisfies $\exists r.(A \sqcap B)$. Or b_1 and b_2 are identified, then a satisfies $\exists r^3.\top$, where $\exists r^3$ denotes the three-fold nesting of $\exists r$. It follows that we can reduce satisfiability of 2+2 formulas using a reduction very similar to that for $\mathcal{EL}^{(\geq 3r)}$ above. Observe that we do not need a TBox at all to make this work. We take the original ABox $\mathcal{A}_\varphi$ defined for $\mathcal{EL}^{\neg A}$, add a copy of $\mathcal{A}''$ for each q_i with a replaced by q_i, and replace $A(1)$ with $\{r(1, e), A(e), B(e)\}$ and $\overline{A}(0)$ with $\{r(0, e_0), r(e_0, e_1), r(e_1, e_2)\}$. Thus, 1 satisfies $\exists r.(A \sqcap B)$ (representing true) and 0 satisfies $\exists r^3.\top$ (representing false). It remains to modify the query concept to $C' := \exists c.(\exists p_1.\exists r^3.\top \sqcap \exists p_2.\exists r^3.\top \sqcap \exists n_1.\exists r.(A \sqcap B) \sqcap \exists n_2.\exists r.(A \sqcap B))$.

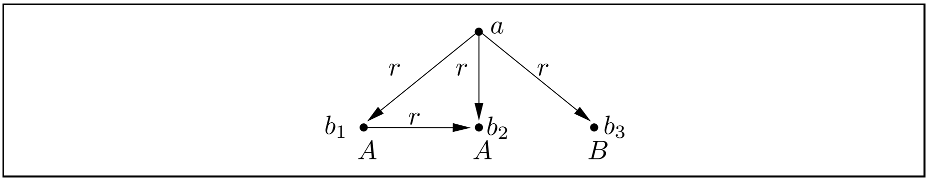


Fig. 2. Auxiliary ABox $\mathcal{A}''$ for $\mathcal{EL}^{2f}$ without UNA

With the UNA and without TBoxes, instance checking in $\mathcal{EL}^{kf}$, $k \geq 2$ is tractable regarding data complexity. The same holds for conjunctive query answering. In a nutshell, a polytime algorithm is obtained by considering the input ABox as a (complete) description of an interpretation and then checking all possible matches of the conjunctive query. A special case that has to be taken into account are inconsistent ABoxes such as those containing $\{r(a, b_1), r(a, b_2), r(a, b_3)\}$ for a 2-functional role r and with the b_i mutually distinct. Such inconsistencies are easily detected. If found, the algorithm returns "yes" because an inconsistent ABox entails every consequence.

If we add acyclic TBoxes, instance checking in $\mathcal{EL}^{kf}$, $k \geq 2$, becomes co-NP-hard also with the UNA. We only treat the case $k = 3$, but our arguments generalize. As in the case of $\mathcal{EL}^{2f}$ without UNA, we have to give additional structure to the ABox. Consider the TBox $\mathcal{T}'' = \{V \doteq \exists r.B\}$ and the ABox

$$\mathcal{A}''' = \{V(a), r(a, b_1), r(a, b_2), r(a, b_3), s(a, b_1), s'(a, b_2), s'(a, b_3)\}.$$

with r a 3-functional role. Then a satisfies $\exists r.B$ in all models $\mathcal{I}$ of $\mathcal{A}'''$ and $\mathcal{T}''$. Because of the UNA, we can distinguish two cases: either b_1 satisfies B or one of b_2, b_3 does. In the first case, a satisfies $\exists s.B$ and in the second case, it satisfies $\exists s'.B$. We can then continue the reduction as in the previous cases.

Theorem 3. *For $\mathcal{EL}^{kf}$ with $k \geq 2$, instance checking is*

- *tractable w.r.t. the empty TBox and with UNA;*
- *co-*NP*-hard in the following cases: (i) w.r.t. the empty TBox and without UNA, and (ii) w.r.t. acyclic TBoxes and with UNA.*

4 Upper Bound

The only remaining extensions of $\mathcal{EL}$ introduced in Section 2 are $\mathcal{EL}^{\exists r^-.C}$ and $\mathcal{EL}^{1f}$. For both of them, instance checking w.r.t. general TBoxes is ExpTime-complete regarding combined complexity [2]. In this section, we consider the union $\mathcal{ELI}^f$ of $\mathcal{EL}^{\exists r^-.C}$ and $\mathcal{EL}^{1f}$, i.e., the extension of $\mathcal{EL}$ with both inverse roles and globally functional roles. It follows from the results on Horn-$\mathcal{SHIQ}$ in [12] that instance checking in $\mathcal{ELI}^f$ w.r.t. general TBoxes is tractable regarding data complexity. A direct proof can be found in [14]. Here, we show that even conjunctive query answering in $\mathcal{ELI}^f$ is tractable regarding data complexity.

An *inverse role* is an expression r^- with r a role name. The interpretation of an inverse role is $(r^-)^\mathcal{I} = \{(e,d) \mid (d,e) \in r^\mathcal{I}\}$. In $\mathcal{ELI}^f$, roles and also their inverses can be declared functional using statements $\top \sqsubseteq (\leq 1\, r)$ in the TBox. For conveniently dealing with inverse roles, we use the following convention: if $r = s^-$ (with s a role name), then r^- denotes s. Observe that w.l.o.g., we do not admit inverse roles in the ABox and the query.

As a preliminary, we assume that TBoxes are in a normal form, i.e., all concept inclusions are of one of the following forms, where A, A_1, A_2, and B are concept names or $\top$ and r is a role name or an inverse role:

$$\begin{array}{ccc} A \sqsubseteq B, & A \sqsubseteq \exists r.B, & \top \sqsubseteq (\leq 1\, r) \\ A_1 \sqcap A_2 \sqsubseteq B, & \exists r.A \sqsubseteq B & \end{array}$$

Let $\mathcal{T}$ be a TBox. $\mathcal{T}$ can be converted into normal form $\mathcal{T}'$ in polytime, by introducing additional concept names. See [2] for more details. Moreover, it is not too difficult to see that for every ABox $\mathcal{A}$ and conjunctive query q not using any of the concept names that occur in $\mathcal{T}'$ but not in $\mathcal{T}$, we have $\mathcal{A}, \mathcal{T} \models q$ iff $\mathcal{A}', \mathcal{T}' \models q$.

Two other (standard) assumptions that we make w.l.o.g. is that (i) in all atoms $C(v)$ in a conjunctive query q, C is a concept name; and (ii) conjunctive queries are connected, i.e., for all $u, v \in \mathsf{Var}(q)$, there are atoms $r(u_0, u_1), \ldots, r(u_{n-1}, u_n) \in q$, $n \geq 0$, such that $u = u_0$ and $v = u_n$. It is easy to achieve (i) by replacing $C(v)$ with $A(v)$ and adding $A \doteq C$ to the TBox, with A a fresh concept name. Regarding (ii), it is well-known that entailment of non-connected queries can easily (and polynomially) be reduced to entailment of connected queries: if q is a non-connected query, then $\mathcal{A}, \mathcal{T} \models q$ iff $\mathcal{A}, \mathcal{T} \models q'$ for all connected components q' of q; see e.g. [10].

Our algorithm for conjunctive query answering in $\mathcal{ELI}^f$ is based on canonical models. To introduce canonical models, we need some preliminaries. Let $\mathcal{T}$ be a TBox and Γ a finite set of concept names. We use $\mathsf{N}_\mathsf{C}^\mathcal{T}$ to denote the set of all concept names occurring in $\mathcal{T}$ and "$\sqsubseteq_\mathcal{T}$" to denote subsumption w.r.t. $\mathcal{T}$, i.e., $C \sqsubseteq_\mathcal{T} D$ iff $C^\mathcal{I} \subseteq D^\mathcal{I}$ for all models $\mathcal{I}$ of $\mathcal{T}$. We write

$$\mathsf{sub}_\mathcal{T}(\Gamma) := \{A \in \mathsf{N}_\mathsf{C}^\mathcal{T} \mid \bigsqcap_{A' \in \Gamma} A' \sqsubseteq_\mathcal{T} A\}$$

to denote the *closure* of Γ under subsuming concept names w.r.t. $\mathcal{T}$. For the next definition, the reader should intuitively assume that we want to make all elements of Γ (jointly) true at a domain element in a model of $\mathcal{T}$. If $A \in \Gamma$ and $A \sqsubseteq \exists r.B \in \mathcal{T}$, then we say that Γ *has* $\exists r.B$*-obligation* O, where

$$O = \{B\} \cup \{B' \in \mathsf{N}_\mathsf{C}^\mathcal{T} \mid \exists A' \in \Gamma : \exists r^-.A' \sqsubseteq B' \in \mathcal{T}\} \cup O',$$

with $O' = \emptyset$ if $\top \sqsubseteq (\leq 1\, r) \notin \mathcal{T}$ and $O' = \{B' \in \mathsf{N}_\mathsf{C}^\mathcal{T} \mid \exists A' \in \Gamma : A' \sqsubseteq \exists r.B' \in \mathcal{T}\}$ otherwise.

Let $\mathcal{T}$ be a TBox in normal form and $\mathcal{A}$ an ABox, for which we want to decide conjunctive query entailment (for a yet unspecified query q). We use $\mathsf{Ind}(\mathcal{A})$ to denote the set of individual names occurring in $\mathcal{A}$. To define a canonical model for $\mathcal{A}$ and $\mathcal{T}$, we have to require that $\mathcal{A}$ is *admissible* w.r.t. $\mathcal{T}$. What admissibility means depends on whether or not we make the UNA: $\mathcal{A}$ is admissible w.r.t. $\mathcal{T}$ if (i) the UNA is made and $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ or (ii) the UNA is not made and $(\top \sqsubseteq (\leq 1\, r)) \in \mathcal{T}$ implies that there are no $a, b, c \in \mathsf{Ind}(\mathcal{A})$ with $r(a, b), r(a, c) \in \mathcal{A}$ and $b \neq c$. As will be discussed later, admissibility can be ensured by an easy (polytime) preprocessing step.

We define a sequence of interpretations $\mathcal{I}_0, \mathcal{I}_1, \ldots$, and the canonical model for $\mathcal{A}$ and $\mathcal{T}$ will then be the limit of this sequence. To facilitate the construction, it is helpful to use domain elements that have an internal structure. An *existential* for $\mathcal{T}$ is a concept $\exists r.A$ that occurs on the right-hand side of some inclusion in $\mathcal{T}$. A *path* p for $\mathcal{T}$ is a finite (possibly empty) sequence of existentials for $\mathcal{T}$. We use $\mathsf{ex}(\mathcal{T})$ to denote the set of all existentials for $\mathcal{T}$, $\mathsf{ex}(\mathcal{T})^*$ to denote the set of all paths for $\mathcal{T}$, and ε to denote the empty path. All interpretations $\mathcal{I}_i$ in the above sequence will satisfy

$$\Delta^{\mathcal{I}_i} \subseteq \{\langle a, p\rangle \mid a \in \mathsf{Ind}(\mathcal{A}) \text{ and } p \in \mathsf{ex}^*(\mathcal{T})\}$$

For convenience, we use a slightly non-standard representation of interpretations when defining the sequence $\mathcal{I}_0, \mathcal{I}_1, \ldots$ and canonical interpretations: the function $\cdot^{\mathcal{I}}$ maps every element $d \in \Delta^{\mathcal{I}}$ to a set of concept names $d^{\mathcal{I}}$ instead of every concept name A to a set of elements $A^{\mathcal{I}}$. It is obvious how to translate back and forth between the standard representation and this one, and we will switch freely in what follows.

To start the construction of the sequence $\mathcal{I}_0, \mathcal{I}_1, \ldots$, define $\mathcal{I}_0$ as follows:

$$\begin{aligned}
\Delta^{\mathcal{I}_0} &:= \{\langle a, \varepsilon\rangle \mid a \in \mathsf{Ind}(\mathcal{A})\}\\
r^{\mathcal{I}_0} &:= \{(\langle a, \varepsilon\rangle, \langle b, \varepsilon\rangle) \mid r(a, b) \in \mathcal{A}\}\\
\langle a, \varepsilon\rangle^{\mathcal{I}_0} &:= \{A \in \mathsf{N_C} \mid \mathcal{A}, \mathcal{T} \models A(a)\}\\
a^{\mathcal{I}_0} &:= \langle a, \varepsilon\rangle
\end{aligned}$$

Now assume that $\mathcal{I}_i$ has already been defined. We want to construct $\mathcal{I}_{i+1}$. If it exists, select a $\langle a, p\rangle \in \Delta^{\mathcal{I}_i}$ and an $\alpha = \exists r.A \in \mathsf{ex}(\mathcal{T})$ such that $\langle a, p\rangle^{\mathcal{I}_i}$ has α-obligation O, and (i) $(\top \sqsubseteq (\leq 1\ r)) \notin \mathcal{T}$ and $\langle a, p\alpha\rangle \notin \Delta^{\mathcal{I}_i}$ or (ii) there is no $\langle b, p'\rangle \in \Delta^{\mathcal{I}_i}$ with $(\langle a, p\rangle, \langle b, p'\rangle) \in r^{\mathcal{I}_i}$. Then do the following:

- add $\langle a, p\alpha\rangle$ to $\Delta^{\mathcal{I}_i}$;
- if r is a role name, add $(\langle a, p\rangle, \langle a, p\alpha\rangle)$ to $r^{\mathcal{I}_i}$;
- if $r = s^-$, add $(\langle a, p\alpha\rangle, \langle a, p\rangle)$ to $s^{\mathcal{I}_i}$;
- set $\langle a, p\alpha\rangle^{\mathcal{I}_i} := \mathsf{sub}_{\mathcal{T}}(O)$.

The resulting interpretation is $\mathcal{I}_{i+1}$ (and $\mathcal{I}_{i+1} = \mathcal{I}_i$ if there are no $\langle a, p\rangle$ and α to be selected). We assume that the selected $\langle a, p\rangle$ is such that the length of p is minimal, and thus all obligations are eventually satisfied. To ensure that the constructed canonical model is unique, we also assume that the set $\mathsf{ex}(\mathcal{T})$ is well-ordered and the selected α is minimal for the node $\langle a, p\rangle$.

A proof of the following result can be found in the full version of this paper [15].

Lemma 3. *The canonical model $\mathcal{I}$ for $\mathcal{T}$ and $\mathcal{A}$ is a model of $\mathcal{T}$ and of $\mathcal{A}$.*

Our aim is to prove that we can verify whether $\mathcal{A}$ and $\mathcal{T}$ entail a conjunctive query q by checking whether the canonical model $\mathcal{I}$ for $\mathcal{A}$ and $\mathcal{T}$ matches q. Key to this result is the observation that the canonical model of $\mathcal{A}$ and $\mathcal{T}$ can be homomorphically embedded into any model of $\mathcal{A}$ and $\mathcal{T}$. We first define homomorphisms and then state the relevant lemma.

Let $\mathcal{I}$ and $\mathcal{J}$ be interpretations. A function $h : \Delta^{\mathcal{I}} \to \Delta^{\mathcal{J}}$ is a *homomorphism* from $\mathcal{I}$ to $\mathcal{J}$ if the following holds:

1. for all individual names a, $h(a^{\mathcal{I}}) = a^{\mathcal{J}}$;
2. for all concept names A and all $d \in \Delta^{\mathcal{I}}$, $d \in A^{\mathcal{I}}$ implies $h(d) \in A^{\mathcal{J}}$;
3. for all (maybe inverse) roles r and $d, e \in \Delta^{\mathcal{I}}$, $(d, e) \in r^{\mathcal{I}}$ implies $(h(d), h(e)) \in r^{\mathcal{J}}$.

Lemma 4. *Let $\mathcal{I}$ be the canonical model for $\mathcal{A}$ and $\mathcal{T}$, and $\mathcal{J}$ a model of $\mathcal{A}$ and $\mathcal{T}$. Then there is a homomorphism h from $\mathcal{I}$ to $\mathcal{J}$.*

Proof. Let $\mathcal{I}$ and $\mathcal{J}$ be as in the lemma. For each interpretation $\mathcal{I}_i$ in the sequence $\mathcal{I}_0, \mathcal{I}_1, \ldots$ used to construct $\mathcal{I}$, we define a homomorphism h_i from $\mathcal{I}_i$ to $\mathcal{J}$. The limit of the sequence $h_0, h_1, \ldots$ is then the desired homomorphism h from $\mathcal{I}$ to $\mathcal{J}$. To start, define h_0 by setting $h_0(\langle a, \varepsilon\rangle) := a^{\mathcal{J}}$ for all individual names a. Clearly, h_0 is a homomorphism:

- Condition 1 is satisfied by construction.
- For Condition 2, let $\langle a, \varepsilon\rangle \in A^{\mathcal{I}_0}$. Then $\mathcal{A}, \mathcal{T} \models A(a)$. Since $\mathcal{J}$ is a model of $\mathcal{A}$ and $\mathcal{T}$, $h_0(\langle a, \varepsilon\rangle) = a^{\mathcal{J}} \in A^{\mathcal{J}}$.
- For Condition 3, let $(\langle a, \varepsilon\rangle, \langle b, \varepsilon\rangle) \in r^{\mathcal{I}_0}$. Then $r(a, b) \in \mathcal{A}$ and since $\mathcal{J}$ is a model of $\mathcal{A}$ and by definition of h_0, we have $(h_0(\langle a, \varepsilon\rangle), h_0(\langle b, \varepsilon\rangle)) \in r^{\mathcal{J}}$.

Now assume that h_i has already been defined. If $\mathcal{I}_{i+1} = \mathcal{I}_i$, then $h_{i+1} = h_i$. Otherwise, there is a unique $\langle a, p\alpha\rangle \in \Delta^{\mathcal{I}_{i+1}} \setminus \Delta^{\mathcal{I}_i}$. Then $\langle a, p\rangle \in \Delta^{\mathcal{I}_i}$, and $\langle a, p\rangle^{\mathcal{I}_i}$ has $\alpha = \exists r.B$-obligation O such that $\langle a, p\alpha\rangle^{\mathcal{I}_{i+1}} = \mathsf{sub}_{\mathcal{T}}(O)$. Let $A \in \langle a, p\rangle^{\mathcal{I}_i}$ such that $A \sqsubseteq \exists r.B \in \mathcal{T}$. By Condition 2 of homomorphisms, we have $d = h_i(\langle a, p\rangle) \in A^{\mathcal{J}}$. Since $A \sqsubseteq \exists r.B \in \mathcal{T}$, there is an $e \in B^{\mathcal{J}}$ with $(d, e) \in r^{\mathcal{J}}$. Define h_{i+1} as the extension of h_i with $h_{i+1}(\langle a, p\alpha\rangle) := e$. We prove that the three conditions of homomorphisms are preserved:

- Condition 1 is untouched by the extension.
- Now for Condition 2. Since $\langle a, p\alpha\rangle^{\mathcal{I}_{i+1}} = \mathsf{sub}_{\mathcal{T}}(O)$ and $\mathcal{J}$ is a model of $\mathcal{T}$, it suffices to show that for all $B' \in O$, we have $e \in B'^{\mathcal{J}}$. Let $B' \in O$. By definition of O, we can distinguish three cases.
  First, let $B' = B$. Then we are done by choice of e.
  Second, let there be an $A' \in \langle a, p\rangle^{\mathcal{I}_i}$ such that $\exists r^-.A' \sqsubseteq B' \in \mathcal{T}$. Since h_i satisfies Condition 2 of homomorphisms, we have $d \in A'^{\mathcal{J}}$. Since $\mathcal{J}$ is a model of $\mathcal{T}$ and $(d, e) \in r^{\mathcal{J}}$, it follows that $e \in B'^{\mathcal{J}}$.
  The third case is that $\top \sqsubseteq (\leq 1\, r) \in \mathcal{T}$ and there is an $A' \in \langle a, p\rangle^{\mathcal{I}_i}$ such that $A' \sqsubseteq \exists r.B' \in \mathcal{T}$. It is similar to the previous case.
- Condition 3 was satisfied by $\mathcal{I}_i$ and is clearly preserved by the extension to $\mathcal{I}_{i+1}$. ❑

Lemma 5. *Let $\mathcal{I}$ be the canonical model for $\mathcal{A}$ and $\mathcal{T}$, and q a conjunctive query. Then $\mathcal{A}, \mathcal{T} \models q$ iff $\mathcal{I} \models q$.*

Proof. Let $\mathcal{I}$ and q be as in the lemma, and n, m, and k as above. If $\mathcal{I} \not\models q$, then $\mathcal{A}, \mathcal{T} \not\models q$ since, by Lemma 3, $\mathcal{I}$ is a model of $\mathcal{A}$ and $\mathcal{T}$. Now assume $\mathcal{I} \models^{\pi} q$, and let $\mathcal{J}$ be a model of $\mathcal{A}$ and $\mathcal{T}$. By Lemma 4, there is a homomorphism h from $\mathcal{I}$ to $\mathcal{J}$. Define $\pi' : \mathsf{Var}(q) \to \Delta^{\mathcal{J}}$ by setting $\pi'(v) := h(\pi(v))$. It is easily seen that $\mathcal{J} \models^{\pi'} q$. ❏

Thus, we can decide query entailment by looking only at the canonical model. At this point, we are faced with the problem that we cannot simply construct the canonical model $\mathcal{I}$ and check whether $\mathcal{I} \models q$ since $\mathcal{I}$ is infinite. However, we can show that if $\mathcal{I} \models q$, then $\mathcal{I} \models^{\pi} q$ for some match π that maps all variables to elements that can be reached by travelling only a bounded number of role edges from some ABox individual. Thus, it suffices to construct a sufficiently large "initial part" of $\mathcal{I}$ and check whether it matches q.

To make this formal, let n be the size of $\mathcal{A}$, m the size of $\mathcal{T}$, and k the size of q. In the following, we use $|p|$ to denote the length of a path p. The *initial canonical model* $\mathcal{I}'$ for $\mathcal{A}$ and $\mathcal{T}$ is obtained from the canonical model $\mathcal{I}$ for $\mathcal{A}$ and $\mathcal{T}$ by setting

$$\begin{aligned}
\Delta^{\mathcal{I}'} &:= \{\langle a, p\rangle \mid |p| \leq 2^m + k\} \\
A^{\mathcal{I}'} &:= A^{\mathcal{I}} \cap \Delta^{\mathcal{I}'} \\
r^{\mathcal{I}'} &:= r^{\mathcal{I}} \cap (\Delta^{\mathcal{I}'} \times \Delta^{\mathcal{I}'}) \\
a^{\mathcal{I}'} &:= a^{\mathcal{I}}
\end{aligned}$$

Lemma 6. *Let $\mathcal{I}$ be the canonical model for $\mathcal{A}$ and $\mathcal{T}$, $\mathcal{I}'$ the initial canonical model, and q a conjunctive query. Then $\mathcal{I} \models q$ iff $\mathcal{I}' \models q$.*

Proof. Let $\mathcal{I}, \mathcal{I}'$, and q be as in the lemma. It is obvious that $\mathcal{I}' \models q$ implies $\mathcal{I} \models q$. For the converse direction, let $\mathcal{I} \models^{\pi} q$. First assume that there is an $a \in \mathsf{Ind}(\mathcal{A})$ and a $v \in \mathsf{Var}(q)$ such that $\pi(v) = a^{\mathcal{I}}$. Since q is connected, this means that for all $v \in \mathsf{Var}(q)$, we have $\pi(v) = \langle a, p\rangle$ such that $|p| \leq k$. It follows that $\mathcal{I}' \models^{\pi} q$.

Now assume that there are no such a and v. Again since q is connected, this means that there is an $a \in \mathsf{Ind}(\mathcal{A})$ such that for all $v \in \mathsf{Var}(q)$, we have $\pi(v) = \langle a, p\rangle$, for some $p \in \mathsf{ex}^*(\mathcal{T})$. If $\pi(v) = \langle a, p\rangle$ with $|p| \leq 2^m + k$ for all $v \in \mathsf{Var}(q)$, then $\mathcal{I}' \models^{\pi} q$. Otherwise, there is a $v \in \mathsf{Var}(q)$ such that $\pi(v) = \langle a, p\rangle$ with $p \in \mathsf{ex}^*(\mathcal{T})$ such that $|p| > 2^m + k$. Since q is connected, this implies that for *all* $v \in \mathsf{Var}(q)$, we have $\pi(v) = \langle a, p\rangle$, for some $p \in \mathsf{ex}^*(\mathcal{T})$ with $|p| > 2^m$. Once more since q is connected, there is a $v_0 \in \mathsf{Var}(q)$ such that $\pi(v_0) = \langle a, p_0\rangle$ and for all $v \in \mathsf{Var}(q)$, we have $\pi(v) = \langle a, p\rangle$ with p_0 a prefix of p.

Since $|p_0| > 2^m$ and the number of distinct labels $d^{\mathcal{I}}$, $d \in \Delta^{\mathcal{I}}$, is bounded by 2^m, we can split p_0 into $p_1p_2p_3$ such that $\langle a, p_1\rangle^{\mathcal{I}} = \langle a, p_1p_2\rangle^{\mathcal{I}}$, and $p_2 \neq \varepsilon$. Now, let $\pi' : \mathsf{Var}(q) \to \Delta^{\mathcal{I}}$ be obtained by setting $\pi'(v) := \langle a, p_1p_3p\rangle$ if $\pi(v) = \langle a, p_1p_2p_3p\rangle$. In the full version of the proof given in [15], we show that $\mathcal{I} \models^{\pi'} q$. Moreover, for each $v \in \mathsf{Var}(q)$ with $\pi(v) = \langle a, p\rangle$ and $\pi'(v) = \langle a, p'\rangle$, we have that the length of p' is strictly smaller than that of p. It follows that we can repeat the described construction to construct a new match from an existing one only a finite number of times. We ultimately end up with a π^* such that $\mathcal{I} \models^{\pi^*} q$ and for all $v \in \mathsf{Var}(q)$, $\pi^*(v) = \langle a, p \rangle$ with $|p| \leq 2^m + k$. ❏

The initial canonical model $\mathcal{I}'$ for $\mathcal{A}$ and $\mathcal{T}$ can be constructed in time polynomial in the size of $\mathcal{A}$. In particular, (i) $\mathcal{I}_0$ can be constructed in polytime since, due to the results of [12,14], instance checking in $\mathcal{ELI}^f$ is tractable regarding data complexity; (ii) obligations can be computed in polytime since subsumption in $\mathcal{ELI}^f$ w.r.t. general TBoxes is decidable and the required checks are independent of the size of $\mathcal{A}$; (iii) the number of elements in the initial canonical model is bounded by $\ell := n \cdot m^{2^m+k}$ and is thus independent of the size of $\mathcal{A}$.

Our algorithm for deciding entailment of a conjunctive query q by a TBox $\mathcal{T}$ in normal form and an ABox $\mathcal{A}$ is as follows. If the UNA is made, we first check consistency of $\mathcal{A}$ w.r.t. $\mathcal{T}$ using one of the polytime algorithms from [12,14]. If $\mathcal{A}$ is inconsistent w.r.t. $\mathcal{T}$, we answer "yes". If the UNA is not made, then we convert $\mathcal{A}$ into an ABox $\mathcal{A}'$ that is admissible w.r.t. $\mathcal{T}$, and continue working with $\mathcal{A}'$. Obviously, the conversion can be done in time polynomial in the size of $\mathcal{A}$ simply by identifying ABox individuals. Both with and without UNA, at this point we have an ABox that is admissible w.r.t. $\mathcal{T}$. The next step is to construct the initial canonical structure $\mathcal{I}'$ for $\mathcal{T}$ and $\mathcal{A}$, and then check matches of q against this structure. The latter can be done in time polynomial in the size of $\mathcal{A}$: there are at most ℓ^k (and thus polynomially many) mappings $\tau : \mathsf{Var}(q) \to \Delta^{\mathcal{I}'}$, and each of them can be checked for being a match in polynomial time. We thus obtain a time bound for our algorithm of $p(n^k \cdot m^{k \cdot 2^m + k^2})$, with $p()$ a polynomial. This bound is clearly polynomial in n

Theorem 4. *In $\mathcal{ELI}^f$, conjunctive query answering w.r.t. general TBoxes is in P regarding data complexity.*

We conjecture that the time bound can be improved to $\mathcal{O}((n + 2^m)^k)$ (only single-exponential in m) by a more refined approach to canonical models. Basically, the idea is to work with the filtration of the canonical model instead of with the initial part.

A matching lower bound can be taken from [8] (which relies on the presence of general TBoxes and already applies to the instance problem), and thus we obtain P-completeness.

5 Summary and Outlook

The results of our investigation are summarized in Table 2. In all cases the lower bounds apply to instance checking and the upper bounds to conjunctive query entailment. The co-NP upper bounds are a consequence of the results in [10]. When the UNA is not explicitly mentioned, the results hold both with and without UNA. We point out two interesting issues. First, for all of the considered extensions we were able to show tractability regarding data complexity if and only if the logic is *convex regarding instances*, i.e., $\mathcal{A}, \mathcal{T} \models C(a)$ with $C = D_0 \sqcup \cdots \sqcup D_{n-1}$ implies $\mathcal{A}, \mathcal{T} \models D_i(a)$ for some $i < n$. It would be interesting to capture this phenomenon in a general result. And second, it is interesting to point out that subtle differences such as the UNA or local

Table 2. Complexity of instance checking and conjunctive query entailment

Extensions of $\mathcal{EL}$	w.r.t. acyclic TBoxes	w.r.t. general TBoxes
$\mathcal{EL}^{\neg A}$	coNP-complete	coNP-complete
$\mathcal{EL}^{C \sqcup D}$	coNP-complete	coNP-complete
$\mathcal{EL}^{\forall r.\bot}$, $\mathcal{EL}^{\forall r.C}$	coNP-complete	coNP-complete
$\mathcal{EL}^{(\leq kr)}$, $k \geq 0$	coNP-complete	coNP-complete
$\mathcal{EL}^{kf}$ w/o UNA, $k \geq 2$	coNP-complete (even w/o TBox)	coNP-complete
$\mathcal{EL}^{kf}$, $k \geq 2$ with UNA	coNP-complete (in P w/o TBox)	coNP-complete
$\mathcal{EL}^{(\geq kr)}$, $k \geq 2$	coNP-complete	coNP-complete
$\mathcal{EL}^{\exists \neg r.C}$	coNP-hard	coNP-hard
$\mathcal{EL}^{\exists r \cup s.C}$	coNP-hard	coNP-hard
$\mathcal{EL}^{\exists r^+.C}$	coNP-hard	coNP-hard
$\mathcal{ELI}^f$	in P	P-complete

versus global functionality (for the latter, see $\mathcal{EL}^{(\leq 1r)}$ vs. $\mathcal{ELI}^f$) can have an impact on tractability.

As future work, it would be interesting to extend our upper bound by including more operators from the tractable description logic $\mathcal{EL}^{++}$ as proposed in [2]. For a start, it is not hard to show that conjunctive query entailment in full $\mathcal{EL}^{++}$ is undecidable due to the presence of role inclusions $r_1 \circ \cdots \circ r_n \sqsubseteq s$. In the following, we briefly sketch the proof, which is by reduction of the problem of deciding whether the intersection of two languages defined by given context-free grammars $G_i = (N_i, T, P_i, S_i)$, $i \in \{1, 2\}$, is empty. We assume w.l.o.g. that the set of non-terminals N_1 and N_2 are disjoint. Then define a TBox

$$\mathcal{T} := \{\top \sqsubseteq \exists r_a.\top \mid a \in T\} \cup \{r_{A_1} \circ \cdots \circ r_{A_n} \sqsubseteq r_A \mid A \to A_1 \cdots A_n \in P_1 \cup P_2\}.$$

It is not too difficult to see that $L(G_1) \cap L(G_2) \neq \emptyset$ iff the conjunctive query $S_1(u, v) \wedge S_2(u, v)$ is entailed by the ABox $\{\top(a)\}$ and TBox $\mathcal{T}$.

We have learned recently that the same undecidability result has been shown independently and in parallel in the workshop papers [17,18]. For people interested in the complexity of conjunctive querying entailment in the $\mathcal{EL}$ family of DLs, both papers are recommended reading. In particular, the algorithms for query answering presented there seem more suitable for implementation than the brute-force canonical model approach pursued in Section 4. We have also learned that our undecidability result is very similar to a number of undecidability results for subsumption in extensions of $\mathcal{EL}$ proved in [13].

Acknowledgement. We are grateful to Markus Krötzsch and Meng Suntisrivaraporn for valuable comments on earlier versions of this paper.

References

1. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: DL-Lite in the light of first-order logic. In: AAAI 2007. Proc. of the 22nd Conf. on AI, AAAI Press, Stanford, California, USA (2007)
2. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI 2005. Proc. of the 19th Int. Joint Conf. on AI, pp. 364–369. Morgan Kaufmann, San Francisco (2005)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. (submitted to a Journal, 2007)
4. Baader, F., Lutz, C., Suntisrivaraporn, B.: Is tractable reasoning in extensions of the description logic $\mathcal{EL}$ useful in practice? In: M4M 2005. Proc. of the 4th Int. WS on Methods for Modalities (2005)
5. Baader, F., McGuiness, D.L., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, implementation and applications. Cambridge University Press, Cambridge (2003)
6. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and—what else? In: ECAI 2004. Proc. of the 16th European Conf. on AI, pp. 298–302. IOS Press, Amsterdam (2004)
7. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: DL-lite: Tractable description logics for ontologies. In: AAAI 2005. Proc. of the 20th National Conf. on AI, pp. 602–607. AAAI Press, Stanford, California, USA (2005)
8. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: KR 2006. Proc. of the 10th Int. Conf. on KR, AAAI Press, Stanford, California, USA (2006)
9. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R., Vetere, G.: DL-lite: Practical reasoning for rich dls. In: DL 2004. CEUR Workshop Proceedings. CEUR-WS.org, vol. 104 (2004)
10. Glimm, B., Horrocks, I., Lutz, C., Sattler, U.: Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. In: IJCAI 2007. Proc. of the 20th Int. Joint Conf. on AI, AAAI Press, Stanford, California, USA (2007)
11. Giacomo, G.D., Lenzerini, M.: Boosting the correspondence between description logics and propositional dynamic logics. In: AAAI 1994. Proc. of the 12th National Conf. on AI, vol. 1, pp. 205–212. AAAI Press, Stanford, California, USA (1994)
12. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: IJCAI 2005. Proc. of the 19th Int. Joint Conf. on AI, Professional Book Center, pp. 466–471 (2005)
13. Kazakov, Y.: Saturation-based decision procedures for extensions of the guarded fragment, PhD thesis, University of Saarland (2005)
14. Krisnadhi, A.: Data complexity of instance checking in the $\mathcal{EL}$ family of description logics. Master thesis, TU Dresden, Germany (2007)
15. Krisnadhi, A., Lutz, C.: Data complexity of instance checking in the $\mathcal{EL}$ family of description logics. available from `http://lat.inf.tu-dresden.de/~clu/papers/`
16. Krötzsch, M., Rudolph, S., Hitzler, P.: On the complexity of horn description logics. In: Proc. of the 2nd WS on OWL: Experiences and Directions. CEUR-WS, vol. 216 (2006), `http://ceur-ws.org/`
17. Krötzsch, M., Rudolph, S.: Conjunctive Queries for $\mathcal{EL}$ with Composition of Roles. In: DL 2007. Proc. of the 2007 Int. WS on DLs, CEUR-WS.org (2007)
18. Rosati, R.: On conjunctive query answering in $\mathcal{EL}$. In: DL 2007. Proc. of the 2007 Int. WS on DLs, CEUR-WS.org (2007)
19. Schaerf, A.: On the complexity of the instance checking problem in concept languages with existential quantification. Journal of Intelligent Information Systems 2, 265–278 (1993)

An Extension of the Knuth-Bendix Ordering with LPO-Like Properties

Michel Ludwig[1] and Uwe Waldmann[2]

[1] Department of Computer Science, University of Liverpool
Liverpool L69 3BX, United Kingdom
[2] Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany

Abstract. The Knuth-Bendix ordering is usually preferred over the lexicographic path ordering in successful implementations of resolution and superposition, but it is incompatible with certain requirements of hierarchic superposition calculi. Moreover, it does not allow non-linear definition equations to be oriented in a natural way. We present an extension of the Knuth-Bendix ordering that makes it possible to overcome these restrictions.

1 Introduction

In theorem proving calculi like Knuth-Bendix completion, resolution, or superposition, reduction orderings such as the Knuth-Bendix ordering (KBO) [11] or the lexicographic path ordering (LPO) by Kamin and Lévy [10] are crucial to reduce the search space. Among these orderings, the Knuth-Bendix ordering is usually preferred in state-of-the-art implementations of theorem provers. There are several reasons for this: it can be efficiently implemented – the most efficient known algorithm needs only linear time – and it correlates well with the sizes of terms; so, reductions w. r. t. a KBO usually lead to terms with fewer nodes. In comparison, computing term comparisons for the lexicographic path ordering requires at least quadratic time and reductions w. r. t. an LPO may result in arbitrarily larger terms.

On the other hand, it is exactly this correlation between the KBO and term sizes that renders the KBO incompatible with special requirements occurring in certain applications. One example is hierarchic theorem proving [3,6,15], where one considers two signatures $\Sigma \supseteq \Sigma_0$ and needs an ordering in which every ground term involving a symbol from $\Sigma \setminus \Sigma_0$ is larger than every ground term over Σ_0. With an LPO, this property is easy to establish, with a KBO it is usually impossible.

A second example is definitions of the form $f(t_1, \ldots, t_n) \approx t_0$ where f does not occur in t_0. Such definitions can easily be ordered from left to right using an LPO where f is larger in the precedence than every symbol occurring in t_0. With a KBO, however, we have the additional requirement that no variable occurs more often in t_0 than in $f(t_1, \ldots, t_n)$; non-linear definitions cannot be handled adequately using a KBO.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 348–362, 2007.
© Springer-Verlag Berlin Heidelberg 2007

In this paper, we present a variant of the Knuth-Bendix ordering that preserves as much as possible of the spirit of KBO, yet satisfies the requirements for hierarchic theorem proving and non-linear definitions. Like the original KBO, our ordering is a simplification ordering that can optionally be made total on ground terms. This is an essential property for theorem proving calculi such as superposition (Bachmair and Ganzinger [2]) and it goes beyond what can be shown using approaches for showing modular termination of rewrite systems (e. g., Fernández, Godoy, and Rubio [5]).

Due to lack of space we cannot give complete proofs in this paper, for which we refer the reader to (Ludwig [13]).

2 Preliminaries

We assume that the reader is familiar with standard concepts and notations in the area of rewriting (see Baader and Nipkow [1]). We use the notation $f/n \in \Sigma$ to denote that the signature Σ contains the n-ary function symbol f; if $n = 0$, f is also called a constant symbol. The set of terms over a signature Σ and a set X of variables is written $\mathrm{T}_\Sigma(X)$; $\mathrm{T}_\Sigma(\emptyset)$ is the set of ground terms over Σ. For a term $t \in \mathrm{T}_\Sigma(X)$, $|t|$ denotes the size, i. e., the number of nodes of t; if x is a variable in X, $|t|_x$ denotes the number of occurrences of x in t, and $\mathrm{P}(x,\, t)$ denotes the set of all positions of occurrences of x in t. Signatures are assumed to be finite.

Definition 1. Let X be a set of variables, let Σ be a signature, and let $\succ\, \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ be a binary relation on the terms over X and Σ. Then $\succ$ is said to be *compatible with Σ-operations*, if $s \succ s'$ implies $f(t_1, \ldots, t_{i-1}, s, t_{i+1}, \ldots, t_n) \succ f(t_1, \ldots, t_{i-1}, s', t_{i+1}, \ldots, t_n)$ for all symbols $f/n \in \Sigma$ with arity $n \in \mathbb{N}$, for all terms $s, s', t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X)$ and for all coefficients $i \in \mathbb{N}, 1 \leq i \leq n$; $\succ$ is called *stable under substitutions* if $s \succ s'$ implies $s\sigma \succ s'\sigma$ for all terms $s, s' \in \mathrm{T}_\Sigma(X)$ and for all substitutions $\sigma \colon X \to \mathrm{T}_\Sigma(X)$. The relation $\succ$ has the *subterm property* if $s \succ s'$ whenever s' is a proper subterm of s; $\succ$ is a *rewrite relation* if $\succ$ is compatible with Σ-operations and stable under substitutions; it is a *rewrite ordering* if it is a strict partial ordering and a rewrite relation; it is a *simplification ordering* if $\succ$ is a rewrite ordering and has the subterm property.

The Knuth-Bendix ordering (KBO) is an example of a simplification ordering. It is parameterized by a "precedence" on signature symbols, and a weight function. The KBO was originally introduced by Knuth and Bendix [11] with a stricter variable condition; the version presented in this document can be found in (Dick, Kalmus, and Martin [4]) and also in (Baader and Nipkow [1]).[1]

First of all, in order to develop later a function that computes the weight of terms, we need to assign weights to signature symbols, which will be natural numbers in the case of the KBO.

---

[1] In fact, Dick, Kalmus, and Martin [4] and Baader and Nipkow [1] also permit positive real coefficients.

Let Σ be a signature, let X be a set of variables and let $\sqsupset$ be a strict partial ordering on Σ. A ([illegible]) [illegible] is a function $\lambda\colon \Sigma \cup X \to \mathbb{N}$. It is called [illegible] $\sqsupset$ if the following two conditions are satisfied:

(i) There exists a $\lambda_0 \in \mathbb{N}^{>0}$ such that for all $x \in X$: $\lambda(x) = \lambda_0$ and for all $c/0 \in \Sigma$: $\lambda(c) \geq \lambda_0$.
(ii) If there exists an $f/1 \in \Sigma$ such that $\lambda(f) = 0$, then $f \sqsupseteq g$ for all $g \in \Sigma$.

A symbol weight assignment λ is extended recursively to a weight function $\mathrm{w}_\lambda\colon \mathrm{T}_\Sigma(X) \to \mathbb{N}$ on terms as follows:

- $\mathrm{w}_\lambda(x) = \lambda(x)$ for $x \in X$.
- $\mathrm{w}_\lambda\big(f(t_1, \ldots, t_n)\big) = \lambda(f) + \sum_{i=1}^{n} \mathrm{w}_\lambda(t_i)$ for $f/n \in \Sigma$, $n \in \mathbb{N}$.

At first, the Knuth-Bendix ordering compares two terms by using the weight function. If both terms have the same weight, the precedence is considered, and only ultimately, if the top symbol is equal as well, recursion is used to compare two terms.

Definition 2 (Knuth-Bendix Ordering). Let Σ be a signature and let X be a set of variables. Additionally, let $\sqsupset$ be a strict partial ordering, the precedence, on Σ and $\lambda\colon \Sigma \cup X \to \mathbb{N}$ be a regular symbol weight assignment that is admissible for $\sqsupset$. Finally, let $\mathrm{w} = \mathrm{w}_\lambda\colon \mathrm{T}_\Sigma(X) \to \mathbb{N}$ be the regular term weight function induced by λ.

We define the [illegible] $\succ_{\mathrm{KBO}} \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ induced by $(\sqsupset, \lambda)$ on terms $s, t \in \mathrm{T}_\Sigma(X)$ in the following way: $s \succ_{\mathrm{KBO}} t$ if

(KBO1) $\forall\, x \in X\colon\ |s|_x \geq |t|_x$ and $\mathrm{w}(s) > \mathrm{w}(t)$,
or
(KBO2) $\forall\, x \in X\colon\ |s|_x \geq |t|_x$, $\mathrm{w}(s) = \mathrm{w}(t)$ and one of the following cases holds:
  (KBO2a) $\exists\, f/1 \in \Sigma$, $\exists\, x \in X$, $\exists\, n \in \mathbb{N}^{>0}$ such that $s = f^n(x)$ and $t = x$,
  (KBO2b) $\exists\, f/m, g/n \in \Sigma$ $(m, n \in \mathbb{N})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X)$ such that $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ with $f \sqsupset g$,
  (KBO2c) $\exists\, f/m \in \Sigma$ $(m \in \mathbb{N}^{>0})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_m \in \mathrm{T}_\Sigma(X)$, $\exists\, i, 1 \leq i \leq m$ such that $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$ and such that $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$, $s_i \succ_{\mathrm{KBO}} t_i$.

The Knuth-Bendix ordering is a simplification ordering; moreover, if the precedence is a total ordering, it is total on ground terms. It can be computed in time $O(|s| + |t|)$, where s and t are the terms to be compared (Löchner [12]).[2]

3 Transfinite KBO

3.1 Motivation

The Knuth-Bendix ordering correlates well with the sizes of terms, which is often a desirable property, since it implies that reductions w. r. t. a KBO usually lead to terms with fewer nodes. On the other hand, it is exactly this correlation between the KBO and term sizes that renders the KBO incompatible with some special requirements for certain applications.

---

[2] Using a machine model in which addition of numbers takes constant time.

One example is the problem of orienting definition equations in an intuitive direction. Suppose that we are given a sequence of signatures Σ_i $(0 \leq i \leq n)$ where $\Sigma_i = \{f_i\} \cup \Sigma_{i-1}$ for $i \geq 1$, and that we have a set of non-recursive definition equations of the form $f_i(s_{i1}, \dots, s_{ik}) \approx t_i$, with $t_i, s_{ij} \in \mathrm{T}_{\Sigma_{i-1}}(X)$ and $\mathrm{Var}(t_i) \subseteq \mathrm{Var}(f_i(s_{i1}, \dots, s_{ik}))$ (where the s_{ij} are often, but not necessarily, variables). If we use a lexicographic path ordering with a precedence $f_n \sqsupset \cdots \sqsupset f_2 \sqsupset f_1 \sqsupset \dots$, then every term t with a top symbol f_i is larger than every term in $\mathrm{T}_{\Sigma_{i-1}}(\mathrm{Var}(t))$, i. e., all these equations can be oriented from left to right (and can hopefully be used to eliminate all occurrences of the f_i in the remainder of the specification completely). If we try to get a similar effect with a KBO, we face two problems: the KBO correlates with term sizes, so in general, a term cannot be larger than [illegible] term over some subsignature, and moreover a term cannot be larger than another term in which some variable occurs more often.

Another scenario where the Knuth-Bendix ordering does not work satisfactorily is hierarchic theorem proving. Standard first-order theorem provers are notoriously bad at dealing with integer or real arithmetic – encoding numbers in binary or unary is not really a viable solution in most application contexts. A hierarchic proof system adds theory knowledge to a saturation-based calculus by using a proof system for a base theory, say, a decision procedure for real arithmetic as a black box. The proof system is initially given a set of formulas over some extension of the base theory, e. g., over real arithmetic extended with data structures, free function symbols, etc. As usual, the deduction rules of the calculus are employed to generate formulas from premises and the conclusions are added to the set of formulas; in addition, all derived formulas belonging to the base domain are passed to the decision procedure. As soon as one of the two systems encounters a contradiction, the problem is solved. (Bachmair, Ganzinger, and Waldmann [3], Ganzinger, Sofronie-Stokkermans, and Waldmann [6], Prevosto and Waldmann [15]).

In hierarchic theorem proving calculi, one usually considers a signature Σ_0 of base symbols and a signature $\Sigma \supseteq \Sigma_0$ that extends Σ_0. Similarly to the ordinary superposition calculus, hierarchic superposition calculi are parameterized by a reduction ordering $\succ$ that is total on ground terms. In order to ensure refutational completeness, this ordering must have the property that every ground term in $\mathrm{T}_{\Sigma_0}(\emptyset)$ is strictly smaller than every ground term in $\mathrm{T}_{\Sigma}(\emptyset) \setminus \mathrm{T}_{\Sigma_0}(\emptyset)$. This requirement is easy to establish with an LPO – the precedence just needs to be defined in such a way that all the symbols from Σ_0 are smaller than all the symbols from Σ – but it is generally incompatible with the definition of the Knuth-Bendix ordering.[3]

Our goal is to find a computable (total) simplification ordering that generalises the KBO and satisfies the requirements of hierarchic theorem proving. We will show that such an ordering can be constructed using certain ordinal numbers as weights.

---

[3] Except if Σ_0 consists only of constant symbols and at most one unary function symbol.

In the next sections, we start by presenting a very general version of the ordering, which is computable, but unfortunately not very efficiently. Restrictions that lead to a better runtime behaviour are discussed later.

3.2 Ordinal Numbers

A set α is an ordinal if α is totally ordered with respect to the subset relation and every element of α is also a subset of α. The class of all ordinals is denoted by **ON**. Ordinals are ordered by the element relation, or equivalently, by the subset relation, i. e., $\alpha < \beta$ if and only if $\alpha \in \beta$ if and only if $\alpha \subsetneq \beta$.

If a non-empty ordinal β has a largest element α, then it can be written as $\beta = \alpha \cup \{\alpha\}$. We say that β is the successor of α, denoted by $\beta = S(\alpha)$. A non-empty ordinal γ that is not a successor of another ordinal is called a limit ordinal. Every limit ordinal γ is the union (or least upper bound) of all ordinals that are smaller than γ.

The ordinals $\emptyset$, $\{\emptyset\}$, $\{\emptyset, \{\emptyset\}\}$, and so on, are identified with the natural numbers $0, 1, 2, \ldots$. The smallest limit ordinal is denoted by ω, it corresponds to the set of all natural numbers.

The following operations on ordinal numbers can be seen as the standard extensions of the addition, multiplication and exponentiation on natural numbers, in particular they coincide with the latter if their arguments are natural numbers. For more information we refer to (Just and Weese [9]).

Definition 3 (Regular Ordinal Operations). Let $\alpha, \beta \in \mathbf{ON}$ be ordinals. The ordinal $\alpha + \beta$, $\alpha \cdot \beta$, and α^β are defined by recursion over β:

$$\begin{aligned}
&\alpha + 0 = \alpha\\
&\alpha + \beta = S(\alpha + \gamma) \text{ if } \beta = S(\gamma)\\
&\alpha + \beta = \textstyle\bigcup_{\gamma<\beta} (\alpha + \gamma) \text{ if } \beta \text{ is a limit ordinal} > 0
\end{aligned}$$

$$\begin{aligned}
&\alpha \cdot 0 = 0\\
&\alpha \cdot \beta = (\alpha \cdot \gamma) + \alpha \text{ if } \beta = S(\gamma)\\
&\alpha \cdot \beta = \textstyle\bigcup_{\gamma<\beta} (\alpha \cdot \gamma) \text{ if } \beta \text{ is a limit ordinal} > 0
\end{aligned}$$

$$\begin{aligned}
&\alpha^0 = 1\\
&\alpha^\beta = \alpha^\gamma \cdot \alpha \text{ if } \beta = S(\gamma)\\
&\alpha^\beta = \begin{cases} 0 & \text{if } \beta \text{ is a limit ordinal} > 0 \text{ and } \alpha = 0\\ \bigcup_{\gamma<\beta} \alpha^\gamma & \text{if } \beta \text{ is a limit ordinal} > 0 \text{ and } \alpha > 0 \end{cases}
\end{aligned}$$

We cannot use the regular operations on ordinals to compute the weight of a term from the weights of its subterms, which is mainly due to the fact that ordinal addition and multiplication are only monotonic, but not strictly monotonic with respect to $>$. For instance, $1 > 0$, but $1 + \omega = \omega = 0 + \omega$, so $\alpha > \alpha'$ does in general not imply $\alpha + \beta > \alpha' + \beta$. There is an alternative set of operations on ordinals that is better suited for our purposes. Let us first define a subset of the ordinal numbers on which we will define those operations:

Definition 4 (Set O). We define the set $\mathbf{O} \subseteq \mathbf{ON}$ inductively as follows:

- $0 \in \mathbf{O}$.
- If there exists an $m \in \mathbb{N}^{>0}$, $n_1, \ldots, n_m \in \mathbb{N}^{>0}$, and $\delta_1, \ldots, \delta_m \in \mathbf{O}$ with $\delta_1 > \delta_2 > \cdots > \delta_m$, then

$$\sum\nolimits_{i=1}^{m} (\omega^{\delta_i} \cdot n_i) \in \mathbf{O}.$$

The set $\mathbf{O}$ exactly contains those ordinals that are smaller than ε_0, where the limit ordinal ε_0 is the smallest ordinal such that $\varepsilon_0 = \omega^{\varepsilon_0}$. Elements of $\mathbf{O}$ are sums of finite sequences of ordinals $\omega^{\beta_i} \cdot n_i$, which we call the [illegible]. The decomposition of an ordinal α into a sum $\sum_{i=1}^{m} (\omega^{\delta_i} \cdot n_i)$ with $\delta_1 > \delta_2 > \cdots > \delta_m$ is called the Cantor normal form of α; it is unique. We define

- $\deg(\alpha) = \delta_1$,
- $\text{Exponents}(\alpha) = \{\delta_1, \delta_2, \ldots, \delta_m\}$,
- $\text{coeff}(\alpha, \beta) = \begin{cases} n_i & \text{if } \beta = \delta_i \text{ for some } i \text{ with } 1 \le i \le m, \\ 0 & \text{otherwise.} \end{cases}$

For $\alpha = 0$, we define $\deg(\alpha) = -\infty$, $\text{Exponents}(\alpha) = \emptyset$, and $\text{coeff}(\alpha, \beta) = 0$.

The following operations on ordinals were introduced by Hessenberg [7]. Intuitively, they add and multiply ordinals in $\mathbf{O}$ as if they were polynomials in ω.

Definition 5 (Hessenberg Addition). The function $\oplus \colon \mathbf{O} \times \mathbf{O} \to \mathbf{O}$ is defined as follows:

- $0 \oplus \alpha = \alpha$ for $\alpha \in \mathbf{O}$.
- $\alpha \oplus 0 = \alpha$ for $\alpha \in \mathbf{O}$.
- Suppose that

$$\alpha = \sum\nolimits_{i=1}^{m} (\omega^{\delta_i} \cdot n_i), \quad \beta = \sum\nolimits_{i=1}^{m'} (\omega^{\delta'_i} \cdot n'_i) \in \mathbf{O}$$

for natural numbers $m, m' \in \mathbb{N}^{>0}$, $n_1, \ldots, n_m$, $n'_1, \ldots, n'_{m'} \in \mathbb{N}^{>0}$, ordinals $\delta_1, \ldots, \delta_m$, $\delta'_1, \ldots, \delta'_{m'} \in \mathbf{O}$ such that $\delta_1 > \delta_2 > \cdots > \delta_m$ and $\delta'_1 > \delta'_2 > \cdots > \delta'_{m'}$. Then

$$\alpha \oplus \beta = \sum\nolimits_{i=1}^{m''} \Big(\omega^{c_i} \cdot \big(\text{coeff}(\alpha, c_i) + \text{coeff}(\beta, c_i)\big)\Big)$$

where we set $\text{Exponents}(\alpha) \cup \text{Exponents}(\beta) = \{c_1, c_2, \ldots, c_{m''}\}$ such that $m'' \in \mathbb{N}$ and $c_1 > c_2 > \cdots > c_{m''}$.

Definition 6 (Hessenberg Multiplication). The function $\odot \colon \mathbf{O} \times \mathbf{O} \to \mathbf{O}$ is defined as follows:

- $0 \odot \alpha = 0$ for $\alpha \in \mathbf{O}$.
- $\alpha \odot 0 = 0$ for $\alpha \in \mathbf{O}$.

- Suppose that

$$\alpha = \sum\nolimits_{i=1}^{m} (\omega^{\delta_i} \cdot n_i), \quad \beta = \sum\nolimits_{j=1}^{m'} (\omega^{\delta'_j} \cdot n'_j)$$

  for $m, m' \in \mathbb{N}^{>0}$, $n_1, \ldots, n_m$, $n'_1, \ldots, n'_{m'} \in \mathbb{N}^{>0}$, $\delta_1, \ldots, \delta_m$, $\delta'_1, \ldots, \delta'_{m'} \in \mathbf{O}$ such that $\delta_1 > \delta_2 > \cdots > \delta_m$ and $\delta'_1 > \delta'_2 > \cdots > \delta'_{m'}$. Then

$$\alpha \odot \beta = \bigoplus_{i=1}^{m} \bigoplus_{j=1}^{m'} \Big(\omega^{\delta_i \oplus \delta'_j} \cdot \big(\mathrm{coeff}(\alpha, \delta_i) \cdot \mathrm{coeff}(\beta, \delta'_j) \big) \Big).$$

Lemma 7. *The following properties hold for all* $\alpha, \beta, \gamma \in \mathbf{O}$*:*

- $\alpha \oplus \beta = \beta \oplus \alpha$.
- $\alpha \odot \beta = \beta \odot \alpha$.
- $\alpha \oplus (\beta \oplus \gamma) = (\alpha \oplus \beta) \oplus \gamma$.
- $\alpha \odot (\beta \odot \gamma) = (\alpha \odot \beta) \odot \gamma$.
- $\alpha \odot (\beta \oplus \gamma) = \alpha \odot \beta \oplus \alpha \odot \gamma$.
- $\alpha < \beta$ *implies* $\alpha \oplus \gamma < \beta \oplus \gamma$.
- $\alpha < \beta$ *and* $\gamma > 0$ *imply* $\alpha \odot \gamma < \beta \odot \gamma$.

It is important to note that the Hessenberg addition $\oplus$ on the set $\mathbf{O}$ does not possess the continuity property, i. e., for two ordinals $\alpha, \beta \in \mathbf{O}$ such that $\alpha < \beta$ there does not necessarily exist an ordinal $\gamma \in \mathbf{O}$ such that $\alpha \oplus \gamma = \beta$. A simple example consists in the two ordinals 1 and ω: there is no ordinal $\alpha \in \mathbf{O}$ such that $1 \oplus \alpha = \omega$. This fact makes the proof of the following lemma (which is essential for proving that our ordering is stable under substitutions) rather tedious:

Lemma 8. *Let* $\alpha, \beta, \gamma, \delta, \zeta \in \mathbf{O}$ *be ordinals such that* $\beta \leq \zeta$ *and*

$$\alpha \oplus (\beta \odot \gamma) < \delta \oplus (\zeta \odot \gamma).$$

Furthermore, let $\eta \in \mathbf{O}$ *be an ordinal such that* $\deg(\eta) > \deg(\gamma)$*. Then*

$$\alpha \oplus (\beta \odot \eta) < \delta \oplus (\zeta \odot \eta).$$

3.3 Constructing the Ordering

We start by introducing two functions. Firstly, we assign an ordinal number, the so-called [illegible], to every symbol in the signature and to every variable.

Definition 9 (Ordinal Symbol Weight Assignment). Let X be a set of variables and Σ be a signature. Then, an [illegible] is a function $\Omega\colon \Sigma \cup X \to \mathbf{O}$.

Adding up ordinals as weights is sufficient for hierarchic theorem proving, but it is not sufficient for dealing with non-linear definitions. In addition, we need a factor for each signature symbol, called [illegible], with which we multiply the weights of subterms before the weight of the top symbol is added:[4]

---

[4] The idea of multiplying weights of subterms by some factor can also be found in Otter's ad hoc ordering [14] (which is in general not a reduction ordering, though).

Definition 10 (Subterm Coefficient Function). Let Σ be a signature. Then, a *subterm coefficient function* is a mapping $\Psi\colon \Sigma \to \mathbf{O} \setminus \{0\}$.

Using the two previous definitions, we can construct a function that computes the (ordinal) weight of terms.

Definition 11 (Ordinal Term Weight). Let X be a set of variables and Σ be a signature. Furthermore, let $\Omega\colon \Sigma \cup X \to \mathbf{O}$ be an ordinal symbol weight assignment and $\Psi\colon \Sigma \to \mathbf{O} \setminus \{0\}$ be a subterm coefficient function. Then, we inductively define a function

$$\mathrm{W} = \mathrm{W}_{(\Omega,\,\Psi)}\colon \mathrm{T}_{\Sigma}(X) \to \mathbf{O}$$

which computes the *(ordinal) weight of terms* in the following way:

- For $x \in X$:
$$\mathrm{W}_{(\Omega,\,\Psi)}(x) = \Omega(x).$$
- For $n \in \mathbb{N}$ and terms $t_1, \ldots, t_n \in \mathrm{T}_{\Sigma}(X)$:
$$\mathrm{W}_{(\Omega,\,\Psi)}\bigl(f(t_1, \ldots, t_n)\bigr) = \Omega(f) \oplus \Bigl(\Psi(f) \odot \bigoplus_{i=1}^{n} \mathrm{W}_{(\Omega,\,\Psi)}(t_i)\Bigr).$$

We define now when an ordinal symbol weight assignment is *admissible* for a strict partial ordering on signature symbols.

Definition 12 (Admissible Symbol Weight Assignment). Let Σ be a signature, X be a set of variables and $\sqsupset$ be a strict partial ordering on Σ. We say then that the symbol weight assignment $\Omega\colon \Sigma \cup X \to \mathbf{O}$ is *admissible* for the ordering $\sqsupset$ if the following two conditions are satisfied:

(i) There exists an $\Omega_0 \in \mathbb{N}^{>0}$ such that for all $x \in X$: $\Omega(x) = \Omega_0$ and for all $c/0 \in \Sigma$: $\Omega(c) \geq \Omega_0$.
(ii) If there exists an $f/1 \in \Sigma$ such that $\Omega(f) = 0$, then $f \sqsupseteq g$ for all $g \in \Sigma$.

The coefficient of a position p in a term t is the product of all the coefficients of the function symbols on the path from the root of t to p:

Definition 13 (Coefficient of a Position). Let X be a set of variables and Σ be a signature. Furthermore, let $\Psi\colon \Sigma \to \mathbf{O} \setminus \{0\}$ be a subterm coefficient function, $t \in \mathrm{T}_{\Sigma}(X)$ be a term and $p \in \mathrm{pos}(t)$ be a position in t. We inductively define the coefficient $\mathcal{C}(p,\, t)$ of p in t as follows:

- $\mathcal{C}(\Lambda,\, t) = 1$, where Λ is the empty string.
- If $t = f(t_1, \ldots, t_n)$ for $n \in \mathbb{N}^{>0}$, terms $t_1, \ldots, t_n \in \mathrm{T}_{\Sigma}(X)$ and a position $p = ip'$ such that $1 \leq i \leq n$ and $p' \in \mathrm{pos}(t_i)$, then
$$\mathcal{C}(p,\, t) = \mathcal{C}\bigl(ip',\, f(t_1, \ldots, t_n)\bigr) = \Psi(f) \odot \mathcal{C}(p',\, t_i).$$

We can now define the *transfinite Knuth-Bendix ordering* (TKBO). Compared with the definition of the regular Knuth-Bendix ordering (KBO) (Def. 2) the variable occurrence condition is replaced by two separate conditions on term variables and coefficient sums. It is then possible for a smaller term (with respect to the TKBO) to contain a specific variable more often than the corresponding larger term, which allows to order non-linear term definitions. Note that we obtain the usual variable condition as a special case if $\Psi(f) = 1$ for every symbol f.

Definition 14 (Transfinite Knuth-Bendix Ordering). Let Σ be a signature and X be a set of variables. Additionally, let $\sqsupset$ be a strict ordering on Σ, let $\Omega\colon \Sigma \cup X \to \mathbf{O}$ be an ordinal symbol weight assignment that is admissible for $\sqsupset$ and let $\Psi\colon \Sigma \to \mathbf{O} \setminus \{0\}$ be a subterm coefficient function. Finally, let $\mathrm{W} = \mathrm{W}_{(\Omega,\Psi)}\colon \mathrm{T}_\Sigma(X) \to \mathbf{O}$ be the ordinal term weight function induced by Ω and Ψ.

We define the *transfinite Knuth-Bendix ordering* (TKBO) $\succ_\mathrm{T} \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ induced by $(\sqsupset, \Omega, \Psi)$ on terms $s, t \in \mathrm{T}_\Sigma(X)$ in the following way: $s \succ_\mathrm{T} t$ if

(TKBO1) $\mathrm{Var}(s) \supseteq \mathrm{Var}(t)$, $\mathrm{W}(s) > \mathrm{W}(t)$ and

$$\forall\, x \in \mathrm{Var}(t)\colon \bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) \geq \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t)$$

or

(TKBO2) $\mathrm{Var}(s) \supseteq \mathrm{Var}(t)$, $\mathrm{W}(s) = \mathrm{W}(t)$,

$$\forall\, x \in \mathrm{Var}(t)\colon \bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) \geq \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t)$$

and one of the following cases occurs:

(TKBO2a) $\exists\, f/1 \in \Sigma$, $\exists\, x \in X$, $\exists\, n \in \mathbb{N}^{>0}$ such that $s = f^n(x)$ and $t = x$,

(TKBO2b) $\exists\, f/m, g/n \in \Sigma$ $(m, n \in \mathbb{N})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X)$ such that $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ with $f \sqsupset g$,

(TKBO2c) $\exists\, f/m \in \Sigma$ $(m \in \mathbb{N}^{>0})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_m \in \mathrm{T}_\Sigma(X)$, $\exists\, i \in \mathbb{N}, 1 \leq i \leq m$ such that $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$ with $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$, $s_i \succ_\mathrm{T} t_i$.

Example 15. Let $\Omega(x) = 1$, $\Omega(h) = \Psi(h) = 1$, $\Omega(g) = \Psi(g) = \omega$, $\Omega(f) = \Psi(f) = \omega^\omega$. Let $s = f(h(x))$ and $t = g(g(x,x), g(x,x))$. Then $\mathrm{W}(s) = \omega^\omega \cdot 3 > \mathrm{W}(t) = \omega^2 \cdot 6 + \omega$. Furthermore $\bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) = \omega^\omega > \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t) = \omega^2 \cdot 4$. Hence $f(h(x)) \succ_\mathrm{T} g(g(x,x), g(x,x))$ by (TKBO1).

The following two theorems are proved analogously to the corresponding theorems for KBO (e. g. in Baader and Nipkow [1]):

Theorem 16. *The transfinite Knuth-Bendix ordering $\succ_T$ is a simplification ordering.*

Theorem 17. *If the precedence* $\sqsupset$ *is a total ordering, then the transfinite Knuth-Bendix ordering* $\succ_{\mathrm{T}}$ *is total on ground terms.*

For terms built from symbols with subterm coefficient 1 and natural numbers as weights, the transfinite Knuth-Bendix ordering $\succ_{\mathrm{T}}$ agrees with $\succ_{\mathrm{KBO}}$:

Theorem 18. *Let* Σ_0 *be a subsignature of* Σ *such that* $\Psi(f) = 1$ *and* $\Omega(f) \in \mathbb{N}$ *for all* $f \in \Sigma_0$. *Let* $\succ_{KBO}$ *be the regular Knuth-Bendix ordering on* $\mathrm{T}_{\Sigma_0}(X)$ *with* $\lambda(f) = \Omega(f)$ *for* $f \in \Sigma_0$. *Then, for all terms* $s, t \in \mathrm{T}_{\Sigma_0}(X)$, $s \succ_T t$ *if and only if* $s \succ_{KBO} t$.

4 Ordering Definition Equations

The TKBO is able to orient every set of non-recursive, but possibly non-linear definition equations from left to right, i. e. in an intuitive way. Moreover, if the set of definition equations is finite and given a priori, this is possible even with natural numbers as weights and subterm coefficients:

Suppose that we have a sequence of signatures Σ_i $(0 \le i \le n)$ where $\Sigma_i = \{f_i\} \cup \Sigma_{i-1}$ for $i \ge 1$, and that we have a set of non-recursive definition equations of the form $f_i(s_1, \ldots, s_k) \approx t$, with $t, s_j \in \mathrm{T}_{\Sigma_{i-1}}(X)$ and $\mathrm{Var}(t) \subseteq \mathrm{Var}\,(f_i(s_1, \ldots, s_k))$ (where the s_j are not necessarily variables). We start with arbitrary natural numbers as weights and subterm coefficients for the symbols in Σ_0. Then, for $i = 1, \ldots, n$, we recursively choose $\Omega(f_i)$ and $\Psi(f_i)$ in such a way that $\Omega(f_i) > \mathrm{W}(t)$ and $\Psi(f_i) \ge \max_{x \in \mathrm{Var}(t)} \big(\sum_{p \in \mathrm{P}(x,t)} \mathcal{C}(p, t)\big)$ for every definition equation $f_i(s_1, \ldots, s_k) \approx t$ for f_i. It is clear that this construction implies $f_i(s_1, \ldots, s_k) \succ_{\mathrm{T}} t$ by condition (TKBO1).

If we want to have the property that then [illegible] term t with a top symbol f_i is larger than [illegible] term in $\mathrm{T}_{\Sigma_{i-1}}(\mathrm{Var}(t))$, this is still possible with the transfinite Knuth-Bendix ordering, but now we have to use ordinal numbers beyond ω:

Theorem 19. *Let* Σ_0 *be a subsignature of* Σ *and let* $i \in \mathbb{N}$ *such that* $\Psi(f) < \omega^{\omega^i}$ *and* $\Omega(f) < \omega^{\omega^i}$ *for all* $f \in \Sigma_0$ *and* $\Psi(f) \ge \omega^{\omega^i}$ *and* $\Omega(f) \ge \omega^{\omega^i}$ *for all* $f \in \Sigma \setminus \Sigma_0$. *Let* s *be a term with top symbol in* $\Sigma \setminus \Sigma_0$, *let* $t \in \mathrm{T}_{\Sigma_0}(\mathrm{Var}(s))$ *be a term over* Σ_0 *and the variables of* s. *Then* $s \succ_T t$ *holds.*

Corollary 20. *If we have a a sequence of signatures* Σ_i, $i = 0, \ldots, n$, *where* $\Sigma_i = \{f_i\} \cup \Sigma_{i-1}$ *for* $i \ge 1$, *and an arbitrary KBO* $\succ_{KBO}$ *on* $\mathrm{T}_{\Sigma_0}(X)$ *with weights in* $\mathbb{N}$, *then defining* $\Psi(f_i) = \Omega(f_i) = \omega^{\omega^i}$ *yields a transfinite KBO that agrees with* $\succ_{KBO}$ *on* $\mathrm{T}_{\Sigma_0}(X)$ *and in which moreover every term* s *with top symbol* f_i *is larger than every term in* $\mathrm{T}_{\Sigma_{i-1}}(\mathrm{Var}(s))$.

It is clear that the transfinite Knuth-Bendix ordering is computable: Ordinals from **O** can easily be encoded as nested list structures, on which the Hessenberg operations can be performed. Neither addition nor multiplication can be performed in constant time, though. Consequently, the efficiency advantage of the KBO over the LPO is essentially lost, and Cor. 20 is mostly a theoretical result. On the other hand, both the criteria from Thm. 18 and from Thm. 19 can be efficiently checked, and together, they are often sufficient in practice. Cor. 20 then ensures that completeness proofs, etc., which require the existence of a reduction ordering total on ground terms still hold.

5 Hierarchic KBO

5.1 Simple Simplification Orderings

As mentioned earlier, for refutational completeness a hierarchic proof calculus that operates on a base signature Σ_0 and an extension $\Sigma \supseteq \Sigma_0$ of Σ_0 needs a reduction ordering $\succ$ that is total on ground terms and has the property that every ground term in $\mathrm{T}_{\Sigma_0}(\emptyset)$ is strictly smaller than every ground term in $\mathrm{T}_{\Sigma}(\emptyset) \setminus \mathrm{T}_{\Sigma_0}(\emptyset)$. It is easy to see that the transfinite Knuth-Bendix ordering satisfies this property, for instance, if the weight symbol assignment Ω maps every symbol in Σ_0 to a natural number and every symbol in $\Sigma \setminus \Sigma_0$ to an ordinal number $\omega \cdot m + n$ with $m > 0$, and if $\Psi(f) \in \mathbb{N}$ for all $f \in \Sigma$. Note that ordinals of the form $\omega \cdot m + n$ with $m \geq 0$, $n \geq 0$, can be written as tuples (m, n); the Hessenberg addition then corresponds to the componentwise addition of tuples, the Hessenberg multiplication with positive integers to scalar multiplication, and the ordering on ordinals is equivalent to the lexicographic ordering over $\mathbb{N} \times \mathbb{N}$.[5]

Moreover, a small refinement of the transfinite Knuth-Bendix ordering for the hierarchic case is possible: In hierarchic superposition calculi there may be variables for which we only have to consider instantiations with terms from $\mathrm{T}_{\Sigma_0}(X)$ and other variables for which we only have to consider instantiations with terms from $\mathrm{T}_{\Sigma}(X) \setminus \mathrm{T}_{\Sigma_0}(X)$.[6] This motivates a relaxation of the definitions of reduction and simplification orderings.

Definition 21 (Simple Substitution). Let Σ_0, Σ be two signatures such that $\Sigma_0 \subseteq \Sigma$ and let X_{l}, X_{s}, X_{u} be disjoint sets of variables with $X = X_{\mathrm{l}} \cup X_{\mathrm{s}} \cup X_{\mathrm{u}}$. We say that a substitution $\sigma\colon X \to \mathrm{T}_{\Sigma}(X)$ is a *simple substitution* for $(X_{\mathrm{l}}, X_{\mathrm{s}}, X_{\mathrm{u}})$ and $\Sigma_0 \subseteq \Sigma$ if $\sigma(x) \in \mathrm{T}_{\Sigma_0}(X_{\mathrm{s}})$ for all $x \in X_{\mathrm{s}}$ and $\sigma(x) \in \mathrm{T}_{\Sigma}(X) \setminus \mathrm{T}_{\Sigma_0}(X_{\mathrm{s}} \cup X_{\mathrm{u}})$ for all $x \in X_{\mathrm{l}}$.

In other words, variables in X_{s} ("small variables") may only be mapped to terms over base symbols and small variables ("small terms"); variables in X_{l} ("large variables") may only be mapped to terms containing at least one proper extension symbol or large variable ("large terms"); for variables in X_{u} ("unspecified variables") there is no restriction.

The next definition is analogous to Def. 1 and introduces the concept of simple simplification orderings.

---

[5] A very restricted case of such a behaviour can also be found in the DomPred mechanism implemented in Vampire [16] and SPASS [17].

[6] A similar requirement appears in superposition for finite domains (Hillenbrand and Weidenbach [8]), where it is sufficient to consider substitutions that map variables to a given set of constant symbols (which are smaller than complex terms).

Definition 22 (Simple Simplification Ordering). Let Σ_0, Σ be two signatures such that $\Sigma_0 \subseteq \Sigma$ and let X_l, X_s, X_u be disjoint sets of variables with $X = X_\mathrm{l} \cup X_\mathrm{s} \cup X_\mathrm{u}$. Furthermore, let $\succ \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ be a binary relation on terms. Then we say that

- $\succ$ is [illegible] if for all terms $s, s' \in \mathrm{T}_\Sigma(X)$ and for all simple substitutions $\sigma \in \mathrm{Subst}_{X,\Sigma}$ for $(X_\mathrm{l}, X_\mathrm{s}, X_\mathrm{u})$ and $\Sigma_0 \subseteq \Sigma$ it holds that $s \succ s' \implies s\sigma \succ s'\sigma$,
- $\succ$ is a [illegible] if $\succ$ is compatible with Σ-operations and stable under simple substitutions,
- $\succ$ is a [illegible] if $\succ$ is a strict partial ordering and a simple rewrite relation,
- $\succ$ is a [illegible] if $\succ$ is a simple rewrite ordering and has the subterm property.

Note that if $X_\mathrm{s} = X_\mathrm{l} = \emptyset$, the notion of simple simplification ordering coincides with the notion of (regular) simplification ordering.

5.2 Constructing the Ordering

In order to turn the transfinite Knuth-Bendix ordering into a simple simplification ordering, we use ordinals of the form $\omega \cdot m + n$ with $m \geq 0$, $n \geq 0$ as weights and positive integers as subterm coefficients.

Definition 23 (Admissible Hierarchic Symbol Weight Assignment). Let Σ_0, Σ be signatures such that $\Sigma_0 \subseteq \Sigma$ and let X_l, X_s, X_u be pairwise disjoint sets of variables with $X = X_\mathrm{l} \cup X_\mathrm{s} \cup X_\mathrm{u}$. Additionally, let $\sqsupset$ be a strict partial ordering on Σ and let $\Omega\colon \Sigma \cup X \to \mathbf{O}$ be a symbol weight assignment. We say that Ω [illegible] $\sqsupset$ [illegible] Σ_0 if the following conditions are satisfied:

(i) There exists an $\Omega_0 \in \mathbb{N}^{>0}$ such that for all $x \in X_\mathrm{s} \cup X_\mathrm{u}$: $\Omega(x) = \Omega_0$ and such that for all $c/0 \in \Sigma$: $\Omega(c) \geq \Omega_0$;
(ii) For all $f \in \Sigma_0$: $\Omega(f) \in \mathbb{N}$.
(iii) There exists an $\Omega_1 = \omega \cdot m + n$ with $m \in \mathbb{N}^{>0}$, $n \in \mathbb{N}$, such that for all $x \in X_\mathrm{l}$: $\Omega(x) = \Omega_1$ and such that for all $f \in \Sigma \backslash \Sigma_0$: $\Omega(f) = \omega \cdot m' + n' \geq \Omega_1$;
(iv) If there is a symbol $f/1 \in \Sigma$ such that $\Omega(f) = 0$, then $f \sqsupseteq g$ for all $g \in \Sigma$.

The extension from symbol weights to term weights is defined as before. We can now introduce the [illegible] (HKBO). Compared with the transfinite Knuth-Bendix ordering (Def. 14) there are two major differences: the new case (HKBO1′) implies that small variables can essentially be ignored if the weight difference of the two terms is large enough, and a change in the definition of admissible symbol weight assignments enforces large variables to get assigned the weight Ω_1, which is greater than the weight of every symbol from Σ_0.

Definition 24 (Hierarchic Knuth-Bendix Ordering). Let Σ_0, Σ be signatures such that $\Sigma_0 \subseteq \Sigma$, let X_l, X_s, X_u be pairwise disjoint sets of variables with

$X = X_\mathrm{l} \cup X_\mathrm{s} \cup X_\mathrm{u}$. In addition, let $\sqsupset$ be a strict partial ordering, the precedence, on Σ, let $\Omega\colon \Sigma \cup X \to \{\, \omega \cdot m + n \mid m, n \in \mathbb{N} \,\}$ be a hierarchic symbol weight assignment that is admissible for $\sqsupset$ and Σ_0, and let $\Psi\colon \Sigma \to \mathbb{N}^{>0}$ be a subterm coefficient function. Finally, let $\mathrm{W} = \mathrm{W}_{(\Omega,\, \Psi)}\colon \mathrm{T}_\Sigma(X) \to \mathbf{O}$ be the ordinal term weight function induced by Ω and Ψ.

We define the *hierarchic Knuth-Bendix ordering* (HKBO) $\succ_\mathrm{H} \subseteq \mathrm{T}_\Sigma(X) \times \mathrm{T}_\Sigma(X)$ induced by $(\sqsupset, \Omega)$ on terms $s, t \in \mathrm{T}_\Sigma(X)$ in the following way:
$s \succ_\mathrm{H} t$ if

(HKBO1) $\mathrm{Var}(s) \supseteq \mathrm{Var}(t)$, $\mathrm{W}(s) > \mathrm{W}(t)$ and

$$\forall\, x \in \mathrm{Var}(t)\colon \bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) \geq \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t)$$

or

(HKBO1′) $\mathrm{Var}(s) \supseteq \mathrm{Var}(t) \cap (X_\mathrm{l} \cup X_\mathrm{u})$, $\mathrm{W}(s) = \omega \cdot m + n$, $\mathrm{W}(t) = \omega \cdot m' + n'$, $m > m'$ and

$$\forall\, x \in \mathrm{Var}(t) \cap (X_\mathrm{l} \cup X_\mathrm{u})\colon \bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) \geq \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t)$$

or

(HKBO2) $\mathrm{Var}(s) \supseteq \mathrm{Var}(t)$, $\mathrm{W}(s) = \mathrm{W}(t)$,

$$\forall\, x \in \mathrm{Var}(t)\colon \bigoplus_{p \in \mathrm{P}(x,\, s)} \mathcal{C}(p,\, s) \geq \bigoplus_{p \in \mathrm{P}(x,\, t)} \mathcal{C}(p,\, t)$$

and one of the following cases occurs:

(HKBO2a) $\exists\, f/1 \in \Sigma$, $\exists\, x \in X$, $\exists\, n \in \mathbb{N}^{>0}$ such that $s = f^n(x)$ and $t = x$,

(HKBO2b) $\exists\, f/m, g/n \in \Sigma$ $(m, n \in \mathbb{N})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_n \in \mathrm{T}_\Sigma(X)$ such that $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$ with $f \sqsupset g$,

(HKBO2c) $\exists\, f/m \in \Sigma$ $(m \in \mathbb{N}^{>0})$, $\exists\, s_1, \ldots, s_m, t_1, \ldots, t_m \in \mathrm{T}_\Sigma(X)$, $\exists\, i \in \mathbb{N}, 1 \leq i \leq m$ such that $s = f(s_1, \ldots, s_m)$, $t = f(t_1, \ldots, t_m)$ with $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$, $s_i \succ_\mathrm{T} t_i$.

It is easy to show that terms built over Σ_0 and "small" variables are smaller w. r. t. the HKBO than terms which contain at least one large variable or one symbol from $\Sigma \setminus \Sigma_0$, as required for hierarchic superposition:

Lemma 25. *For every term $s \in \mathrm{T}_\Sigma(X) \setminus \mathrm{T}_{\Sigma_0}(X_\mathrm{s} \cup X_\mathrm{u})$ and for every term $t \in \mathrm{T}_{\Sigma_0}(X_\mathrm{s})$ we have $s \succ_H t$.*

Proof. By part (iii) of Def. 23, we have $\mathrm{W}(s) \geq \omega$; by part (i) and (ii) we have $\omega > \mathrm{W}(t)$; hence $s \succ_\mathrm{H} t$ by (HKBO1′).

The following theorems are proved analogously to the corresponding propositions for TKBO:

Theorem 26. *The hierarchic Knuth-Bendix ordering $\succ_H$ is a simple simplification ordering.*

Theorem 27. *If the precedence $\sqsupset$ is a total ordering, then the hierarchic Knuth-Bendix ordering $\succ_H$ is total on ground terms.*

Furthermore, if we restrict to subterm coefficient functions that map every symbol to 1, then Löchner's proof [12] that KBO can be computed in linear time can easily be extended to HKBO:

Theorem 28. *If $\Psi(f) = 1$ for all $f \in \Sigma$, then there exists an algorithm with worst-case time complexity $\mathrm{O}(|s| + |t|)$ that tests for two terms s and t whether $s = t$, $s \succ_H t$, $t \succ_H s$, or s and t are incomparable.*

6 Conclusions

We have described a generalisation of the Knuth-Bendix ordering that possesses certain properties that are typical for LPO, such as the usability for hierarchic theorem proving or the ability to handle non-linear definition equations adequately.

As long as we restrict ourselves to subterm coefficient functions that map every signature symbol to 1, the transfinite and the hierarchic KBO not only inherit the general computation scheme of KBO but also its runtime behaviour, which in particular turns the HKBO into a useful tool for actual implementations of hierarchic theorem proving. In SPASS+T [15], we have implemented a three-level version of the HKBO, with numeric constants on the lowest level, numeric operators and predicates on the middle level, and other operators and predicates on the top level. This ordering ensures that (a) terms and literals are primarily compared using their non-numeric parts; (b) terms that differ only by their numeric constants are essentially compared by the sum of the absolute values of these constants, e. g., $g(20, 4) \succ g(5, 5)$ and $g(4, 20) \succ g(5, 5)$; (c) complex numeric expressions are always larger than the numbers to which they evaluate, e. g., $4 \cdot 5 \succ 20$.

On the other hand, choosing subterm coefficients that are larger than 1 is clearly detrimental to the runtime behaviour of the TKBO. This holds already when positive integers greater than 1 are used as subterm coefficients (in this case one needs arbitrary precision integer arithmetic), and even more so when one uses ordinal numbers beyond ω as subterm coefficients. In its full generality, the transfinite KBO is mostly a theoretical device which ensures that using the regular KBO on "small terms" and applying definition equation on "large terms" are both compatible with a single reduction ordering over the whole signature that is total on ground terms and whose existence may be required for refutational completeness of a calculus. Actual computation of the TKBO is possible, but it is essentially a last resort.

Acknowledgements. We are grateful to Andrei Voronkov and the LPAR reviewers for providing useful comments on previous versions of this paper.

References

1. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press, New York (1998)
2. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. Journal of Logic and Computation 4(3), 217–247 (1994)
3. Bachmair, L., Ganzinger, H., Waldmann, U.: Refutational theorem proving for hierarchic first-order theories. Applicable Algebra in Engineering, Communication and Computing (AAECC) 5(3/4), 193–212 (1994)
4. Dick, J., Kalmus, J., Martin, U.: Automating the Knuth-Bendix ordering. Acta Informatica 28(2), 95–119 (1990)
5. Fernández, M.-L., Godoy, G., Rubio, A.: Recursive path orderings can also be incremental. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 230–245. Springer, Heidelberg (2005)
6. Ganzinger, H., Sofronie-Stokkermans, V., Waldmann, U.: Modular proof systems for partial functions with Evans equality. Information and Computation 204, 1453–1492 (2006)
7. Hessenberg, G.: Grundbegriffe der Mengenlehre. Vandenhoeck & Ruprecht, Göttingen (1906)
8. Hillenbrand, T., Weidenbach, C.: Superposition for finite domains. Research Report MPI-I-2007-RG1-002, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany (April 2007)
9. Just, W., Weese, M.: Discovering modern set theory. I: The Basics, Graduate Studies in Mathematics, vol. 8. American Mathematical Society (1996)
10. Kamin, S., Lévy, J.-J.: Attempts for generalising the recursive path orderings. Manuscript Department of Computer Science, University of Illinois, Urbana-Champaign (1980), available at `http://perso.ens-lyon.fr/pierre.lescanne/not_accessible.html`
11. Knuth, D.E., Bendix, P.B.: Simple word problems in universal algebras. In: Leech, J. (ed.) Computational Problems in Abstract Algebra, pp. 263–297. Pergamon Press, Oxford (1970)
12. Löchner, B.: Things to know when implementing KBO. Journal of Automated Reasoning 36, 289–310 (2006)
13. Ludwig, M.: Extensions of the Knuth-Bendix ordering with LPO-like properties. Diploma thesis, Universität des Saarlandes, Saarbrücken, Germany (July 2006)
14. McCune, W.: Otter 3.3 Reference Manual. Argonne National Laboratory, Argonne, IL, USA, Technical Memorandum No. 263 (August 2003)
15. Prevosto, V., Waldmann, U.: SPASS+T. In: Sutcliffe, G., Schmidt, R., Schulz, S. (eds.) ESCoR: FLoC 2006 Workshop on Empirically Successful Computerized Reasoning, Seattle, WA, USA. CEUR Workshop Proceedings, vol. 192, pp. 18–33 (August 2006)
16. Riazanov, A., Voronkov, A.: The design and implementation of Vampire. AI Communications 15, 91–110 (2002)
17. Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobalt, C., Topić, D.: SPASS version 2.0. In: Voronkov, A. (ed.) CADE-18. LNCS (LNAI), vol. 2392, pp. 275–279. Springer, Heidelberg (2002)

Retractile Proof Nets of the Purely Multiplicative and Additive Fragment of Linear Logic

Roberto Maieli*

Dipartimento di Filosofia
Università degli Studi "Roma Tre"
`maieli@uniroma3.it`

Abstract. Proof nets are a parallel syntax for sequential proofs of linear logic, firstly introduced by Girard in 1987. Here we present and intrinsic (geometrical) characterization of proof nets, that is a correctness criterion (an algorithm) for checking those proof structures which correspond to proofs of the purely multiplicative and additive fragment of linear logic. This criterion is formulated in terms of simple graph rewriting rules and it extends an initial idea of a retraction correctness criterion for proof nets of the purely multiplicative fragment of linear logic presented by Danos in his Thesis in 1990.

1 Introduction

Proof nets are a parallel syntax (a graphical presentation) for sequential proofs of linear logic (LL), firstly introduced by Girard in [3]. An interesting challenge is to find intrinsic (geometrical) characterizations of proof nets, that is correctness criteria (naively, algorithms) for checking those proof structures which correspond to LL proofs; this is particularly true for proof nets of the pure multiplicative and additive fragment of linear logic (MALL).

Our starting idea is that correctness for MALL proof nets should be formulated as simple as possible, following the spirit of correctness for proof nets of the pure multiplicative fragment of linear logic (MLL, see [3] and [1]). In our work correctness is formulated by an algorithm which implements simple graph rewriting rules. In particular, we extend an initial idea of a retraction correctness criterion for MLL proof nets presented in Danos's Thesis ([2]) and subsequently reformulated as a parsing criterion for MELL proof nets by Guerrini and Masini ([6]). Naively, retractility is a way to simulate sequentialization steps: each retracted (sub)graph corresponds to a correct (sequentializable) (sub)proof structure. Compared with other existing syntaxes for MALL proof nets, like that

---

* The author thanks Michele Abrusci, Paul Ruet and the anonymous referees for their suggestions and remarks. This work was partially supported by the MIUR Project *Rete italo-francese di ricerca in logica e geometria della computazione* (Prot. IIO4CE27L4).

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 363–377, 2007.
© Springer-Verlag Berlin Heidelberg 2007

one due to Girard ([4]) or Hughes-van Glabbeek ([7]), our retractile correctness criterion does not rely on any notion of [illegible] or [illegible]. This effort should simplify the complexity of checking correctness. However here we do not discuss complexity aspects of our criterion; moreover, for simplicity reasons, we restrict to consider only cut-free proof nets.

After recalling, in next sub-section, some basic notions of the MALL fragment we introduce, in Section 2, a notion of (abstract) proof structure; then, in Section 3, we characterize correctness in terms of a rewriting algorithm which is shown confluent, correct (sequentializable) and complete (de-sequentializable) w.r.t. MALL sequent calculus. Finally, in Section 4, we discuss some directions in the way we could extend our criterion to proof nets with cuts.

1.1 The MALL Fragment of Linear Logic

MALL formulas $A, B, ...$ are built from literals (propositional variables $P, Q, ...$ and their negations $P^{\perp}, Q^{\perp}, ...$) by the binary connectives $\otimes$ ([illegible]), $\parr$ ([illegible]), $\&$ ([illegible]) and $\oplus$ ([illegible]). Negation $(.)^{\perp}$ extends to arbitrary formulas by the de Morgan laws: $(A \otimes B)^{\perp} = (A^{\perp} \parr B^{\perp})$, $(A \parr B)^{\perp} = (A^{\perp} \otimes B^{\perp})$, $(A \& B)^{\perp} = (A^{\perp} \oplus B^{\perp})$, and $(A \oplus B)^{\perp} = (A^{\perp} \& B^{\perp})$. A MALL sequent Γ is a non empty set of formula occurrences $A_1, ..., A_n$. We omit turnstiles ($\vdash$) since all sequents are right-sided. Sequents are proved using the following rules:

$$\frac{}{A, A^{\perp}}\,\text{ax} \qquad \frac{\Gamma, A \qquad \Delta, A^{\perp}}{\Gamma, \Delta}\,\text{cut} \qquad \frac{\Gamma, A \qquad \Delta, B}{\Gamma, \Delta, A \otimes B}\,\otimes \qquad \frac{\Gamma, A, B}{\Gamma, A \parr B}\,\parr$$

$$\frac{\Gamma, A \qquad \Gamma, B}{\Gamma, A \& B}\,\& \qquad \frac{\Gamma, A}{\Gamma, A \oplus B}\,\oplus_1 \qquad \frac{\Gamma, B}{\Gamma, A \oplus B}\,\oplus_2$$

2 Proof Structures

Definition 1 (proof structure). [illegible] ([illegible]) proof structure [illegible] PS [illegible] ([illegible]) [illegible] conclusions [illegible] premise [illegible] conclusion [illegible] link [illegible] ( ) [illegible]

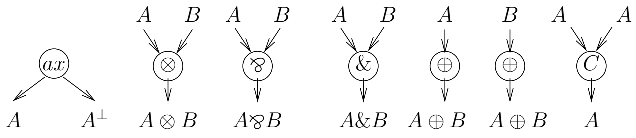
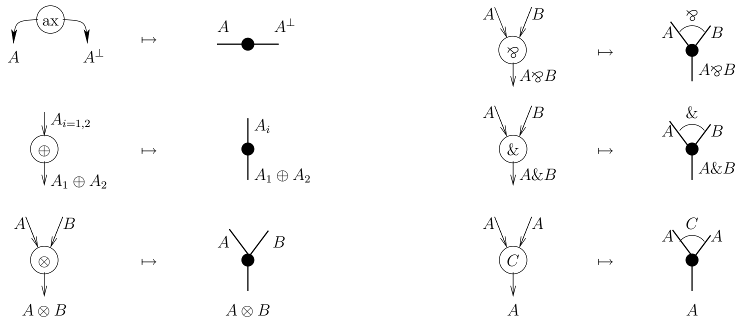


Fig. 1. MALL links

Definition 2 (abstract proof structure). [illegible] abstract structure (AS) [illegible] G [illegible] $\mathcal{C}(G)$ [illegible] ([illegible] coincident [illegible]) [illegible] pair [illegible] $\mathcal{C}(G)$ [illegible] base [illegible] ([illegible]) [illegible] ([illegible]) [illegible] arc [illegible]

[illegible] abstract proof structure (APS) [illegible] ⅋, & [illegible] C [illegible] (mapping [illegible])



ax
A
$A^\perp$
↦
A
$A^\perp$
A
B
⅋
$A ⅋ B$
↦
⅋
A
B
$A ⅋ B$
$A_{i=1,2}$
⊕
$A_1 \oplus A_2$
↦
A_i
$A_1 \oplus A_2$
A
B
&
$A \& B$
↦
&
A
B
$A \& B$
A
B
⊗
$A \otimes B$
↦
A
B
$A \otimes B$
A
A
C
A
↦
C
A
A
A

Fig. 2. Mapping PS in to APS

[illegible]: if π is a PS then π^* denotes its corresponding APS; variables $e_1,\ e_2, ...$ denote edges and $v_1,\ v_2, ...$ denote vertices of an APS; a dotted edge incident to a vertex v and (eventually) labelled by variables $a, b, ...$, is a compact representation of possibly several edges incident to v; finally, $\delta(v)$ states for the [illegible] (the number of incident edges) of a vertex v.

Definition 3 (multiplicative retraction). [illegible] multiplicative retraction [illegible] π [illegible] π [illegible] π' ([illegible] $\pi \rightsquigarrow \pi'$) [illegible]

R_1 ([illegible]) [illegible] π
- [illegible] v_1 [illegible] v_2 [illegible]
- [illegible] e_1 [illegible] $\mathcal{C}(\pi)$

R_2 ([illegible]) [illegible] π
- [illegible] v_1 [illegible] v_2 [illegible]
- [illegible] e_1 [illegible] e_2 [illegible] ⅋ [illegible]

Definition 4 (additive retraction)
[illegible] additive retraction [illegible] π [illegible] π [illegible] π' [illegible]

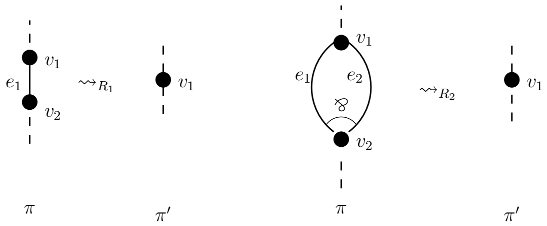

v_1
e_1
v_2
$\rightsquigarrow_{R_1}$
v_1
π
π'
v_1
e_1
e_2
⅋
v_2
$\rightsquigarrow_{R_2}$
v_1
π
π'

Fig. 3. Multiplicative retraction rules R_1 and R_2

R_3 ([illegible]) [illegible] π

- [illegible] v_i, $1 \leq i \leq 4$ [illegible]
- [illegible] e_1 [illegible] e_2 [illegible] C [illegible]

R_4 ([illegible]) [illegible] π

- [illegible] v_i, $1 \leq i \leq 3$ [illegible]
- [illegible] e_1 [illegible] e_2 [illegible] & [illegible]
- $\delta(v_2) = 1$ [illegible] $\delta(v_3) = 1$

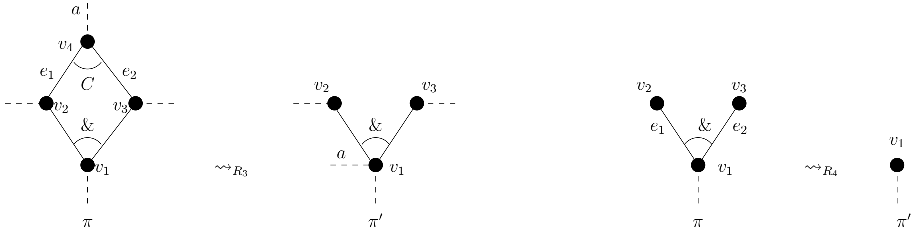

a
v_4
e_1
e_2
C
v_2
v_3
&
v_1
π
$\rightsquigarrow_{R_3}$
v_2
v_3
&
a
v_1
π'
v_2
v_3
e_1
&
e_2
v_1
π
$\rightsquigarrow_{R_4}$
v_1
π'

Fig. 4. Additive retraction rules R_3 and R_4

Definition 5 (distributive retraction). [illegible] distributive retraction [illegible][1] [illegible] π [illegible] π [illegible] π' [illegible] R_5 [illegible] π

- [illegible] v_i, $1 \leq i \leq 8$ [illegible]
- [illegible] e_1 [illegible] e_2 [illegible] C [illegible]
- [illegible] e_3 [illegible] e_4 [illegible] ⅋ [illegible]
- $\delta(v_4) = 2$, $\delta(v_5) = 2$, $\delta(v_6) = 3$ [illegible] $\delta(v_7) = 3$

Fixed a retraction rule R_i, $1 \leq i \leq 5$, the subgraph of π (resp., of π') depicted on the left (resp., on the right) hand side of the $\rightsquigarrow_{R_i}$ map is called the [illegible] (resp., [illegible]) [illegible] of R_i.

We say that two instances R_i and R_j, with $1 \leq i, j \leq 5$, [illegible] (resp., [illegible]) when the intersection of their retraction graphs is not empty (resp.,

---

[1] This rule reflects the *distributive linear law* $(b⅋c)\&(b⅋d) \vdash b⅋(c\&d)$.

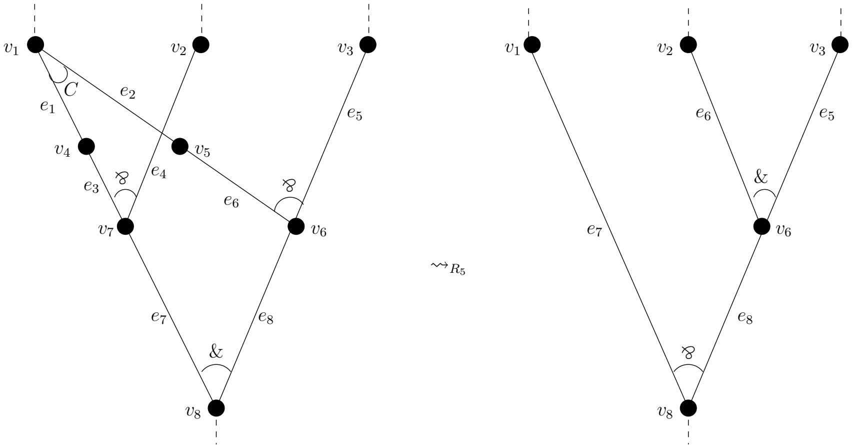


Fig. 5. Distributive retraction rule R_5

empty). R_i and R_j are said [illegible] when they can be applied in any order, i.e. R_i immediately before R_j or R_j immediately before R_i.

An APS π with conclusions $A_1, ..., A_n$ is [illegible] when there exists a sequence of retraction instances starting with π and terminating with a single node ($\bullet$) with n incident edges labelled by $A_1, ..., A_n$.

Definition 6 (proof net). [illegible] π [illegible] $A_1, ..., A_n$ [illegible] $n \geq 1$ [illegible] correct ([illegible] proof net) [illegible] π^* [illegible]

Theorem 1 (confluence). [illegible] π [illegible] π [illegible] ([illegible]) [illegible] π

[illegible] Assume Σ is a retractions sequence $\pi \rightsquigarrow \pi_1 \rightsquigarrow ... \rightsquigarrow \pi_n = \bullet$ with R_i as first retraction (i.e. $\pi \rightsquigarrow_{R_i} \pi_1$) and assume there exists a σ such that $\pi \rightsquigarrow_{R_j} \sigma$. We show that σ is retractile too. We reason by induction on the length l of Σ, where l is the number of retraction instances of Σ.

Assume $l = 1$, then $\pi \rightsquigarrow_{R_i} \pi_1 = \bullet$ and so R_i and R_j must be the same instance with $\sigma = \pi_1$. This follows from the definition of the retraction rules (if R_i and R_j are two different instances then the retraction graphs of R_i and R_j can be disjoint or partially overlapping but not included each other).

Assume $l > 1$, then we split our reasoning in two sub-cases.

1. R_i and R_j are independent. Since, by assumption π_1 is retractile in $n - 1$ steps, then by hypothesis of induction applied to π_1 we conclude that any π_1' obtained by $\pi_1 \rightsquigarrow_{R_j} \pi_1'$ must be retractile. This means that σ is retractile since $\sigma \rightsquigarrow_{R_i} \pi_1'$ and R_i and R_j are independent (see Figure 6).
2. R_i and R_j are not independent; this means that R_i and R_j must be two overlapping instances of the R_5 rule like in left hand side of Figure 7. Again,

$$\begin{tikzcd}
& \sigma \arrow[rd, rightsquigarrow, "R_i"] & & & \\
\pi \arrow[ru, rightsquigarrow, "R_j"] \arrow[rd, rightsquigarrow, "R_i"'] & & \pi'_1 \arrow[r, rightsquigarrow] & \ldots \arrow[r, rightsquigarrow] & \bullet \\
& \pi_1 \arrow[ru, rightsquigarrow, "R_j"] \arrow[r, rightsquigarrow] & \pi_2 \arrow[r, rightsquigarrow] & \ldots \arrow[r, rightsquigarrow] & \bullet : \Sigma
\end{tikzcd}$$

Fig. 6. Confluence of independent retractions

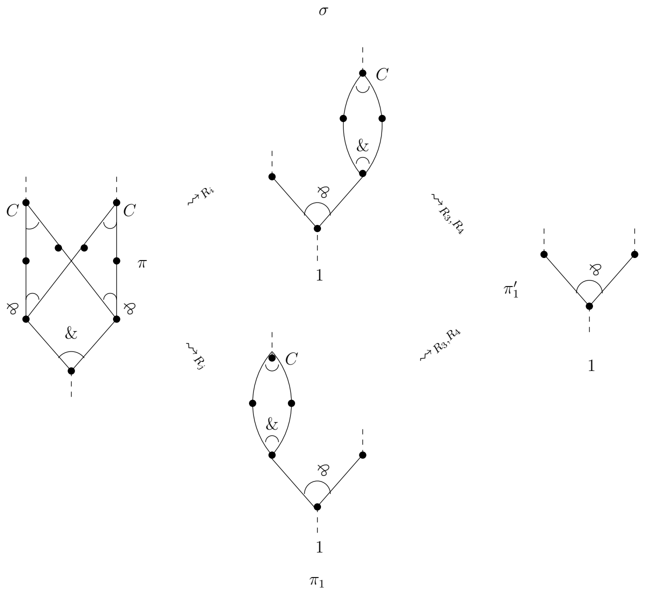


Fig. 7. Confluence of non independent retractions

we reason like in the previous case. Since, by assumption π_1 is retractile in $n-1$ steps, then we can apply the hypothesis of induction to π_1 and conclude that π'_1 is also retractile since $\pi_1 \rightsquigarrow_{R_3,4} \pi'_1$. This means that σ will be retractile too, since $\sigma \rightsquigarrow_{R_3,4} \pi'_1$ (see Figure 6).

3 (De-)Sequentialization

In this section we show that any sequent proof of can be de-sequentialized into a proof net with the same conclusions (Theorem 2) and vice-versa (Theorem 5).

Theorem 2 (de-sequentialization). π^- $\Gamma = A_1, ...,$ $A_{n\geq 1}$ π Γ

[illegible] By induction on the [illegible][2] of the given sequential proof π^-. We only consider the case when last rule of π is a &-rule (the other cases are very simple and we omit them). Assume π^- like in the left hand side of Figure 8, then by hypothesis of induction π_1^- and π_2^- desequentialize respectively into two retractile APS π_1^* and π_2^*, like in the middle side of Figure 8. Clearly the resulting APS π^* (see the right hand side of Figure 8) will be retracted to • by applying (iteratively) rule R_3 and (an instance of) rule R_4.

$$\pi^- : \dfrac{\begin{array}{cc} \pi_1^- & \pi_2^- \\ \Gamma, A & \Gamma, B \end{array}}{\Gamma, A\&B}\ \&$$

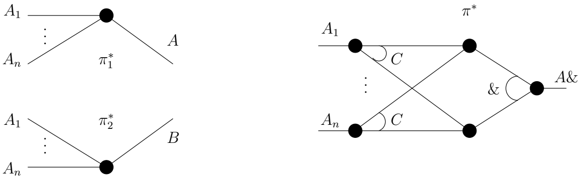


Fig. 8. De-sequentialization of the &-rule

In the following we give an indirect proof of the sequentialization: first we show that any proof net can be weighted in such a way of becoming a [illegible] (Section 3.1), then sequentialization follows as a consequence of Girard's one (Section 3.2).

3.1 Girard's Proof Nets

In this section we recall the basic notions of Girard's proof net; for simplification reasons we adopt the syntax of [9].

Definition 7 (Girard's proof structure). [illegible] proof structure à la Girard (GPS) [illegible] weights [illegible] (weights assignment)

- [illegible] boolean variable [illegible] eigen weight p [illegible] & [illegible] ([illegible] ff [illegible])
- [illegible] weight [illegible] ([illegible]) [illegible] $(p, \overline{p}, q, \overline{q}...)$ [illegible] & [illegible] C [illegible]
- [illegible] conclusion node [illegible] 1
- [illegible] w [illegible] & [illegible] p [illegible] w' [illegible] p [illegible] $w' \le w$ ([illegible] w depends on p [illegible] p [illegible] $\overline{p}$ [illegible] w)

---

[2] The height, $h(\pi^-)$, of π^- is defined inductively as usual. We consider last rule R of π^-: if $R = ax$ then $h(\pi^-) = 1$ otherwise if R is an unary rule, ⅋ or ⊕, (resp., a binary rule, ⊗ or &) then $h(\pi^-) = h(\pi_1^-)+1$ (resp., $h(\pi^-) = max(h(\pi_1^-), h(\pi_2^-))+1$) where π_1^- (resp., π_1^- and π_2^-) is the immediate sub-proof (resp., are the immediate sub-proofs) of π^-.

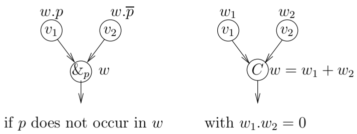

$w.p$
$w.\overline{p}$
v_1
v_2
$\&_p$
w
if p does not occur in w
w_1
w_2
v_1
v_2
C
$w = w_1 + w_2$
with $w_1.w_2 = 0$

Fig. 9. Weights for GPS

A node L with weight w [illegible] the eigen weight p if w depends on p or L is a C-node and one of the weights just above it depends on p.

Definition 8 (slice and switchings). [illegible] valuation φ [illegible] π [illegible] π [illegible] $\{0,1\}$ [illegible] φ [illegible] π [illegible]

- [illegible] slice $\varphi(\pi)$ [illegible] π [illegible] 1 [illegible]
- [illegible] multiplicative switching S [illegible] π [illegible] $\varphi(\pi)$ [illegible] ⅋ [illegible] *(*left/right ⅋-switch*)*
- [illegible] additive switching *(* [illegible] switching*)* [illegible] & [illegible] $\varphi(\pi)$ [illegible] jump [illegible] & [illegible] L [illegible] & [illegible]

Definition 9 (Girard's proof net). [illegible] π [illegible] correct [illegible] proof net à la Girard *(*GPN*)* [illegible] π [illegible] *(*ACC*)*

Theorem 3 (sequentialization). [illegible]

[illegible] omitted (see[4]).

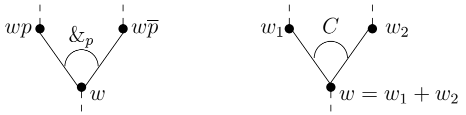
3.2 Sequentialization

Definition 10 (GAPS). [illegible] Girard abstract proof structure [illegible]

fi [illegible] & [illegible] & [illegible] & [illegible] C [illegible]

[illegible] conclusion node [illegible] 1

[illegible] w [illegible] $\&_p$ [illegible] w' [illegible] p [illegible] $w' \leq w$



p does not occur in w with $w_1.w_2 = 0$

Fig. 10. Weights for GAPS

[illegible] The notions of valuation and slice are still well defined w.r.t. GAPS. Only the definition of switching needs a slight modification. Fixed a valuation φ for an GAPS π then:

- a multiplicative switching S for π is the (non oriented) graph built on $\varphi(\pi)$ with the modification that for each ⅋-pair we take only one edge ([illegible] ⅋ [illegible]);
- an additive switching is a multiplicative switching where for each $\&_p$-pair we erase the (unique) edge in $\varphi(\pi)$ and we add a jump from the base of this $\&_p$-pair to an L-node whose weight depends on p.

Lemma 1. [illegible] π [illegible] $\pi \rightsquigarrow_{R_i} \pi'$, [illegible] $1 \leq i \leq 5$, [illegible] π' [illegible] π [illegible]

[illegible] We reason by cases, according to the retraction R_i.

[illegible] R_1. The weight w associated to node v_1 in the retracted graph of R_1 in Figure 3 is inherited by both nodes v_1 and v_2 of the retraction graph. Since all other weights remain unchanged π is a GAPS. Case R_2 is similar.

[illegible] R_2 is similar.

[illegible] R_3. Assume, in the retracted graph of Figure 4, p is the eigen weight of the &-pair, w is the weight of node v_1, wp is the weight of node v_2 and $w\overline{p}$ is the weight of node v_3. We can easily extend this weight assignment to the corresponding nodes of the retraction graph: it is easy to verify that this assignment preserves the property of being a AGPS w.r.t. π, since all pair of $\mathcal{C}(\pi)$ are pairwise disjoint and all other weights remain unchanged.

[illegible] R_4. Assume node v_1 has weight w in the retracted graph of Figure 4; then w is trivially inherited by the corresponding node v_1 of the retraction graph. Now, chosen a new eigen weight p for the &-pair, we can assign weights wp and $w\overline{p}$ to nodes, respectively, v_2 and v_3. Now, since all other weights remain unchanged, π is a GAPS.

[illegible] R_5. Assume a weight assignment for the retracted graph of Figure 5 as follows: the $\&_p$-pair together with nodes v_1, v_6 and v_8 have the same weight w, while nodes v_2 and v_3 have weights, respectively, wp and $w\overline{p}$. This assignment can be easily extended to π with a slight modifications: the &-pair with base in v_8 inherits the eigen weight p and v_8 inherits the weight w; then weight wp is assigned to v_7, v_4 and v_2, and weight $w\overline{p}$ to v_6, v_5 and v_3; finally weight $w = wp + w\overline{p}$ is assigned to v_1. It is easy to verify that this new assignment preserves the property of being a AGPS w.r.t. π, since all pair of $\mathcal{C}(\pi)$ are pairwise disjoint and all other weights remain unchanged.

What follows is a well known graph theoretical property (see [5], pages 250-251) we will exploit in the proof of the Lemma 2.

[illegible] ([illegible]) Given a graph $\mathcal{G}$, then $\sharp CC - \sharp Cy = \sharp V - \sharp E$, where $\sharp CC$, $\sharp Cy$, $\sharp V$ and $\sharp E$ denotes, respectively, the number of connected components, cycles, vertices and edges of $\mathcal{G}$.

We use the predicate $Gir(\pi)$ for saying that a GAPS is correct in the sense of Definition 9, i.e., any switching $S(\pi)$, w.r.t. a fixed valuation $\varphi(\pi)$, is ACC.

Lemma 2. [illegible] π [illegible] $\pi \rightsquigarrow_{R_i} \pi'$, $1 \leq i \leq 5$, [illegible] $Gir(\pi)$ [illegible] $Gir(\pi')$

[illegible] Let us fix a valuation φ for π. First observe that by Lemma 1 we have only to verify that every switching $S(\pi)$ is ACC. The proof idea, illustrated in Figure 11, relies on the fact that if $\pi \rightsquigarrow_{R_i} \pi'$ then any switching S for π is nothing else that a switching S' for π' except for the fact we replace the switched retracted graph σ' of R_i by a corresponding switched retraction graph σ. We reason by cases, according to the retraction rule R_i.

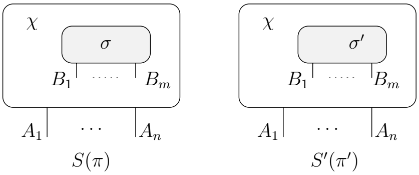


Fig. 11. Recovering $S(\pi)$ from $S'(\pi')$

1. If $\pi \rightsquigarrow_{R_1} \pi'$ (see the left hand side of Figure 3) then trivially any switching S for π can be recovered from a S' for π' where we replaced the retracted graph σ' with the retraction graph σ of Figure 12. Clearly $S(\pi)$ is ACC.

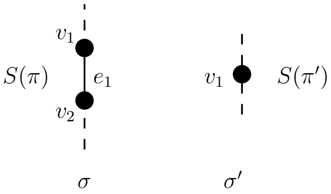


Fig. 12. Recovering $S(\pi)$ from $S'(\pi')$ after a retraction R_1

2. If $\pi \rightsquigarrow_{R_2} \pi'$ (see the right hand side of Figure 3) then any switching S for π can be recovered from a switching S' for π' where we replaced the switched retracted graph σ' by a corresponding switched retraction graph σ (resp., χ) in case we set a left (resp., right) switch for the $\parr$-pair (see Figure 13). Clearly, $S_{\parr_l}(\pi)$ (resp., $S_{\parr_r}(\pi)$) is ACC.

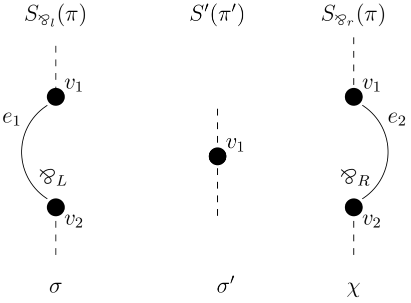


Fig. 13. Recovering $S(\pi)$ from $S'(\pi')$ after a retraction R_2

3. If $\pi \rightsquigarrow_{R_3} \pi'$ (see the left hand side of Figure 4) then any switching S for π can be recovered from a switching S' of π'. We need to consider two cases, according to the jump emerging from the base v_1 of the $\&_p$-pair in S'.
[illegible]: assume S' contains an immediate jump j from the base v_1 of the $\&_p$-pair to its (unique) premise v_2 (see the right hand side of Figure 14). Then S for π can be recovered from S' where we replaced the switched retracted graph σ' by the corresponding switched retraction graph σ on the left hand side of Figure 14. We have to show that S is ACC. First, observe that edges a, b and c are connected in S as well in S', so the number of connected components in S is 1. Moreover in S the difference $\sharp V - \sharp E$ must be the same as that one in S', that is 1, since in S there is only one more edge and one more vertex than in S'. So by the Euler-Poincaré invariance in S we have $\sharp CC - \sharp Cy = 1$, therefore $\sharp Cy$ in S must be 0. So S is ACC.

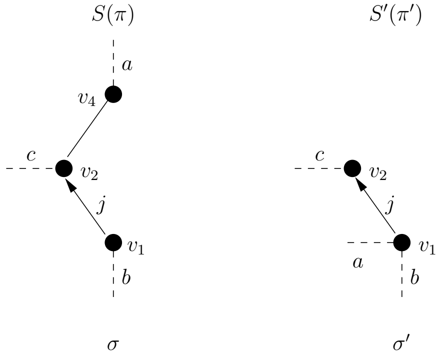


Fig. 14. Recovering $S(\pi)$ from $S'(\pi')$, with an immediate jump, after a retraction R_3

[illegible]: S contains a remote jump j from the base v_1 of the $\&_p$ pair to a node v depending on p that is different from the (unique) premise of $\&_p$ (see the right hand side of Figure 15). Clearly S can be recovered from a switching S' where we replaced the switched retracted graph σ' by the switched retraction graph σ on the left hand side of Figure 15. We show that S is ACC. Although vertices v_1 and v_2 are connected in S' by assumption, they cannot be connected through a, otherwise we could easily set a switching S'' for π' that is identical to S' except for the immediate jump from v_1 to v_2 and get a cycle, contradicting the assumption $Gir(\pi')$. This means that the retraction step R_3 preserves backwards the connection of S, so $\sharp CC$ of S is 1. Now, observe that the difference $\sharp V - \sharp E$ of S is the same as that one of S' (i.e. 1), so by the Euler-Poincaré invariance we have, in S, $\sharp CC - \sharp Cy = 1$. This means that, in S, $\sharp Cy$ must be 0. So S is ACC.

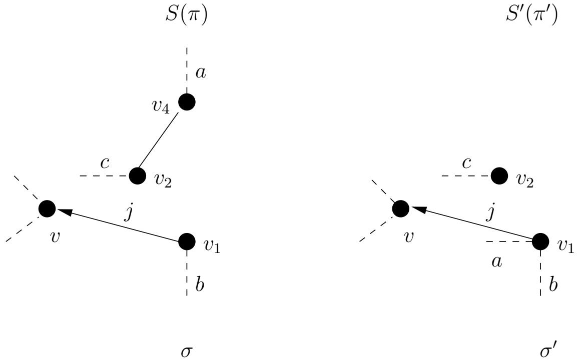


Fig. 15. Recovering $S(\pi)$ from $S'(\pi')$, with a remote jump, after a retraction R_3

4. If $\pi \rightsquigarrow_{R_4} \pi'$ (see the right hand side of Figure 4) then a switching S for π can be recovered from a switching S' for π' plus a jump j emerging from the base v_1 of the $\&_p$ pair. Now, observe this jump j in S can only be directed to its (unique) premise, otherwise: ([illegible]) either there would exist in π' a node whose weight depends on a variable p that is not an eigen weight of any $\&$-pair or ([illegible]) there would exist in π two $\&$-pairs with the same eigen-weight variable p. Both cases contradict that π is an AGPS (Lemma 1).
5. If $\pi \rightsquigarrow_{R_5} \pi'$ (see Figure 5) then a switching S for π is exactly a switching S' for π' except for the the jump emerging from the $\&_p$-pair and the switch for the $\parr$ pair occurring in the retraction R_5. So, let us fix in S' a left $\parr$-switch and jump from the base v_6 of the $\&_p$-pair (the case with the right $\parr$-switch, is analogous): there are two possible jumps.

   [illegible]: assume in $S'(\pi')$ we jump from the base v_6 of the $\&_p$ pair to its (unique) premise. Then a switching S for π is exactly a switching S' for π' except for the fact we replaced the switched retracted graph σ' with σ of Figure 16. Clearly S is connected, so via the Euler-Poincaré invariance, we conclude that S is also acyclic (we reason like in the previous point 3).

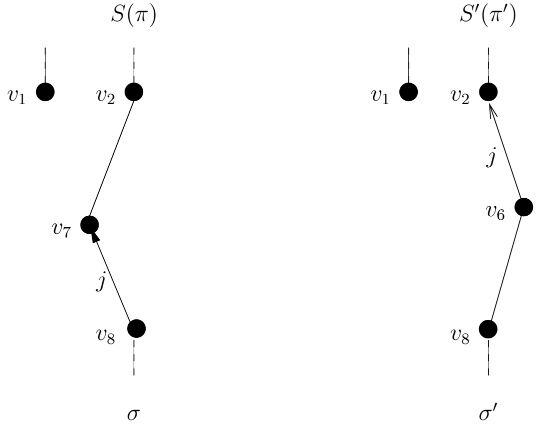


Fig. 16. Recovering $S(\pi)$ form $S'(\pi')$, with immediate jump, after a reduction R_5

[illegible]: assume in $S'(\pi)$ we jump from the base v_6 of the $\&_p$ pair to a remote node v depending on p. Then a switching S for π can be recovered from a switching S' where we replaced the switched retracted graph σ' with σ of Figure 17. Now observe that σ' does not induce any cycle in S, otherwise this cycle would already occur in S'. Moreover, the number of vertices and edges in S is the same as in S' so, via the Euler-Poincaré invariance, we conclude that S is connected.

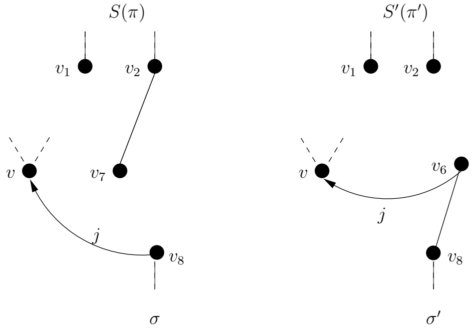


Fig. 17. Recovering $S(\pi)$ form $S'(\pi')$, with a remote jump, after a reduction R_5

Theorem 4 (PN↦GPN). [illegible] π [illegible] π [illegible]

[illegible] If π is a PN then π^* retracts to a node v with possibly several incident edges labelled by the conclusions of π. Trivially v is an AGPS satisfying the predicate *Gir*, then by iteration of Lemma 2 we conclude that π is a GPN.

Theorem 5 (sequentialization). [illegible] π [illegible] Γ [illegible] Γ [illegible]

[illegible] It follows from Girard's sequentialization (see [4]) via Theorem 4.

4 Conclusions and Forthcoming Work

We presented a simple system of graph rewriting rules which can be viewed as a geometrical correction criterion for cut free proof structures of MALL. Each proof structure that is correct in our sense it will be so also in Girard's sense but not vice-versa. In general the other direction of Theorem 4 does not hold: the proof structure π_1, on the left hand side of Figure 18 is not correct for us (it is not retractile); nevertheless we can find a weight assignment transforming π_1 in to a Girard proof net. Actually we only accept correct the proof structure π_2 depicted on the right hand side of Figure 18 which (in our opinion) better embeds the two different sequentializations induced by the permutability of the &-rule w.r.t. the ⊗-rule of the sequent calculus.

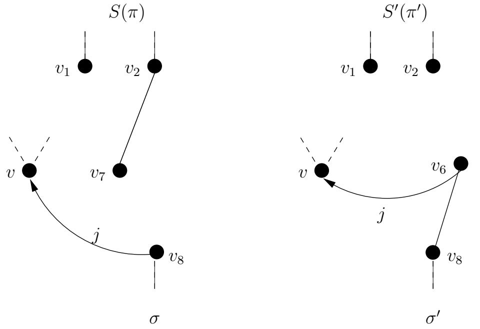


Fig. 18. Examples of PS w.r.t. sequentialization

As future work we aim at comparing the complexity of the retractility correctness criterion w.r.t. Girard and Hughes-van Glabbeek's criteria. Moreover we aim at extending retraction rules in such a way to take in to account proof nets with cuts. At the moment we are investigating some local (commutative) cut reduction steps, following the style of the Interaction Nets ([8]). Our idea is sketched in Figure 19 where the $\star$ symbol states for a binary MALL connective

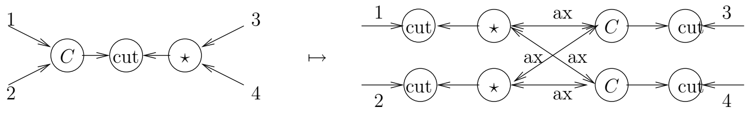


Fig. 19. Commutative cut step reduction

or a contraction C. But in that case the correspondence with monomial GPS is lost: as soon as we replace the $\star$ symbol with the & connective we are suddenly faced to proof structures weighted with polynomials.

References

1. Danos, V., Regnier, L.: The Structure of Multiplicatives. Archive for Mathematical Logic 28, 181–203 (1989)
2. Danos, V.: La Logique Linéaire appliquée à l'étude de divers processus de normalisation (principalment du λ-calcul). PhD Thesis, Univ. Paris VII (Juin 1990)
3. Girard, J.-Y.: Linear Logic. Theoretical Computer Science 50, 1–102 (1987)
4. Girard, J.-Y.: Proof nets: the parallel syntax for proof theory. In: Logic and Algebra, Marcel Dekker (1996)
5. Girard, J.-Y.: Le point aveugle. In: Hermann (ed.) Cours de Logique. Vers la Perfection, Paris, vol. I (2006)
6. Guerrini, S., Masini, A.: Parsing MELL proof nets. Theoretical Computer Science 254, 317–335 (2001)
7. Hughes, D., van Glabbeek, R.: Proof Nets for Unit-free Multiplicative-Additive Linear Logic. In: Proc. of IEEE Logic in Computer Science, IEEE Computer Society Press, Los Alamitos (2003)
8. Lafont, Y.: From proof nets to interaction nets. In: Girard, J.-Y., Lafont, Y., Regnier, L. (eds.) Advanced in Linear Logic, pp. 225–247. Cambridge Press, Cambridge (1995)
9. Laurent, O.: Polarized Proof Nets: Proof Nets for LC. In: Girard, J.-Y. (ed.) TLCA 1999. LNCS, vol. 1581, pp. 213–227. Springer, Heidelberg (1999)

Integrating Inductive Definitions in SAT

Maarten Mariën, Johan Wittocx, and Marc Denecker

Department of Computer Science, Katholieke Universiteit Leuven, Belgium
`{maartenm,johan,marcd}@cs.kuleuven.be`

Abstract. We investigate techniques for supporting inductive definitions (IDs) in SAT, and report on an implementation, called MidL, of the resulting solver. This solver was first introduced in [11], as a part of a declarative problem solving framework. We go about our investigation by proposing a new formulation of the semantics of IDs as presented in [2]. This new formulation suggests a way to perform the computational task involved, resulting in an algorithm supporting IDs. We show in detail how to integrate our algorithm with traditional SAT solving techniques. We also point out the similarities with another algorithm that was recently developed for ASP [1]. Indeed, our formulation reveals a very tight relation with stable model semantics. We conclude by an experimental validation of our approach using MidL.

1 Introduction

This work is motivated by the following observations:

- contemporary SAT solvers exhibit impressive performance;
- SAT provides a poor modelling language;
- the ability to express inductive definitions (or recursion) is present in many important knowledge representation formalisms.

The first two observations provide an impetus to extend SAT with language constructs that yield a better modelling language, without giving up too much on the performance side. Adding to this, the third observation shows what would make a better modelling language. Hence this paper focuses on providing computational support for an extension of SAT with (propositional) inductive definitions.

An extension of classical logic with inductive definitions, FO(ID), is given in [2]. A first model generator for the propositional fragment of this logic, SAT(ID), was presented in [10]. An improved version of this solver was reported on in [11], as a part of a declarative problem solving framework. This paper presents the algorithms of this new solver.

Our work is aimed at closing the computational gap between SAT(ID) and "pure" SAT. Hence we try to show how ID support can be integrated in state-of-the-art SAT solving techniques. First, we give an alternative definition of the semantics of SAT(ID). This alternative definition leads to a natural understanding of the computational task of a SAT(ID) solver. In particular, the new definition contains two properties that could be used as invariants for such a solver. We

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 378–392, 2007.
© Springer-Verlag Berlin Heidelberg 2007

show that applying SAT's *unit propagation* technique [14] on (Clark's completion of) an ID suffices to satisfy the first property. A more involved algorithm to satisfy the second property is worked out. It turns out to be very similar to an algorithm for finding unfounded sets proposed by [1].[1] As we show a strong relation between SAT(ID) and the stable model semantics [5], which is the semantics used in Answer Set Programming (ASP) [9], this similarity is not unsurprising.

Combining these two algorithms, we end up with an extension of the DPLL algorithm. As such, a lot of techniques that lead to the impressive performance of SAT solvers can be used in a SAT(ID) solver.

Finally, we present results obtained with MIDL, an implementation of the discussed algorithms. A comparison with some SAT and ASP solvers shows that MIDL is competitive with state-of-the art ASP solvers, while there is still an efficiency gap with SAT solvers on problems containing no recursion.

2 Preliminaries

2.1 SAT(ID)

In this section, we introduce SAT(ID), an extension of propositional logic with inductive definitions. We assume familiarity with propositional logic.

A vocabulary Σ is a set of atoms. A literal is an atom P or its negation $\neg P$. An atom P is called a *positive* literal, $\neg P$ a *negative* one. For a literal L, we identify $\neg\neg L$ with L. For a set S of literals, we denote by $\overline{S}$ the set $\{\neg L \mid L \in S\}$, and by $S^\star$ the set $S \cup \overline{S}$.

A *definition* over Σ is a finite set of rules of the form $P \leftarrow \varphi$ where $P \in \Sigma$ is an atom and φ is an arbitrary propositional formula over Σ. P is called the *head* of the rule and φ the *body*. For a definition Δ, an atom P occuring as head of a rule in Δ is called a *defined* atom of Δ. All other atoms are called *open* atoms of Δ. The set of all defined, respectively open atoms of Δ is denoted by $Def(\Delta)$, respectively $Open(\Delta)$. We say that an atom occurs positively (negatively) in a propositional formula if it occurs in the scope of an even (odd) number of negations. We call a definition Δ *positive* if each occurrence of an atom in the body of a rule in Δ is positive.

A *SAT(ID) theory* is a set of propositional formulae and definitions.

A three-valued Σ-interpretation I is a function $I : \Sigma \to \{\boldsymbol{t}, \boldsymbol{u}, \boldsymbol{f}\}$. An interpretation is two-valued if it maps no atom to $\boldsymbol{u}$. The restriction of I to a set $\sigma \subset \Sigma$ is denoted $I|_\sigma$. The *truth order* $\leq$ on $\{\boldsymbol{t}, \boldsymbol{u}, \boldsymbol{f}\}$ is induced by $\boldsymbol{f} \leq \boldsymbol{u} \leq \boldsymbol{t}$ and the *precision order* $\leq_p$ by $\boldsymbol{u} \leq_p \boldsymbol{f}$ and $\boldsymbol{u} \leq_p \boldsymbol{t}$. Both orders pointwise extend to interpretations. Define $\boldsymbol{f}^{-1} = \boldsymbol{t}$, $\boldsymbol{u}^{-1} = \boldsymbol{u}$ and $\boldsymbol{t}^{-1} = \boldsymbol{f}$. An interpretation I on Σ can be extended inductively to propositional formulae over Σ:

$$I(\varphi \wedge \psi) = \min_{\leq}(\{I(\varphi), I(\psi)\}),\ I(\varphi \vee \psi) = \max_{\leq}(\{I(\varphi), I(\psi)\}),\ \text{and}\ I(\neg\varphi) = I(\varphi)^{-1}.$$

We say that I satisfies φ, denoted by $I \models \varphi$, if $I(\varphi) = \boldsymbol{t}$.

---

[1] We independently developed our algorithm, and believe that the new insights gained by the alternative presentation are a valuable asset.

We now introduce both the stable and well-founded semantics for definitions. The semantics of definitions in SAT(ID) will be given by the latter. Our presentation is based on [16], where the first three-valued characterisation of the stable model semantics was given: such a characterisation is closer to actual computational processes, where partial interpretations are used.

Let Σ be a vocabulary, Δ a definition over Σ and I_O an $Open(\Delta)$-interpretation. Denote by L_Δ the set of all Σ-interpretations extending I_O and define the operator $\Psi_\Delta : L_\Delta \to L_\Delta$ by $\Psi_\Delta(I)(P) = I(\bigvee_{P\leftarrow\varphi\in\Delta}\varphi)$ if $P \in Def(\Delta)$, and $\Psi_\Delta(I)(P) = I(P)$ otherwise. If Δ is a positive definition, Ψ_Δ is $\leq$-monotone, and the [illegible] Δ [illegible] I_O is defined as the $\leq$-least fixpoint of Ψ_Δ.

Let I be a 3-valued Σ-interpretation. The [illegible] of Δ in I, denoted by Δ^I, is the definition obtained by replacing in every rule all open atoms and all negative occurrences of defined atoms P by $I(P)$. The reduct is a positive definition.

Definition 1 (Stable model). [illegible] Δ [illegible] Σ [illegible] I_O [illegible] $Open(\Delta)$ [illegible] Σ [illegible] I [illegible] stable model of Δ extending I_O [illegible] $I|_{Def(\Delta)}$ [illegible] Δ^I [illegible] I_O [illegible] $I|_{Open(\Delta)} = I_O$

Note that in the standard definition of stable models [5] atoms in $Open(\Delta)$ are considered false. Intuitively, an atom $P \in Open(\Delta)$ here corresponds to an atom P defined by "$P \leftarrow \texttt{not}\, P'$. $P' \leftarrow \texttt{not}\, P$." in the standard definition.

It is shown in [16] that for every definition Δ and $Open(\Delta)$-interpretation I_O, there exists at least one stable model of Δ extending I_O. Also, the greatest lower bound with respect to $\leq_p$ of the set of all stable models of Δ extending I_O is itself a stable model of Δ extending I_O.

Definition 2 (Well-founded model). [illegible] Δ [illegible] I_O [illegible] $Open(\Delta)$ [illegible] well-founded model of Δ extending I_O [illegible] $\leq_p$ [illegible] Δ [illegible] I_O

An interpretation I satisfies a definition Δ, denoted $I \models \Delta$, if I is the well-founded model of Δ extending $I|_{Open(\Delta)}$ and I is two-valued. Finally, I satisfies a SAT(ID) theory T if I satisfies every formula and every definition of T.

Observe that we are only interested in two-valued models of definitions. We call a definition Δ [illegible] if for every two-valued $Open(\Delta)$-interpretation I_O, the well-founded model M of Δ extending I_O is two-valued. In this case, M is also the unique stable model of Δ extending I_O. Definitions that are encountered in practice are total, and for these, the well-founded and stable semantics coincide.

[illegible] The definition $\{P \leftarrow \neg P', \quad P' \leftarrow \neg P\}$ is not total (its well-founded model is three-valued) and hence has no model.

[illegible] In the following definition E_{xy} represents the existence of an edge between nodes x and y in a graph, and R_{xy} the reachability of x to y. $\Delta_2 = \{R_{ab} \leftarrow E_{ab} \vee I_{acb}, \quad R_{ac} \leftarrow E_{ac} \vee I_{abc}, \quad I_{acb} \leftarrow R_{ac} \wedge E_{cb}, \quad I_{abc} \leftarrow R_{ab} \wedge E_{bc}\}$. For any interpretation I_O of $\{E_{ab}, E_{bc}, E_{cb}, E_{ac}\}$, Δ_2 has a two-valued well-founded model extending I_O, e.g. for $I_1 = \{E_{ab} \mapsto \boldsymbol{f}, E_{bc} \mapsto \boldsymbol{t}, E_{cb} \mapsto \boldsymbol{t}, E_{ac} \mapsto \boldsymbol{f}\}$, the model is $I_1 \cup \{R_{ab} \mapsto \boldsymbol{f}, R_{ac} \mapsto \boldsymbol{f}, I_{acb} \mapsto \boldsymbol{f}, I_{abc} \mapsto \boldsymbol{f}\}$.

2.2 MIDL Normal Form

We extend the CNF format, as used by SAT solvers, to a normal form for SAT(ID) theories. A *clause* is a disjunction $L_1 \vee \ldots \vee L_n$ of literals. We denote clauses also by $[L_1, \ldots, L_n]$. A CNF theory is a set (conjunction) of clauses. A definition Δ is in MIDL *normal form* (MNF) if each rule $r \in \Delta$ is of the form $P \leftarrow L_1 \vee \cdots \vee L_n$ or $P \leftarrow L_1 \wedge \cdots \wedge L_n$, with L_i literals and $n \geq 0$, and each atom in $Def(\Delta)$ occurs exactly once as head. A SAT(ID) theory T is in MNF if $T = \Delta \cup \Gamma$, where Δ is a definition in MNF and Γ is a CNF theory.

As with CNF, the advantage of MNF is its simplicity. In particular, MNF makes explicit the data structures that are implemented in ASP systems such as Smodels [18] and clasp [4], where both literals and bodies have a truth value.

There exists a linear transformation from an arbitrary SAT(ID) theory T over Σ to an MNF theory T' over $\Sigma' \supset \Sigma$ such that there is a one-to-one correspondence between models M of T and models M' of T' (with $M'|_{\Sigma} = M$). Hence without loss of generality, we can from now on assume MNF theories.

3 Theory

In the rest of the paper, T denotes a SAT(ID) theory over vocabulary Σ, with definition Δ and CNF part Γ, and I denotes a three-valued Σ-interpretation. A trivial but naive SAT(ID) solving algorithm consists of (1) applying traditional SAT solving techniques to find a model of Γ and (2) subsequently checking whether this model satisfies Δ. Several existing algorithms can be used to compute (in quadratic time) the well-founded model extending a given two-valued interpretation, e.g. [19]. If it turns out that the model does not satisfy Δ, the algorithm tries to find another model of Γ. This algorithm is very inefficient, since in general, most models of Γ are not a model of Δ.[2] Hence the goal of this work is to interleave (1) and (2), i.e., while constructing the model of Γ, making sure that it can still satisfy Δ.

3.1 Justifications

In this section, we introduce the notion of a *justification*, and use it to provide an alternative characterization of the stable and well-founded model. We then return to model generation for SAT(ID), following an approach suggested by this new characterization.

For a directed graph $G = (V, E)$ and an element $v \in V$, we denote by $Ch_G(v)$ the set $\{w \mid (v, w) \in E\}$. If V is a set of literals, we call a cycle in G *positive*,

---

[2] This is a simplified representation of a more intelligent approach, applied by several ASP solvers, e.g. ASSAT [8] and Cmodels [6]. These apply (1) on Clark's completion of the given theory, and in (2) they add a *nogood* loop formula to the original theory for every failed SAT model. Though a viable and sometimes competitive approach, both experimental evaluation and theoretical considerations [7] show that exponentially many iterations may be needed, i.e., the original conclusion still holds. A direct integration of IDs in SAT seems more promising in principle.

[illegible] or [illegible] if it contains respectively only positive, only negative or both kind of literals.

Definition 3 (Justification). [illegible] justification J for Δ [illegible] $(\Sigma^\star, E)$ [illegible]

- [illegible] $C \leftarrow C_1 \wedge \ldots \wedge C_N \in \Delta$ $Ch_J(C) = \{C_1, \ldots, C_N\}$ [illegible] $Ch_J(\neg C) = \{\neg C_i\}$ [illegible] $i \in [1, N]$ [illegible]
- [illegible] $D \leftarrow D_1 \vee \ldots \vee D_N \in \Delta$ $Ch_J(\neg D) = \{\neg D_1, \ldots, \neg D_N\}$ [illegible] $Ch_J(D) = \{D_i\}$ [illegible] $i \in [1, N]$ [illegible]
- [illegible] $L \notin Def(\Delta)^\star$ $Ch_J(L)$ [illegible]

We denote by J^Δ the greatest common subgraph of all justifications for Δ. This contains at least the subgraphs determined by $Ch_J(C)$ resp. $Ch_J(\neg D)$, for conjunctively resp. disjunctively defined atoms C resp. D. We denote the unique descendant of $\neg C$ resp. D by $\mathcal{D}_J$: $\mathcal{D}_J(\neg C) = \neg C_i$, $\mathcal{D}_J(D) = D_i$.

[illegible] Let $\Delta_3 = \{P \leftarrow Q \vee A, \quad Q \leftarrow P\}$. Then Δ_3 has two justifications, J_1 and J_2, which differ in $\mathcal{D}_J(P)$: $J_1 = J^{\Delta}3 \cup \{(P, Q)\}$, $J_2 = J^{\Delta}3 \cup \{(P, A)\}$, with $J^{\Delta}3 = \left[\ \neg A \leftarrow \neg P \rightleftarrows \neg Q \qquad Q \rightarrow P\ \right]$.

Definition 4 (Stable, well-founded). [illegible] J [illegible] Δ [illegible] I [illegible] Σ [illegible] J is stable in I iff

- [illegible] $L \in Def(\Delta)^\star$ [illegible] $I(L) \geq_p I(\bigwedge Ch_J(L))$. [illegible]
- [illegible] J [illegible] I

[illegible] J supports I [illegible] J is cycle-safe in I. J is well-founded in I iff J [illegible] I [illegible] J [illegible] I

Intuitively, the cycle-safeness property expresses that atoms in a positive cycle without external support must be false.[3]

[illegible] Continuing from Example 3, let $I = \{A \mapsto \boldsymbol{f}, P \mapsto \boldsymbol{t}, Q \mapsto \boldsymbol{t}\}$. Then J_1 supports I and J_2 does not, while J_2 is cycle-safe in I and J_1 is not.

Theorem 1 (Stable, well-founded model). [illegible] I [illegible] Δ [illegible] $I|_{Open(\Delta)}$ iff [illegible] J [illegible] Δ [illegible] I

We omit the proof for well-founded models due to space restrictions; we include the proof for stable models, however, because it is more instructive.

[illegible] Denote by I_1 the Σ-interpretation such that $I_1(P) = I(P)$ for every $P \in Open(\Delta)$ and $I_1(P) = \boldsymbol{f}$ for all $P \in Def(\Delta)$, let $I_{n+1} = \Psi_{\Delta^I}(I_n)$ for every $n \geq 1$, and finally denote the limit of $(I_n)_{n \geq 1}$ by I'.

---

[3] This intuition can be made explicit by the concept of a *loop formula* [8].

($\Leftarrow$) Assume there exists a stable justification J for Δ in I. From the facts that J supports I, and I is 2-valued, one can easily show by induction that for every $n \geq 1$, $I_n \leq I$. Now, let $V = \{P \in Def(\Delta) \mid I(P) = \boldsymbol{t} \neq I'(P)\}$. We show that $V = \emptyset$, hence that $I = I'$.

Assume towards contradiction $V \neq \emptyset$ and let $P \in V$. If P is defined by the rule $P \leftarrow D_1 \vee \ldots \vee D_n$, then for all $1 \leq i \leq n$, $I'(D_i) < \boldsymbol{t}$. Hence $I'(\mathcal{D}_J(P)) < \boldsymbol{t}$, while $I(\mathcal{D}_J(P)) = \boldsymbol{t}$. Therefore, $\mathcal{D}_J(P)$ must be a defined atom and $\mathcal{D}_J(P) \in V$. If P is defined by the rule $P \leftarrow C_1 \wedge \ldots \wedge C_n$, there exists a C_i such that $I'(C_i) < \boldsymbol{t}$. As $I(C_i) = \boldsymbol{t}$, C_i must be defined atom and $C_i \in V$. This proves that every atom of V has at least one child of V in J. Hence, J contains a positive cycle whose atoms are true in I. This contradicts the cycle-safeness of J in I.

($\Rightarrow$) Conversely, assume I is a stable model of Δ, i.e. $I = I'$. We construct a justification J for Δ as follows.

- J contains J^{Δ}.
- For every $D \in Def(\Delta)$ with rule $D \leftarrow D_1 \vee \ldots \vee D_m$ in Δ, let $\mathcal{D}_J(D) = D_1$ if $I(D) = \boldsymbol{f}$. If $I(D) = \boldsymbol{t}$, we distinguish between two cases. If there exists a $D_i \in Open(\Delta) \cup Def(\Delta)$ with $I(D_i) = \boldsymbol{t}$, let $\mathcal{D}_J(D) = D_i$ for such a D_i. Otherwise, for some $n \geq 3$ we have $I_n(D) \neq \boldsymbol{t}$, but $I_{n+1}(D) = \boldsymbol{t}$. Then let $\mathcal{D}_J(D) = D_i$ for a D_i such that $I_n(D_i) = \boldsymbol{t}$, but $I_{n-1}(D_i) \neq \boldsymbol{t}$.
- For every $C \in Def(\Delta)$ with rule $C \leftarrow C_1 \wedge \ldots \wedge C_m$, let $\mathcal{D}_J(\neg C) = \neg C_1$ if $I(C) = \boldsymbol{t}$. If $I(C) = \boldsymbol{f}$, let $\mathcal{D}_J(\neg C) = \neg C_i$ for some C_i such that $I(C_i) = \boldsymbol{f}$.

It can easily be verified that J supports I. We define the [illegible] of a literal L, denoted $dl(L)$, as follows: if $L \in Def(\Delta)$ with $I(L) = \boldsymbol{t}$, then $dl(L) = n + 1$, with n such that $I_n(L) \neq \boldsymbol{t}$ and $I_{n+1}(L) = \boldsymbol{t}$; in all other cases, $dl(L) = 0$. Hence, our construction of J is such that for each child Q in J of any atom $P \in Def(\Delta)$ with $I(P) = \boldsymbol{t}$, $dl(Q)$ is strictly lower than $dl(P)$. Therefore J cannot contain a positive cycle with true atoms, i.e., J is also cycle-safe in I.

Theorem 1 suggests an approach to compute models of a definition: namely, maintain a justification that is stable for the partial interpretation at each moment in the computation. We show in Section 3.2 how to maintain support, and in Section 3.3 how to maintain cycle-safeness.

3.2 Two Watched Literals

Contemporary SAT solvers have [illegible] as their propagation mechanism: whenever in an input clause $[L_1, \ldots, L_n]$ all L_i are false, except one, this so-called [illegible] is made true. In all state-of-the art solvers, unit propagation is executed by means of the [illegible] (2WL). In this scheme, in each clause $[L_1, \ldots, L_n]$ two literals $W_1 = L_i$ and $W_2 = L_j$ for some $i \neq j$ are "watched", and the statement $I(W_1 \vee W_2) = \boldsymbol{t} \vee I(W_1 \wedge W_2) = \boldsymbol{u}$, called the [illegible], must be satisfied at all times. Hence, when either W_1 or W_2 becomes false, a replacement watch has to be found (the watch has to be"moved"). When no suitable replacement is found, i.e., all other literals are false, the remaining watch is made true. In contrast, when any non-watched literal becomes false, nothing needs to be done. We denote the function mapping clauses $[L_1, \ldots, L_n]$ to the set of their watched literals $\{W_1, W_2\}$ by F_{2WL}.

To apply 2WL on definitional rules, we first introduce some terminology.

Definition 5. [illegible] completion [illegible][4] $P \leftarrow \varphi$ [illegible] $P \equiv \varphi$ [illegible] completion [illegible] Δ [illegible] $comp(\Delta)$ [illegible] $r \in \Delta$

[illegible] $comp(\Delta_3) = \{\neg P \vee Q \vee A, \quad P \vee \neg Q, \quad P \vee \neg A, \quad \neg P \vee Q\}$.

Observe that for a clause A in the completion or a rule r, either the head of r or its negation occurs in A: we denote this occurence by $Head_A$. I.e., $Head_{[D,\neg D_i]} = D$, $Head_{[\neg D, D_1, \ldots, D_n]} = \neg D$, etc. Now, when applying 2WL on $comp(\Delta)$, F_{2WL} naturally [illegible] justifications: let for every rule $r \in \Delta$ and every clause A in the completion of r, W_A be a literal in $F_{2WL}(A)$ such that $W_A \neq Head_A$. Then the set of all edges $(\neg Head_A, W_A)$ forms a justification for Δ. Remark that the binary clauses in the completion induce the graph J^{Δ}. Multiple justifications are induced when for some clauses A in the completion $Head_A \notin F_{2WL}(A)$. We call the watches in $[\neg D, D_1, \ldots, D_n]$ respectively $[C, \neg C_1, \ldots, \neg C_n]$ of the completion of a rule r the [illegible] r and denote them by $W_1(r)$ and $W_2(r)$, or simply W_1, W_2 if r is clear from the context. The following property is easy to verify.

Proposition 1. [illegible] I [illegible] F_{2WL} [illegible] $comp(\Delta)$ [illegible] F_{2WL} [illegible] J [illegible] Δ [illegible] J [illegible] I [illegible] supported justification

[illegible] Let $\Delta_6 = \{P \leftarrow Q \wedge R, \quad Q \leftarrow P \vee \neg S\}$, and let $F_{2WL}([P, \neg Q, \neg R]) = \{\neg Q, \neg R\}$ and $F_{2WL}([\neg Q, P, S]) = \{\neg Q, P\}$. Then F_{2WL} induces $J_1 = J^{\Delta}6 \cup \{(Q, P), (\neg P, \neg Q)\}$ and $J_2 = J^{\Delta}6 \cup \{(Q, P), (\neg P, \neg R)\}$. The interpretation $I = \{P \mapsto \boldsymbol{f}, Q \mapsto \boldsymbol{f}, R \mapsto \boldsymbol{t}, S \mapsto \boldsymbol{t}\}$ satisfies the 2WL invariants on $comp(\Delta_6)$. J_1 supports I, but J_2 does not: $I(\neg P) = \boldsymbol{t}$, whereas $I(\bigwedge Ch_{J_2}(\neg P)) = I(\neg R) = \boldsymbol{f}$.

Our algorithm applies the 2WL scheme on the completion of Δ, hence maintains at least one supported justification. Next section shows to maintain amongst the supported justifications at least one that is also cycle-safe, hence stable.

3.3 Cycle-Safeness

Assume that for a given I and F_{2WL}, at least one supported justification for Δ is cycle-safe. Now, moving a watch in $comp(\Delta)$ changes the induced justifications. Hence we have to evaluate which type of watch moves may lead to the introduction of a positive cycle in any of those, i.e., to the invalidation of cycle-safeness.[5] If r is a conjunctive rule with head C, $Ch_J(C)$ is fixed, hence moving $W_1(r)$

---

[4] Recall that in MNF each defined atom has exactly one defining rule.

[5] The same reasoning was used to introduce the concept of a *source pointer* in Smodels [18], which corresponds to the unique descendant $(\mathcal{D}_J(\cdot))$ concept here.

or $W_2(r)$ cannot introduce a positive cycle in any induced justification. If r is a disjunctive rule with head D, and $W_1(r)$ or $W_2(r)$ is moved, a positive cycle may be introduced if the move is to a defined atom in r's body.

It is highly inefficient—mainly for technical reasons, related to cache behaviour—to interrupt unit propagations to verify whether there is indeed a positive cycle, and to fix that problem if so. The alternative is to delay this cycle testing/repairing. To do so, atoms that could be in a positive cycle are marked as "cycle sources". We formalize this by means of the following concept.

Definition 6 (Cycle-safe up to a set of atoms). [illegible] J [illegible] S [illegible] J is cycle-safe up to S [illegible] J [illegible] S [illegible] $s \in S$ [illegible] cycle sources

Observe that when all atoms in S are false in I, cycle-safeness up to S implies cycle-safeness in I. Hence we consider an algorithm that maintains a partial interpretation I, a watch function F_{2WL}, and a set of cycle sources S, and has as invariant: at least one justification induced by F_{2WL} and supported by I is cycle-safe up to S. To satisfy this invariant without interrupting unit propagations, the head of a disjunctive rule r must be added to S whenever one of r's watches is moved to a defined atom. To also obtain a [illegible] justification, the algorithm must try to remove all non-false atoms from S, but only [illegible] unit propagations reached a fixpoint. Atoms in S should be made false only when they are false in all three-valued stable models of Δ that are refinements of I. The following algorithm makes S smaller while preserving the invariant.

1. Select some $P \in S$ with $I(P) \neq \boldsymbol{f}$.
2. If F_{2WL} induces some J with J cycle-safe up to $S \setminus \{P\}$, remove P from S and stop.
3. Try to change F_{2WL} such that it satisfies the 2WL invariant and induces J with J cycle-safe up to $S \setminus \{P\}$. If this succeeds: remove P from S and stop.
4. Set $I(P) := \boldsymbol{f}$.

This algorithm can be repeated until all remaining atoms in S are false in I. The main challenge is how to perform Step 3 in an efficient way. Also, we must be able to prove that this sub-algorithm is complete, so that step 4 is justified, i.e., any justification supported in I necessarily exhibits a positive cycle. This is the subject of Section 4.1.

3.4 Identifying a Unique Supported Justification

Let $P_1, P_2 \in S$, and let J_1, J_2 be different supported justifications, and suppose J_1 is cycle-safe up to $S \setminus \{P_1\}$, while J_2 is cycle-safe up to $S \setminus \{P_2\}$. We cannot conclude from this that there is a supported justification that is cycle-safe up to $S \setminus \{P_1, P_2\}$. The task of making S smaller would be greatly simplified if a unique justification J could be maintained during the whole algorithm.

One way to do so is to change F_{2WL}, such that the functionality of the two watches is distinguished, i.e., the function now maps to a pair (W_1, W_2) instead of a set $\{W_1, W_2\}$. We assign W_1 as the single watch that induces a justification.

As such, for each clause A in $comp(\Delta)$ with $F_{2WL}(A) = (W_{1,A}, W_{2,A})$, $W_{1,A}$ should not be equal to $Head_A$. Then the set of all edges $(\neg Head_A, W_{1,A})$ forms a [illegible] justification. This change requires some adaptations of 2WL: whenever $W_{1,A}$ is moved to $Head_A$, and whenever $W_{1,A}$ becomes false and all other literals in the clause, except for $W_{2,A}$, are false, $W_{1,A}$ and $W_{2,A}$ have to be swapped. Proposition 1 can be adapted accordingly: by applying the new 2WL scheme, the unique induced justification supports the partial interpretation.[6]

Observe that this new strategy also means that moving $W_2(r)$ in a disjunctive rule r does not change the induced justification, hence does not generate a cycle source: thus only about half as many cycle sources are produced.

4 Algorithm

For the whole of Section 4, let J be a justification and S a set of atoms, such that J supports I and is cycle-safe up to S, and let $P \in S$.

4.1 Justifying Cycle Sources

We want to remove P from S. This can be done when J is cycle-safe up to $S \setminus \{P\}$. If this is not the case for the current J, a search for an alternative justification has to begin, i.e., for some atoms A, $\mathcal{D}_J(A)$ has to be changed. We now investigate which atoms need to be considered.

[illegible] Consider following sub-graph of a justification J, where P is a cycle source: $\left[\; U \rightarrow P \rightleftarrows Q \rightarrow R \qquad V \;\right]$. At least one of the outgoing edges from P or Q has to change its child node. Changing it to U, however, would introduce a new cycle, while changing it to R or V would not. It follows that only $\mathcal{D}_J(P)$, $\mathcal{D}_J(Q)$ and/or $\mathcal{D}_J(U)$ need to be changed. Generalizing: the ancestor atoms of a cycle may need to change their outgoing edges, the descendants do not.

We express the observation from this example using following concepts.

Definition 7. [illegible] A [illegible] fi [illegible] J [illegible] fi [illegible] fi $Top_J(A)$ [illegible] $\{B \mid$ [illegible] J [illegible] B [illegible] $A\}$

Definition 8. [illegible] A [illegible] fi [illegible] J [illegible] fi [illegible] S [illegible] A is justified in J up to S [illegible] ff [illegible] J [illegible] S [illegible] $A \notin Top_J(B)$ [illegible] B [illegible]

Clearly, if $P \notin Top_J(P)$, there is no positive cycle through P. The intuition behind Definition 8 is that J might contain positive cycles through P: if J contains no path of atoms from A to any atom B in such a cycle, then A is justified. Hence the following formalization of the intuition from Example 7: any atom outside $Top_J(P)$ is justified in J up to $S \setminus \{P\}$. This means that we can restrict the search for an alternative justification to $Top_J(P)$.

---

[6] Actually, the 2WL invariant must be refined to the following: if one of the watches is false, then all other literals of the clause are false, except for the other watch. Current SAT solvers already maintain this invariant for efficiency reasons.

[illegible] It is easy to adjust the definition of Top_J to take into account the [illegible] (SCCs) of the positive atom dependency graph. Then the search can be further restricted to those atoms in $Top_J(P)$ that are in the same SCC as P. For simplicity, we stick to definition 7 in this presentation.

During the search, a set $\mathcal{N}$ of atoms is maintained, of which it is not known yet whether they are justified up to $S \setminus \{P\}$ in the current justification J. In other words, these atoms may be in, or may positively depend on, a positive cycle in J. Initially, $\mathcal{N}$ is set to $Top_J(P)$; after that, the goal of the algorithm is to decrease $\mathcal{N}$ until $P \notin \mathcal{N}$. Informally, we call removing an atom A from $\mathcal{N}$, "justifying" A. Any non-false literal from outside $\mathcal{N}$ can be used for justifying atoms in $\mathcal{N}$. E.g., let $D \leftarrow D_1 \vee \ldots \vee D_N$ be a rule with $\mathcal{D}_J(D) = D_i$, $D, D_i \in \mathcal{N}$ and for some D_j, $D_j \notin \mathcal{N}$ and $I(D_j) \neq \boldsymbol{f}$, then we can safely remove D from $\mathcal{N}$ by changing $\mathcal{D}_J(D)$ to D_j. For a conjunction, the head can only be justified by showing that all body literals are outside $\mathcal{N}$. The algorithm employs a function $f_\wedge$ for this, which maps conjunctively defined atoms to one of their body literals. Note that other cycle sources than P could be in $Top_J(P)$; justifying them during the process effectively means that they can be safely removed from S.

This search for an alternative justification J' proceeds downward from P, from head to body atoms. As such it tries to justify atoms as close to P as possible, the intuition being that in most cases, a solution can be found close to P. To avoid double work, a set $\mathcal{T}$ of "touched" atoms is also maintained. When an atom is touched, it is added to $\mathcal{Q}$: this set contains the atoms for which search can still be expanded, i.e., it is the "working queue". When an atom is justified, it is not only removed from $\mathcal{N}$, but also from $\mathcal{T}$. Hence when $\mathcal{Q}$ is empty (the whole search space has been visited), but $\mathcal{T} \neq \emptyset$, the search proved unable to produce a good justification for the atoms in $\mathcal{T}$—i.e., in the current interpretation, they have to be false.[7]

Algorithm 1 is the result of the above reasoning. Here we provide a correctness and completeness result with respect to this algorithm.

Theorem 2. [illegible] I [illegible] S [illegible] J [illegible] J [illegible] I [illegible] S [illegible] $P \in S$ [illegible] U [illegible] S' [illegible] J' [illegible] J' [illegible] I [illegible] S' [illegible]

- $U = \emptyset$ [illegible] $J' = J$ [illegible] $S' = S \setminus \{P\}$.
- $U = \emptyset$ [illegible] $P \in Top_J(P)$ [illegible] $S \setminus Top_J(P) \subseteq S' \subseteq S \setminus \{P\}$.
- $U \neq \emptyset$ [illegible] $S \setminus Top_J(P) \subsetneq S' \subseteq S$ [illegible] $Q \in U$ [illegible] $\max_{\leq}\{I'(Q) \mid I'$ [illegible] $I|_{Open(\Delta)}\} = \boldsymbol{f}$

In the third of the possible outcomes, U is an unfounded set. It then also holds that each $Q \in U$ is in a positive cycle in J' through P. Observe that in the second and third possible outcome the set S' is not precisely defined: some atoms from $Top_J(P)$ may be removed from S in Step 1 of the algorithm.

---

[7] They are in an *unfounded set* [20].

4.2 Overall Algorithm

The difference between the overall MidL algorithm and the DPLL algorithm as used in SAT solvers lies entirely in an adapted initialization phase, and an augmentation of DPLL's unit propagation. The former must make sure that the invariants are satisfied: a straightforward (though slightly naive) initialization phase may set $S := Def(\Delta)$. The latter is replaced by Algorithm 2. Here, the function `UnitProp()` applies unit propagation by means of the (new) 2WL scheme on Γ and on $comp(\Delta)$, as described in Section 3. At the end of this step, the invariants "J supports I" and "J is cycle-safe up to S" are satisfied, and a while-loop begins, trying to decrease S. The loop applies `Justify(`P`)` on every non-false cycle source P,[8] continuing unit propagations as soon as an unfounded set is found. At the end of Algorithm 2, S is made as small as possible. We then have "J is cycle-safe up to S", and "$\forall A \in S : I(A) = \boldsymbol{f}$", hence also "$J$ is cycle-safe in I". We can conclude that I is a three-valued stable model.

By repeated application of Algorithm 2 on new choice literals, we obtain a two-valued stable model. A standard well-founded model checking algorithm can subsequently verify whether that model is also well-founded. This check may fail when Δ is not total, causing backtracking. However, all definitions encountered in practice are total. We demonstrate the algorithm in next example.

[illegible] Let $\Delta_8 = \{P \leftarrow Q \vee R.\ \ Q \leftarrow P.\ \ R \leftarrow A.\}$. Initially, everything is unknown. Suppose $\mathcal{D}_J(P) = R$. A sample run of the algorithm might make P true (by choice), which makes Q true by unit propagation. S is still empty, so a new choice is made: say, $\neg A$. This makes R false by unit propagation, which forces $\mathcal{D}_J(P)$ to change to Q, so that P must be added to S. Next, `Justify(`P`)` finds the unfounded set $\{P, Q\}$, so both atoms are made false, which yields a conflict. By backtracking, A is made true, propagating to R and P. P is still in S, so `Justify(`P`)` is run again, changing $\mathcal{D}_J(P)$ back to R, thus creating a cycle safe justification. The resulting interpretation is 2-valued, hence a model of Δ_8.

Theorem 3 (Correctness & Completeness). [illegible] T [illegible] T MidL [illegible] T [illegible] T [illegible] fi [illegible]

This can easily be proven by completeness of standard backtracking, and correctness and completeness of both `UnitProp()` and `Justify(·)`.

5 Evaluation

On MidL**.** We have implemented the above algorithms in a system called MidL [12,13]. The following features that make contemporary SAT solvers efficient and robust have been implemented in MidL: good choice heuristics (Vsids

---

[8] Observe that not removing false atoms from S renders it unnecessary to change S during backtracking.

```
Result: A set of atoms $U$
$\mathcal{N} := Top_J(P)$;
if $P \notin \mathcal{N}$ then $S := S \setminus \{P\}$; return $\emptyset$ ;
$\mathcal{T} := \mathcal{Q} := \{P\}$;
while $\mathcal{Q} \neq \emptyset$ do
    Remove an atom $Q$ from $\mathcal{Q}$ ;
    if $Q$ is disjunctively defined then
        % Let the rule for $Q$ be $Q \leftarrow D_1 \vee ... \vee D_n$.
        $\mathcal{D} := \{D_i \mid 1 \leq i \leq n, I(D_i) \neq \boldsymbol{f}\}$.  % Note that $\mathcal{D} \neq \emptyset$.
        if $\exists D_i \in \mathcal{D} \setminus \mathcal{N}$ then
            $\mathcal{D}_J(Q) := D_i$ ;  BUProp($Q$) ;
        else
            Add $\mathcal{D} \setminus \mathcal{T}$ to $\mathcal{Q}$ ;   Add $\mathcal{D}$ to $\mathcal{T}$ ;
    else
        % Let the rule for $Q$ be $Q \leftarrow C_1 \wedge ... \wedge C_n$.
        $\mathcal{C} := \{C_i, 1 \leq i \leq n \mid C_i \in \mathcal{N}\}$;
        if $\mathcal{C} = \emptyset$ then
            BUProp($Q$) ;
        else
            if $\exists C_i \in \mathcal{T}$ then $f_\wedge(Q) := C_i$ ;
            else Choose a $C_i \in \mathcal{C}$;   $f_\wedge(Q) := C_i$;   Add $C_i$ to $\mathcal{Q}$ and to $\mathcal{T}$ ;
return $\mathcal{T}$ ;
function BUProp $Q$
    Remove $Q$ from $\mathcal{N}$, $\mathcal{T}$ and $\mathcal{Q}$ ;
    if $Q \in S$ then $S := S \setminus \{Q\}$ ;
    if $Q = P$ then return $\emptyset$ ;
    for each $C \in \mathcal{T}$ with $f_\wedge(C) = Q$ do
        Push $C$ on $\mathcal{Q}$ ;
    for each $D \in \mathcal{T}$ with rule $r_D = D \leftarrow \ldots \vee Q \vee \ldots$ do
        $\mathcal{D}_J(D) := Q$ ;   BUProp($D$) ;
end function
```

Algorithm 1. $Justify(P)$: justifying a cycle source P

```
repeat
    UnitProp();
    $U := \emptyset$;
    while $U = \emptyset$ and $\exists P \in S$ such that $I(P) \neq \boldsymbol{f}$ do
        Let $P \in S$ such that $I(P) \neq \boldsymbol{f}$;
        $U :=$ Justify($P$);
        $I(Q) := \boldsymbol{f}$ for every $Q \in U$;
until $U = \emptyset$ ;
```

Algorithm 2. SAT(ID) replacement of SAT's unit propagation

and VMTF), clause learning and backjumping, restarts, compact encoding of binary clauses. Some other important features could make it more mature still, such as preprocessing, clause deletions, and compact encoding of ternary clauses.

On SAT(ID) experimenting. SAT(ID) is a relatively new logic, hence few solvers exist. The main purpose of this paper, however, is to demonstrate how to integrate ID support in a SAT solver, and to illustrate that this integration is viable. Hence we perform experiments on two different fronts: one concerning ID support, where we compare with ASP solvers on problems containing inductive definitions, and one concerning SAT solving, where we compare with both ASP and SAT solvers on problems not containing any inductive definitions.[9]

Are IDs needed? Observe that the need for using IDs can sometimes be avoided by using elaborate encodings, e.g. for the Hamiltonian circuit problem.[10] The propositional instances obtained this way are over 50 times the size of those using an ID encoding, for graphs with 150 nodes, and were simply too big for graphs with 200 nodes. In Table 1, the column for MiniSAT refers to these instances. Also a reduction from SAT(ID) to SAT is a possibility [15]. The IDSAT solver implements this, but yielded only time-outs on the instances of Table 1 (using MiniSAT). Clearly, an integrated approach is superior.

Experiments and discussion. We show the experimental results in Table 1.[11] The **first set** of results concerns Hamiltonian cycle problems, where we compare MIDL with the ASP solvers clasp [4], Smodels [18], Cmodels [6], and Smodels$_{cc}$ [21], and with MiniSAT [3] on an alternative encoding. The instances have respectively $(100, 800)$, $(150, 1200)$ and $(200, 1800)$ nodes and edges (4 instances of each kind). Too big ground files are denoted '#'. We can conclude that MIDL and clasp are comparable, and outperform all other solvers. The **second set** concerns Hitori puzzles of size 50×50, compared against the same ASP solvers. These puzzles contain inductive definitions, but with very few cycles: on most instances, Cmodels needed to create no or only one loop formula. It outperformed both MIDL and the other ASP solvers, despite its use of a somewhat outdated SAT solver. The **last set** concerns Blocked 28-queens problems, where we also compare with the SAT solvers siege [17] and MiniSAT. We observe an efficiency gap between pure SAT solvers, and ID supporting solvers. Interestingly, Cmodels' performance is not as good as that of siege and MiniSAT. By analogy,

---

[9] Naturally, problem encodings differ according to the formalism used. We have tried to ensure fair comparisons, e.g., we haven't used aggregate expressions.

[10] See `http://www.cs.sfu.ca/research/groups/mxp/examples/view.php?f=hc`.

[11] Version numbers of the programs used are: Minisat 1.14, siege 4, Clasp 1.0.3, Smodels 2.32, Cmodels 3.67, SmodelsCC 1.08, IDSAT 0.9.5, and MIDL 2.1.1. All experiments run on a P4 2.8GHz with 1GB memory. Problem instances of Hamiltonian cycle and of Blocked N-queens are taken from the benchmark website Asparagus, `http://asparagus.cs.uni-potsdam.de/`. Results of all bencmark instances are available in attachment, but left out here due to space restrictions.

Table 1. Running times (sec.) of Hamiltonian Cycle (left), Hitori (right) and Blocked N-queens (below) problem instances. First lines show averages.

clasp	MidL	Cmodels	MiniSAT	Smodels$_{cc}$	Smodels
1.05	2.28	19.35	163.8	194.43	210.6
0.47	0.48	3.92	7.22	285.86	> 300
0.48	1.36	2.37	5.77	> 300	> 300
0.48	0.67	2.35	6.18	24.33	8.63
0.48	0.69	1.96	15.95	77.70	> 300
0.79	1.51	5.05	55.15	181.35	> 300
1.07	1.37	7.77	> 300	> 300	> 300
0.91	3.75	5.59	179.18	185.29	> 300
0.87	1.22	8.57	196.12	59.54	19.69
1.78	4.65	146.07	#	151.90	47.51
2.27	4.26	20.73	#	167.15	51.32
1.51	4.02	13.37	#	> 300	> 300
1.45	3.41	14.40	#	> 300	> 300

Cmodels	clasp	MidL	Smodels	Smodels$_{cc}$
1.32	6.04	8.15	17.63	51.55
1.00	6.50	7.19	16.26	45.87
0.94	6.53	7.24	16.40	45.98
1.18	7.62	8.66	20.75	68.21
0.90	5.24	5.70	22.36	55.80
1.25	7.97	6.86	18.72	52.78
0.93	0.38	0.61	1.37	1.37
1.56	6.68	12.82	18.43	58.31
0.67	0.37	0.63	1.64	1.25
3.94	9.00	25.14	27.72	78.76
1.20	5.86	7.08	23.03	62.33
1.15	9.64	8.32	22.36	72.89
1.17	6.69	7.55	22.54	75.05

MiniSat	Siege	clasp	MidL	Cmodels	Smodels	Smodels$_{cc}$
0.54	2.53	3.26	4.68	5.66	24.16	58.29
0.62	2.14	2.18	4.40	8.38	17.44	44.61
0.10	1.65	4.20	2.33	2.90	1.79	2.28
0.23	2.27	1.41	3.61	0.43	8.20	12.74
0.34	0.60	1.02	1.43	1.22	5.16	12.17
0.37	0.85	1.29	2.26	1.79	18.81	70.48
2.26	11.87	14.37	18.54	28.64	124.30	> 300
0.19	0.28	0.58	1.23	0.59	5.36	15.60
0.20	0.54	1.02	3.67	1.30	12.21	8.41

a possible explanation for the worse performance of the ID solvers is that their implementation details are less advanced than those of siege and MiniSAT.

6 Conclusions, Related and Future Work

In this paper, we have illustrated an approach to integrate ID support in contemporary SAT solving algorithms. In particular, we have shown how SAT's 2WL scheme can be reused to also determine justifications, and how such justifications can be used to find models of IDs. Finally, we reported on an implementation of the algorithms discussed.

A related approach to SAT(ID) solving works by reduction to SAT [15].[12] An algorithm for finding unfounded sets, presented in [1], is closely related to our Algorithm 1. In particular, the following sets have similar functionalities: Set $\sim \mathcal{T}$, Ext $\sim \mathcal{Q}$, Sink $\sim (\Sigma \setminus \mathcal{N})$. The precise relation between Source and S is unclear.

The most obvious item of future work is to convert theory into practice, by actually extending an existing contemporary SAT solver such as MiniSat with IDs. On the other hand, extending MidL with additional features such as preprocessing may make it more mature and robust. There is also room for experimenting with variants of Algorithm 1. Finally, it is a goal of our project to make SAT more expressive by adding relevant modelling constructs: for instance, we intend to also support aggregate expressions in MidL.

---

[12] The solver described in this paper had mostly time-outs in our experiments.

References

1. Anger, C., Gebser, M., Schaub, T.: Approaching the core of unfounded sets. In: Dix, J., Hunter, A. (eds.) Proceedings of the International Workshop on Nonmonotonic Reasoning, pp. 58–66 (2006)
2. Denecker, M.: Extending classical logic with inductive definitions. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 703–717. Springer, Heidelberg (2000)
3. Eén, N., Sörensson, N.: An extensible sat-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
4. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: LPNMR 2007 (2007)
5. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: JICSLP 1988, pp. 1070–1080. MIT Press, Cambridge (1988)
6. Lierler, Y.: cmodels - sat-based disjunctive answer set solver. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 447–451. Springer, Heidelberg (2005)
7. Lifschitz, V., Razborov, A.A.: Why are there so many loop formulas? ACM Trans. Comput. Log. 7(2), 261–268 (2006)
8. Lin, F., Zhao, Y.: Assat: computing answer sets of a logic program by sat solvers. Artif. Intell. 157(1-2), 115–137 (2004)
9. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., et al. (ed.) The Logic Programming Paradigm: a 25 Years Perspective, pp. 375–398. Springer, Heidelberg (1999)
10. Mariën, M., Mitra, R., Denecker, M., Bruynooghe, M.: Satisfiability checking for PC(ID). In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 565–579. Springer, Heidelberg (2005)
11. Mariën, M., Wittocx, J., Denecker, M.: The IDP framework for declarative problem solving. In: Search and Logic: Answer Set Programming and SAT, pp. 19–34 (2006)
12. Mariën, M., Wittocx, J., Denecker, M.: MidL: a SAT(ID) solver. In: ASP 2007 (to appear, 2007)
13. MidL. `http://www.cs.kuleuven.be/~dtai/krr/software/midl.html`
14. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC, pp. 530–535. ACM, New York (2001)
15. Pelov, N., Ternovska, E.: Reducing inductive definitions to propositional satisfiability. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 221–234. Springer, Heidelberg (2005)
16. Przymusinski, T.C.: Well founded semantics coincides with three valued Stable Models. Fundamenta Informaticae 13, 445–463 (1990)
17. Ryan, L.: Efficient algorithms for clause-learning SAT solvers. Master's thesis, Simon Fraser University (2004)
18. Simons, P.: Extending and Implementing the Stable Model Semantics. PhD thesis, Helsinki Univ. of Technology (2000)
19. Van Gelder, A.: The alternating fixpoint of logic programs with negation. Journal of Computer and System Sciences 47(1), 185–221 (1993)
20. Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM 38(3), 620–650 (1991)
21. Ward, J., Schlipf, J.: Answer set programming with clause learning. In: Lifschitz, V., Niemelä, I. (eds.) Logic Programming and Nonmonotonic Reasoning. LNCS (LNAI), vol. 2923, pp. 302–313. Springer, Heidelberg (2003)

The Separation Theorem for Differential Interaction Nets

Damiano Mazza[1],* and Michele Pagani[2],**

[1] Laboratoire d'Informatique de Paris Nord
damiano.mazza@lipn.univ-paris13.fr
http://www-lipn.univ-paris13.fr/~mazza
[2] Dipartimento di Filosofia – Università degli Studi Roma Tre
pagani@uniroma3.it
http://logica.uniroma3.it/~pagani

Abstract. Differential interaction nets (DIN) have been introduced by Thomas Ehrhard and Laurent Regnier as an extension of linear logic proof-nets. We prove that DIN enjoy an internal separation property: given two different normal nets, there exists a dual net separating them, in analogy with Böhm's theorem for the λ-calculus. Our result implies in particular the faithfulness of every non-trivial denotational model of DIN (such as Ehrhard's finiteness spaces). We also observe that internal separation does not hold for linear logic proof-nets: our work points out that this failure is due to the fundamental asymmetry of linear logic exponential modalities, which are instead completely symmetric in DIN.

Keywords: Differential interaction nets, faithfulness, linear logic, observational equivalence, proof-nets.

1 Introduction

The question of separation is an important one in computer science and, more recently, also in proof theory. The best known example of separation result is Böhm's theorem for the pure λ-calculus [1]: if t, t' are two distinct closed $\beta\eta$-normal terms, then there exist terms $u_1, \ldots, u_n$, such that $t u_1 \ldots u_n \to_\beta^* \mathbf{0}$ and $t' u_1 \ldots u_n \to_\beta^* \mathbf{1}$.[1] This result has consequences both at the semantical level as well as at the syntactical one: on the one hand it entails that a model of the λ-calculus cannot identify two different $\beta\eta$-normal forms without being trivial (in this case we say that the model is [illegible], or [illegible]); on the other hand it establishes a balance between syntactical constructs and β-reduction: any difference in the structure of a $\beta\eta$-normal form implies a difference in the value of that normal form on suitable arguments.

After Böhm, this kind of question was studied by Friedman and Statman in the simply typed framework [2,3], leading to what is often called "typed Böhm's

---

* Supported by a post-doc fellowship of French ANR project "NOCoST".

** Post-doc fellow, research project: "Ricerche sulla geometria della logica".

[1] As it is usual in the λ-calculus, $\mathbf{1} = \lambda xy.x$ and $\mathbf{0} = \lambda xy.y$.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 393–407, 2007.
© Springer-Verlag Berlin Heidelberg 2007

theorem".[2] In this case the two distinct $\beta\eta$-normal terms have the same type $A_1, \ldots, A_n \to X$, and they are not separated directly on that type, but on an instance of it: that is, there is a type B and, for each $1 \leq i \leq n$, an argument u_i of type $A_i[B/X]$ such that $tu_1 \ldots u_n \to_\beta^* \mathbf{0}$ and $t'u_1 \ldots u_n \to_\beta^* \mathbf{1}$.[3]

After the introduction of linear logic [5], the question of separation has been addressed also in proof theoretical frameworks. The first work on the subject is [6], where the authors deal with "pure proof-nets", a linear logical system corresponding to the pure λ-calculus. But it is only with Girard's work on ludics [7] that separation became a key property of proof theory, which may now be seen as a fundamental step in analyzing the structure of our representation of proofs.

There is a good reason why syntactical, interactive separation in the style of Böhm's theorem has taken so many years to shift from computer science to proof theory: the lack of results was essentially due to the absence of interesting logical systems where proofs could be represented in a "nice" canonical way. The only existing exception was natural deduction for minimal logic which, being isomorphic to the simply typed λ-calculus, had already been fully covered by Friedman and Statman's results.

In linear logic, canonical representations of proofs do exist, under the form of directed graphs called *proof-nets* [5]. A key ingredient of proof-nets is to forget the context of logical rules (except for the so-called promotion rule), so as to allow a higher degree of parallelism in the representation of proofs, which becomes thus more canonical. The typical (and most fundamental) form of parallelism we refer to here is the one needed to obtain the associativity of deduction: from three lemmas proving that A implies B, B implies C, and C implies D, we should only obtain one proof that A implies D, even if there are two ways of composing the lemmas. This is true in proof-nets (as it is true in natural deduction), but is strikingly false in sequent calculus.

In recent years, Tortora de Falco studied the canonicity of linear logic proof-nets by addressing the question of faithfulness (injectivity) in coherent spaces (which is, as cited above, strictly related to syntactical separation). With the exception of certain subsystems of linear logic, this study yielded a series of negative results: coherent spaces are not in general a faithful model of proof-nets, and separation fails [8]. The problem lies in the exponential modalities of linear logic, and more precisely in their *uniform* behavior: during cut-elimination, if at some point there is the need for two proofs of the same exponential formula to be provided, the procedure always answers this need with two copies of *the same* proof. In an interactive setting, this corresponds to the environment giving the same answer to a program querying multiple times for the value of an argument.

---

[2] Actually Friedman and Statman proved the faithfulness of standard models of the simply typed λ-calculus; from those semantic results however one can easily infer the syntactical separation (see for example [4]).

[3] Here we are supposing that X is the only variable occurring in the type of t, t'. To consider a term of type A also as a term of type $A[B/X]$, for any formula B, is sometimes called "Statman's typical ambiguity".

A new, potentially very powerful tool for the analysis of linear logic proofs came from the work of Ehrhard and Regnier, which led to the introduction of [illegible] (DIN, [9]). Based on Lafont's interaction nets [10], DIN are a syntax corresponding to the semantical constructions defined by Ehrhard in his [illegible] [11]. This semantical interpretation models linear proofs with linear functions on certain topological vector spaces, on which one can define an operation of derivative. Non-linear proofs (i.e., proofs using exponential modalities) become analytic functions, in the sense that they can be arbitrarily approximated by the equivalent of a Taylor expansion, which becomes available thanks to the presence of a derivative operator.

In syntactical terms, these constructions take a very interesting form: they correspond to "symmetrizing" the exponential modalities, i.e., in the logical system arising from finiteness spaces the rules handling the two dual exponential modalities [illegible] / [illegible] are perfectly symmetrical (although the logic is not self-dual). What is equally interesting is that the "old" rules of linear logic exponentials are not lost: proof-nets can be encoded in DIN.

This paper considers the question of separation for DIN, giving a positive answer in Theorem 1: given two different normal nets, we find another (dual) net separating them, up to Statman's typical ambiguity. This separation is as meaningful as that of Böhm's theorem, as it implies the faithfulness of every denotational semantics of DIN (Corollary 1), so in particular of finiteness spaces themselves.

We then apply Theorem 1 to the framework of proof-nets, and show with a few examples that pairs of proof-nets which cannot be interactively distinguished can on the contrary be easily separated once encoded in DIN, by heavily exploiting the symmetry of DIN exponentials. This shows concretely one of the main insight provided by our work: separation in proof-nets fails because of the asymmetry of linear logic exponentials.

2 Differential Interaction Nets

[illegible] In what follows, the set of all permutations over n elements is denoted by $\mathfrak{S}_n$.

The formulas of propositional multiplicative exponential linear logic (**MELL**) are generated by the following grammar, where $X, X^\perp$ range over a denumerable set of propositional variables:

$$A, B ::= X \mid X^\perp \mid 1 \mid A \otimes B \mid \perp \mid A \parr B \mid !A \mid ?A.$$

Linear negation is defined through De Morgan laws:

$$\begin{array}{cc}
(X)^\perp = X^\perp & (X^\perp)^\perp = X \\
(1)^\perp = \perp & (\perp)^\perp = 1 \\
(A \otimes B)^\perp = A^\perp \parr B^\perp & (A \parr B)^\perp = A^\perp \otimes B^\perp \\
(!A)^\perp = ?A^\perp & (?A)^\perp = !A^\perp
\end{array}$$

Lists of occurrences of formulas will be ranged over by Γ, Δ, Σ. If $\Gamma = A_1, \ldots, A_n$, we shall denote by $\Gamma^\perp$ the list $A_1^\perp, \ldots, A_n^\perp$.

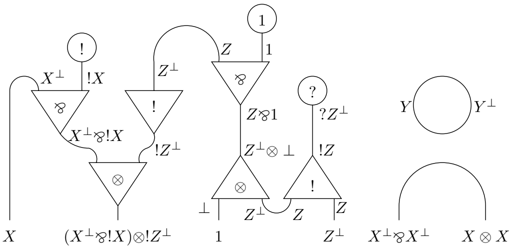


Fig. 1. A simple net

[illegible] Differential interaction nets are defined on top of simple nets, which are particular interaction nets [10]. Here we give an informal definition; for a more detailed one, see [12].

A *simple net* is a set of *cells* and *wires*, graphically represented as in Fig. 1. Each cell has a *type*, which is a **MELL** connective, i.e., a symbol belonging to the set $\{1, \otimes, \perp, ⅋, !, ?\}$, and a number of *ports*, exactly one of which is called *principal port*, while the others (if any) are called *auxiliary ports*. The arity of a cell is equal to the number of its auxiliary ports; cells of type 1 and $\perp$ are required to be nullary, and those of type $\otimes$ and $⅋$ must be binary. Graphically, the principal port of a non-nullary cell is seen as one of the "tips" of the triangle representing it, while a nullary cell is represented by a circle.

A wire is represented as… a wire; the extremities of wires not connected to anything are called *free ports* of the net. For example, the net in Fig. 1 has six free ports. In the case of cyclic wires like the one at the top-right of Fig. 1, which are called *[illegible]*, we stipulate that there are two wires connecting the same two *internal ports*. Hence, there are four kinds of ports: principal, auxiliary, free, and internal. A wire connecting two non-principal ports is said to be an *axiom*; a wire connecting two principal or internal ports is said to be a *cut*. Note that a wire may be an axiom and a cut at the same time; this is the case of deadlocks. Those wires that are neither axioms nor cuts are called *simple*.

Each port i has a *type* $T(i)$, which is a **MELL** formula. These types must satisfy the following:

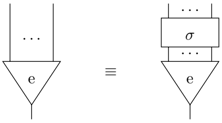
- if i, j are connected by an axiom or a cut, then $T(i) = T(j)^\perp$;
- if i, j are connected by a simple wire, then $T(i) = T(j)$;
- if i_0 is the principal port of a cell of type 1, then $T(i_0) = 1$;
- if i_0 is the principal port of a cell of type $\otimes$, whose two auxiliary ports are i_1, i_2, then $T(i_0) = T(i_1) \otimes T(i_2)$;
- if i_0 is the principal port of a cell of type $\perp$, then $T(i_0) = \perp$;
- if i_0 is the principal port of a cell of type $⅋$, whose two auxiliary ports are i_1, i_2, then $T(i_0) = T(i_1) ⅋ T(i_2)$;
- if i_0 is the principal port of a cell of type !, whose auxiliary ports are $i_1, \ldots, i_n$, then $T(i_1) = \cdots = T(i_n) = A$, and $T(i_0) = !A$;
- if i_0 is the principal port of a cell of type ?, whose auxiliary ports are $i_1, \ldots, i_n$, then $T(i_1) = \cdots = T(i_n) = A$, and $T(i_0) = ?A$;

If a simple net α has n free ports, we assume that they are numbered by the integers $1, \dots, n$, so that p_k is the kth free port. Then, we refer to the list of occurrences of formulas $T(p_1), \dots, T(p_n)$ as the [illegible] of α.

The empty simple net will be denoted by $\mathbf{1}$.

We now introduce a fundamental equivalence on simple nets, accounting for the fact that the auxiliary ports of exponential cells are unordered:

Definition 1 (σ-equivalence). [illegible] σ [illegible] $\equiv$ [illegible]



[illegible] σ [illegible] e [illegible] ! [illegible] ?

Unless otherwise stated, simple nets will be considered modulo $\equiv$, i.e., whenever we refer to "the simple net α", we actually mean "the σ-equivalence class containing α".

Definition 2 (Differential interaction net). [illegible] differential interaction net [illegible] net [illegible] σ [illegible] Γ [illegible] μ, ν [illegible] $\emptyset$ [illegible] $\mathbf{0}$

Definition 3 (Composition). [illegible] α [illegible] β [illegible] Δ, Γ [illegible] $\Gamma^{\perp}, \Sigma$ [illegible] $\langle \alpha \,|\, \beta \rangle$ [illegible] Δ, Σ [illegible] Γ [illegible] α [illegible] $\Gamma^{\perp}$ [illegible] β [illegible] μ [illegible] ν [illegible] Δ, Γ [illegible] $\Gamma^{\perp}, \Sigma$ [illegible] $\langle \mu \,|\, \nu \rangle = \{ \langle \alpha \,|\, \beta \rangle \mid \alpha \in \mu, \beta \in \nu \}$ [illegible] Δ, Σ

In the sequel, we shall confuse a simple net α with the net $\{\alpha\}$ whenever this is not source of ambiguity. In particular, the net with no conclusions containing only the empty simple net, i.e. $\{\mathbf{1}\}$, will simply be denoted by $\mathbf{1}$.

Nets are provided with two rewriting relations, corresponding to **MELL** cut-elimination (β-reduction) and non-atomic axiom-elimination (η-expansion). On simple nets, these are the contextual closures of the rules resp. given in Fig. 2 and Fig. 3, and are resp. denoted by $\succ_\beta$ and $\succ_\eta$.

The two topmost rules of Fig. 2 are called [illegible]; the bottom rule is called [illegible]. In all rules, a simple net reduces to a net; in the multiplicative cases, the right member must be seen as a singleton. In the exponential case, if the arities of the two interacting cells do not match, the rule yields the empty net; in case the arities match, the rule yields a net containing all simple nets obtained by connecting in all possible ways the auxiliary ports of the two cells.

Similarly, the topmost two rules of Fig. 3 are called multiplicative, and the bottom rule exponential. These rules also yield a net out of a simple net, with

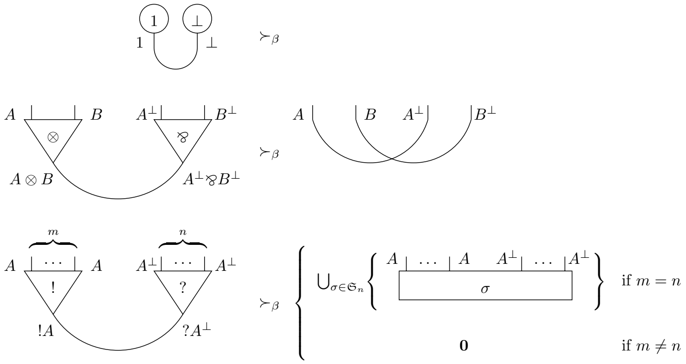

1
⊥
$\succ_\beta$
A
B
$A^\perp$
$B^\perp$
$A \otimes B$
$A^\perp \parr B^\perp$
m
n
!
?
$!A$
$?A^\perp$
$\bigcup_{\sigma\in\mathfrak{S}_n}$
σ
if $m = n$
$\mathbf{0}$
if $m \neq n$

Fig. 2. Cut-elimination rules (β-reduction)

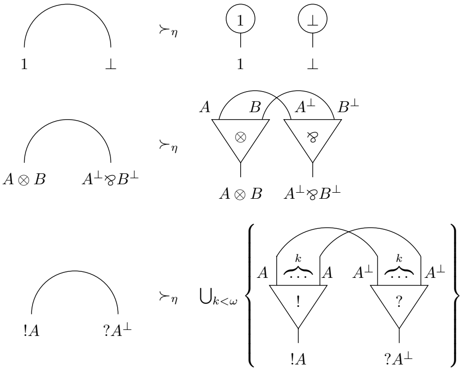

1
⊥
$\succ_\eta$
$A \otimes B$
$A^\perp \parr B^\perp$
A
B
$A^\perp$
$B^\perp$
k
!
?
$!A$
$?A^\perp$
$\bigcup_{k<\omega}$

Fig. 3. Non-atomic axiom-elimination rules (η-expansion)

the right member of the multiplicative rules equal to a singleton. In the exponential rule, an axiom is replaced by its expansions using all possible arities for exponential cells.

Definition 4 (β-reduction and η-expansion). [illegible] $\to_\beta$ [illegible] $\mu \to_\beta \mu'$ [illegible]

$$\mu' = \bigcup_{\alpha\in\mu} \nu_\alpha,$$

[illegible] $\alpha \succ_\beta \nu_\alpha$ [illegible] $\nu_\alpha = \{\alpha\}$ [illegible] $\alpha \succ_\beta \nu_\alpha$ [illegible] $\alpha \in \mu$ [illegible] $\to_\eta$ [illegible] $\succ_\eta$ [illegible] $\succ_\beta$ [illegible] $\to \; = \; \to_\beta \cup \to_\eta$ [illegible] μ [illegible] normal [illegible] μ' [illegible] $\mu \to \mu'$ [illegible] μ [illegible] normalizable [illegible] ν [illegible] $\mu \to^* \nu$

Proposition 1 (Confluence). [illegible] $\to^*$ [illegible]

[illegible] Actually $\to^*$ is strongly confluent: if $\mu \to \mu', \mu''$ with $\mu' \neq \mu''$, then there exists ν s.t. $\mu', \mu'' \to^* \nu$ in at most one step. □

Moreover notice that every finite net is strongly β-normalizing. In fact, define for any simple net α with k ports, $\sharp\alpha = \prod_{n\leq k} n!$. By simply inspecting Fig. 2, we see that $\alpha \rightarrow_\beta \nu_\alpha$ implies $\sharp\alpha > \sum_{\alpha'\in\nu_\alpha} \sharp\alpha'$. Now suppose μ finite and $\mu \rightarrow_\beta \mu'$. Observe that μ' is also finite and $\sum_{\alpha\in\mu} \sharp\alpha > \sum_{\alpha'\in\mu'} \sharp\alpha'$, so μ is strongly β-normalizing.

However, strong normalization may fail in case of infinite nets, even if we ignore deadlocks (which of course are not normalizable).[4] In fact, for each non-negative integer k, it is easy to find a simple net α_k such that $\{\alpha_k\}$ reduces to a normal net in at least k steps. Then, the net $\bigcup_{k<\omega}\{\alpha_k\}$ obviously has no normal form.

[illegible] There are two notable differences with respect to the definition of DIN given in [9].
(i) We consider generalized exponential cells, corresponding in [9] to trees of binary (co)contraction cells with (co)dereliction and (co)weakening cells on the leaves, modulo associativity, commutativity, and neutrality of (co)weakening. This is so-called [illegible] [13], and provides more canonical nets. In fact, only commutativity needs to be explicitly treated in our framework (through σ-equivalence); associativity and neutrality are built-in.
(ii) In [9] nets are defined as *finite* sets of simple nets. The need to consider infinite nets is a consequence of (i), which forbids a conclusion of an axiom to be (co)contracted. In our syntax, such configurations are represented using η-expansion, which yields infinite nets in the exponential case. Additionally, infinite nets are required if one wants to consider the Taylor-Ehrhard expansion of proof-nets, as we do in Sect. 4.

3 The Separation Theorem

In this section, we fix a single propositional variable X, and consider only formulas built on the dual pair $X, X^\perp$. Everything we say can of course be generalized to types containing arbitrary atoms.

Let μ be a net with conclusions Γ and let A be a formula. We denote by $\mu[A/X]$ the net with conclusions $\Gamma[A/X]$ obtained from μ by substituting each occurrence of X with A.

Our main result is the following one:

Theorem 1 (Separation). [illegible] μ μ' [illegible] Γ [illegible] ν [illegible] $\Gamma[?1/X]^\perp$ [illegible] $\langle \nu \,|\, \mu[?1/X]\rangle \rightarrow_\beta^* \mathbf{1}$ [illegible] $\langle \nu \,|\, \mu'[?1/X]\rangle \rightarrow_\beta^* \mathbf{0}$ [illegible]

We remark that the use of multiplicative units is only a convenience: Theorem 1 also holds in their absence, using a formula of the form $?!A$ instead of $?1$, where A is arbitrary (for example X itself).

We now proceed with the proof of Theorem 1. First of all, in case Γ is empty (i.e., μ, μ' have no conclusion), then by definition of normal net, either $\mu = \mu'$

---

[4] By the way, there are geometrical conditions, known as *correctness criteria* [9], which prevent a net satisfying them from producing deadlocks.

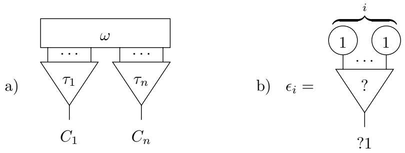


Fig. 4. a) Decomposition of a normal simple net. b) Definition of the net ϵ_i, $i \in \mathbb{N}$.

or $\mu = \mathbf{1}$ and $\mu' = \mathbf{0}$. Otherwise, let us suppose that μ, μ' do have conclusions. Observe that if two normal nets μ, μ' are different, then there is a simple net α in one of them which is different from every simple net in the other one; to separate μ and μ' we shall define a (simple) net ν which has the property of reducing to $\mathbf{1}$ when applied to α, and to $\mathbf{0}$ in all other cases (see Definition 6 and Lemma 1). To do this properly, we need to be careful because nets are defined as sets of σ-equivalence classes, which unfortunately do not have canonical representatives. Therefore, in the rest of the section, α will denote an actual simple net, and we may have $\alpha \neq \alpha'$ without having $\alpha \not\equiv \alpha'$.

We define a [illegible] to be a simple net containing no cells and no deadlock. Wirings will be ranged over by ω, and are said to be [illegible] if their conclusions are all atomic. If A is a formula, a [illegible] of root A is a simple net defined by induction on A:

- if A is atomic, then the only tree of root A is a wire of conclusions $A^\perp, A$;
- if $A = 1$ (resp. $A = \perp$), then the only tree of root A consists of a single cell of type 1 (resp. $\perp$);
- if $A = A_1 \otimes A_2$ (resp. $A = A_1 \parr A_2$), and τ_1, τ_2 are two trees of resp. roots A_1 and A_2, then the net obtained by plugging the roots of τ_1 and τ_2 to the auxiliary ports of a cell of type $\otimes$ (resp. $\parr$) is a tree of root A;
- if $A = !B$ (resp. $A = ?B$), and $\tau_1, \dots, \tau_n$ are trees of root B ($n \in \mathbb{N}$), then the net obtained by plugging the roots of each τ_i to the auxiliary ports of a cell of type ! (resp. ?) is a tree of root A.

The perfect symmetry of DIN cells allows the following definition, which is a crucial point in the proof of Theorem 1:

Definition 5 (Mirror tree). [illegible] τ [illegible] A [illegible] mirror tree [illegible] τ [illegible] $\tau^\perp$ [illegible] $A^\perp$ [illegible] $\tau[X^\perp/X]$ [illegible] $1 \leftrightarrow \perp$ $\otimes \leftrightarrow \parr$ $! \leftrightarrow ?$

Definition 6 (Antagonist). [illegible] α [illegible] $C_1, \dots, C_n$ [illegible] α [illegible] ω [illegible] n [illegible] $\tau_1, \dots, \tau_n$ [illegible] $\alpha^\dagger$ [illegible] $C_1[?1/X]^\perp, \dots, C_n[?1/X]^\perp$ [illegible] antagonist [illegible] α [illegible] $\alpha^\dagger$ [illegible] $0, \dots, k$ [illegible] ω [illegible] X [illegible] $X^\perp$ [illegible] $\omega(i) = j$ [illegible] i [illegible] X [illegible] j [illegible] $X^\perp$ [illegible] ω [illegible] $\varphi = \tau_1^\perp, \dots, \tau_n^\perp$ [illegible] $\alpha^\dagger$ [illegible]

$\varphi[?1/X]$ [illegible] $i \in \{0, \dots, k\}$ [illegible] ϵ_i [illegible] i [illegible] $\epsilon_i^\perp$ [illegible] $\omega(i)$ [illegible] ϵ_i [illegible]

Observe that, in the decomposition of Fig. 4a, ω may as well be empty; in that case, α is a forest $\tau_1, \dots, \tau_n$, and its only antagonist is $\tau_1^\perp, \dots, \tau_n^\perp$.

Lemma 1. [illegible] α [illegible] $C_1, \dots, C_n$ [illegible] $\alpha^\dagger$ [illegible] σ [illegible] α [illegible] α' [illegible] α [illegible]

$\alpha \equiv \alpha'$ [illegible] $\langle \alpha^\dagger \,|\, \alpha'[?1/X] \rangle \rightarrow_\beta^* \mathbf{1}$.
$\alpha \not\equiv \alpha'$ [illegible] $\langle \alpha^\dagger \,|\, \alpha'[?1/X] \rangle \rightarrow_\beta^* \mathbf{0}$

[illegible] By induction on the number of cells in α. If α is a wiring, then it must be atomic, so α' is also an atomic wiring. In that case, $\alpha^\dagger$ is an antagonist of α, $\alpha \equiv \alpha'$ iff $\alpha = \alpha'$, and it is then easy to prove points 1 and 2 of the lemma. If α has a cell, then one of its conclusions must be connected to the principal port of a cell c, because α is normal. We can suppose w.l.o.g. this conclusion to be C_1. The proof splits into six cases, depending on the type of c. We consider only the case in which c is of type !, the ? case being perfectly symmetrical and the other cases being easier.

So we have $C_1 = !A$. This also means that the corresponding conclusion of α' is connected to the principal port of a cell c' of type ! (recall that α' is η-normal). If the arity of c' is different than that of c, then $\alpha' \not\equiv \alpha$ and we immediately have $\langle \alpha^\dagger \,|\, \alpha' \rangle \rightarrow_\beta^* \mathbf{0}$. So suppose that c and c' have same arity k. Let α_0 (resp. α'_0) be the simple net obtained from α (resp. α') by removing c (resp. c'). Observe that the conclusions of α_0 (as those of α'_0) are $A^1, \dots, A^k, C_2, \dots, C_n$, where $A^1, \dots, A^k$ are occurrences of the same formula A and correspond to the type of the auxiliary ports of c and c'. In $\langle \alpha^\dagger \,|\, \alpha' \rangle$ we have a cut between c' and a cell $c^\dagger$ of type ?, which is also of arity k. By reducing this cut, we obtain

$$\langle \alpha^\dagger \,|\, \alpha' \rangle \rightarrow_\beta \{ \langle \delta \,|\, \gamma \rangle \ ; \ \gamma \in P \},$$

where δ is obtained from $\alpha^\dagger$ by removing the cell $c^\dagger$, and P is the set of all simple nets obtained from α'_0 by permuting the conclusions $A^1, \dots, A^k$. Each $\gamma \in P$ has the same conclusions as α_0, so P can be partitioned into $P_{\mathbf{0}} = \{ \gamma \in P \ ; \ \gamma \not\equiv \alpha_0 \}$ and $P_{\mathbf{1}} = \{ \gamma \in P \ ; \ \gamma \equiv \alpha_0 \}$. Now, it is possible that δ is not σ-equivalent to an antagonist of α_0: this may be because $\alpha^\dagger$ is σ-equivalent to an antagonist of α thanks to a permutation $\sigma \in \mathfrak{S}_k$ on the auxiliary ports of $c^\dagger$. But in that case one can always include this permutation in the ones generated by the β-reduction ($\mathfrak{S}_k$ is a group, so $\sigma\mathfrak{S}_k = \mathfrak{S}_k$), so that actually δ can always be considered to be σ-equivalent to an antagonist of α_0. This latter contains strictly fewer cells than α, so by induction hypothesis $\langle \delta \,|\, \gamma \rangle \rightarrow_\beta^* \mu$ iff $\gamma \in P_\mu$, where $\mu \in \{\mathbf{0}, \mathbf{1}\}$. But $\alpha \equiv \alpha'$ iff $P_{\mathbf{1}} \neq \emptyset$, hence the lemma holds. □

We can now conclude the proof of Theorem 1. Take two different normal nets μ, μ'. As remarked above, we can assume w.l.o.g. that μ contains a σ-equivalence class not contained in μ'. Take any representative α of this equivalence class, and define ν to be the net containing only the equivalence class of an antagonist of α. By Lemma 1, we have $\langle \nu \,|\, \mu[?1/X] \rangle \rightarrow_\beta^* \mathbf{1}$, while $\langle \nu \,|\, \mu'[?1/X] \rangle \rightarrow_\beta^* \mathbf{0}$.

3.1 An Application: Faithfulness

A denotational semantics $\mathfrak{M}$ of DIN is a $*$-autonomous category with some additional structure (refer to [14] for the details) interpreting **MELL** formulas as objects and nets as morphisms. More precisely, having associated with the variable X an object $\mathcal{X}$, then $\mathfrak{M}$ associates with each **MELL** formula A an object $[\![A]\!]_{\mathcal{X}}$ and with each net μ of conclusions $C_1, \dots, C_n$ (for $n \geq 0$) a morphism $[\![\mu]\!]_{\mathcal{X}}$ from $[\![1]\!]_{\mathcal{X}} = I$ (the identity object of the monoidal structure) to $[\![C_1 \parr \dots \parr C_n]\!]_{\mathcal{X}}$, in such a way that:

composition: $[\![\langle \mu \,|\, \nu \rangle]\!]_{\mathcal{X}} = [\![\mu]\!]_{\mathcal{X}} \circ [\![\nu]\!]_{\mathcal{X}}$.[5]
invariance: if $\mu \rightarrow \mu'$ then $[\![\mu]\!]_{\mathcal{X}} = [\![\mu']\!]_{\mathcal{X}}$.

A semantics is faithful (or injective, see [8]) if for any two distinct normal nets μ, μ', there is an object $\mathcal{X}$, s.t. $[\![\mu]\!]_{\mathcal{X}} \neq [\![\mu']\!]_{\mathcal{X}}$. A notable corollary of Theorem 1 is the faithfulness of every non-trivial denotational semantics (for example Ehrhard's fi[illegible], introduced in [11]).

Corollary 1 (Faithfulness). [illegible] $\mathfrak{M}$ [illegible] μ μ' [illegible] $\mathcal{X}$ $[\![\mu]\!]_{\mathcal{X}} = [\![\mu']\!]_{\mathcal{X}}$ [illegible] $\mathcal{X}$ [illegible] ν $[\![\nu]\!]_{\mathcal{X}} = [\![\mathbf{0}]\!]_{\mathcal{X}}$

[illegible] Suppose that for every object $\mathcal{X}$, $[\![\mu]\!]_{\mathcal{X}} = [\![\mu']\!]_{\mathcal{X}}$. By Theorem 1, there is a simple net α such that $\langle \{\alpha\} \,|\, \mu[?1/X] \rangle \rightarrow_{\beta}^{*} \mathbf{1}$ and $\langle \{\alpha\} \,|\, \mu'[?1/X] \rangle \rightarrow_{\beta}^{*} \mathbf{0}$. Now let ν be a net and consider the net $\nu\alpha$ obtained by juxtaposing α to each simple net of ν. Remark that $\langle \nu\alpha \,|\, \mu[?1/X] \rangle \rightarrow_{\beta} \nu$ and $\langle \nu\alpha \,|\, \mu'[?1/X] \rangle \rightarrow_{\beta} \mathbf{0}$. By hypothesis we have that, for every object $\mathcal{X}$, $[\![\mu[?1/X]]\!]_{\mathcal{X}} = [\![\mu'[?1/X]]\!]_{\mathcal{X}}$, hence by composition $[\![\langle \mu[?1/X] \,|\, \nu\alpha \rangle]\!]_{\mathcal{X}} = [\![\langle \mu'[?1/X] \,|\, \nu\alpha \rangle]\!]_{\mathcal{X}}$, and finally by invariance $[\![\nu]\!]_{\mathcal{X}} = [\![\mathbf{0}]\!]_{\mathcal{X}}$. □

4 Proof-Nets in DIN

We shall now apply Theorem 1 to the framework of proof-nets, and show a few examples of interactively indistinguishable proof-nets which can be easily separated once encoded in DIN.

Our definition of **MELL** proof-nets follows closely that of Danos and Regnier [15]:

Definition 7 (Proof-net). [illegible] proof-net [illegible] ? [illegible] ! [illegible] promotion cells [illegible]

---

[5] Here we are implicitly exploiting the $*$-autonomous structure of the category: a morphism of type $I \rightarrow [\![\Delta]\!] \parr [\![\Gamma]\!]$ and a morphism of type $I \rightarrow [\![\Gamma^{\perp}]\!] \parr [\![\Sigma]\!]$ can be seen resp. as morphisms of type $[\![\Delta^{\perp}]\!] \rightarrow [\![\Gamma]\!]$ and $[\![\Gamma]\!] \rightarrow [\![\Sigma]\!]$, so it makes sense to compose them.

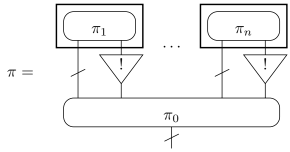
boxing condition: [illegible] c [illegible] $\mathcal{B}$ [illegible] !-box [illegible] $\mathcal{B}$ [illegible] principal door [illegible] c [illegible] auxiliary doors [illegible] ? [illegible] ! [illegible] ! [illegible]

sequentialization condition: [illegible] fi [illegible] $\bot$ [illegible] 1 [illegible] π [illegible] ⅋ [illegible] ? [illegible] π [illegible] π_1 π_2 [illegible] π_1 [illegible] π_2 [illegible] $\otimes$ [illegible] π [illegible] $A, ?C_1, \ldots, ?C_n$ [illegible] π [illegible] ! [illegible] A [illegible] ! [illegible] π [illegible] ? [illegible] $?C_1, \ldots, ?C_n$ [illegible]

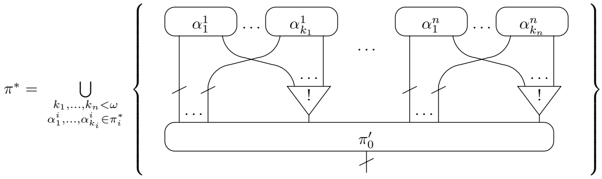
[illegible] π [illegible] depth [illegible] π [illegible] ! [illegible] π [illegible] $\partial(\pi)$ [illegible]

A proof-net is not a simple net because it contains additional information, namely that carried by !-boxes. However, this additional information can be accommodated in DIN thanks to the Taylor-Ehrhard formula, which is the reformulation in terms of nets of the usual Taylor expansion of analytic functions around the origin [11].

Definition 8 (Taylor-Ehrhard expansion). [illegible] π [illegible] Γ [illegible] Taylor-Ehrhard expansion π^* [illegible] Γ [illegible] fi [illegible] π [illegible] $\partial(\pi) = 0$ [illegible] $\pi^* = \{\pi\}$ [illegible] $\partial(\pi) > 0$ [illegible]



[illegible] π_0 [illegible] ! [illegible]



[illegible] π'_0 [illegible] fi [illegible] π_i [illegible]

[illegible] ? [illegible] c [illegible] π_0 [illegible] π'_0 [illegible] π_0 [illegible] c [illegible] c [illegible] $k+1$ [illegible] π_0 [illegible] $k+k_i$ [illegible] π'_0

Cut-elimination is defined exactly as in Fig. 2, except for exponential cuts. In these cuts !-boxes play a crucial role, since they delimit subnets to be erased or duplicated as a whole in one step. Fig. 5 defines exponential cut-elimination for proof-nets: what happens is that the !-box dispatches n copies of π_1 ($n \geq 0$ being the arity of ? cell shown) inside the !-boxes (if any) crossed by the auxiliary doors of the ? cell.

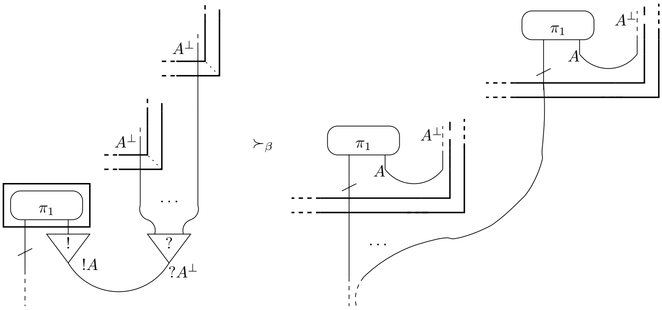


Fig. 5. Cut-elimination rule for promotion

The Taylor-Ehrhard expansion preserves **MELL** reductions on proof-nets, in the sense given by the following proposition:

Proposition 2 (Simulation). [illegible] π_1 [illegible] π_2 [illegible] A, Γ [illegible] $A^\perp, \Delta$ [illegible] $\langle \pi_1 \,|\, \pi_2 \rangle \to^*_\beta \pi_3$ [illegible] $\langle \pi_1^* \,|\, \pi_2^* \rangle \to^*_\beta \pi_3^*$

[illegible] It is enough to prove that, given a generic proof-net π, if $\pi \to_\beta \pi'$ by reducing a cut at depth 0, then $\pi^* \to_\beta \pi'^*$. We omit the details. ☐

Coherent spaces provide the most classical denotational semantics for proof-nets (see [5]). Lorenzo Tortora de Falco proves in [8] that this semantics fails to be faithful: there are distinct normal proof-nets which are associated with the same morphism, for any interpretation of the variables. We reproduce in Fig. 6a and in Fig. 7a two examples of pairs of non-separable proof-nets. This means that **MELL** proof-nets cannot verify the separation property, at least in the strong form of Theorem 1, as this would contradict Corollary 1 (which would hold also for **MELL** in that case).

The example of Fig. 6a morally[6] corresponds to the following PCF terms:

$$\lambda x.\texttt{if}\,x\,\texttt{then}\,(\texttt{if}\,x\,\texttt{then}\,\texttt{false}\,\texttt{else}\,\frac{\texttt{true}}{\texttt{false}})\,\texttt{else}\,(\texttt{if}\,x\,\texttt{then}\,\frac{\texttt{false}}{\texttt{true}}\,\texttt{else}\,\texttt{false})$$

---

[6] We cheated a bit in order to have simpler proof-nets. The exact proof-nets should use additives and have conclusions $?(\bot \& \bot), 1 \oplus 1$.

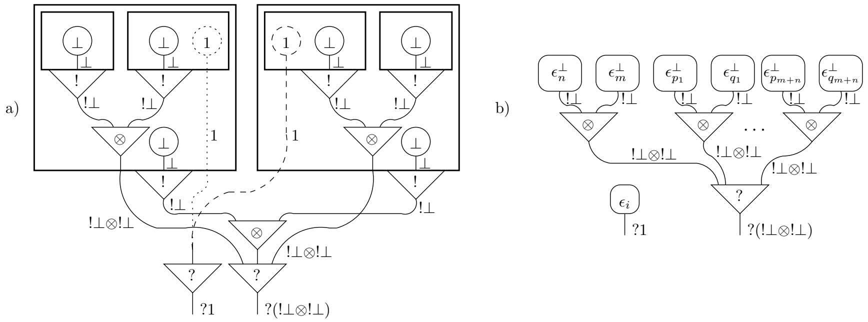


Fig. 6. a) Example 1 of non-separable proof-nets: π_1 is defined by considering the dotted wire and cell, instead π_2 by considering the dashed ones. b) Definition of $\alpha\langle i, n, m, p_1, q_1, \ldots, p_{n+m}, q_{n+m}\rangle$, where ϵ_i is the simple net of Fig. 4b.

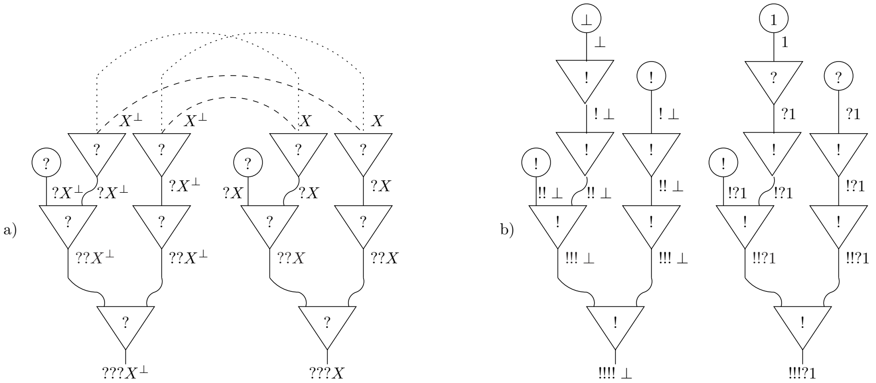


Fig. 7. a) Example 2 of non-separable proof-nets: π_1 is defined by considering the dashed wires, instead π_2 by considering the dotted ones. b) Simple net separating π_1^*, π_2^* of a).

where the term corresponding to π_1 is obtained by choosing `true` from $\frac{\texttt{true}}{\texttt{false}}$ and `false` from $\frac{\texttt{false}}{\texttt{true}}$, while π_2 by making the opposite choices. It is well-known that this two terms are indistinguishable also in PCF, since the nested `if then else` have all the same argument x (corresponding to the ? cell of conclusion $?(!\perp\otimes!\perp)$ in Fig. 6a).

The example of Fig. 7a is reported[7] in [8]. Notice that this example does not use promotion: the proof-nets of Fig. 7a are already simple nets, i.e. $\pi_i^* = \pi_i$. The problem of proof-net separation therefore lies in the contraction rule, and not in the exponential box.

Although the examples of Fig. 6a and Fig. 7a are not separable in **MELL** proof-nets, they become easily separable when translated in DIN. Let us start with the proof-nets π_1, π_2 defined in Fig. 6a. Their Taylor-Ehrhard expansions in DIN are:

---

[7] To be pedantic, the example we show here is a slight simplification of that defined in [8], since we are using units and mix.

$$\pi_1^* = \left\{ \alpha \left\langle i, n, m, p_1, q_1, \ldots, p_{n+m}, q_{n+m} \right\rangle \mid i = \textstyle\sum_{j=1}^{n} q_j \right\}$$

$$\pi_2^* = \left\{ \alpha \left\langle i, n, m, p_1, q_1, \ldots, p_{n+m}, q_{n+m} \right\rangle \mid i = \textstyle\sum_{j=n+1}^{n+m} p_j \right\}$$

where $\alpha \langle i, n, m, p_1, q_1, \ldots, p_{n+m}, q_{n+m} \rangle$ is the simple net defined in Fig. 6b, and n, m, p_j, q_j range over all non-negative integers.

An example of separating simple net is $\alpha^{\perp} \langle 2, 1, 0, 0, 2 \rangle$. In fact, we have $\langle \pi_1^* \mid \alpha^{\perp} \langle 2, 1, 0, 0, 2 \rangle \rangle \rightarrow_{\beta}^{*} \mathbf{1}$ and $\langle \pi_2^* \mid \alpha^{\perp} \langle 2, 1, 0, 0, 2 \rangle \rangle \rightarrow_{\beta}^{*} \mathbf{0}$.

Let us turn to π_1, π_2 as defined in Fig. 7a. Let α be the simple net of Fig. 7b; we have $\langle \pi_1[?1/X] \mid \alpha \rangle \rightarrow_{\beta}^{*} \mathbf{1}$ and $\langle \pi_2[?1/X] \mid \alpha \rangle \rightarrow_{\beta}^{*} \mathbf{0}$.

Observe that both examples are separable by means of simple nets which make a crucial use of ! cells of arity different (in particular, higher) than 1, which are exactly the cells not allowed in **MELL** proof-nets.

5 Concluding Remarks

[illegible] It is known that in general a denotational semantics for **MELL** does not provide a semantics for DIN: for example, coherent semantics does not interpret cocontraction (i.e., ! cells of arity higher than 1).

On the other hand, any denotational semantics of DIN provides a semantics for **MELL** through the Taylor-Ehrhard expansion: given a proof-net π, one can define $[\![\pi]\!]$ as $[\![\pi^*]\!]$.[8]

An immediate consequence of our result is that any semantics of DIN separates any two proof-nets with different Taylor-Ehrhard expansions. This sheds more light upon the faithfulness of relational semantics: a few years ago Tortora de Falco conjectured in [8] that such semantics is faithful for **MELL** proof-nets, a question which still waits to be settled. Since relational semantics is a semantics for DIN, we can reduce Tortora's conjecture to the question of whether different normal proof-nets have different Taylor-Ehrhard expansions.

$\lambda\mu$ [illegible] The question of separation for the $\lambda\mu$-calculus has been addressed by David and Py in [16]. In that paper the authors produce two $\lambda\mu$-terms which are indistinguishable: the two terms are a variant of the nested `if then else` analyzed in the example of Fig. 6a. Very recently Lionel Vaux has introduced in [17] a differential extension of the $\lambda\mu$-calculus: in such extension, it is not hard to find a term separating David and Py's terms in exactly the same way as we did for separating the two proof-nets of Fig. 6a. It is then likely that a separation result similar to Theorem 1 holds for Vaux's extension of the $\lambda\mu$-calculus.

---

[8] This is possible only if one considers DIN modulo commutativity of exponential cells, as we do with our σ-equivalence. For a more detailed discussion of the link between DIN and linear logic denotational semantics see [14].

References

1. Böhm, C.: Alcune proprietà delle forme $\beta\eta$-normali nel λ-K-calcolo. Pubblicazioni dell'IAC 696, 1–19 (1968)
2. Friedman, H.: Equality between functionals. In: Proceedings of LC 72-73. Lecture Notes in Math., vol. 453, pp. 22–37 (1975)
3. Statman, R.: Completeness, invariance and λ-definability. J. Symbolic Logic, 17–26 (1983)
4. Joly, T.: Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types. Ph.D. Thesis, Université de Paris 7 (2000)
5. Girard, J.Y.: Linear logic. Theoret. Comput. Sci. 50, 1–102 (1987)
6. Mascari, G., Pedicini, M.: Head linear reduction and pure proof net extraction. Theoret. Comput. Sci. 135, 111–137 (1994)
7. Girard, J.Y.: Locus solum. Math. Struct. Comput. Sci. 11, 301–506 (2001)
8. de Falco, L.T.: Obsessional experiments for linear logic proof-nets. Math. Struct. Comput. Sci. 13, 799–855 (2003)
9. Ehrhard, T., Regnier, L.: Differential interaction nets. Theoret. Comput. Sci. 364, 166–195 (2006)
10. Lafont, Y.: Interaction nets. In: POPL 1990, pp. 95–108 (1990)
11. Ehrhard, T.: Finiteness spaces. Math. Struct. Comput. Sci. 15, 615–646 (2005)
12. Mazza, D.: Interaction Nets: Semantics and Concurrent Extensions. Ph.D. Thesis, Universitée de la Méditerranée/Universitá degli Studi Roma Tre (2006)
13. Regnier, L.: Lambda-calcul et réseaux. Ph.D. Thesis, Université de Paris 7 (1992)
14. De Carvalho, D.: Sémantique de la logique linéaire et temps de calcul. Ph.D. Thesis, Université d'Aix-Marseille 2 (2007)
15. Danos, V., Regnier, L.: Proof nets and the Hilbert space. In: Advances in Linear Logic, pp. 307–328. Cambridge University Press, Cambridge (1995)
16. David, R., Py, W.: $\lambda\mu$-calculus and Böhm's theorem. J. Symbolic Logic 66, 407–413 (2001)
17. Vaux, L.: The differential $\lambda\mu$-calculus. Theoret. Comput. Sci. 379, 166–209 (2007)

Complexity of Planning in Action Formalisms Based on Description Logics

Maja Miličić*

Institut für Theoretische Informatik
TU Dresden, Germany
maja@tcs.inf.tu-dresden.de

Abstract. In this paper, we continue the recently started work on integrating action formalisms with description logics (DLs), by investigating planning in the context of DLs. We prove that the plan existence problem is decidable for actions described in fragments of $\mathcal{ALCQIO}$. More precisely, we show that its computational complexity coincides with the one of projection for DLs between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$ if operators contain only unconditional post-conditions. If we allow for conditional post-conditions, the plan existence problem is shown to be in 2-ExpSpace.

1 Introduction

Description Logics (DLs) are a well-known family of knowledge representation formalisms that may be viewed as decidable fragments of first-order logic (FO). The main strength of DLs is that they offer a nice compromise between expressiveness and complexity of reasoning [1].

The idea to investigate action formalisms based on description logics was inspired by the expressiveness gap between existing action formalisms: they were either based on FO logic and undecidable, like the Situation Calculus [14] and the Fluent Calculus [17], or decidable but only propositional.

First results on integrating DLs with action formalisms from [2] show that reasoning remains decidable even if an action formalism is based on the expressive DL $\mathcal{ALCQIO}$. In [2], ABoxes give incomplete descriptions of the current state of the world, and describe the pre- and post-conditions of actions. Domain constraints are captured by [illegible] TBoxes, and post-conditions may contain only atomic concept and role assertions. This formalism is in fact a decidable fragment of SitCalc. It is shown in [2] that the projection and executability problem for actions can be reduced to standard DL reasoning problems. Further work in this line [11,10] treat the problem of computing ABox updates and the ramification problem induced by GCIs.

However, in the mentioned DL-action-framework, planning, an important reasoning task, has not yet been considered. Intuitively, given an initial state $\mathcal{A}$, final state Γ and a finite set of actions Op, the [illegible] [illegible] [illegible] is the following: "is there a plan (a sequence of actions from Op) which transforms

---

* The author is supported by the EU project TONES.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 408–422, 2007.
© Springer-Verlag Berlin Heidelberg 2007

$\mathcal{A}$ into a state where Γ is satisfied?". It is known that, already in the propositional case, planning is a hard problem. For example, the plan existence problem for propositional STRIPS-style actions with complete state descriptions is PSPACE-complete [4,7], while it is EXPSPACE-complete for conformant planning (incomplete state descriptions) where actions have conditional post-conditions [8,15].

The planning problem in DL action formalisms is not only interesting from the theoretical point of view. It is well known that the semantic web ontology language OWL [9] is based on description logics; thus actions described in DLs can be viewed as simple semantic web services. In this context, planning is a very important reasoning task as it supports, e.g., web service discovery which is needed for an automatic service execution.

This paper is, to our best knowledge, the first attempt to formally define the planning problem in a DL fragment of SitCalc. We investigate the computational complexity of the plan existence problem for the description logics "between" $\mathcal{ALC}$ and $\mathcal{ALCQIO}$. By using a compact representation of possible states obtained by action application, we show that, if we allow only for actions with unconditional post-conditions, in these logics the plan existence problem is decidable, and of the same computational complexity as projection. If conditional post-conditions are allowed, we show that the plan existence problem is in 2-EXPSPACE. In the last section we discuss possible ways of developing practical planning algorithms for DLs.

2 The Description Logic $\mathcal{ALCQIO}$

The action formalism used in this paper is not restricted to a particular DL. However, for our complexity results we consider the DL $\mathcal{ALCQIO}$ and a number of its sublanguages. The reason for choosing this family of DLs is that they are very expressive, but nevertheless admit practical reasoning. Moreover, $\mathcal{ALCQIO}$ forms the core of OWL-DL, the description logic variant of OWL. As discussed in [2], the additional OWL-DL constructors can be easily added, except for transitive roles which lead to semantic and computational problems. Indeed, DLs from this family underlie highly optimized DL systems such as FaCT++, RacerPro, and Pellet.

In DL, concepts are inductively defined with the help of a set of [illegible], starting with a set $\mathsf{N_C}$ of [illegible] [illegible], a set $\mathsf{N_R}$ of [illegible] [illegible], and a set $\mathsf{N_I}$ of [illegible] [illegible]. The constructors determine the expressive power of the DL. Table 1 shows a minimal set of constructors from which all constructors of $\mathcal{ALCQIO}$ can be defined. The first row contains the only role constructor: in $\mathcal{ALCQIO}$, a [illegible] s is either a role name $r \in \mathsf{N_R}$ or the inverse r^- of a role name r. [illegible] of $\mathcal{ALCQIO}$ are formed using the remaining constructors shown in Table 1, where r is a role, n a positive integer, and a an individual name. Using these constructors, several other constructors can be defined as abbreviations:

- $C \sqcup D := \neg(\neg C \sqcap \neg D)$ (disjunction),
- $\top := A \sqcup \neg A$ for a concept name A (top-concept),

Table 1. Syntax and semantics of $\mathcal{ALCQIO}$

Name	Syntax	Semantics
inverse role	r^-	$\{(y, x) \mid (x, y) \in r^{\mathcal{I}}\}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
at-least restriction	$(\geqslant n\ s\ C)$	$\{x \in \Delta^{\mathcal{I}} \mid \mathit{card}\{y \in C^{\mathcal{I}} \mid (x, y) \in s^{\mathcal{I}}\} \geq n\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$

- $\exists s.C := (\geqslant 1\ s\ C)$ (existential restriction),
- $\forall s.C := \neg(\exists s.\neg C)$ (value restriction),
- $(\leqslant n\ s\ C) := \neg(\geqslant (n + 1)\ s\ C)$ (at-most restriction).

The DL that allows for negation, conjunction, and value restrictions is called $\mathcal{ALC}$. The availability of additional constructors is indicated by concatenating the corresponding letter: $\mathcal{Q}$ stands for number restrictions; $\mathcal{I}$ stands for inverse roles, and $\mathcal{O}$ for nominals. This explains the name $\mathcal{ALCQIO}$ for our DL, and also allows us to refer to sublanguages in a simple way.

The semantics of $\mathcal{ALCQIO}$-concepts and roles is defined in terms of an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The domain $\Delta^{\mathcal{I}}$ of $\mathcal{I}$ is a non-empty set of individuals and the interpretation function $\cdot^{\mathcal{I}}$ maps each concept name $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name $r \in \mathsf{N_R}$ to a binary relation $r^{\mathcal{I}}$ on $\Delta^{\mathcal{I}}$, and each individual name $a \in \mathsf{N_I}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ to arbitrary concepts and roles is inductively defined, as shown in the third column of Table 1. Here, the function *card* yields the cardinality of the given set. Note that the third column of Table 1 suggests a straightforward translation of DL concepts into first-order formulas with one free variable, as explicated e.g. in [1].

A *concept definition* is an identity of the form $A \doteq C$, where A is a concept name and C an $\mathcal{ALCQIO}$-concept. A *TBox* $\mathcal{T}$ is a finite set of concept definitions with unique left-hand sides. Concept names occurring on the left-hand side of a definition of $\mathcal{T}$ are called *defined in* $\mathcal{T}$ whereas the others are called *primitive in* $\mathcal{T}$. The TBox $\mathcal{T}$ is *acyclic* iff there are no cyclic dependencies between the definitions [1]. The *semantics* of TBoxes is defined in the obvious way: the interpretation $\mathcal{I}$ is a *model* of the TBox $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$) iff it satisfies all its definitions, i.e., $A^{\mathcal{I}} = C^{\mathcal{I}}$ holds for all $A \doteq C$ in $\mathcal{T}$. In the case of acyclic TBoxes, any interpretation of the primitive concepts and of the role names can uniquely be extended to a model of the TBox [12].

An *ABox assertion* is of the form $C(a)$, $r(a, b)$ or $\neg r(a, b)$, where $a, b \in \mathsf{N_I}$, C is a concept, and r a role name.[1] An *ABox* is a finite set of ABox assertions. The interpretation $\mathcal{I}$ is a *model* of the ABox $\mathcal{A}$ ($\mathcal{I} \models \mathcal{A}$) iff it satisfies all its assertions, i.e., $a^{\mathcal{I}} \in C^{\mathcal{I}}$ ($(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin r^{\mathcal{I}}$) for all assertions $C(a)$ ($r(a, b)$, $\neg r(a, b)$) in $\mathcal{A}$. If φ is an assertion, then we write $\mathcal{I} \models \varphi$ to indicate that $\mathcal{I}$ satisfies φ.

---

[1] Disallowing inverse roles in ABox assertions is not a restriction since $r^-(a, b)$ can be expressed by $r(b, a)$.

Various reasoning problems are considered for DLs. For the purpose of this paper, it suffices to introduce ABox consistency and ABox consequence: the ABox $\mathcal{A}$ is [illegible] w.r.t. the TBox $\mathcal{T}$ iff there exists an interpretation $\mathcal{I}$ that is a model of both $\mathcal{T}$ and $\mathcal{A}$; the ABox assertion φ is a [illegible] of $\mathcal{A}$ w.r.t. $\mathcal{T}$ (written $\mathcal{T}, \mathcal{A} \models \varphi$) iff every model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$ is also a model of φ.

3 Action Formalism

In this section, we present (a slightly extended version of) the action formalism from [2]. Since we focus on planning in this paper, the central notion become parameterized actions (operators), rather than ground actions like in [2].

The main ingredients of our framework are operators and actions (as defined below), ABoxes describing the current knowledge about the state of affairs in the application domain, and acyclic TBoxes for describing general knowledge about the application domain similar to state constraints in the SitCalc and Fluent Calculus.

Definition 1 (Action, operator). [illegible] $\mathsf{N_X}$ [illegible] variables [illegible] $\mathsf{N_C}$ $\mathsf{N_R}$ [illegible] NI [illegible] $\mathcal{T}$ [illegible] primitive literal for $\mathcal{T}$ [illegible]

$$A(a), \neg A(a), r(a,b), \quad \neg r(a,b)$$

[illegible] A [illegible] $\mathcal{T}$ r [illegible] $a, b \in \mathsf{N_I}$ [illegible] atomic atomic $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ [illegible] $\mathcal{T}$ [illegible]

- [illegible] pre [illegible] pre-conditions.
- [illegible] occ [illegible] occlusions [illegible] $A(a)$ [illegible] $r(a,b)$ [illegible] A [illegible] $\mathcal{T}$ r [illegible] $a, b \in \mathsf{N_I}$.
- [illegible] post [illegible] conditional post-conditions [illegible] φ/ψ [illegible] φ [illegible] ψ [illegible] primitive literal for $\mathcal{T}$

[illegible] composite action [illegible] $\mathcal{T}$ [illegible] $\alpha_1, \ldots, \alpha_k$ [illegible] $\mathcal{T}$ [illegible] operator for $\mathcal{T}$ [illegible] $\mathcal{T}$ [illegible] $\mathsf{N_X}$ [illegible]

We call post-conditions of the form $\top(t)/\psi$ [illegible] and write just ψ instead.

Applying an action changes the state of affairs, and thus transforms an interpretation $\mathcal{I}$ into an interpretation $\mathcal{I}'$. Intuitively, the pre-conditions specify under which conditions the action is applicable. The post-condition φ/ψ says that, if φ is true in the original interpretation $\mathcal{I}$, then ψ is true in the interpretation $\mathcal{I}'$ obtained by applying the action. The rôle of occlusions is to indicate those primitive literals that can change arbitrarily.

When defining the semantics of actions, we assume that states of the world correspond to interpretations. Thus, the semantics of actions can be defined by means of a transition relation on interpretations. We do not give semantics of operators explicitly and assume that they are grounded before their application. Let $\mathcal{T}$ be an acyclic TBox, $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ an action for $\mathcal{T}$, and $\mathcal{I}$ an interpretation. For each primitive concept name A and role name r, set:

$$\begin{aligned}
A^+ &:= \{b^{\mathcal{I}} \mid \varphi/A(b) \in \mathsf{post} \wedge \mathcal{I} \models \varphi\} \\
A^- &:= \{b^{\mathcal{I}} \mid \varphi/\neg A(b) \in \mathsf{post} \wedge \mathcal{I} \models \varphi\} \\
I_A &:= (\Delta^{\mathcal{I}} \setminus \{b^{\mathcal{I}} \mid A(b) \in \mathsf{occ}\}) \cup (A^+ \cup A^-) \\
r^+ &:= \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \varphi/r(a,b) \in \mathsf{post} \wedge \mathcal{I} \models \varphi\} \\
r^- &:= \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid \varphi/\neg r(a,b) \in \mathsf{post} \wedge \mathcal{I} \models \varphi\} \\
I_r &:= ((\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \setminus \{(a^{\mathcal{I}}, b^{\mathcal{I}}) \mid r(a,b) \in \mathsf{occ}\}) \cup (r^+ \cup r^-)
\end{aligned}$$

The transition relation on interpretations should ensure that $A^+ \subseteq A^{\mathcal{J}}$ and $A^- \cap A^{\mathcal{J}} = \emptyset$ if $\mathcal{J}$ is the result of applying α in $\mathcal{I}$. It should also ensure that nothing else changes, with the possible exception of the occluded literals. Intuitively, I_A and I_r describe those parts of the model that are [illegible] exempted from this restriction by the presence of an occlusion. Since we restrict our attention to acyclic TBoxes, for which the interpretation of defined concepts is uniquely determined by the interpretation of primitive concepts and role names, it is not necessary to consider defined concepts when defining the transition relation.

Definition 2. [illegible] $\mathcal{T}$ [illegible] $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ [illegible] $\mathcal{T}$ [illegible] $\mathcal{I}, \mathcal{I}'$ [illegible] $\mathcal{T}$ [illegible] α may transform $\mathcal{I}$ [illegible] $\mathcal{I}'$ $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha} \mathcal{I}'$ iff [illegible] A [illegible] r [illegible]

$$\begin{aligned}
A^+ \cap A^- = \emptyset \quad \text{[illegible]} \quad r^+ \cap r^- &= \emptyset \\
A^{\mathcal{I}'} \cap I_A &= ((A^{\mathcal{I}} \cup A^+) \setminus A^-) \cap I_A \\
r^{\mathcal{I}'} \cap I_r &= ((r^{\mathcal{I}} \cup r^+) \setminus r^-) \cap I_r.
\end{aligned}$$

[illegible] $\alpha_1, \dots, \alpha_k$ may transform $\mathcal{I}$ [illegible] $\mathcal{I}'$ $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha_1,\dots,\alpha_k} \mathcal{I}'$ iff [illegible] $\mathcal{I}_0, \dots, \mathcal{I}_k$ [illegible] $\mathcal{T}$ [illegible] $\mathcal{I} = \mathcal{I}_0$ $\mathcal{I}' = \mathcal{I}_k$ [illegible] $\mathcal{I}_{i-1} \Rightarrow^{\mathcal{T}}_{\alpha_i} \mathcal{I}_i$ [illegible] $1 \leq i \leq k$

Note that the semantics is such that the changes are minimized w.r.t. the initial interpretations. Also note that this definition does not check whether the action is indeed executable, i.e., whether the pre-conditions are satisfied. It just says what the result of applying the action is, irrespective of whether it is executable or not. Since we use acyclic TBoxes to describe background knowledge, if occ is the empty set, there cannot exist more than one $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha} \mathcal{I}'$. Thus, actions with empty occlusions are deterministic.

Like in [2], we assume that actions $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post})$ are [illegible] $\mathcal{T}$ in the following sense: for every model $\mathcal{I}$ of $\mathcal{T}$, there exists $\mathcal{I}'$, such that $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha} \mathcal{I}'$. It is not difficult to see that this is the case iff $\{\varphi_1/\psi, \varphi_2/\neg\psi\} \subseteq \mathsf{post}$ implies that the ABox $\{\varphi_1, \varphi_2\}$ is inconsistent w.r.t. $\mathcal{T}$.

Two standard reasoning problems about actions, [illegible] and [illegible], are thoroughly investigated in [2] in the context of DLs. Executability is the problem of whether an action can be applied in a given situation, i.e. if preconditions are satisfied in the states of the world considered possible.

Formally, let $\mathcal{T}$ be an acyclic TBox, $\mathcal{A}$ an ABox, and let $\alpha_1, \ldots, \alpha_n$ be a composite action with $\alpha_i = (\mathsf{pre}_i, \mathsf{occ}_i, \mathsf{post}_i)$ atomic actions for $\mathcal{T}$ for $i = 1, \ldots, n$.

We say that $\alpha_1, \ldots, \alpha_n$ is [illegible] $\mathcal{A}$ [illegible] $\mathcal{T}$ iff the following conditions are true for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$:

- $\mathcal{I} \models \mathsf{pre}_1$
- for all i with $1 \leq i < n$ and all interpretations $\mathcal{I}'$ with $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha_1,\ldots,\alpha_i} \mathcal{I}'$, we have $\mathcal{I}' \models \mathsf{pre}_{i+1}$.

Projection is the problem of whether applying an action achieves the desired effect, i.e., whether an assertion that we want to make true really holds after executing the action. Formally,the assertion φ is a [illegible] $\alpha_1, \ldots, \alpha_n$ [illegible] $\mathcal{A}$ [illegible] $\mathcal{T}$ iff for all models $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$ and for all $\mathcal{I}'$ with $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha_1,\ldots,\alpha_n} \mathcal{I}'$, we have $\mathcal{I}' \models \varphi$.

In [2] it was shown that projection and executability are decidable for the logics between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$. More precisely, projection in $\mathcal{L}$ can be reduced to (in)consistency of an ABox relative to an acyclic TBox in $\mathcal{LO}$. The following theorem from [2] states that upper complexity bounds obtained in this way are optimal:

Theorem 1. [illegible]

[illegible] PSpace[illegible] $\mathcal{ALC}$ $\mathcal{ALCO}$ $\mathcal{ALCQ}$ [illegible] $\mathcal{ALCQO}$.
[illegible] ExpTime[illegible] $\mathcal{ALCI}$ [illegible] $\mathcal{ALCIO}$.
[illegible] NExpTime[illegible] $\mathcal{ALCQI}$ [illegible] $\mathcal{ALCQIO}$

Looking carefully at the reduction of projection in $\mathcal{L}$ to ABox inconsistency in $\mathcal{LO}$ from [2,3], we conclude that the upper complexity bounds from Theorem 1 hold even for the "stronger" projection problem, namely the one where instead of a single ABox assertion φ, we have an ABox Γ. We will need this strengthened complexity result in the coming sections.

4 Planning Problem

We continue by defining the plan existence problem in the introduced framework. First we introduce a bit of notation. If o is an operator (for a TBox $\mathcal{T}$), we use $\mathsf{var}(o)$ to denote the set of variables in o. A substitution v for o is a mapping $v : \mathsf{var}(o) \rightarrow \mathsf{N_I}$. An action α that is obtained by applying a substitution v to o is denoted as $\alpha := o[v]$. Intuitively, the plan existence problem is: given an acyclic TBox $\mathcal{T}$ which describes the background knowledge, ABoxes $\mathcal{A}$ and Γ giving incomplete descriptions of the initial and the goal state, and a set of operators Op, is there a plan (sequence of actions obtained by instantiating operators from Op) which "transforms" the stated described by $\mathcal{A}$ into a state where Γ is satisfied?

In this paper, we assume that operators can be instantiated with individuals from a finite set $\mathsf{Ind} \subset \mathsf{N_I}$. Moreover, we assume that $\mathcal{T}$, $\mathcal{A}$ and Γ contain only individuals from Ind (we say that they are based on Ind). For an operator o, we set $o[\mathsf{Ind}] := \{o[v] \mid v : \mathsf{var}(o) \to \mathsf{Ind}\}$ and for Op a set of operators, we set $\mathsf{Op}[\mathsf{Ind}] := \{o[\mathsf{Ind}] \mid o \in \mathsf{Op}\}$, i.e. Op[Ind] is the set of all actions obtained by instantiating operators from Op with individuals from Ind. In the following definition, we formally introduce the notion of a planing task:

Definition 3 (Planning task). [illegible] $\Pi = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma)$ [illegible]

- Ind [illegible]
- $\mathcal{T}$ [illegible] Ind.
- Op [illegible] $\mathcal{T}$.
- $\mathcal{A}$ [illegible] Ind.
- Γ [illegible] Ind

[illegible] plan [illegible] Π [illegible] $\alpha = \alpha_1, \ldots, \alpha_k$ [illegible] $\alpha_i \in \mathsf{Op}[\mathsf{Ind}]$ $i = 1..k$. [illegible] $\alpha = \alpha_1, \ldots, \alpha_k$ [illegible] Π [illegible] solution [illegible] Π [illegible]

- α [illegible] $\mathcal{A}$ [illegible] $\mathcal{T}$ [illegible]
- [illegible] $\mathcal{I}$ [illegible] $\mathcal{I}'$ [illegible] $\mathcal{I} \models \mathcal{A}, \mathcal{T}$ [illegible] $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha} \mathcal{I}'$ [illegible] $\mathcal{I}' \models \Gamma$

[illegible] We illustrate the previous definition by the following example describing a (simplified) process of opening a bank account in the UK.

Let the set of individuals be defined as

$$\mathsf{Ind} = \{\mathsf{dirk}, \mathsf{uni\_liv}, \mathsf{yoga\_center}, \mathsf{UK}, \mathsf{el}, \mathsf{el}', \mathsf{l}, \mathsf{ba}\}.$$

The initial state - ABox $\mathcal{A}$ states that Dirk is a resident of the UK who has gotten two jobs – at the University of Liverpool and in the Yoga Center, but still does not hold a bank account in the UK.

$$\begin{aligned} \mathcal{A} := \{&\mathsf{resident}(\mathsf{dirk}, \mathsf{UK}), \mathsf{employs}(\mathsf{uni\_liv}, \mathsf{dirk}), \mathsf{employs}(\mathsf{yoga\_center}, \mathsf{dirk}), \\ &\mathsf{University}(\mathsf{uni\_liv}), \neg\exists \mathsf{holds}.(\mathsf{B\_acc} \sqcap \exists \mathsf{in}.\{\mathsf{UK}\})(\mathsf{dirk})\} \end{aligned}$$

Moreover, the set Op contains the operators for obtaining a lease, a letter from employer, and a bank account. The set of occlusions occ is empty for all three operators, so we will state only the sets of pre- and post-conditions pre and post.

- Suppose the pre-condition of obtaining a lease is that the customer x holds a letter from his employer. This is formalized by the operator get_Lease:

$$\begin{aligned} \mathsf{pre}: &\ \{\exists \mathsf{holds}.\mathsf{EmployerLetter}(x)\} \\ \mathsf{post}: &\ \{\mathsf{holds}(x, y), \mathsf{Lease}(y)\} \end{aligned}$$

- The operator $\mathsf{get\_Letter}$ describes the process of an employee x getting a letter y from his employer z:

$$\begin{aligned} \mathsf{pre}: &\ \{\mathsf{employs}(z,x)\} \\ \mathsf{post}: &\ \{\mathsf{holds}(x,y), \mathsf{EmployerLetter}(y), \mathsf{signed}(z,y)\} \end{aligned}$$

- Suppose the pre-condition of opening a bank account is that the customer x is a resident in the UK and holds a proof of address. Moreover, suppose that, if x is rated as "reliable", then the bank account comes with a credit card, otherwise not. This service can be formalized by the following operator $\mathsf{get\_B\_acc}$:

$$\begin{aligned} \mathsf{pre}: &\ \{\exists \mathsf{resident}.\{\mathsf{UK}\}(x), \exists \mathsf{holds}.\mathsf{Proof\_address}(x)\} \\ \mathsf{post}: &\ \{\mathsf{holds}(x,y), \mathsf{in}(y,\mathsf{UK}), \\ &\quad \mathsf{Reliable}(x)/\mathsf{B\_acc\_credit}(y), \\ &\quad \neg\mathsf{Reliable}(x)/\mathsf{B\_acc\_no\_credit}(y)\} \end{aligned}$$

The meaning of the concepts used in $\mathcal{A}$ and Op is defined in the following acyclic TBox $\mathcal{T}$:

$$\begin{aligned} \mathsf{Reliable} &\doteq \exists \mathsf{holds}.(\mathsf{B\_acc} \sqcap \mathsf{Good\_credit\_rating} \sqcap \exists \mathsf{in}.\{\mathsf{UK}\}) \\ &\quad \sqcup \exists \mathsf{holds}.(\mathsf{EmployerLetter} \sqcap \exists \mathsf{signed}^{-}.\mathsf{University}\} \\ \mathsf{Proof\_address} &\doteq \mathsf{Electricity\_contract} \sqcup \mathsf{Lease} \\ \mathsf{B\_acc} &\doteq \mathsf{B\_acc\_credit} \sqcup \mathtt{B\_acc\_no\_credit} \end{aligned}$$

The first concept definition tells us that a person is rated as reliable if and only if he already holds a bank account in the UK with a good credit rating, or holds a letter stating that he is employed at the university. The second definition defines a proof of the address to be either en electricity contract or a lease, while the last one states that a bank account can come either with or without a credit card.

Finally, we have two goals, $\Gamma_1 = \{\exists \mathsf{holds}.(\mathsf{B\_acc} \sqcap \exists \mathsf{in}.\{\mathsf{UK}\})(\mathsf{dirk})\}$, requiring that Dirk holds a bank account in the UK, and a more ambitious one, $\Gamma_2 = \{\exists \mathsf{holds}.(\mathsf{B\_acc\_credit} \sqcap \exists \mathsf{in}.\{\mathsf{UK}\})(\mathsf{dirk})\}$, namely that Dirk holds a bank account in the UK with a credit card. We define corresponding planing tasks Π_1 and Π_2 as $\Pi_1 = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma_1)$ and $\Pi_2 = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma_2)$. It is not difficult to see that the plan:

$$\mathsf{get\_Letter}[x/\mathsf{dirk}, y/\mathsf{el}, z/\mathsf{yoga\_center}], \mathsf{get\_Lease}[x/\mathsf{dirk}, y/\mathsf{l}], \mathsf{get\_B\_acc}[x/\mathsf{dirk}, y/\mathsf{ba}]$$

is a solution to Π_1, but not Π_2, while the plan:

$$\mathsf{get\_Letter}[x/\mathsf{dirk}, y/\mathsf{el}', z/\mathsf{uni\_liv}], \mathsf{get\_Lease}[x/\mathsf{dirk}, y/\mathsf{l}], \mathsf{get\_B\_acc}[x/\mathsf{dirk}, y/\mathsf{ba}]$$

is a solution both to Π_1 and Π_2.

The [illegible], c.f. [7], is the problem of whether a given planning task Π has a solution. If operators in Π contain conditional post-conditions, we will call it [illegible], and otherwise [illegible].

5 Complexity of Planning: Unconditional Post-Conditions

In this section, we will focus on the plan existence problem in the case operators have only unconditional post-conditions. It turns out that unconditional PLANEX is not harder, at least in theory, than projection in the fragments of $\mathcal{ALCQIO}$ from Theorem 1.

Obviously, the plan existence problem is closely related to projection and executability. First we introduce some notation. Let $\mathcal{A}$ be an ABox, $\mathcal{T}$ an acyclic TBox, α a (possibly composite) action, and φ an ABox assertion. We will write $\mathcal{T}, \mathcal{A}^\alpha \models \varphi$ iff φ is a consequence of applying α in $\mathcal{A}$ w.r.t. $\mathcal{T}$. For an ABox $\mathcal{B}$, we write $\mathcal{T}, \mathcal{A}^\alpha \models \mathcal{B}$ iff $\mathcal{T}, \mathcal{A}^\alpha \models \varphi$ for all $\varphi \in \mathcal{B}$.

Let $\mathit{\Pi} = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \mathit{\Gamma})$ be a planning task for which we want to decide if it has a solution. This means that we want to check if there is a sequence of actions from $\mathsf{Op}[\mathsf{Ind}]$ which transform the initial state (described by $\mathcal{A}$) into a state where goal $\mathit{\Gamma}$ holds.

In the propositional case, planning is based on step-wise computation of the next state – which corresponds to computing updated ABoxes. However, in [11], it is shown that an updated ABox may be exponentially large in the size of the initial ABox and the update, which makes this approach unsuitable. We base our approach in this paper on the following observation: instead of computing a sequence of (exponentially large) updated ABoxes, it suffices to compute a sequence of [illegible] which are applied to the initial ABox $\mathcal{A}$. Intuitively, these updates are lists of accumulated triggered post-conditions. Similarly, we keep track of accumulated occlusions. Thus, states of the search space can be compactly described as pairs: (occlusion, update).

We define the set of possible (negated) atomic changes as:

$$\mathcal{L}_{\mathsf{post}} := \{\psi, \neg\psi \mid \psi \in \mathsf{post},\ \alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post}),\ \alpha \in \mathsf{Op}[\mathsf{Ind}]\}$$

and the set of possible occlusions:

$$\mathcal{L}_{\mathsf{occ}} := \{\psi \mid \psi \in \mathsf{occ},\ \alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post}),\ \alpha \in \mathsf{Op}[\mathsf{Ind}]\}$$

An [illegible] for $\mathit{\Pi}$ is a consistent subset of $\mathcal{L}_{\mathsf{post}}$. Let $\mathfrak{U}$ be a set of all updates for $\mathit{\Pi}$. Moreover, let $\mathfrak{O} := 2^{\mathcal{L}_{\mathsf{occ}}}$. Then $\mathfrak{O} \times \mathfrak{U}$ is our search space, the size of which $|\mathfrak{O}| \cdot |\mathfrak{U}|$ is exponential in the size of $\mathit{\Pi}$, since the sizes of $\mathcal{L}_{\mathsf{post}}$ and $\mathcal{L}_{\mathsf{occ}}$ are polynomial in $\mathit{\Pi}$. For a $\mathcal{U} \in \mathfrak{U}$, we set $\neg\mathcal{U} := \{\neg l \mid l \in \mathcal{U}\}$ and $\overline{\mathcal{U}} := \{l \mid l \in \mathcal{U} \cup \neg\mathcal{U}$ and l positive$\}$.

Intuitively, $(\emptyset, \emptyset)$ represents the initial state, and all tuples $(\mathcal{O},\mathcal{U}) \in \mathfrak{O} \times \mathfrak{U}$ such that $\mathcal{T}, \mathcal{A}^{(\emptyset,\mathcal{O},\mathcal{U})} \models \mathit{\Gamma}$ represent goal states. In the next step, we define the transition relation "$\xrightarrow{\alpha}_{\mathcal{T},\mathcal{A}}$" on $\mathfrak{O} \times \mathfrak{U}$. Let $(\mathcal{O},\mathcal{U}),(\mathcal{O}',\mathcal{U}') \in \mathfrak{O} \times \mathfrak{U}$. We say that $(\mathcal{O},\mathcal{U}) \xrightarrow{\alpha} (\mathcal{O}',\mathcal{U}')$ iff:

(i) $\mathcal{O}' = (\mathcal{O} \cup \mathsf{occ}) \setminus \overline{\mathsf{post}}$
(ii) $\mathcal{U}' = (\mathcal{U} \setminus (\mathsf{occ} \cup \neg\mathsf{occ} \cup \neg\mathsf{post})) \cup \mathsf{post}$

Obviously, the relation "$\xrightarrow{\alpha}$" is functional for every α. In the following lemma, we show that "$\xrightarrow{\alpha}$" simulates "$\Rightarrow_{\alpha}^{\mathcal{T}}$" on the set $\mathfrak{O} \times \mathfrak{U}$. We omit the proof, which can be done by an easy induction.

Lemma 1. [illegible] $\Pi = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma)$ [illegible] $\alpha = \alpha_1, \dots, \alpha_k$ [illegible] Π [illegible] $\mathcal{U}_0 = \mathcal{O}_0 := \emptyset$ [illegible] $(\mathcal{O}_1,\mathcal{U}_1), \dots, (\mathcal{O}_k,\mathcal{U}_k)$ [illegible]

$$(\mathcal{O}_0,\mathcal{U}_0) \xrightarrow{\alpha_1} (\mathcal{O}_1,\mathcal{U}_1) \cdots \xrightarrow{\alpha_k} (\mathcal{O}_k,\mathcal{U}_k)$$

[illegible]

[illegible] $\mathcal{I}, \mathcal{I}'$ [illegible] $\mathcal{I} \models \mathcal{A}$ [illegible] $1 \leq i \leq k$ [illegible] $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\alpha_1,\dots,\alpha_i} \mathcal{I}'$ iff $\mathcal{I} \Rightarrow^{\mathcal{T}}_{(\emptyset,\mathcal{O}_i,\mathcal{U}_i)} \mathcal{I}'$

[illegible] $\mathcal{T}, \mathcal{A}^{(\emptyset,\mathcal{O}_i,\mathcal{U}_i)} \models \mathsf{pre}_{i+1}$ [illegible] $i < k$ iff $\alpha_1, \dots, \alpha_k$ [illegible] $\mathcal{A}$ [illegible] $\mathcal{T}$.

We now present a non-deterministic procedure which decides whether the planning task Π has a solution. The procedure searches for an executable sequence of actions from $\mathsf{Op}[\mathsf{Ind}]$ which transforms the initial state $S_0 = (\emptyset, \emptyset)$ into a state $\mathcal{S}_\Gamma = (\mathcal{O}_\Gamma,\mathcal{U}_\Gamma) \in \mathfrak{O} \times \mathfrak{U}$ such that $\mathcal{T}, \mathcal{A}^{\mathcal{S}_\Gamma} \models \Gamma$ [2] (goal state). We use ϵ do denote the empty action $(\emptyset, \emptyset, \emptyset)$. Since the search space $\mathfrak{O} \times \mathfrak{U}$ is of size $2^{|\mathcal{L}_{\mathsf{occ}}|} \cdot 3^{\frac{|\mathcal{L}_{\mathsf{post}}|}{2}}$ $(< 2^{|\mathcal{L}_{\mathsf{occ}}|+|\mathcal{L}_{\mathsf{post}}|})$, there is no need to search for longer sequences than $2^{|\mathcal{L}_{\mathsf{occ}}|+|\mathcal{L}_{\mathsf{post}}|}$.

```
PLANEX($\Pi$)
    $i := 0$; $\mathcal{S}_0 := (\emptyset, \emptyset)$;
    while $i < 2^{|\mathcal{L}_{\mathsf{occ}}|+|\mathcal{L}_{\mathsf{post}}|}$
        guess $\alpha = (\mathsf{pre}, \mathsf{occ}, \mathsf{post}) \in \mathsf{Op}[\mathsf{Ind}] \cup \{\epsilon\}$
        if $\mathcal{T}, \mathcal{A}^{\mathcal{S}_i} \not\models \mathsf{pre}$
            then return FALSE
        compute $\mathcal{S}_{i+1}$ such that $\mathcal{S}_i \xrightarrow{\alpha} \mathcal{S}_{i+1}$
        $i := i + 1$
    if $\mathcal{T}, \mathcal{A}^{\mathcal{S}_i} \not\models \Gamma$
        then return FALSE
    return TRUE
```

It is not difficult to show that Lemma 1 implies that **PLANEX**(Π) returns TRUE iff Π has a solution. Clearly, **PLANEX**(Π) works in NPSpace with the "projection oracle". If projection is in PSpace, then PLANEX is obviously in NPSpace. By using Savitch's result [16] that PSpace = NPSpace, we obtain that PLANEX is then in PSpace. Similarly, if projection is in ExpTime, since NPSpace $\subseteq$ ExpTime, we have that PLANEX can be decided in ExpTime. Finally, we will show the less straightforward result that PLANEX is in co-NExpTime if projection is in co-NExpTime. To this end, we develop an alternative NExpTime algorithm which returns TRUE iff the planning task Π has no solution. Let $\mathfrak{S} = \mathfrak{O} \times \mathfrak{U}$ and $\mathfrak{Q} = \{\mathsf{pre}(\alpha) \mid \alpha \in \mathsf{Op}[\mathsf{Ind}]\} \cup \{\Gamma\}$. The alternative algorithm has three steps: (i) guess an (exponentially big) set T of tuples $(\mathcal{S}, \mathcal{Q})$ from $\mathfrak{S} \times \mathfrak{Q}$; (ii) check whether for all tuples $(\mathcal{S}, \mathcal{Q})$ it holds that $\mathcal{T}, \mathcal{A}^{\mathcal{S}} \not\models \mathcal{Q}$; (iii) check if the following holds: in every run of the original **PLANEX**(Π) procedure, there is at least one projection test $\mathcal{T}, \mathcal{A}^{\mathcal{S}} \models \mathcal{Q}$?

---

[2] From now on, if $\mathcal{S} = (\mathcal{O},\mathcal{U})$, we write $\mathcal{S}$ as an abbreviation for the action $(\emptyset, \mathcal{O},\mathcal{U})$.

such that $(\mathcal{S}, \mathcal{Q}) \in T$. If (ii) and (iii) give positive answers, return TRUE. Since (ii) and (iii) can be checked in ExpTime, the steps (i)-(iii) can be executed in NExpTime. Thus, we obtained the following lemma:

Lemma 2. $\mathcal{L} \in \{\mathcal{ALC}, \mathcal{ALCO}, \mathcal{ALCI}, \mathcal{ALCQ}, \mathcal{ALCIO}, \mathcal{ALCQO}, \mathcal{ALCQI}, \mathcal{ALCQIO}\}$ $\mathcal{L}$ $\mathcal{L}$

We show that the upper complexity bounds established in Lemma 1 are tight by the following easy reduction of projection to PLANEX. Let $\mathcal{A}$ be an ABox, α an action with empty pre-conditions and empty occlusions and only with unconditional post-conditions, and φ an assertion.We define the planning task $\Gamma_{\mathcal{A},\alpha,\varphi}$ as $\Gamma_{\mathcal{A},\alpha,\varphi} := (\emptyset, \emptyset, \{\alpha\}, \mathcal{A}, \{\varphi\})$. It is not difficult to see that $\mathcal{A}^{\alpha} \models \varphi$ iff $\Gamma_{\mathcal{A},\alpha,\varphi}$ has a solution.

Since the lower bounds for projection from Theorem 1 hold already in the case of the empty TBox and an atomic action with empty pre-conditions and occlusions and only with unconditional post-conditions [2], we conclude that the complexity bounds from Lemma 2 are optimal, i.e. plan existence problem is of exactly the same computational complexity as projection.

Theorem 2.

PSpace $\mathcal{ALC}$ $\mathcal{ALCO}$ $\mathcal{ALCQ}$ $\mathcal{ALCQO}$.
ExpTime $\mathcal{ALCI}$ $\mathcal{ALCIO}$.
NExpTime $\mathcal{ALCQI}$ $\mathcal{ALCQIO}$

6 Complexity of Planning: Conditional Post-Conditions

If we allow for conditional post-conditions in operators, the complexity results from the previous section do not hold anymore. With conditional post-conditions, already in the propositional case, conformant PLANEX is ExpSpace-hard [8,15]. In this section we will show that conditional PLANEX is decidable for DLs between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$. Decidability will be shown by an 2-ExpSpace algorithm.

Let $\Pi = (\mathsf{Ind}, \mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma)$ be a planning task for which we want to decide if it has a solution. For the sake of simplicity, we assume that occlusions in operators from Op are empty, i.e. operators are of the form $(\mathsf{pre}, \mathsf{post})$. Non-empty occlusions can be treated similarly as in the previous section. We will also use abbreviations introduced in the previous section. Moreover, we set

$$\mathcal{L}_{\mathsf{post}} := \{\psi, \neg\psi \mid \varphi/\psi \in \mathsf{post}, \alpha = (\mathsf{pre}, \mathsf{post}), \alpha \in \mathsf{Op}[\mathsf{Ind}]\}$$

and

$$\mathcal{C}_{\mathsf{post}} := \{\varphi \mid \varphi/\psi \in \mathsf{post}, \alpha = (\mathsf{pre}, \mathsf{post}), \alpha \in \mathsf{Op}[\mathsf{Ind}]\}.$$

An in Π is a consistent subset of $\mathcal{L}_{\mathsf{post}}$. Let $\mathfrak{U}$ be the set of all updates in Π. A $\mathcal{C}$ in Π is a consistent subset of $\mathcal{C}_{\mathsf{post}} \cup \neg\mathcal{C}_{\mathsf{post}}$ such that for every

$\varphi \in \mathcal{C}_{\mathsf{post}}$, it is the case that either $\varphi \in \mathcal{C}$ or $\neg\varphi \in \mathcal{C}$. Moreover let $\mathfrak{C}$ be the set of all contexts in Π. Let $\mathfrak{M}$ be the set of [illegible] mappings $m : \mathfrak{U} \to \mathfrak{C}$, where a mapping m is admissible iff there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{A}, \mathcal{T}$ and for all $\mathcal{U} \in \mathfrak{U}$ it holds that $\mathcal{I}^{\mathcal{U}}_{\mathcal{T}} \models m(\mathcal{U})$.[3]. Intuitively, if $m(\mathcal{U}) = \mathcal{C}$, it means that after updating a model $\mathcal{I}$ of $\mathcal{A}$ and $\mathcal{T}$ with $\mathcal{U}$, all assertions from $\mathcal{C}$ will hold. Thus every admissible m describes a relevant class of possible initial models of $\mathcal{A}$ and $\mathcal{T}$. The number of different mappings $m : \mathfrak{U} \to \mathfrak{C}$ is at most $2^{|\mathcal{C}_{\mathsf{post}}| \cdot 2^{\mathcal{L}_{\mathsf{post}}}}$, and for every m it can be checked in ExpSpace if it is admissible, if projection is in ExpSpace. The search space $\mathfrak{S}$ is the set of all mappings $\mathcal{S} : \mathfrak{M} \to \mathfrak{U}$, the size of which $|\mathfrak{S}| \leq 2^{|\mathcal{L}_{\mathsf{post}}| \cdot 2^{|\mathcal{C}_{\mathsf{post}}| \cdot 2^{\mathcal{L}_{\mathsf{post}}}}}$. Similarly as in the previous section, we define the transition relation $\xrightarrow{\alpha}$ on $\mathfrak{S} \times \mathfrak{S}$. For $\mathcal{S}, \mathcal{S}' \in \mathfrak{S}$, $m \in \mathfrak{M}$, and $\alpha = (\mathsf{pre}, \mathsf{post})$ we set $\mathsf{post}_{\alpha,\mathcal{S},m} := \{\psi \mid \varphi/\psi \in \mathsf{post}, \varphi \in m(\mathcal{S}(m))\}$. We say that $\mathcal{S} \xrightarrow{\alpha} \mathcal{S}'$ iff

$$\mathcal{S}'(m) = (\mathcal{S}(m) \setminus \neg\mathsf{post}_{\alpha,\mathcal{S},m}) \cup \mathsf{post}_{\alpha,\mathcal{S},m} \quad \text{for all } m \in \mathfrak{M}$$

Moreover, for $\mathcal{B}$ be an ABox, we say that $\mathcal{T}, \mathcal{A}^{\mathcal{S}} \models^* \mathcal{B}$ iff for all $m \in \mathfrak{M}$ the following holds: for all interpretations $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{A}, \mathcal{T}$, if for all $\mathcal{U} \in \mathfrak{U}$ it holds that $\mathcal{I}^{\mathcal{U}}_{\mathcal{T}} \models m(\mathcal{U})$, then $\mathcal{I}^{\mathcal{S}(m)}_{\mathcal{T}} \models \mathcal{B}$.

```
condPLANEX($\Pi$)
    $i := 0$; $\mathcal{S}_0(m) = \emptyset$ for all $m \in \mathfrak{M}$;
    while $i < 2^{|\mathcal{L}_{\mathsf{post}}| \cdot 2^{|\mathcal{C}_{\mathsf{post}}| \cdot 2^{\mathcal{L}_{\mathsf{post}}}}}$
        if $\mathcal{T}, \mathcal{A}^{\mathcal{S}_i} \models^* \Gamma$
            then return TRUE
        guess $\alpha = (\mathsf{pre}, \mathsf{post}) \in \mathsf{Op}[\mathsf{Ind}]$
        if $\mathcal{T}, \mathcal{A}^{\mathcal{S}_i} \not\models^* \mathsf{pre}$
            then return FALSE
        compute $\mathcal{S}_{i+1}$ such that $\mathcal{S}_i \xrightarrow{\alpha} \mathcal{S}_{i+1}$
        $i := i + 1$
    return FALSE
```

It is not difficult to show that **condPLANEX**(Π) indeed decides if Π has a solution. The search space $\mathfrak{S}$ is 3-exponential in the size of Π, thus **condPLANEX** requires 2-NExpSpace with a "$\models^*$ oracle". It is not difficult to see that $\models^*$ can be decided in 2-ExpSpace if projection is in ExpSpace. Thus we have that conditional PLANEX in 2-NExpSpace. Since 2-NExpSpace =2-ExpSpace [16], we obtain the following theorem:

Theorem 3. [illegible] ExpSpace [illegible] $\mathcal{ALC}$ [illegible] $\mathcal{ALCQIO}$

7 Individuals Not Part of Input

The previous decidability and complexity results are obtained under assumption that the set of individuals Ind used to instantiate operators is finite and a part of the input. This assumption is rather natural and in the line with the standard definitions of planning tasks for STRIPS operators from [4,7]. Intuitively, Ind is the set of individuals the planning agent has control over.

---

[3] $\mathcal{I}^{\mathcal{U}}_{\mathcal{T}}$ denotes the unique interpretation $\mathcal{I}'$ such that $\mathcal{I} \Rightarrow^{\mathcal{T}}_{\mathcal{U}} \mathcal{I}'$.

Alternatively, one can omit individuals from the input and define a planning task Π as $\Pi = (\mathcal{T}, \mathsf{Op}, \mathcal{A}, \Gamma)$. The [illegible] is the one of whether there is a solution for Π, defining a plan for Π to be a sequence of actions $\alpha_1, \dots, \alpha_k$, where each α_i is obtained by instantiating an operator from Op with individuals from an infinitely countable set $\mathsf{N_I}$.

In the case of the datalog STRIPS, it is shown that the extended plan existence problem is undecidable [7,6]. However, this undecidability result does not automatically carry over to the action formalism instantiated by DLs used in this paper. Indeed, the undecidability result from [7,6] relies on the closed world assumption and negative pre-conditions. By using these two, one can define operators which are applicable only if instantiated with "unused" individuals. Such operators would have $\neg\mathsf{Used}(x)$ among its pre-conditions, and $\mathsf{Used}(x)$ in the list of post-conditions. Like this, one can enforce the usage of infinitely many individuals.

In the case of DLs considered in the previous sections, due to the open world assumption (OWA), it is not possible to state that all individuals not appearing in the initial ABox are instances of the concept $\neg\mathsf{Used}$. However, in the presence of the universal role U, we can make assertions over the whole domain. For example, the assertion $\forall U.\neg\mathsf{Used}(a)$ can ensure that all element domains are unused in the initial state. We will show that extended planning in $\mathcal{ALC}_U$ (extension of $\mathcal{ALC}$ with the universal role) is undecidable. Undecidability is shown by reducing the halting problem of a deterministic Turing machine to the extended plan existence problem, similar to [6].

Let $M = (Q, \Sigma, \delta, q_0, q_f)$ be a deterministic Turing machine, where

- $Q = \{q_0, \dots, q_n\}$ a finite set of states;
- $\Sigma = \{\mathsf{blank}, a_1, \dots, a_m\}$ a finite alphabet;
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is a transition function;
- q_0 is the initial state;
- $q_f \in Q$ is the final state.

Let $a = a_{i_0}, \dots a_{i_k} \in \Sigma^*$ be an input word. We will define a planning task $\Pi_{M,a} = (\emptyset, \mathsf{Op}_{M,a}, \mathcal{A}_{M,a}, \Gamma_{M,a})$ such that a planner for Π simulates moves of the Turing Machine M. In the reduction, we use concept names $Q_0, \dots, Q_n$, $\mathsf{Blank}, A_1, \dots, A_m$, Used, Last, M, Done, and the role name right. We define the initial state $\mathcal{A}_{M,a}$, the goal $\Gamma_{M,a}$, and the set of operators $\mathsf{Op}_{M,a}$ as:

$$
\begin{aligned}
\mathcal{A}_{M,a} &:= \{(M \sqcap \forall U.\neg\mathsf{Used})(t_0)\} \cup \{A_{i_0}(t_0), \dots, A_{i_k}(t_k)\} \\
&\quad \cup \{\mathsf{right}(t_0, t_1), \dots \mathsf{right}(t_{k-1}, t_k)\} \\
\Gamma_{M,a} &:= \{\mathsf{Done}(t_0)\} \\
\mathsf{Op}_{M,a} &:= \{\mathsf{start}, \mathsf{create\_succ}(x.y), \mathsf{done}(x), \mathsf{done\_to\_left}(x, y)\} \cup \\
&\qquad \bigcup_{\delta(q,a)=(q',b,R)} \{\mathsf{right}_{q,a,q',b}(x, y)\} \cup \bigcup_{\delta(q,a)=(q',b,L)} \{\mathsf{left}_{q,a,q',b}(x, y)\}
\end{aligned}
$$

where the single operators (of the form $(\mathsf{pre}, \mathsf{post})$, $\mathsf{occ} = \emptyset$ for all operators) are defined as follows :

$$\begin{aligned}
\mathsf{start} &:= (\{M(t_0)\}, \{\mathsf{Used}(t_0), ..., \mathsf{Used}(t_k), \mathsf{Last}(t_k), \neg M(t_0), Q_0(t_0)\}) \\
\mathsf{create\_succ}(x.y) &:= (\{\mathsf{Last}(x), \neg\mathsf{Used}(y)\}, \\
&\qquad \{\mathsf{right}(x, y), \neg\mathsf{Last}(x), \mathsf{Last}(y), \mathsf{Used}(y), \mathsf{Blank}(y)\} \\
\mathsf{right}_{q,a,q',b}(x, y) &:= (\{Q(x), A(x), \mathsf{right}(x, y)\}, \{\neg Q(x), \neg A(x), B(x), Q'(y)\} \\
\mathsf{left}_{q,a,q',b}(x, y) &:= (\{Q(x), A(x), \mathsf{right}(y, x)\}, \{\neg Q(x), \neg A(x), B(x), Q'(y)\} \\
\mathsf{done}(x) &:= (\{Q_f(x)\}, \{\mathsf{Done}(x)\}) \\
\mathsf{done\_to\_left}(x, y) &:= (\{\mathsf{Done}(x), \mathsf{right}(y, x)\}, \{\mathsf{Done}(y)\})
\end{aligned}$$

It is not difficult to show that the following lemma holds:

Lemma 3. [illegible] M [illegible] a [illegible] $\Pi_{M,a} = (\emptyset, \mathsf{Op}_{M,a}, \mathcal{A}_{M,a}, \Gamma_{M,a})$

Thus, we obtained the following theorem:

Theorem 4. [illegible] $\mathcal{ALC}_U$

We conjecture that in the fragments of $\mathcal{ALCQIO}$ (i.e. without the universal role) the extended plan existence problem is decidable. However, a proof is yet to be done.

8 Conclusion and Future Work

In this paper, we have shown that the plan existence problem (PLANEX) is decidable in the action formalism based on fragments of $\mathcal{ALCQIO}$. More precisely, PLANEX is shown to be of the same computational complexity as projection in the logics between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$ if operators have only unconditional post-conditions. It is also shown that occlusions do not make planning harder. If operators have conditional post-conditions, planning is shown to be in 2-ExpSpace. At the moment, it remains an open problem if this complexity bound is optimal. Finally, we have shown that the extended plan existence problem is undecidable in DLs providing for the universal role but we conjecture that it is decidable in the fragments of $\mathcal{ALCQIO}$ (without the universal role).

A future work will include a development and implementation of efficient planners for description logics. Unfortunately, the complexity results we obtained are quite discouraging. Unlike the propositional case, for DLs between $\mathcal{ALC}$ and $\mathcal{ALCQIO}$, looking for polynomial-length plans is not easier than PLANEX, since the hardness results from Theorem 2 hold already for the plans of constant length. Thus it looks reasonable to start with “small” DLs, like $\mathcal{EL}$ or $\mathcal{EL}^{(\neg)}$, for which projection is in co-NP, and try to adapt some of the known techniques for SAT-based conformant planning [5,13].

Acknowledgements. Thanks to Carsten Lutz for inspiring discussions. Many thanks to Ricard Gavaldà for his help concerning complexity classes.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, Cambridge (2003)
2. Baader, F., Lutz, C., Milicic, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. In: Proceedings of AAAI 2005, Pittsburgh, PA, USA (2005)
3. Baader, F., Lutz, C., Milicic, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms for reasoning about web services. Technical Report LTCS 05-02, TU Dresden (2005), See `http://lat.inf.tu-dresden.de/research/reports.html`
4. Bylander, T.: The computational complexity of propositional STRIPS planning. Artificial Intelligence 69(1-2), 165–204 (1994)
5. Castellini, C., Giunchiglia, E., Tacchella, A.: Sat-based planning in complex domains: Concurrency, constraints and nondeterminism. Artif. Intell. 147(1-2), 85–117 (2003)
6. Erol, K., Nau, D.S., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain-independent planning: A detailed analysis. Technical Report CS-TR-2797, University of Maryland College Park (1991)
7. Erol, K., Nau, D.S., Subrahmanian, V.S.: Complexity, decidability and undecidability results for domain-independent planning. Artificial Intelligence 76(1-2), 75–88 (1995)
8. Haslum, P., Jonsson, P.: Some results on the complexity of planning with incomplete information. In: Biundo, S., Fox, M. (eds.) ECP 1999. LNCS, vol. 1809, pp. 308–318. Springer, Heidelberg (2000)
9. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
10. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Reasoning about actions using description logics with general TBoxes. In: Fisher, M., van der Hoek, W., Konev, B., Lisitsa, A. (eds.) JELIA 2006. LNCS (LNAI), vol. 4160, pp. 266–279. Springer, Heidelberg (2006)
11. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating description logic ABoxes. In: Proceedings of KR 2006, pp. 46–56 (2006)
12. Nebel, B.: Terminological reasoning is inherently intractable. Artificial Intelligence 43, 235–249 (1990)
13. Palacios, H., Geffner, H.: Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In: Proc. of AAAI 2006 (2006)
14. Reiter, R.: Knowledge in Action. MIT Press, Cambridge (2001)
15. Rintanen, J.: Complexity of planning with partial observability. In: Proceedings of (ICAPS 2004), pp. 345–354 (2004)
16. Savitch, W.J.: Relationship between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4, 177–192 (1970)
17. Thielscher, M.: Introduction to the Fluent Calculus. Electronic Transactions on Artificial Intelligence 2(3–4), 179–192 (1998)

Faster Phylogenetic Inference with MXG

David G. Mitchell, Faraz Hach, and Raheleh Mohebali

Computational Logic Laboratory
Simon Fraser University, Burnaby BC, Canada
{mitchell,fhach,rmohebal}@cs.sfu.ca

Abstract. We apply the logic-based declarative programming approach of Model Expansion (MX) to a phylogenetic inference task. We axiomatize the task in multi-sorted first-order logic with cardinality constraints. Using the model expansion solver MXG and SAT+cardinality solver MXC, we compare the performance of several MX axiomatizations on a challenging set of test instances. Our methods perform orders of magnitude faster than previously reported declarative solutions. Our best solution involves polynomial-time pre-processing, redundant axioms, and symmetry-breaking axioms. We also discuss our method of test instance generation, and the role of pre-processing in declarative programming.

Keywords: Phylogeny, Declarative Programming, Model Expansion.

1 Introduction

We apply a declarative programming approach, based on the logical task of model expansion (MX), to a problem in phylogenetic inference. In the approach, an instance is a finite structure; a solution is a finite structure; a problem specification is an axiomatization, in a suitable logic, of the relationship between instances and solutions [16].

A phylogenetic tree is a directed graph representing the evolutionary relationships among a collection of species. Phylogenetic inference (or re-construction) is the task of constructing a phylogenetic tree (or other network) from species data. It has many applications in biology and elsewhere, producing a variety of particular computational problems. Our interest in these problems is primarily as developers of declarative programming tools: In trying to make our tools more effective, it is useful to work on challenging applications, especially those where success may not be immediate, but benefits may be significant. Phylogenetic inference is interesting because of the following observations. Most interesting variants are NP-hard optimization problems, and there are many data sets too hard to solve well in practice; Many particular problems are variants of, or combinations of, a few basic problems; And, the optimality metrics often do not precisely match subjective solution quality, so users could benefit from a method to interactively add ad-hoc constraints, which is not possible with current tools.

Contributions. We describe a method that is faster, by many orders of magnitude, than the only previous declarative solution we know of. In doing so, we demonstrate that MX-based tools can be effective on more realistic domains than has previously been shown. Effectively measuring progress in solving NP-hard problems is tricky, and we

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 423–437, 2007.
© Springer-Verlag Berlin Heidelberg 2007

believe the method we use here is interesting. Our pre-processing method, in addition to benefiting our own solution, could improve the performance of existing software packages. We point out that instance pre-processing, often important in problem solving, can be done declaratively.

The Problem. The particular problem we study is the binary cladistic Camin-Sokal (CCS) problem, which we chose because it is a simple problem which is NP-hard; to which standard tools apply; for which we have suitably challenging real data; and for which there is a previous declarative programming solution to compare with. Our primary solving mechanism is the model expansion grounder/solver MXG [16], with the SAT+cardinality solver MXC [2]. Our test set consists of several hundred instances of graduated difficulty derived from biological data. We compare the performance of:

- MXG and MXC, with various axiomatizations using cardinality constraints;
- MXG and Minisat [6], a high-performance SAT solver, with non-cardinality MX axiomatizations;
- clasp [9], a high-performance answer set programming (ASP) solver, with the best-performing ASP axiomatization from [13];
- MXG and MXC, aided by polynomial-time instance pre-processing;
- PAUP, a widely-used phylogeny software package [19].

Related Work. Kavanagh et al [13] reported answer set programming (ASP) based solutions to binary CCS. Their best solution established optimal trees for instances for which the phylogeny software package PENNY [8] could not, but could not solve their largest instance (which is identical to our largest instance), or even moderate-sized subsets. ASP solutions to some other phylogeny problems, which are not directly comparable, are reported in [3,7,1]. [3] use "large compatibility", where the goal is to find the maximum number of characters, for a given set of species, for which there is a perfect phylogeny. In contrast, we use "large parsimony", where we find a (perhaps not perfect) phylogeny for the input species with the minimum number of evolutionary changes. The task in [7] is to construct a "perfect phylogenetic network", from given phylogenetic trees (the "species" there are natural languages). [1] studied the "Maximum Quartet Consistency" problem, and evaluated an ASP solution on synthetic data.

2 The Binary Camin-Sokal Phylogeny Problem

We study a simple "large parsimony" problem in character-based cladistics. A group of species is characterized by a set of *characters*. Each character can take one of several *states*, and each species is described by a vector giving a state for each character. The input is a set of species vectors and the goal is to construct a tree with nodes labeled by character vectors, so that the vector of every input species labels some node. Changes of a character's state along an edge are mutations. Problem variations result from differing cost metrics and restrictions on character changes. A tree minimizing the cost metric is a "most parsimonious tree". In the cladistic Camin-Sokal (CCS) problem, the states of each character are ordered and mutations must be increasing on this order. This is appropriate when the direction of evolutionary change of characters is assumed to be known. The goal is to minimize the total number of mutations. (Examples of biological application of CCS include [5,18]). The decision version of the problem, even in the binary case where each character has just two states, is NP-complete [4].

Definition 1. *The binary cladistic Camin-Sokal problem (binary CCS) is:*

Instance: *Set S of n distinct vectors from $\{0,1\}^m$; natural number B.*
Question: *Is there a directed tree $T = (V, E)$, such that: 1) T is rooted at 0^m; 2) $S \subseteq V \subseteq \{0,1\}^m$; 3) Every leaf of T is in S; 4) For each directed edge $(v_1, v_2) \in E$, v_1 and v_2 differ in exactly one character, which is 0 at v_1 and 1 at v_2; 5) $|V| \leq B$.*

Remark 1. An alternate definition allows multiple mutations on an edge. The definition we use here was also used in [13], and is easier to axiomatize in MXG.

In a *perfect phylogeny*, each character mutation occurs only once. For the binary CCS, this is equivalent to setting $B = m$, provided that both states of every character occur in S. (Note that if some character has only one state, we may safely delete it.) When an instance does not have a perfect phylogeny, some character mutations must occur more than once in the tree. Since mutations are irreversible in CCS, the same character cannot change more than once on a directed path from the root, so the same mutation will occur in distinct subtrees. The goal is to find a tree that minimizes the number of these “extra mutations”. We allow only one mutation per edge, so the number of extra mutations is equivalent to the number of “extra vertices” or “extra edges” needed to construct a phylogeny. Since mutation is irreversible, we may assume that all mutations are from state 0 to state 1, and the tree is rooted at the zero vector.

3 Model Expansion and MXG Basics

We give a minimal and informal description of MXG. For further details, including formal aspects of MX, the MXG language and grounding algorithm, examples and performance on other problems, see [16]. MXG is a model expansion grounder/solver. It takes as input a problem specification S and an instance I. The problem specification is essentially an axiomatization in multi-sorted first-order logic (FO) extended with inductive definitions and cardinality constraints.

Vocabulary symbols in an axiomatization have three distinct roles: Instance vocabulary consists of symbols whose interpretation is given by an instance; Solution vocabulary is symbols whose interpretations comprise a solution; Auxiliary vocabulary is symbols that are not part of the instance or solution. The solution and auxiliary vocabulary together form the *expansion vocabulary*, the symbols whose interpretations must be constructed by the solver. Problem specifications contain declarations of types, typed declarations of vocabulary symbols, and axioms. They have three parts: Given: has declarations of all types and instance vocabulary; Find: has the declaration of solution vocabulary; Satisfying: has axioms, plus declaration of auxiliary vocabulary, if any.

As an example, Figure 1 is an MXG specification for the graph colouring problem. The sorts are Vtx (vertices) and Clr (colours); the instance vocabulary is the binary

```
Given:    type  Vtx Clr;
          Edge(Vtx, Vtx)

Find:     Colour(Vtx, Clr)

Satisfying:
          ∀ x : ∃ y : Colour(x, y)
          ∀ x y : (Edge(x,y) ⊃ (∀ z : ¬(Colour(x, z) & Colour(y,z))))
          ∀ x y z : ((Colour(x, y) ∧ Colour(x, z)) ⊃ (y=z))
```

Fig. 1. An MXG problem specification for graph colouring

relation Edge; the solution vocabulary is the binary relation Colour. The axioms say that the interpretation of Colour must give a proper colouring of the given graph. The MXC instance file for the instance with colours $\{1, 2\}$, vertices $\{1, 2, 3\}$, and edges $(1, 2)$ and $(1, 3)$ contains: Vtx = [1..3] Clr = [1; 2] Edge = {1, 2; 1, 3}.

MXG combines a specification and instance, producing a propositional formula ϕ which is a "reduced grounding" of S with respect to I. That is, satisfying assignments of ϕ correspond one-to-one with solutions of I. Currently, ϕ is a CNF formula, possibly extended with cardinality constraints. A propositional solver searches for a satisfying assignment to ϕ, and if found MXG maps the assignment back to the FO language to produce a solution. Other relevant aspects of the (current) MXG language are:

- Vocabulary symbols are typed, by declarations in the specification. The domain of each variable is inferred from its use, which must be consistent.
- Each type is an ordered finite set given by the instance. The ordering is determined by the form of the instance description: Numerical if expressed as a range of integers; otherwise as enumerated. For each type, constant symbols MIN and MAX, binary relation symbols $<$, $\leq$, etc., and binary relation SUCC are all implicitly defined, with the natural semantics. Types are disjoint, so two elements are comparable only if of identical type.
- Bounded quantifiers are supported: $\forall$x y $<$ x : ϕ(x,y) is equivalent to $\forall$x $\forall$y : (y$<$x $\supset$ ϕ(x,y)); $\exists$x y $<$ x : ϕ(x,y) is equivalent to $\exists$x $\exists$y : (y$<$x $\wedge$ ϕ(x,y)).
- Simple cardinality constraints are supported, which are universal sentences of the form $\forall \boldsymbol{x} : \odot(n; \boldsymbol{y}; \phi(\boldsymbol{x}, \boldsymbol{y}))$, where $\odot$ is one of UB, LB, or CARD, for upper bound, lower bound, and equivalence, respectively. For example $\forall$u : UB(1;v;Edge(v,u)) says the in-degree of every vertex is at most 1.
- A limited form of inductive definition is supported (see [16] for details).

4 MX Axiomatizations of Binary CCS

Here we give three MX axiomatizations of Binary CCS. One we find natural and simple, using cardinality constraints; one uses no cardinality constraints, and can be solved by straightforward reduction to SAT; and one is a translation to MX of the best ASP encoding from [13]. We produced other distinct axiomatizations, but since none performed better than our basic one (except when using enhancements such as described in Section 6), we do not report them.

```
Given:  type  Char Vertex State;
        A(Vertex, Char, State)
        NSpecies: Vertex
        NEdges: Vertex

Find:   Edge(Vertex, Vertex)
        Vector(Vertex, Char)

Satisfying:
        ∀ s ≤ NSpecies c : (A(s,c,MAX) ⇔ Vector(s, c))                     (1)
        ∀ u v : UB(1; c; (Edge(u,v) ∧ ¬ Vector(u,c) ∧ Vector(v,c)))        (2)
        ∀ u v : (Edge(u,v) ⊃ (∃c : (¬Vector(u,c) ∧ Vector(v,c))))          (3)
        ∀ u v : UB(0; c; (Edge(u,v) ∧ Vector(u,c) ∧ ¬ Vector(v,c)))        (4)
        CARD(NEdges; v, u; Edge(v,u))                                      (5)
        ∀ v>MIN : CARD(1; u; Edge(u, v))                                   (6)
```

Fig. 2. Basic MX axiomatization of binary CCS phylogeny re-construction

Basic MX Axiomatization. The types are Vertex, vertices of the tree; Char, the set of characters; and State $(= \{0, 1\})$, the set of states. We identify the n species with the first n vertices. The (too simple) type system requires that variables and constant symbols which are to range over species must be of type Vertex. The instance vocabulary consists of:

- A(Vertex, Char, State): the set of triples specifying the matrix of species of data. (The first argument is the species.)
- NSpecies: a constant symbol denoting the number of species.
- NEdges: a constant symbol which is always set to $|\text{Vertex}| - 1$.

The solution vocabulary has two binary relation symbols: Edge, the set of edges, and Vector, which labels vertices with character vectors. Vector(v,c) holds if character c has state 1 in the vector labeling v. The axioms (see Figure 2) state:

- The label of vertex i, for $i \in \{1, \dots, n\}$, must be species vector i (Axiom 1);
- Each edge has exactly one character changing from 0 to 1, and no characters changing from 1 to 0 (Axioms 2–4);
- Every node, except the root, has in-degree exactly one, and the number of edges is exactly the number of vertices less one (Axioms 5,6).

Axioms 2 through 4 ensure edges have only allowed mutations, and in particular that every path is monotone increasing in the set of characters with state 1; Axioms 5 and 6 ensure the graph is a tree, which is rooted at the zero vector by a convention that species 1 is the zero vector.

Non-Cardinality Axiomatization. To determine if we obtain a speed-up over pure SAT solving by using MXC with cardinality constraints, we produced several axiomatizations without cardinality constraints, which MXG grounds to SAT. Figure 3 shows the best-performing of these. The axioms state: The input species vector i must label vertex i (Axiom 1 - as before); Each edge has exactly one character changing from 0 to 1 (Axiom 2); On a directed path the set of 1-characters is monotone increasing (Axiom 3), so there are no reverse mutations; The graph is a tree (Axioms 4 and 5), since every node but the root has in-degree one and there are no cycles in the transitive closure of

```
Given:  type  Char Vertex State;
        A(Vertex, Char, State)
        NSpecies: Vertex
        NEdges: Vertex

Find:   Edge(Vertex, Vertex)
        Vector(Vertex, Char)

Satisfying:
        TC(Vertex, Vertex)   // TC will be the transitive closure of Edge.
        ∀ s ≤ NSpecies  c :  (A(s,c,MAX) ⇔ Vector(s,c))                              (1)
        ∀ v1 v2: (Edge(v1,v2)⊃(∃ c1 : (¬Vector(v1,c1)∧Vector(v2, c1)                 (2)
                      ∧ (∀c2 : ((¬Vector(v1,c2) ∧ Vector(v2, c2)) ⊃ (c1=c2))))))
        ∀ u v c : ((TC(u, v) ∧ Vector(u, c)) ⊃ Vector(v,c))                          (3)
        ∀ u>MIN : ∃ v : (Edge(v,u) ∧ (∀v2 : (Edge(v2, u) ⊃ (v2 = v))))               (4)
        ∀ u v>u : ¬(TC(u, v) ∧ TC(v, u))                                              (5)
        ∀ u v : (TC(u,v) ⇔ ((u = v) ∨ Edge(u,v) ∨ (∃x : (TC(u, x) ∧ Edge(x,v)))))    (6)
```

Fig. 3. MXG axiomatization with no cardinality constraints

```
Given:  type Chars Vertex State Species;
        A(Species, Char, State)

Find :  Edge(Vertex, Vertex)

Satisfying :
        M(Vertex, Char)
        P(Species, Vertex)
        TC(Vertex, Vertex)
        ∀ v : CARD(1; c; M(v,c))                                                     (1)
        ∀ s : CARD(1; v; P(s,v))                                                     (2)
        ∀ v : UB(1; u; Edge(u, v))                                                   (3)
        ∀ u v : (TC(u,v) ⇔ ((u = v) ∨ Edge(u,v) ∨ (∃x : (TC(u, x) ∧ Edge(x,v)))))   (4)
        ∀ u v>u : ¬(TC(u,v) ∧ TC(v,u))                                               (5)
        ∀ s c: ( A(s,c,MAX) ⇔ (∃ u v : (TC(u, v) ∧ M(u,c) ∧ P(s, v))))               (6)
        ∀ s v c : ¬(P(s,v) ∧ M(v,c) ∧ A(s,c, MIN))                                   (7)
        ∀ v v1>v c : ¬(TC(v, v1) ∧ M(v1, c) ∧ M(v, c))                               (8)
```

Fig. 4. MXG axiomatization based on ASP encoding "A+" from [13]

Edge; TC is the transitive closure of Edge (Axiom 6 provides the lower bound on TC; Axiom 5 the upper bound). We report results based on the SAT solver minisat, arguably the best all-around SAT solver available.

Translation of ASP to MX. We also used an MX axiomatization based on the best ASP encoding from [13] (denoted "A+" there). It differs from the previous two in that: 1) We have a type Species, distinct from Vertex. 2) Rather than identify the n species with the first n vertices, we construct a function P (represented as a binary relation) from species to vertices. 3) We construct a function M from vertices to characters. The mutation on edge (u,v) is the character c such that M(v,c) holds. In contrast, in our previous axiomatizations the mutation on edge (u,v) is implicit in the difference between the vectors labeling u and v. 4) The (root) vector **0** is left implicit, so a solution is a forest. A tree is obtained by adding an edge from **0** to the root of each forest component.

Figure 4 gives the axioms, which state: Each vertex is mapped to exactly one character (Axiom 1); Each species is mapped to a vertex (Axiom 2); The graph is a tree (Axioms 3–5); The characters which are 1 at species S must have mutated at some ancestor of the node S is mapped to (Axiom 6); If species S is mapped to vertex v, the character which mutated at v must not be 0 at S (Axiom 7); A character mutates at most once on any (directed) path (Axiom 8). Axioms 7 and 8 are redundant, but improve performance.

5 Evaluating Progress in Performance

Evaluating performance of solvers for NP-hard problems has many pitfalls, especially when there is no base-line provided by well-established benchmarks and solvers. Our goal is to have a clear measure of *progress* in performance. Direct comparison of run-times does not work here, because run-times for the methods we test vary by many orders of magnitude (see Figure 8). A better measure is the number of instances that can be solved within reasonable time. For this, a collection of related instances of graduated difficulty is needed, but in practice this is often hard to arrange. For example, [13] obtained three real data sets: Two were too easy and the third was too hard. Randomly generated instances are easily graduated, but their use requires care (see, e.g., [17]), and may be irrelevant to practice.

Instances. Here, we produce a set of suitably graduated instances from the one challenging real data set we have for our problem. This is possible because, if we view a set of n species vectors of length m as an $n \times m$ matrix M, any sub-matrix of M is a valid set of data, as real (or not) as the full matrix. For illustration: $\{$eye-colour,hair-colour$\}$ is as valid a set of characters as $\{$eye-colour,hair-colour,handedness$\}$. (Not every such matrix is scientifically interesting, of course.) Our initial instance is a 36×63 matrix obtained from the experimentally obtained haplotype data of [12] (for Poecilia reticulata – guppies) as described in [13]. Following [13], we produced a set of instances from upper-left sub-matrices of size $k \times l$, for k, l multiples of 3. Thus, we view performance as a function of two natural instance parameters: number of species and number of characters. Unfortunately, the resulting instances were not nicely graduated, as most moderate-size instances were very easy for our methods. The problem was that most sub-matrices had all-zero columns and duplicate rows, which we solved as follows. Keeping the zero vector as species 1, we put all other species in reverse lexicographic order. Thus, the first row was all zero, but the second row had many ones. The set of instances produced from this initial matrix by the scheme described above satisfied our main criteria: the instances are smoothly graded in difficulty for our solvers, and they do not contain significant numbers of trivial or duplicate rows or columns.

Performance Measure. As an objective measure of progress, we require the solver to establish the optimal phylogeny size within a fixed time bound. As with any optimization problem, one may trade solution quality for solving time. In phylogenetic inference, user's often don't care about optimality *per se*, because the cost metrics do not exactly correspond their subjective notion of quality. But if optimality is not a precise measure of quality, surely being within some distance of optimal is not either, so relaxing the optimality requirement does not improve our measure. The requirement to solve to optimality would seem to better measure whether we are making progress

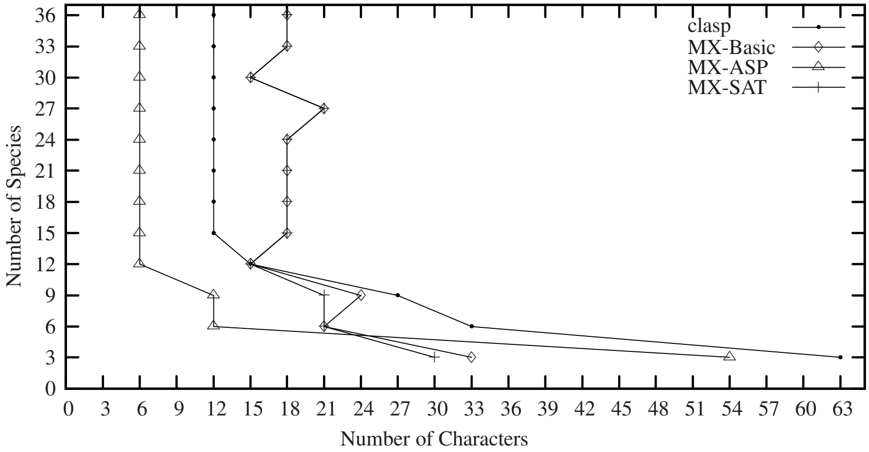


Fig. 5. Frontier comparison of three MX axiomatizations and an ASP solution

in dealing with whatever it is that makes up the combinatorially hard aspect of our instances. For measuring progress toward being able to practically solve larger and harder instances than currently possible, we believe that establishing optimal solutions within a reasonable time cut-off is as good a measure as any we know of.

Evaluation. MXG does not have a built-in optimization facility, so for each optimization instance we solve a sequence of search instances. The first asks for a perfect phylogeny (with the same number of mutations as characters). Successive instances allow one more mutation. We run the solver on the sequence, stopping when a solution is found – which must be optimal – or when the cumulative run time reaches two hours. Sequential search is faster than binary search because instances with too few mutations are typically easier than those with too many. Time for sequential search is dominated, almost without exception, by the two instances needed to establish optimality: the one producing an optimum solution and the one with one fewer mutations. Binary search is often dominated by the instances just beyond optimal, which sequential search never visits. This pattern of hardness also supports our argument in favour of using optimality in our measure of performance.

Tests were run on Sun Fire VZ20 Dual Opteron computers with 2.4 GHz AMD Opteron 250 processors, with 1MB cache and 2GB of RAM per processor, running Suse Enterprise Linux 2.6.11. The software versions were: MXG 0.17; MXC 0.5; minisat_v2s; clasp rc3; and paup4b10-opt-linux-a. Executables for clasp and PAUP were downloaded from the solver web sites, while MXG, MXC, and minisat_v2s were compiled with gcc version 3.3.4.[1]

Figure 5 shows the “frontier” for MXG with the axiomatizations of Section 4, and for the ASP solver clasp using the A+ axiomatization of [13]. We plot a curve for each solving method. A point at (x, y) denotes that x is the largest number of characters for which the method succeeded in solving instances with y species within the two-hour cut-off. Instances left of or below a curve were solved; those above and to the right were not. The basic MX axiomatization is best, except with very few species. MXG performs relatively poorly with few species because it must construct the whole vector for each vertex, and thus with many characters has quite a bit of work to do, while the ASP axiomatizations do not. We ran two ASP solvers, cmodels [14] and clasp [9], on the A+ axiomatization. Since the performance was similar, with clasp being slightly better, we show only the clasp curve. The no-cardinality axiomatization and minisat performed essentially the same as our basic MX axioms, except for being slightly weaker with few species. Our translation of the ASP axioms to MX performed poorly.

---

[1] MXG and MXC are at `www.cs.sfu.ca/research/groups/mxp/`; minisat at `www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`, clasp at `www.cs.uni-potsdam.de/clasp/` and PAUP at `http://paup.csit.fsu.edu/`

We conclude that our basic MX axiomatization, which is already substantially better than the best solution in [13], is a good starting point for further work.

6 Enhancing Performance

In this section, we report on two ways we refined our basic axiomatization that dramatically improved performance. Adding redundant axioms is a standard method in SAT and CSP encodings, and [13] reported significant speedups with this method. It is not well understood why particular redundant axioms help (or hurt) performance. Natural explanations are that they increase the amount of unit propagation performed for some partial assignments, or that they help a clause-learning solver learn more useful clauses. Symmetry-breaking axioms eliminate some - but not all - solutions among a set of symmetric solutions. They often improve performance, even when only one solution is needed, presumably because they help the solver effectively eliminate many symmetric “near-solutions”.

Axioms 13 of Figure 6 states that no extra vertex is a leaf. It is neither symmetry-breaking nor redundant, but has a similar flavour in that it removes only solutions that are not very interesting, and improves performance.

Computing Vertex Depth. In a binary CCS tree, each vector labeling a vertex at depth k has exactly k 1’s. Thus, if K is the maximum number of 1’s in any species vector, the tree has height at most K. We can add axioms requiring labels of vertices to respect this property. These are axioms seven through ten and thirteen of Figure 6, which state:

- Each vertex must be assigned a unique depth (Axiom 8), which must be one greater than that of its parent (Axiom 9).
- The depth of each vertex is the number of 1’s in its label (Axiom 10).
- Only the root has depth 0 (Axiom 11);

These axioms are redundant, but significantly improve performance. They use two auxiliary relation symbols, SpcDepth and VtxDepth. VtxDepth(v,d) holds if vertex v is at depth d. SpcDepth(s, c, d) holds if the number of 1’s among the first c characters for species s is d. Axiom 7 is an inductive definition which plays a special role. The form of this definition is such that MXG can compute the relation SpcDepth *before* grounding Axioms 8, 9 and 10. Thus, it is as if a pre-processor computed this relation

```
Given:  type  Char Vertex State Depth;
        A(Vertex, Char, State)
        NSpecies: Vertex
        NEdges: Vertex

Find:   Edge(Vertex, Vertex)
        Vector(Vertex, Char)

Satisfying:
        // Axioms 1-6 are the Basic MX Axioms of Figure 2

        VtxDepth(Nodes, Depth)
        SpcDepth(Nodes, Chars, Depth)
        { SpcDepth(u,c,d)  ← c=MIN ∧ d=MIN ∧ s = MIN ∧ A(u,c,s)                          (7)
          SpcDepth(u,c,d)  ← c=MIN ∧ SUCC(MIN, d) ∧ s = MAX ∧ A(u,c,s)
          SpcDepth(u,c,d)  ← SpcDepth(u,c1, d) ∧ SUCC(c1, c) ∧ A(u,c,s) ∧ s=MIN
          SpcDepth(u,c,d)  ← SpcDepth(u,c1, d1) ∧ SUCC(c1, c) ∧ SUCC(d1, d)
                               ∧ A(u,c,s) ∧ s=MAX
        }
        ∀ u : CARD(1; d; VtxDepth(u, d))                                                  (8)
        ∀ u v d1 d2 : ((Edge(u,v) ∧ VtxDepth(u, d1) ∧ VtxDepth(v,d2)) ⊃ SUCC(d1, d2))     (9)
        ∀ u≤NSpecies d: (VtxDepth(u,d) ⇔ SpcDepth(u,MAX,d))                              (10)
        ∀ u>MIN : ¬ VtxDepth(u,MIN)                                                       (11)
        ∀ u>NSpecies v>u d1 d2 : ((VtxDepth(u,d1) ∧ VtxDepth(v,d2)) ⊃ d2 ≥ d1)            (12)
        ∀ u>NSpecies : ∃ v : Edge(u,v)                                                    (13)
```

Fig. 6. MX-Depth (Axioms 1-10) and MX-Depth+ (Axioms 1-13) axiomatizations

and added it to the instance. Since SpcDepth(s, MAX, d) says that species s is at depth d, the grounder has computed the depth for each species. (For simplicity of axiomatization, we also added a new type Depth, which is a set the size of the maximum number of ones in species vector. We added this to our instances in a simple pre-processing step, although this could be avoided with a more complex axiomatization.)

Symmetry Breaking. Our final example is a symmetry-breaking axiom. It states that the depth of "extra vertices" (those which allow extra mutations), respects their numerical order (Axiom 12).

Instance Pre-processing. We found that instances (including the largest) often satisfied easily-checked properties that could be used to simplify them with a pre-processing step, which greatly improved performance. We recursively applied the following rules: 1. Delete any all-zero column: The character does not mutate, so we need no node for it. 2. Delete any column having exactly one 1: If c is 1 only in s, we construct a tree without c, then add $c = 0$ to every vector on the tree, adding one new edge and vertex where s appears. 3. Delete any column having exactly one 0: The 0 occurs in the root zero vector. We construct a solution without c, and insert one new node beneath the root in the solution, setting $c = 1$ everywhere except the root. 4. Delete any duplicate species. 5. Delete s_2 for any pair s_1, s_2 of species such that: (a) $s_1 \subset s_2$, i.e., every character that is 1 for s_1 is also 1 for s_2; (b) $|s2 - s1| = k$, i.e., s_2 has k more 1's than s_1; (c) $\forall s_3 \notin \{s_1, s_2\}, |s_2 \backslash s_3| \geq k$. Solve the instance without s_2, then add it to a path of length k below s_1.

Performance with Refined Axioms and Pre-processing. Figure 7 is a frontier plot showing performance improvements obtained with enhanced axiomatizations and

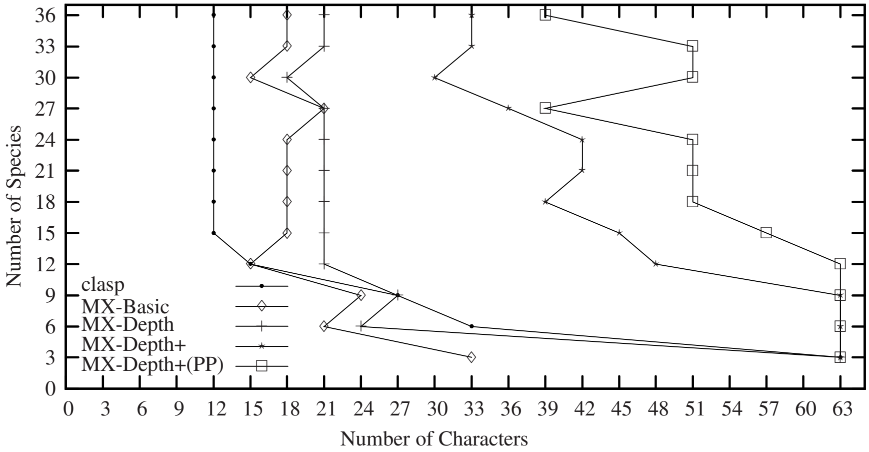


Fig. 7. Frontier plot showing performance improvement with refined axiomatizations (MX-Depth and MX-Depth+) and instance pre-processing (MX-Depth+(PP))

pre-processing. For comparison, we included the curves for clasp and MX-Basic, and in addition three new curves:

- MX-Depth: MX-Basic axioms extended with Axioms 7–10 of Figure 6;
- MX-Depth+: MX-Depth further extended with Axioms 11–13 of Figure 6;
- MX-Depth+(PP): Pre-processing of instances, and solving with the MX-Depth+ axiomatization (all axioms of Figure 6).

Remark 2. The “dip” in performance of MX-Depth+(PP) at 27 species is the consequence of pre-processing being less effective on these.

How Much Better: Frontier *vs.* **Run-time.** The frontier plots show that we have progressed in terms of our chosen measure, but do not show the (dramatic) corresponding changes in run-time. Figure 8 illustrates, showing run-times as a function of number of characters, with number of species fixed at 24. Analogous curves for fewer or more species are similar, except for very small numbers of species. The y (time) axis is log scale, so these run-times appear to be exponential in the number of characters. Notice that the curves have very different slopes, suggesting that the run-time curves for MX-Depth+ and MX-Depth+(PP) have much smaller exponents than the other solutions. We cannot really extrapolate the curves for the ASP or Basic MX solutions to compare run-times for large instances with the best methods, but unless the curves here are completely mis-leading the difference is certainly many orders of magnitude. Solving the hardest instances with those methods is completely infeasible in practice.

7 MXG *vs.* PAUP

The two most widely used phylogeny software packages, PHYLIP [8] and PAUP [19], both use two methods to carry out phylogenetic inference (for CCS and other models).

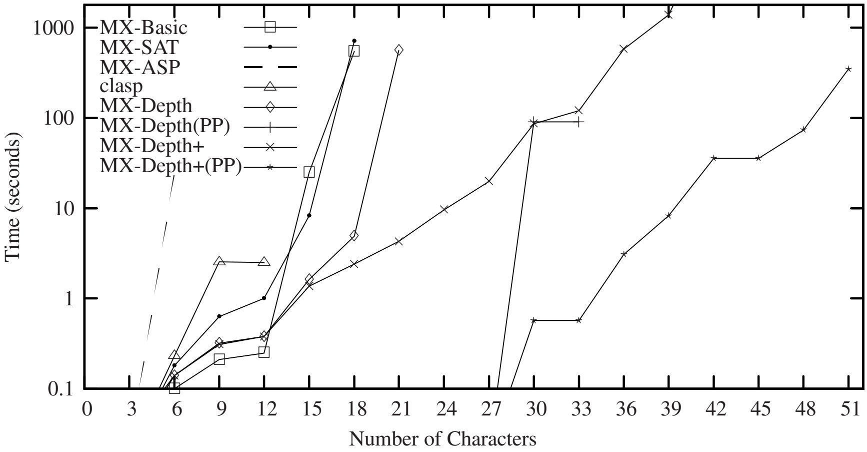


Fig. 8. Run-times for instances with 24 species, as a function of number of characters

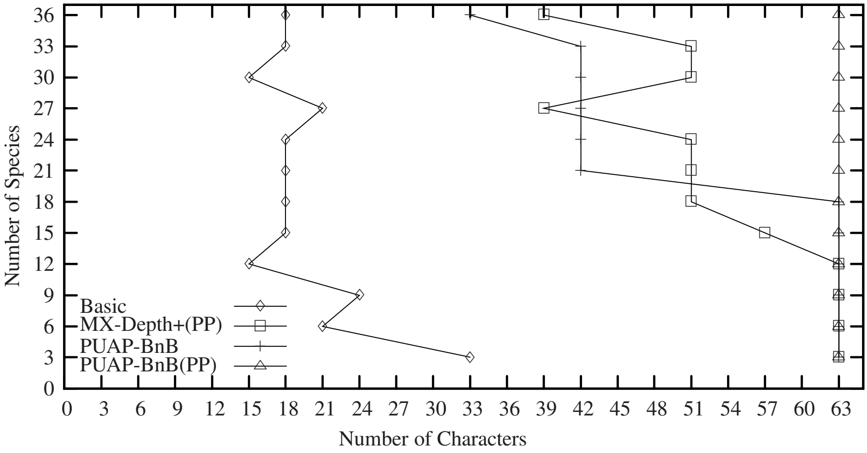


Fig. 9. Frontier for MX-Basic, PAUP-BnB, PAUP-BnB(PP), and MX-Depth+(PP)

One method is based on heuristic search, which cannot guarantee optimality, and one is based on branch-and-bound, which can. The branch-and-bound program for CCS in the PHYLIP package is called PENNY (after the second author of [11], where branch and bound was proposed for this task). In [13], the performance of PENNY was compared with the ASP-based solutions developed there. PENNY was unable to prove optimality of solutions for any instances with more than 18 species.

We compare the performance of our method against the branch and bound implementation in PAUP. (We might expect PAUP (which is not free) to be faster than PHYLIP (which is free), because it has had more development effort, and this seems to be the case.) Figure 9 shows the frontier plots for the PAUP branch and bound implementation (PAUP-BnB), along with that for MX-Depth+(PP), and MX-Basic for comparison. Our MX-based solution is similar overall to PAUP, but comes closer to solving our largest –

and presumably hardest – instances. For completeness, we also ran PAUP branch-and-bound on the instances produced by our pre-processing algorithm. Interestingly, PAUP performance improved, and with our pre-processing it solved all instances.

8 Discussion

We have developed MX-based solutions to a phylogenetic inference problem. A simple and natural axiomatization gave much better performance than the only other declarative solution to this problem we know of, and more refined efforts produced a solution scheme with dramatically better performance. Ultimately, the kinds of methods and tools we use must be validated by demonstrating good performance on a wide range of problems and instances. Here we have tackled one interesting and non-trivial problem, and we believe what we have learned here will usefully inform more general solutions to a variety of phylogeny problems. Our instances here are derived from a single source of data, but we have taken pains to ensure that our instances and performance measures provide a good measure of performance progress. The improvements in running time, which are of many orders of magnitude, strongly suggest that our methods will be significant improvements by any other reasonable measure of performance.

MX *vs.* **ASP.** We remind the reader that, while our Model Expansion based solutions, using the grounder MXG with ground solver MXC, perform dramatically better than the ASP solution we evaluate, our results here do not justify a claim of superiority of MX methodology or solvers over those of ASP. The best ASP solvers are quite powerful, and alternate approaches to ASP axioms, combined with pre-processing of the sort we do, might yield effective ASP-based solutions. (A comparison of MXG with several ASP solvers appears in [16].)

PAUP Heuristics. An easy criticism of the work presented here is that the heuristic methods implemented in PAUP and PHYLIP often perform very well, and we have not compared our methods with those. In fact, the heuristic search method of PAUP finds optimum solutions for all of our instances well within our time limit. PAUP, of course, does not know if they are optimal or not, and neither would a PAUP user. But, if optimality *per se* is not of much value to a biologist, why would they care?

One reply is that declarative solutions are potentially very useful. For example, user's don't worry about optimality because they are interested in criteria that are not captured by the cost function. With standard tools they are limited in how they can address these other preferences. Good declarative tools would allow them to add specific constraints, say that certain species should not be in the same sub-tree, and find solutions satisfying these (see also [7]). Another reason good declarative techniques could pay off is that problems of interest are often variants of a few core problems, and in some cases these are much more complex than the simple problem we studied here. An example is the Galled Tree Network Haplotyping Problem [10]. An instance is genotype data, which consists of vectors of conflated pairs of haplotypes. The task is to infer a set of haplotype vectors from the genotype data for which a parsimonious galled tree network exists. A galled tree network is a significantly more complex phylogeny than our binary

CCS trees. The task of inferring small sets of haplotypes from genotype data, without worrying about phylogenies, is itself NP-hard (although SAT solvers do well at this [15], so MXG should also). Implementing a special-purpose program for this problem would be some effort, and finding simple heuristics which work reliably on large instances of such a problems seems unlikely. However, if we had effective declarative solutions for haplotype inference and construction of galled-tree networks, it would be easy to combine them and have a good start toward a solution for the larger problem.

Declarative Pre-Processing. A point that may be argued against the progress we claim is that pre-processing of the instances before passing to the declarative solver was important, but this step is not declarative. Indeed, pre-processing is important in tackling many problems, seemingly a stumbling block for declarative methods. We point out the technique we used in our MX-Depth axiomatization (Figure 6), where we wrote an inductive definition to compute a set, and then used certain elements of that set in other axioms. MXG computes the defined set directly, while grounding, so the ground solver does not see this part of the axiomatization. Essentially any poly-time preprocessing can be carried out using this technique (not necessarily by the current version of MXG). With suitably refined languages, some users could accomplish such pre-processing more conveniently with declarative descriptions than with procedural code.

Conclusion. We have not, yet, changed the way phylogenetic inference will be done in practice. But we have made progress that justifies our optimism regarding declarative approaches in general, and our MX-based tools in particular.

Acknowledgments. We thank Jonathan Kavanagh for his instances, ASP programs, and bibtex file, and Eugenia Ternovska, Jan Manuch, and Sharon (Xiao-hong) Zhong for helpful discussions on axiomatizations and instances.

References

1. Adn, G.W., You, J.-H., Lin, G.: Quartet-based phylogeny reconstruction with answer set programming. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4(1), 139–152 (2007)
2. Bregman, D.R., Mitchell, D.G.: The sat solver mxc, version 0.5, Solver Description for the 2007 SAT Solver Competition (2007)
3. Brooks, D.R., Erdem, E., Minett, J.W., Rings, D.: Character-based cladistics and answer set programming. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2005. LNCS, vol. 3350, pp. 37–51. Springer, Heidelberg (2005)
4. Day, W.H.E., Johnson, D.S., Sankoff, D.: The computational complexity of inferring rooted phylogenies by parsimony. Mathematical Biosciences 81, 33–42 (1986)
5. Edwards-Ingram, L.C., Gent, M.E., Hoyle, D.C, Hayes, A., Stateva, L.I., Oliver, S.G.: Comparative genomic hybridization provides new insights into the molecular taxonomy of the saccharomyces sensu stricto complex. Genome Research 14, 1043–1051 (2004)
6. Een, N., Sorensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)

7. Erdem, E., Lifschitz, V., Nakhleh, L., Ringe, D.: Reconstructing the evolutionary history of indo-european languages using answer set programming. In: Proc. Practical Aspects of Declarative Languages: 5th Int'l. Symposium, pp. 160–176 (January 2003)
8. Felsenstein, J.: Phylip home page phylip (1980), `http://evolution.genetics.washington.edu/`
9. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: clasp: A conflict-driven answer set solver. In: Baral, C., Brewska, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 260–265. Springer, Heidelberg (2007)
10. Gupta, A., Manuch, J., Zhao, X., Stacho, L.: Characterization of the existence of galled-tree networks. J. Bioinformatics and Computational Biology 4(6), 1309–1328 (2006)
11. Hendy, M.D., Penny, D.: Branch and bound algorithms to determine minimal evolutionary trees. Mathematical Biosciences 59(2), 277–290 (1982)
12. Hoffmann, M., Tripathi, N., Henz, S.R., Lindholm, A.K., Weigel, D., Breden, F., Dreyer, C.: Opsin gene duplication and diversification in the guppy, a model for sexual selection. Proc. of the Royal Society of London Series B 274, 33–42 (2007)
13. Kavanagh, J., Mitchell, D.G., Ternovska, E., Manuch, J., Zhao, X., Gupta, A.: Constructing Camin-Sokal phylogenies via answer set programming. In: Hermann, M., Voronkov, A. (eds.) LPAR 2006. LNCS (LNAI), vol. 4246, pp. 452–466. Springer, Heidelberg (2006)
14. Lierler, Y., Maratea, M.: Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In: Lifschitz, V., Niemelä, I. (eds.) LNMR 2004. LNCS (LNAI), vol. 2923, pp. 346–350. Springer, Heidelberg (2004)
15. Lynce, I., Silva, J.P.M.: Efficient haplotype inference with boolean satisfiability. In: Proc. AAAI 2006 (2006)
16. Mitchell, D., Ternovska, E., Hach, F., Mohebali, R.: A framework for modelling and solving search problems. Technical Report TR 2006-24, School of Computing Science, Simon Fraser University (December 2006)
17. Mitchell, D.G., Levesque, H.J.: Some pitfalls for experimenters with random SAT. Artificial Intelligence 81(1,2) (March 1996) Special Issue – Frontiers in Problem Solving: Phase Transitions and Complexity.
18. Nozaki, H., Ohta, N., Matsuzaki, M., Misumi, O., Kuroiwa, T.: Phylogeny of plastids based on cladistic analysis of gene loss inferred from complete plastid genome sequences. J. Molecular Evolution 57, 377–382 (2003)
19. Swofford, D.L.: Paup* 4.0, Phylogenetic Analysis Using Parsimony (*and Other Methods) (2001)

Enriched μ–Calculus Pushdown Module Checking*

Alessandro Ferrante[1], Aniello Murano[2], and Mimmo Parente[1]

[1] Università di Salerno, Via Ponte don Melillo, 84084 - Fisciano (SA), Italy
[2] Università di Napoli "Federico II", Via Cintia, 80126 - Napoli, Italy

Abstract. The model checking problem for open systems (called *module checking*) has been intensively studied in the literature, both for finite–state and infinite–state systems. In this paper, we focus on pushdown module checking with respect to μ–calculus enriched with graded and nominals (*hybrid graded μ-calulus*). We show that this problem is decidable and solvable in double–exponential time in the size of the formula and in exponential time in the size of the system. This result is obtained by exploiting a classical automata–theoretic approach via *pushdown nondeterministic parity tree automata*. In particular, we reduce in exponential time our problem to the emptiness problem for these automata, which is known to be decidable in Exptime. As a key step of our algorithm, we show an exponential improvement of the construction of a nondeterministic parity tree automaton accepting all models of a formula of the considered logic. This result, not only allows our algorithm to match the known lower bound, but it is also interesting by itself, since it allows investigating decision problems related to enriched μ-calculus formulas in a greatly simplified manner. We conclude the paper with a discussion on the model checking w.r.t. μ-calculus formulas enriched with backward modalities as well.

1 Introduction

In system design, one of the most challenging problems is to check for system correctness. [illegible] is a formal method that allows us to automatically verify, in a suitable way, the ongoing behaviors of [illegible] ([CE81, QS81]). In this verification technique (for a survey, see [CGP99]), the behavior of a system, formally described by a mathematical model, is checked against a behavioral constraint, possibly specified by a formula in an appropriate temporal logic.

In system modeling, we distinguish between [illegible] and [illegible] systems [HP85]. While the behavior of a closed system is completely determined by the state of the system, the behavior of an open system depends on the ongoing interaction with its environment [Hoa85]. Model checking algorithms used for the verification of closed systems are not appropriate for open systems. In the latter case, we

---

* Work partially supported by MIUR FIRB Project no. RBAU1P5SS and the grant *Metodi per la Verifica di Sistemi Software e Real-Time* from University of Salerno.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 438–453, 2007.
© Springer-Verlag Berlin Heidelberg 2007

should check the system with respect to arbitrary environments and take into account uncertainty regarding the environment. In [KVW01], model checking has been extended from closed finite–state systems to open finite–state systems. In such a framework, the open finite–state system is described by a labeled state–transition graph called *module* whose set of states is partitioned into *system states* (where the system makes a transition) and *environment states* (where the environment makes a transition). Given a module $\mathcal{M}$, describing the system to be verified, and a temporal logic formula φ, specifying the desired behavior of the system, the problem of model checking a module, called *module checking*, asks whether for all possible environments, $\mathcal{M}$ satisfies φ. Module checking thus involves not only checking that the full computation tree $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$ obtained by unwinding $\mathcal{M}$ (which corresponds to the interaction of $\mathcal{M}$ with a maximal environment) satisfies φ (which corresponds to model checking $\mathcal{M}$ with respect to φ), but also that all trees obtained from $\langle T_{\mathcal{M}}, V_{\mathcal{M}} \rangle$, by pruning subtrees of environment nodes (these trees correspond to all possible choices of the environment and are collected in $exec(\mathcal{M})$) satisfy φ. To see an example, consider a two-drink dispenser machine that serves, upon request, tea or coffee. The machine is an open system and an environment for the system is an infinite line of thirsty people. Since each person in the line can prefer both tea and coffee, or only tea, or only coffee, each person suggests a different disabling of the external nondeterministic choices. Accordingly, there are many different possible environments to consider. In [KVW01], it has been shown that while for linear–time logics model and module checking coincide, module checking for specifications given in *CTL* and *CTL** is exponentially harder than model checking. Indeed, *CTL* and *CTL** module checking is Exptime–complete and 2Exptime–complete in the size of the formula, respectively, and both Ptime–complete in the size of the system.

Recently, finite-state module checking has been also investigated with respect to formulas of the *fully graded μ-calculus* [FM07], a powerful decidable fragment of the *fully enriched μ-calculus* [BP04, BLMV06]. The *μ-calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [Koz83]. Fully enriched μ–calculus is the extension of the μ–calculus with *inverse programs*, *graded modalities*, and *nominals*. Intuitively, inverse programs allow us to travel backwards along accessibility relations [Var98], nominals are propositional variables interpreted as singleton sets [SV01], and graded modalities enable statements about the number of successors of a state [KSV02]. By dropping at least one of the additional constructs, we get a *fragment* of the fully enriched μ-calculus. In particular, by inhibiting backward modalities we get the fragment we call hybrid graded μ-calculus. In [BP04], it has been shown that satisfiability is undecidable in the *fully enriched μ-calculus*. On the other hand, it has been shown in [SV01, BLMV06] that satisfiability for any of its fragments is decidable and Exptime-complete. The upper bound result is based on an automata–theoretic approach via *two-way graded alternating parity tree automata* (*2GAPT*). Intuitively, these automata generalize alternating automata on infinite trees as inverse programs and graded modalities enrich the standard μ–calculus: 2*GAPT* can move up to a node's predecessor and move down to [illegible] n or [illegible] n successors. Using these automata, along with the fact that each fragment of the fully enriched μ-calculus enjoys the [illegible][1], it has been shown in [SV01, BLMV06] that given a formula φ of a fragment logic, it is possible to construct a 2[illegible] accepting all trees encodings quasi-forests[2] modeling φ. Then, the exponential-upper bound follows from the fact that 2[illegible] can be exponentially translated in [illegible] ([illegible]), and the emptiness problem for [illegible] is solvable in PTIME [KPV02].

Coming back to the finite-state module checking problem for the hybrid graded μ–calculus, this problem has been shown in [FM07] to be EXPTIME–complete. To see an example of its application, consider the previous two-drink dispenser machine such that whenever a customer can choose a drink he can also call a customer service, among $k > 1$ different services. Suppose also that by taking a customer service choice the drink-dispenser machine stops dispensing drinks unless the customer service resets the machine. Suppose now we want to check the property that whenever the customer comes at a choice he can always choose among k different services. This property can be described using a formula of the hybrid graded μ–calculus, whose truth depends on the possibility of jumping to nodes, each labeling the start interaction with a particular service (using nominals), and having exactly k identical of such nodes (using graded modalities). Clearly, such an open system does not satisfy this formula. Indeed, it is not satisfied by the particular behavior that chooses always the same service.

In [BMP05], the module checking technique has been also extended to infinite-state systems by considering [illegible] ([illegible], for short). These are pushdown systems augmented with finite information that allows us to partition the set of configurations (in accordance with the control state and the symbol on the top of the stack) into [illegible] and [illegible]. To see an example of an open pushdown system, consider an extension of the above mentioned two-drink dispenser machine, with the additional constraint that a coffee can be served only if the number of coffees served up to that time is smaller than that of teas served. Such a machine can be clearly modeled as an open pushdown system (the stack is used to guarantee the inequality between served coffees and teas). In [BMP05], it has been shown that pushdown module checking is 2EXPTIME–complete for [illegible] and 3EXPTIME–complete for [illegible]*.

In this paper, we extend the pushdown module checking problem to the hybrid graded μ-calculus and, by exploiting an automata-theoretic approach via pushdown tree automata, we show that this problem is decidable and solvable in 2EXPTIME. In particular, we reduce the addressed problem to the emptiness problem for pushdown tree automata. The algorithm we propose works as follows. Given an [illegible] $\mathcal{S}$, a module $\mathcal{M}$ induced by the configurations of $\mathcal{S}$, and an hybrid graded μ-calculus formula φ, we first construct in polynomial time a pushdown Büchi tree automaton ([illegible]) $\mathcal{A}_{\mathcal{M}}$, accepting $exec(\mathcal{M})$. The construction of $\mathcal{A}_{\mathcal{M}}$ we propose here extends that used in [BMP05] by also taking into account that $\mathcal{M}$ must be unwound in a quasi-forest, rather than a tree, with both nodes and edges labeled. Thus, the set *exec*$(\mathcal{M})$ is a set of quasi-forests, and the automaton $\mathcal{A}_{\mathcal{M}}$ we construct will accept all trees encodings of all quasi-forests of *exec*$(\mathcal{M})$. From the formula side, accordingly to [BLMV06], we can construct in a polynomial time a $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$, with the intent of checking that no models of $\neg\varphi$ are in *exec*$(\mathcal{M})$. Thus, we check that $\mathcal{M}$ models φ for every possible choice of the environment by checking whether $\mathcal{L}(\mathcal{A}_{\mathcal{M}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. To the best of our knowledge, the latter problem can only be solved in triple-exponential time. For example, by using a double-exponential translation of into nondeterministic parity tree automata ( ) [BLMV06, KSV02] and the fact that the emptiness problem for the intersection of a and an is solvable in Exptime [KPV02]. Here, by showing a non-trivial exponential reduction of 2 into , we show a 2Exptime upper bound for the addressed problem. Since the pushdown module checking problem for is 2Exptime-hard, we get that the addressed problem is then 2Exptime-complete. The exponential improvement on translating 2 into does not only allow us to match the known lower bound, but it also turns out to be useful in several automata-theoretic approaches to system verification. In particular, it also allows us to get results concerning decision problems for the hybrid μ–calculus (such as the satisfiability and module checking problems [BLMV06, FM07]) in a simplified way.

---

[1] A quasi forest is a forest where nodes can have roots as successors.

[2] Encoding is done by using a new root node that connects all roots of the quasi-forest and new atomic propositions which are used to encode programs and jumps to roots.

The rest of the paper is organized as follows. In the next section, we give preliminaries on labeled forests, hybrid graded μ-calculus, open Kripke structures, and open pushdown systems. In section 3, we recall 2 and . In section 4, we show an exponential translation of 2 into . In section 5, we give the module checking algorithm and conclude in section 6 with a discussion on model checking w.r.t. fragments of fully enriched μ-calculus also including the backward modality.

2 Preliminaries

Labeled Forests. For a finite set X, we denote the of X by $|X|$, the set of words over X by X^*, the empty word by ε, and with X^+ we denote $X^* \setminus \{\varepsilon\}$. Given a word w in X^* and a symbol a of X, we use $w \cdot a$ to denote the word wa. Let $\mathbb{N}$ be the set of positive integers. For $n \in \mathbb{N}$, let $\mathbb{N}$ denote the set $\{1, 2, \ldots, n\}$. A is a set $F \subseteq \mathbb{N}^+$ such that if $x \cdot c \in F$, where $x \in \mathbb{N}^+$ and $c \in \mathbb{N}$, then also $x \in F$. The elements of F are called , and words consisting of a single natural number are of F. For each root $r \in F$, the set $T = \{r \cdot x \mid x \in \mathbb{N}^* \text{ and } r \cdot x \in F\}$ is a of F (the tree r). For $x \in F$, the nodes $x \cdot c \in F$ where $c \in \mathbb{N}$ are the of x, denoted $sc(x)$, and x is their . The number of successors of a node x is called the of x ($deg(x)$). The degree h of a forest F is the maximum of the degrees of all nodes in F and the number of roots. A forest with degree h is an h forest. A full h-ary forest is a forest having h roots and all nodes with degree h.

Let $F \subseteq \mathbb{N}^+$ be a forest, x a node in F, and $n \in \mathbb{N}$. As a convention, we take $x \cdot \varepsilon = \varepsilon \cdot x = x$, $(x \cdot c) \cdot -1 = x$, and $n \cdot -1$ as undefined. We call x a *leaf* if it has no successors. A *path* π in F is a word $\pi = x_1 x_2 \ldots$ of F such that x_1 is a root of F and for every $x_i \in \pi$, either x_i is a leaf (i.e., π ends in x_i) or x_i is a predecessor of x_{i+1}. Given two alphabets Σ_1 and Σ_2, a (Σ_1, Σ_2)–labeled forest is a triple $\langle F, V, E\rangle$, where F is a forest, $V : F \rightarrow \Sigma_1$ maps each node of F to a letter in Σ_1, and $E : F \times F \rightarrow \Sigma_2$ is a partial function that maps each pair (x, y), with $y \in sc(x)$, to a letter in Σ_2. As a particular case, we consider a forest without labels on edges as a Σ_1–labeled forest $\langle F, V\rangle$, and a *tree* as a forest containing exactly one tree. A *quasi–forest* is a forest where each node may also have roots as successors. For a node x of a quasi–forest, we set $children(x)$ as $sc(x) \setminus \mathbb{N}$. All the other definitions regarding forests easily extend to quasi–forests. Notice that in a quasi–forest, since each node can have a root as successor, a root can also have several predecessors, while every other node has just one. Clearly, a quasi–forest can always be transformed into a forest by removing root successors.

Hybrid Graded μ–Calculus. Let *Var*, *Prog*, *AP*, and *Nom* be finite and pairwise disjoint sets of *variables*, *atomic programs*, *atomic propositions*, and *nominals*. The set of *hybrid graded μ–calculus* formulas is the smallest set such that (*i*) **true** and **false** are formulas; (*ii*) p and $\neg p$, for $p \in AP \cup Nom$, are formulas; (*iii*) $x \in Var$ is a formula; (*iv*) if φ_1 and φ_2 are formulas, $\alpha \in Prog$, n is a non negative integer, and $y \in Var$, then $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, $\langle n, \alpha\rangle\varphi_1$, $[n, \alpha]\varphi_1$, $\mu y.\varphi_1(y)$, and $\nu y.\varphi_1(y)$ are also formulas. Observe that we use positive normal form, i.e., negation is applied only to atomic propositions.

We call μ and ν *fixpoint operators*. A propositional variable y occurs *free* in a formula if it is not in the scope of a fixpoint operator. A *sentence* is a formula that contains no free variables. We refer often to the *graded modalities* $\langle n, \alpha\rangle\varphi_1$ and $[n, \alpha]\varphi_1$ as respectively *atleast formulas* and *allbut formulas* and assume that the integers in these operators are given in binary coding: the contribution of n to the length of the formulas $\langle n, \alpha\rangle\varphi$ and $[n, \alpha]\varphi$ is $\lceil \log n \rceil$ rather than n.

The semantics of the hybrid graded μ–calculus is defined with respect to a *Kripke structure*, i.e., a tuple $\mathcal{K} = \langle W, W_0, R, L\rangle$ where W is a non–empty set of *states*, $W_0 \subseteq W$ is the set of initial states, $R : Prog \rightarrow 2^{W \times W}$ is a function that assigns to each atomic program a transition relation over W, and $L : AP \cup Nom \rightarrow 2^W$ is a labeling function that assigns to each atomic proposition and nominal a set of states such that the sets assigned to nominals are singletons and subsets of W_0. If $(w, w') \in R(\alpha)$, we say that w' is an α*–successor* of w. Informally, an *atleast* formula $\langle n, \alpha\rangle\varphi$ holds at a state w of $\mathcal{K}$ if φ holds in at least $n+1$ α–successors of w. Dually, the *allbut* formula $[n, \alpha]\varphi$ holds in a state w of $\mathcal{K}$ if φ holds in all but at most n α–successors of w. Note that $\neg\langle n, \alpha\rangle\varphi$ is equivalent to $[n, \alpha]\neg\varphi$, and the modalities $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$ of the standard μ–calculus can be expressed as $\langle 0, \alpha\rangle\varphi$ and $[0, \alpha]\varphi$, respectively.

To formalize semantics, we introduce valuations. Given a Kripke structure $\mathcal{K} = \langle W, W_0, R, L\rangle$ and a set $\{y_1, \ldots, y_n\}$ of variables in *Var*, a *valuation* $\mathcal{V} : \{y_1, \ldots, y_n\} \rightarrow 2^W$ is an assignment of subsets of W to the variables $y_1, \ldots, y_n$. For a valuation $\mathcal{V}$, a variable y, and a set $W' \subseteq W$, we denote by $\mathcal{V}[y \leftarrow W']$ the valuation obtained from $\mathcal{V}$ by assigning W' to y. A formula φ with free variables among $y_1, \ldots, y_n$ is interpreted over $\mathcal{K}$ as a mapping $\varphi^{\mathcal{K}}$ from valuations to 2^W, i.e., $\varphi^{\mathcal{K}}(\mathcal{V})$ denotes the set of points that satisfy φ under valuation $\mathcal{V}$. The mapping $\varphi^{\mathcal{K}}$ is defined inductively as follows:

- $\mathbf{true}^{\mathcal{K}}(\mathcal{V}) = W$ and $\mathbf{false}^{\mathcal{K}}(\mathcal{V}) = \emptyset$;
- for $p \in AP \cup$ [illegible], we have $p^{\mathcal{K}}(\mathcal{V}) = L(p)$ and $(\neg p)^{\mathcal{K}}(\mathcal{V}) = W \setminus L(p)$;
- for $y \in$ [illegible], we have $y^{\mathcal{K}}(\mathcal{V}) = \mathcal{V}(y)$;
- $(\varphi_1 \wedge \varphi_2)^{\mathcal{K}}(\mathcal{V}) = \varphi_1^{\mathcal{K}}(\mathcal{V}) \cap \varphi_2^{\mathcal{K}}(\mathcal{V})$ and $(\varphi_1 \vee \varphi_2)^{\mathcal{K}}(\mathcal{V}) = \varphi_1^{\mathcal{K}}(\mathcal{V}) \cup \varphi_2^{\mathcal{K}}(\mathcal{V})$;
- $(\langle n, \alpha \rangle \varphi)^{\mathcal{K}}(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \in \varphi^{\mathcal{K}}(\mathcal{V})\}| \geq n+1\}$;
- $([n, \alpha] \varphi)^{\mathcal{K}}(\mathcal{V}) = \{w : |\{w' \in W : (w, w') \in R(\alpha) \text{ and } w' \notin \varphi^{\mathcal{K}}(\mathcal{V})\}| \leq n\}$;
- $(\mu y.\varphi(y))^{k}(\mathcal{V}) = \bigcap \{W' \subseteq W : \varphi^{\mathcal{K}}([y \leftarrow W']) \subseteq W'\}$;
- $(\nu y.\varphi(y))^{k}(\mathcal{V}) = \bigcup \{W' \subseteq W : W' \subseteq \varphi^{\mathcal{K}}([y \leftarrow W'])\}$.

For a state w of a Kripke structure $\mathcal{K}$, we say that $\mathcal{K}$ [illegible] φ at w if $w \in \varphi^{\mathcal{K}}$. In what follows, a formula φ [illegible] up to b if the maximal integer in [illegible] and [illegible] formulas used in φ is $b - 1$.

Open Kripke Structures. In this paper we consider open systems, i.e., systems that interact with their environment and whose behavior depends on this interaction. The (global) behavior of such a system is described by a [illegible] $\mathcal{M} = \langle W_s, W_e, W_0, R, L \rangle$, which is a Kripke structure where the set of states $W = W_s \cup W_e$ is partitioned in [illegible] W_s and [illegible] W_e.

Given a module $\mathcal{M}$, we assume that its states are ordered and the number of successors of each state w is finite. For each $w \in W$, we denote by $succ(w)$ the ordered tuple (possibly empty) of w's α-successors, for all $\alpha \in Prog$. When $\mathcal{M}$ is in a system state w_s, then all states in $succ(w_s)$ are possible next states. On the other hand, when $\mathcal{M}$ is in an environment state w_e, the possible next states (that are in $succ(w_e)$) depend on the current environment. Since the behavior of the environment is not predictable, we have to consider all the possible sub–tuples of $succ(w_e)$. The only constraint, since we consider environments that cannot block the system, is that not all the transitions from w_e are disabled.

The set of all (maximal) computations of $\mathcal{M}$ starting from W_0 is described by a $(W, Prog)$–labeled quasi–forest $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$, called [illegible], which is obtained by unwinding $\mathcal{M}$ in the usual way. The problem of deciding, for a given branching–time formula φ over $AP \cup Nom$, whether $\langle F_{\mathcal{M}}, L \circ V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ satisfies φ at a root node, denoted $\mathcal{M} \models \varphi$, is the usual [illegible] [CE81, QS81]. On the other hand, for an open system $\mathcal{M}$, the quasi–forest $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ corresponds to a very specific environment, i.e., a maximal environment that never restricts the set of its next states. Therefore, when we examine a branching–time formula φ w.r.t. $\mathcal{M}$, the formula φ should hold not only in $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$, but in all quasi-forests obtained by pruning from $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ subtrees rooted at children of environment nodes, as well as inhibiting some of their jumps to roots, if there are any. The set of these quasi–forests, which collects all possible behaviors of the environment, is denoted by $exec(\mathcal{M})$ and is formally defined as follows. A quasi–forest $\langle F, V, E \rangle \in exec(\mathcal{M})$ iff for each $w_i \in W_0$, we have $V(i) = w_i$, and for $x \in F$, with $V(x) = w$, $succ(w) =$

$\langle w_1, \ldots, w_n, w_{n+1}, \ldots, w_{n+m}\rangle$, and $succ(w) \cap W_0 = \langle w_{n+1}, \ldots, w_{n+m}\rangle$, there exists $S = \langle w'_1, \ldots, w'_p, w'_{p+1}, \ldots, w'_{p+q}\rangle$ sub-tuple of $succ(w)$ such that $p + q \geq 1$, $S = succ(w)$ if $w \in W_s$ and the following hold:

- $children(x) = \{x \cdot 1, \ldots, x \cdot p\}$ and, for $1 \leq j \leq p$, we have $V(x \cdot j) = w'_j$ and $E(x, x \cdot j) = \alpha$ if $(w, w'_j) \in R(\alpha)$;
- for $1 \leq j \leq q$, let $x_j \in \mathbb{N}$ such that $V(x_j) = w'_{p+j}$, then $E(x, x_j) = \alpha$ if $(w, w'_{p+j}) \in R(\alpha)$.

In the following, we consider quasi–forests in $exec(\mathcal{M})$ as labeled with $(2^{AP\cup Nom}, Prog)$, i.e., the label of a node x is $L(V(x))$. For a module $\mathcal{M}$ and a formula φ of the hybrid graded μ–calculus, we say that $\mathcal{M}$ *reactively* satisfies φ, denoted $\mathcal{M} \models_r \varphi$, if all quasi-forests in $exec(\mathcal{M})$ satisfy φ. The problem of deciding whether $\mathcal{M} \models_r \varphi$ is called *enriched μ-calculus module checking*.

Open Pushdown Systems (*OPD*). An *OPD* over $AP \cup Nom \cup Prog$ is a tuple $\mathcal{S} = \langle Q, \Gamma, \flat, C_0, \Delta, \rho_1, \rho_2, Env\rangle$, where Q is a finite set of (control) *states*, Γ is a finite *stack alphabet*, $\flat \notin \Gamma$ is the *stack bottom symbol*. We set $\Gamma_\flat = \Gamma \cup \{\flat\}$, $Conf = Q \times (\Gamma^* \cdot \flat)$ to be the set of *(pushdown) configurations*, and for each configuration $(q, A \cdot \gamma)$, we set $top((q, A \cdot \gamma)) = (q, A)$ to be a *top configuration*. The function $\Delta : Prog \rightarrow 2^{(Q\times\Gamma_\flat)\times(Q\times\Gamma_\flat^*)}$ is a finite set of transition rules such that $\flat$ is always present at the bottom of the stack and nowhere else (thus whenever $\flat$ is read, it is pushed back). Note that we make this assumption also about the various pushdown automata we use later. The set $C_0 \subseteq Conf$ is a finite set of *initial configurations*, $\rho_1 : AP \rightarrow 2^{Q\times\Gamma_\flat}$ and $\rho_2 : Nom \rightarrow C_0$ are labeling functions associating respectively to each atomic proposition p a set of top configurations in which p holds and to each nominal exactly one initial configuration. Finally, $Env \subseteq Q \times \Gamma_\flat$ specifies the set of *environment configurations*. The size $|\mathcal{S}|$ of $\mathcal{S}$ is $|Q| + |\Delta| + |\Gamma|$.

The *OPD* moves in accordance with the transition relation Δ. Thus, $((q, A), (q', \gamma)) \in \Delta(\alpha)$ implies that if the *OPD* is in state q and the top of the stack is A, it can move along with an α *transition* to state q', and substitute γ for A. Also note that the possible operations of the system, the labeling functions, and the designation of configurations as environment configurations, are all dependent only on the current control state and the top of the stack.

An *OPD* $\mathcal{S}$ induces a module $\mathcal{M}_\mathcal{S} = \langle W_s, W_e, W_0, R, L\rangle$, where:

- $W_s \cup W_e = Conf$, i.e. the set of pushdown configurations, and $W_0 = C_0$;
- $W_e = \{c \in Conf \mid top(c) \in Env\}$.
- $((q, A \cdot \gamma), (q', \gamma' \cdot \gamma)) \in R(\alpha)$ *iff* there is $((q, A), (q', \gamma')) \in \Delta(\alpha)$;
- $L(p) = \{c \in Conf \mid top(c) \in \rho_1(p)\}$ for $p \in AP$; $L(o) = \rho_2(o)$ for $o \in Nom$.

The *enriched μ-calculus pushdown module checking* problem is to decide, for a given *OPD* $\mathcal{S}$ and an enriched μ–calculus formula φ, whether $\mathcal{M}_\mathcal{S} \models_r \varphi$.

3 Tree Automata

Two-way Graded Alternating Parity Tree Automata (2*GAPT*). These automata are an extension of nondeterministic tree automata in such a way that a 2[illegible] can send several copies of itself to the same successor ([illegible]), send copies of itself to the predecessor ([illegible]), specify a number n of successors to which copies of itself are sent without specifying which successors these exactly are ([illegible]), and accept trees along with a [illegible], cf. [BLMV06]. To give a formal definition, let us start with some technicalities.

For a given set Y, let $B^+(Y)$ be the set of positive Boolean formulas over Y (i.e., Boolean formulas built from elements in Y using $\wedge$ and $\vee$), where we also allow the formulas **true** and **false** and $\wedge$ has precedence over $\vee$. For a set $X \subseteq Y$ and a formula $\theta \in B^+(Y)$, we say that X satisfies θ iff assigning **true** to elements in X and assigning **false** to elements in $Y \setminus X$ makes θ **true**. For $b > 0$, let $\langle[b]\rangle = \{\langle 0\rangle, \langle 1\rangle, \ldots, \langle b\rangle\}$, $[[b]] = \{[0], [1], \ldots, [b]\}$, and $D_b = \langle[b]\rangle \cup [[b]] \cup \{-1, \varepsilon\}$.

Formally, a 2[illegible] on Σ-labeled trees is a tuple $\mathcal{A} = \langle \Sigma, b, Q, \delta, q_0, \mathcal{F}\rangle$, where Σ is the input alphabet, $b > 0$ is a counting bound, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow B^+(D_b \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and $\mathcal{F}$ is a parity acceptance condition (see below). Intuitively, an atom $(\langle n\rangle, q)$ (resp. $([n], q)$) means that $\mathcal{A}$ sends copies in state q to $n+1$ (resp. all but n) different successors of the current node, (ε, q) means that $\mathcal{A}$ sends a copy (in state q) to the current node, and $(-1, q)$ means that $\mathcal{A}$ sends a copy to the predecessor of the current node. A [illegible] of $\mathcal{A}$ on an input Σ-labeled tree $\langle T, V\rangle$ is a tree $\langle T_r, r\rangle$ in which each node is labeled by an element of $T \times Q$. Intuitively, a node in T_r labeled by (x, q) describes a copy of the automaton in state q that reads the node x of T. Runs start in the initial state and satisfy the transition relation. Thus, a run $\langle T_r, r\rangle$ with root z has to satisfy the following: (i) $r(z) = (1, q_0)$ for the root 1 of T and (ii) for all $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq D_b \times Q$, such that S satisfies θ, and for all $(d, s) \in S$, the following hold:

- If $d \in \{-1, \varepsilon\}$, then $x \cdot d$ is defined, and there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot d, s)$;
- If $d = \langle n\rangle$ (resp., $d = [n]$), there are at least $t = n + 1$ (resp., $t = deg(x) - n$) distinct indexes $i_1, \ldots, i_t$ such that for all $1 \leq j \leq t$, there is $j' \in \mathbb{N}$ such that $y \cdot j' \in T_r$, $x \cdot i_j \in T$, and $r(y \cdot j') = (x \cdot i_j, s)$.

Note that if $\theta =$ **true**, then y does not need to have successors. This is the reason why T_r may have leaves. Also, since there exists no set S as required for $\theta =$ **false**, we cannot have a run that takes a transition with $\theta =$ **false**.

A run $\langle T_r, r\rangle$ is [illegible] if all its infinite paths satisfy the acceptance condition. In the parity acceptance condition, $\mathcal{F}$ is a set $\{F_1, \ldots, F_k\}$ such that $F_1 \subseteq \ldots \subseteq F_k = Q$ and k is called the [illegible] of the automaton. An infinite path π on T_r [illegible] $\mathcal{F}$ if there is an even i such that π contains infinitely many states from F_i and finitely many states from F_{i-1}. An automaton [illegible] a tree iff there exists an accepting run of the automaton on the tree. We denote by $\mathcal{L}(\mathcal{A})$ the set of all Σ-labeled trees that $\mathcal{A}$ accepts.

A 2[illegible] is a [illegible] if $\delta : Q \times \Sigma \to B^+(D_b \setminus \{-1\} \times Q)$ and a 2[illegible] if $\delta : Q \times \Sigma \to B^+(\{-1, \varepsilon, 1, \ldots, h\} \times Q)$. Moreover, it is an [illegible] if $\delta : Q \times \Sigma \to B^+(\{1, \ldots, h\} \times Q)$ and the transition relation δ is in disjunctive normal form, where in each conjunct each direction appears at most once [KVW00]. We now recall a result on [illegible] and hybrid graded μ-calculus formulas.

Lemma 1 ([BLMV06]). [illegible] μ [illegible] φ [illegible] ℓ atleast [illegible] b [illegible] GAPT [illegible] $\mathcal{O}(|\varphi|^2)$ [illegible] $|\varphi|$ [illegible] b [illegible] φ [illegible] $\max\{|Nom|+1, \ell(b+1)\}$

Nondeterministic Pushdown Parity Tree Automata (*PD–NPT*). A [illegible] (without ε-transitions), on Σ-labeled full h-ary trees, is a tuple $\mathcal{P} = \langle \Sigma, \Gamma, \flat, Q, q_0, \gamma_0, \rho, \mathcal{F} \rangle$, where Σ is a finite input alphabet, Γ, $\flat$, $\Gamma_\flat$, and Q are as in [illegible], (q_0, γ_0) is the initial configuration, $\rho : Q \times \Sigma \times \Gamma_\flat \to 2^{(Q \times \Gamma_\flat^*)^h}$ is a transition function, and $\mathcal{F}$ is a parity condition over Q. Intuitively, when $\mathcal{P}$ is in state q, reading an input node x labeled by $\sigma \in \Sigma$, and the stack contains a word $A \cdot \gamma \in \Gamma^* \cdot \flat$, then $\mathcal{P}$ chooses a tuple $\langle (q_1, \gamma_1), \ldots, (q_h, \gamma_h) \rangle \in \rho(q, \sigma, A)$ and splits in h copies such that for each $1 \leq i \leq h$, a copy in configuration $(q_i, \gamma_i \cdot \gamma)$ is sent to the node $x \cdot i$ in the input tree. A run of $\mathcal{P}$ on a Σ-labeled full h-ary tree $\langle T, V \rangle$ is a $(Q \times \Gamma^* \cdot \flat)$-labeled tree $\langle T, r \rangle$ such that $r(\varepsilon) = (q_0, \gamma_0)$ and for each $x \in T$ with $r(x) = (q, A \cdot \gamma)$, there is $\langle (q_1, \gamma_1), \ldots, (q_h, \gamma_h) \rangle \in \rho(q, V(x), A)$ where, for all $1 \leq i \leq h$, we have $r(x \cdot i) = (q_i, \gamma_i \cdot \gamma)$. The notion of accepting path is defined with respect to the control states that appear infinitely often in the path (thus without taking into account any stack content). Then, the notions given for 2[illegible] regarding accepting runs, accepted trees, and accepted languages, along with the parity condition, easily extend to [illegible]. We also consider Büchi condition $\mathcal{F} \subseteq Q$, which simply is a special parity condition $\{\emptyset, \mathcal{F}, Q\}$. In the following, we denote with [illegible] a [illegible] with a Büchi condition. The [illegible] problem for an automaton $\mathcal{P}$ is to decide whether $\mathcal{L}(\mathcal{P}) = \emptyset$. We now recall two useful results on the introduced automata.

Proposition 1 ([KPV02]). [illegible] PD–NPT [illegible] Σ [illegible] h [illegible] m n [illegible] ρ [illegible] $n \cdot m \cdot h \cdot |\rho|$

Proposition 2 ([BMP05]). [illegible] Σ [illegible] h [illegible] PD–NBT $\mathcal{P} = \langle \Sigma, \Gamma, Q, q_0, \gamma_0, \rho, Q \rangle$ [illegible] NPT $\mathcal{A} = \langle \Sigma, Q', q'_0, \delta, \mathcal{F}' \rangle$ [illegible] PD–NPT $\mathcal{P}'$ [illegible] $\mathcal{L}(\mathcal{P}') = \mathcal{L}(\mathcal{P}) \cap \mathcal{L}(\mathcal{A})$ [illegible] $\mathcal{P}'$ [illegible] $|Q| \cdot |Q'|$ [illegible] $\mathcal{A}$ [illegible] $|\rho| \cdot |\delta| \cdot h$

4 From 2*GAPT* to *NPT*

In this section, we give a nontrivial exponential-time translation from 2[illegible] to [illegible]. To the best of our knowledge this exponentially improves the known result from the literature, e.g. using results from [BLMV06, KSV02] one can easily get a double exponential-time translation. The translation we propose uses the notions of *strategies*, *promises* and *annotations*, which we now recall.

Let $\mathcal{A} = \langle \Sigma, b, Q, \delta, q_0, \mathcal{F}\rangle$ be a 2*GAPT* with $\mathcal{F} = \langle F_1, \ldots, F_k\rangle$ and $\langle T, V\rangle$ be a Σ-labeled tree. Recall that $D_b = \langle [b]\rangle \cup [[b]] \cup \{-1, \varepsilon\}$ and $\delta : (Q \times \Sigma) \rightarrow B^+(D_b \times Q)$. For each control state $q \in Q$, let $index(q)$ be the minimal i such that $q \in F_i$. A *strategy* for $\mathcal{A}$ on $\langle T, V\rangle$ is a $2^{Q\times D_b \times Q}$-labeled tree $\langle T, \mathsf{str}\rangle$ such that, defined $head(w) = \{q : (q, d, q') \in w\}$ as the set of *heads* of w, it holds that (i) $q_0 \in head(\mathsf{str}(root(T)))$ and (ii) for each node $x \in T$ and state q, the set $\{(q, q') : (q, d, q') \in \mathsf{str}(x)\}$ satisfies $\delta(q, V(x))$.

A *promise* for $\mathcal{A}$ on $\langle T, V\rangle$ is a $2^{Q\times Q}$-labeled tree $\langle T, \mathsf{pro}\rangle$. We say that pro *fulfills* str for V if the states promised to be visited by pro satisfy the obligations induced by str as it runs on V. Formally, pro fulfills str for V if for every node $x \in T$, the following hold: "for every $(q, \langle n\rangle, q') \in \mathsf{str}(x)$ (resp. $(q, [n], q') \in \mathsf{str}(x)$), at least $n+1$ (resp $deg(x) - n$) successors $x \cdot j$ of x have $(q, q') \in \mathsf{pro}(x \cdot j)$".

An *annotation* for $\mathcal{A}$ on $\langle T, \mathsf{str}\rangle$ and $\langle T, \mathsf{pro}\rangle$ is a $2^{Q\times\{1,\ldots,k\}\times Q}$-labeled tree $\langle T, \mathsf{ann}\rangle$ such that for each $x \in T$ and $(q, d_1, q_1) \in \mathsf{str}(x)$ the following hold:

- if $d_1 = \varepsilon$, then $(q, index(q_1), q_1) \in \mathsf{ann}(x)$;
- if $d_1 \in \{1, \ldots, k\}$, then for all $d_2 \in \{1, \ldots, k\}$ and $q_2 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(x)$, we have $(q, \min(d_1, d_2), q_2) \in \mathsf{ann}(x)$;
- if $d_1 = -1$ and $x = y \cdot i$, then for all $d_2, d_3 \in \{1, \ldots, k\}$ and $q_2, q_3 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(y)$, $(q_2, d_3, q_3) \in \mathsf{str}(y)$, and $(q_2, q_3) \in \mathsf{pro}(x)$, it holds that $(t, \min(index(q_1), d_2, index(q_3)), q_3) \in \mathsf{ann}(x)$;
- if $d_1 \in [[b]] \cup \langle [b]\rangle$, $y = x \cdot i$, and $(q, q_1) \in \mathsf{pro}(y)$, then for all $d_2, d_3 \in \{1, \ldots, k\}$ and $q_2, q_3 \in Q$ such that $(q_1, d_2, q_2) \in \mathsf{ann}(y)$ and $(q_2, -1, q_3) \in \mathsf{str}(y)$, it holds that $(t, \min(index(q_1), d_2, index(q_3)), q_3) \in \mathsf{ann}(x)$.

A downward path induced by str, pro, and ann on $\langle T, V\rangle$ is a sequence $\langle x_0, q_0, t_0\rangle, \langle x_1, q_1, t_1\rangle, \ldots$ such that $x_0 = root(T)$, q_0 is the initial state of $\mathcal{A}$ and, for each $i \geq 0$, it holds that $x_i \in T$, $q_i \in Q$, and $t_i = \langle q_i, d, q_{i+1}\rangle \in \mathsf{str}(x_i) \cup \mathsf{ann}(x_i)$ is such that either *(i)* $d \in \{1, \ldots, k\}$ and $x_{i+1} = x_i$, or *(ii)* $d \in \langle [b]\rangle \cup [[b]]$ and there exists $c \in \{1, \ldots, deg(x_i)\}$ such that $x_{i+1} = x_i \cdot c$ and $(q_i, q_{i+1}) \in \mathsf{pro}(x_{i+1})$. In the first case we set $index(t_i) = d$ and in the second case we set $index(t_i) = \min\{j \in \{1, \ldots, k\} \mid q_{i+1} \in F_j\}$. Moreover, for a downward path π, we set $index(\pi)$ as the minimum index that appears infinitely often in π. Finally, we say that π is *good* if $index(\pi)$ is even.

The following lemma relates languages accepted by 2*GAPT* with strategies, promises, and annotations.

Lemma 2 ([BLMV06]). *Let* $\mathcal{A}$ *be a* 2GAPT *on* Σ*-labeled trees.* $\langle T, V\rangle$ *is accepted by* $\mathcal{A}$ *iff there exist a strategy* $\langle T, \mathsf{str}\rangle$ *and a promise* $\langle T, \mathsf{pro}\rangle$ *for* $\mathcal{A}$ *on* $\langle T, V\rangle$ *such that* pro *fulfills* str *for* V*, and an annotation* $\langle T, \mathsf{ann}\rangle$ *for* $\mathcal{A}$ *on* $\langle T, V\rangle$*,* $\langle T, \mathsf{str}\rangle$ *and* $\langle T, \mathsf{pro}\rangle$ *such that every downward path induced by* str*,* pro *and* ann *on* $\langle T, V\rangle$ *is good.*

Given an alphabet Σ for the input tree of a 2*GAPT* with transition function δ, let D_b^δ be the subset containing only the elements of D_b appearing in δ.

Then we denote by Σ' the extended alphabet for the combined trees, i.e., $\Sigma' = \Sigma \times 2^{Q \times D_b^\delta \times Q} \times 2^{Q \times Q} \times 2^{Q \times \{1,\dots k\} \times Q}$.

Lemma 3. [illegible] $\mathcal{A}$ [illegible] 2GAPT [illegible] Σ [illegible] n [illegible] k [illegible] b [illegible] h [illegible] NPT $\mathcal{A}'$ [illegible] Σ' [illegible] h [illegible] ff $\mathcal{A}$ [illegible] Σ

[illegible] Let $\mathcal{A} = \langle \Sigma, b, Q, q_0, \delta, \mathcal{F} \rangle$ with $\mathcal{F} = \langle F_1, \dots, F_k \rangle$. By Lemma 2, we construct $\mathcal{A}'$ as the intersection of three [illegible] $\mathcal{A}'$, $\mathcal{A}''$, and $\mathcal{A}'''$, each having size exponential in the size of $\mathcal{A}$, such that, given a Σ'-labeled tree $T' = \langle T, (V, \mathsf{str}, \mathsf{pro}, \mathsf{ann}) \rangle$, [illegible] $\mathcal{A}'$ accepts T' iff str is a strategy for $\mathcal{A}$ on $\langle T, V \rangle$ and pro fulfills str for V, [illegible] $\mathcal{A}''$ accepts T' iff ann is an annotation for $\mathcal{A}$ on $\langle T, V \rangle$, $\langle T, \mathsf{str} \rangle$ and $\langle T, \mathsf{pro} \rangle$, and [illegible] $\mathcal{A}'''$ accepts T' iff every downward path induced by str, pro, and ann on $\langle T, V \rangle$ is accepting.

The automaton $\mathcal{A}' = \langle \Sigma', Q', q'_0, \delta', \mathcal{F}' \rangle$ works as follows: on reading a node x labeled $(\sigma, \eta, \rho, \omega)$, then it locally checks whether η satisfies the definition of strategy for $\mathcal{A}$ on $\langle T, V \rangle$. In particular, when $\mathcal{A}'$ is in its initial state, we check that η contains a transition starting from the initial state of $\mathcal{A}$. Moreover, the automaton $\mathcal{A}'$ sends to each child $x \cdot i$ the pairs of states that have to be contained in $\mathsf{pro}(x \cdot i)$, in order to verify that pro fulfills str. To obtain this, we set $Q' = 2^{Q \times Q} \cup \{q'_0\}$ and $\mathcal{F}' = \{\emptyset, Q'\}$. To define δ', we first give the following definition. For each node $x \in T$ labeled $(\sigma, \eta, \rho, \omega)$, we set

$$
\begin{aligned}
S(\eta) = \{ & \langle S_1, \dots, S_{deg(x)} \rangle \in (2^{Q \times Q})^{deg(x)} \text{ such that} \\
& [\text{for each } (q, \langle m \rangle, p) \in \eta \ \text{ there is } P \subseteq \{1, \dots deg(x)\} \text{ with } |P| = m+1 \\
& \text{such that for all } i \in P,\ (q,p) \in S_i] \text{ and} \\
& [\text{for each } (q, [m], p) \in \eta \ \text{ there is} P \subseteq \{1, \dots deg(x)\} \text{ with } |P| = deg(x) - m \\
& \text{such that for all } i \in P,\ (q,p) \in S_i]\}
\end{aligned}
$$

to be the set of all tuples with size $deg(x)$, each [illegible] all graded modalities in $\mathsf{str}(x)$. Notice that $|S(\eta)| \leq 2^{hn^2}$. Then we have

$$
\delta'(q, (\sigma, \eta, \rho, \omega)) = \begin{cases} S(\eta) & \text{if } \forall\, p \in head(\eta),\ \{(d, p') \mid (p, d, p') \in \eta\} \text{ satisfies } \delta(p, \sigma) \\ & \text{and } [(q = q_0^1 \text{ and } q_0 \in head(\eta)) \text{ or } (q \neq q_0^1 \text{ and } q \subseteq \rho)] \\ \textbf{false} & \text{otherwise.} \end{cases}
$$

Hence, in $\mathcal{A}'$ we have $|Q'| = 2^{n^2}$, $|\delta'| \leq 2^{n^2(k+1)}$, and index 2.

$\mathcal{A}'' = \langle \Sigma', Q'', q''_0, \delta'', \mathcal{F}'' \rangle$ works in a similar way to $\mathcal{A}'$. That is, for each node x, it first locally checks whether the constraints of the annotations are verified; then it sends to the children of x the strategy and annotation associated with x, in order to successively verify whether the promises associated with the children nodes are consistent with the annotation of x. Therefore, in A'' we have $Q'' = 2^{Q \times D_b^\delta \times Q} \times 2^{Q \times \{1,\dots,k\} \times Q}$, $q''_0 = (\emptyset, \emptyset)$, $\mathcal{F}'' = \{\emptyset, Q''\}$, and for a state $(\eta_{prev}, \omega_{prev})$ and a letter $(\sigma, \eta, \rho, \omega)$ we have

$$
\delta''((\eta_{prev}, \omega_{prev}), (\sigma, \eta, \rho, \omega)) = \begin{cases} \langle (\eta, \omega), \dots, (\eta, \omega) \rangle & \text{if the local conditions for the} \\ & \text{annotations are verified} \\ \textbf{false} & \text{otherwise.} \end{cases}
$$

Hence, in $\mathcal{A}''$ we have $|Q''| \leq 2^{n^2(|\delta|+k)}$, $|\delta''| \leq h \cdot 2^{n^2(|\delta|+k)}$, and index 2.

Finally, to define $\mathcal{A}'''$ we start by constructing a 2[illegible] $\mathcal{B}$ whose size is polynomial in the size of $\mathcal{A}$ and accepts $\langle T, (V, \mathsf{str}, \mathsf{pro}, \mathsf{ann})\rangle$ iff there is a non accepting downward path (w.r.t. $\mathcal{A}$) induced by str, pro, and ann on $\langle T, V\rangle$. The automaton $\mathcal{B} = \langle \Sigma', Q^B, q_0^B, \delta^B, \mathcal{F}^B\rangle$ (which in particular does not need direction -1) essentially chooses, in each state, the downward path to walk on, and uses an integer to store the index of the state. We use a special state $\sharp$ not belonging to Q to indicate that $\mathcal{B}$ proceeds in accordance with an annotation instead of a strategy. Therefore, $Q^B = ((Q \cup \{\sharp\}) \times \{1, \ldots, k\} \times Q) \cup \{q_0^B\}$.

To define the transition function on a node x, let us introduce a function f that for each $q \in Q$, strategy $\eta \in 2^{Q \times D_b^\delta \times Q}$, and annotation $\omega \in 2^{Q \times \{1,\ldots,k\} \times Q}$ gives a formula satisfied along downward paths consistent with η and ω, starting from a node reachable in $\mathcal{A}$ with the state q. That is, in each node x, the function f either proceeds according to the annotation ω or the strategy η (note that f does not check that the downward path is consistent with any promise). Formally, f is defined as follows, where $index(p)$ is the minimum i such that $p \in F_i$:

$$f(q, \eta, \omega) = \bigvee_{\substack{(q,d,p) \in \omega \\ d \in \{1,\ldots,k\}}} \langle \varepsilon, (\sharp, d, p)\rangle \quad \vee \quad \bigvee_{\substack{(q,d,p) \in \eta \\ d \in \langle [b]\rangle \cup [[b]]}} \ \bigvee_{c \in \{1,\ldots,deg(x)\}} \langle c, (q, index(p), p)\rangle$$

Then, we have $\delta^B(q_0^B, (\sigma, \eta, \rho, \omega)) = f(q_0, \eta, \omega)$ and

$$\delta^B((q, d, p), (\sigma, \eta, \rho, \omega)) = \begin{cases} \textbf{false} & \text{if } q \neq \sharp \text{ and } (q, p) \notin \rho \\ f(p, \eta, \omega) & \text{otherwise.} \end{cases}.$$

A downward path π is non accepting for $\mathcal{A}$ if the minimum index that appears infinitely often in π is odd. Therefore, $\mathcal{F}^B = \langle F_1^B, \ldots, F_{k+1}^B, Q^B\rangle$ where $F_1^B = \emptyset$ and, for all $i \in \{2, \ldots, k+1\}$, we have $F_i^B = \{(q, d, p) \in Q^B \mid d = i - 1\}$. Thus, $|Q^B| = kn(n+1) + 1$, $|\delta^B| = k \cdot |\delta| \cdot |Q^B|$, and the index is $k+2$. Then, since $\mathcal{B}$ is alternating, we can easily complement it in polynomial-time into a 2[illegible] $\overline{\mathcal{B}}$ that accepts a tree iff all downward paths induced by str, pro, and ann on $\langle T, V\rangle$ are accepting. Finally, following [Var98] we construct in exponential-time the desired automaton $\mathcal{A}'''$. $\square$

5 Deciding Hybrid Graded Pushdown Module Checking

In this section, we show that hybrid graded pushdown module checking is decidable and solvable in 2Exptime. Since [illegible] pushdown module checking is 2Exptime-hard, we get that the addressed problem is 2Exptime-complete. For the upper bound, the algorithm works as follows. Given an [illegible] $\mathcal{S}$ and the module $\mathcal{M}_\mathcal{S}$ induced by $\mathcal{S}$, by combining and extending the constructions given in [BMP05] and [FM07], we first build in polynomial-time a [illegible] $\mathcal{A}_\mathcal{S}$ accepting each tree that encodes a quasi-forest belonging to $exec(\mathcal{M}_\mathcal{S})$. Then, given an hybrid graded μ-calculus formula φ, according to [BLMV06], we build in polynomial-time a [illegible] $\mathcal{A}_{\neg\varphi}$ (Lemma 1) accepting all models of $\neg\varphi$, with the intent of checking that no models of $\neg\varphi$ are in $exec(\mathcal{M}_{\mathcal{S}})$ [3]. Then, accordingly to the basic idea of [KVW01], we check that $\mathcal{M} \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_{\mathcal{S}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. Finally, we get the result by using an exponential-time reduction of the latter to the emptiness problem for , which from Proposition 1 can be solved in EXPTIME. As a key step of the above reduction, we use the exponential-time translation from into showed in Section 4.

Let us start dealing with $\mathcal{A}_{\mathcal{S}}$. Before building the automaton, there are some technical difficulties to overcome, which are also new with respect to [BMP05]. First note that since $\mathcal{A}_{\mathcal{S}}$ is a , it only deals with trees having labels on nodes, while $exec(\mathcal{M})$ contains quasi-forests with both edges and nodes labeled. To solve this problem, for each quasi-forest in $exec(\mathcal{M})$, the automaton $\mathcal{A}_{\mathcal{S}}$ accepts a corresponding encoding tree obtained by adding a new root connecting all roots of the quasi-forest, moving the label of each edge to the target node of the edge (using a new atomic proposition p_α, for each program α), and substituting "jumps to roots" with new atomic propositions $\uparrow_o^\alpha$ (representing an α-labeled jump to the unique root node labeled by nominal o). Let ${}^{*} = \cup \{p_\alpha \mid \alpha \text{ is a program}\} \cup \{\uparrow_o^\alpha \mid \alpha \text{ is a program and } o \text{ is a nominal}\}$, we denote with $\langle T, V^* \rangle$ the $2^{AP^* \cup Nom}$-labeled tree of a quasi-forest $\langle F, V, E \rangle \in exec(\mathcal{M})$, obtained using the above transformations.

Another technical difficulty to handle with is related to the fact that quasi-forests of $exec(\mathcal{M})$ (and thus their encodings) may not be full h-ary, since the nodes of the from which $\mathcal{M}$ is induced may have different degrees. Also, quasi-forests of $exec(\mathcal{M})$ may not share the same structure, since they are obtained by pruning subtrees from the computation quasi-forest $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ of $\mathcal{M}$. Let $\langle T_{\mathcal{M}}, V_{\mathcal{M}}^* \rangle$ be the h-ary of $\mathcal{M}$ obtained from $\langle F_{\mathcal{M}}, V_{\mathcal{M}}, E_{\mathcal{M}} \rangle$ using the above encoding. By extending an idea of [KVW01, BMP05], we consider each tree $\langle T, V^* \rangle$, encoding of a quasi-forest $\langle F, V, E \rangle$ of $exec(\mathcal{M})$, as a $2^{AP^* \cup Nom \cup \{t\}} \cup \{\perp\}$-labeled full h-ary tree $\langle T_{\mathcal{M}}, V^{**} \rangle$ (where $\perp$ and t are fresh proposition names not belonging to $AP^* \cup Nom$) in the following way: first we add proposition t to the label of all leaf nodes of the forest; second, for each node $x \in T_{\mathcal{M}}$ with p children $x \cdot 1, \ldots, x \cdot p$ (note that $0 \leq p \leq h$), we add the children $x \cdot (p+1), \ldots, x \cdot h$ and label these new nodes with $\perp$; finally, for each node x labeled by $\perp$ we add recursively h children labeled by $\perp$. Thus, for each node $x \in T_{\mathcal{M}} \backslash \{root(T_{\mathcal{M}})\}$, if $x \in F$ then $V^{**}(x) = V^*(x)$, otherwise $V^{**}(x) = \{\perp\}$ and therefore the proposition $\perp$ is used to denote both "disabled" states and "completion" states. In this way, all trees encoding quasi-forests belonging to $exec(\mathcal{M})$ are full h-ary trees, and they differ only in their labeling. Moreover, the environment can also disable jumps to roots. This is performed by removing from enabled environment nodes some of the $\uparrow_o^\alpha$ labels. Notice that since we consider environments that do not block the system, nodes associated with environment states have at least one successor not labeled by $\{\perp\}$, unless they have $\uparrow_o^\alpha$ in their labels. Putting in practice the construction proposed above, we obtain the following result, where $\widehat{exec}(\mathcal{M})$ is the set of all $2^{AP^* \cup Nom \cup \{t\}} \cup \{\perp\}$-labeled full h-ary trees obtained from $\langle T_{\mathcal{M}}, V^*_{\mathcal{M}}\rangle$ in the above described manner (the detailed construction is reported in the full version of the paper).

---

[3] For better readability, in the rest of the paper we use $\mathcal{M}$ instead of $\mathcal{M}_{\mathcal{S}}$.

Lemma 4. [illegible] OPD $\mathcal{S} = \langle Q, \Gamma, \flat, C_0, \Delta, \rho_1, \rho_2, Env\rangle$ [illegible] h [illegible] PD–NBT $\mathcal{A}_{\mathcal{S}} = \langle \Sigma, \Gamma, \flat, Q', q'_0, \gamma_0, \delta, Q\rangle$ [illegible] $\widehat{exec}(\mathcal{M})$ [illegible] $\Sigma = 2^{AP^* \cup Nom \cup \{t\}} \cup \{\bot\}$ $|Q'| = \mathcal{O}(|Q|^2 \cdot |\Gamma|)$ [illegible] $|\delta|$ [illegible] $h \cdot |\Delta|$

Let us now go back to the hybrid graded μ-calculus formula φ. Using Lemmas 1 and 3, we get that given an hybrid graded μ-calculus formula φ, we can build in exponential-time an [illegible] $\mathcal{A}_{\neg\varphi}$ accepting all models of $\neg\varphi$. Now, recall that given a module $\mathcal{M}$ induced by an [illegible] S and the automaton $\mathcal{A}_{\mathcal{S}}$ accepting all trees encoding of quasi-forests belonging to $exec(\mathcal{M})$ (see Lemma 4), it is possible to check whether $\mathcal{M} \models_r \varphi$ by checking whether $\mathcal{L}(\mathcal{A}_{\mathcal{S}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ is empty. Also, recall by Proposition 2 that $\mathcal{L}(\mathcal{A}_{\mathcal{S}}) \cap \mathcal{L}(\mathcal{A}_{\neg\varphi})$ can be accepted by a [illegible] $\mathcal{A}$ whose size is exponential in the size of φ and polynomial in the size of $\mathcal{S}$. Finally, by recalling that the emptiness problem for $\mathcal{A}$ can be checked in EXPTIME (Proposition 1) and that the pushdown module checking problem for [illegible] is 2EXPTIME-hard with respect to the size of the formula and EXPTIME-hard in the size of the system [BMP05], we get the following result.

Theorem 1. [illegible] μ [illegible] 2EXPTIME [illegible] EXPTIME [illegible]

6 Discussion

As a direct consequence of the algorithm we have proposed, we get that "pushdown model checking" with respect to formulas of the hybrid graded μ-calculus is also solvable in 2EXPTIME. Indeed, we recall that model checking a closed system is equivalent to check a module in which the maximal environment, which corresponds to the full computation tree, satisfies the specification. Consider now the automaton described in Lemma 4. We can easily simplify the construction to get an automaton that accepts only the full computation tree. Then, by applying our module checking algorithm we easily get the result.

The above idea can be also extended to other fragments of the fully enriched μ-calculus. We recall that this calculus extends the hybrid graded one by also allowing "backward" modalities. Syntactically, it is obtained by allowing a program α in [illegible] and [illegible] formulas to be either an atomic program a or its inverse a^-. To deal with inverse programs, we also extend R as follows: for each $a \in$ [illegible], we set $R(a^-) = \{(v, u) : (u, v) \in R(a)\}$. The semantics given for hybrid graded μ-calculus extends to fully enriched μ-calculus, accordingly and in a natural way. We recall that fully enriched μ-calculus is undecidable (see [BP04]), while any of its fragments is decidable. We argue that also for those fragments including backwards modalities, the pushdown model checking is solvable in 2EXPTIME. The main technical difficulty here is that since we are dealing with a backward modality, the unwinding must take care also of the past configurations. In particular, since each node in the tree can only have one parent, we need to simulate in forward all past configurations, but one. This can be accomplished by inverting the modality (i.e., inverting the program). Using such an unwinding, along with the above idea of constructing an automaton (by simplifying that given in Lemma 4) that accepts only the full computation tree (which now also needs to keep track of the past computations), and the pushdown module checking algorithm idea, we get the result.

Acknowledgments. We thank the anonymous referee for his useful comments regarding backward modalities.

References

[BLMV06] Bonatti, P.A., Lutz, C., Murano, A., Vardi, M.Y.: The complexity of enriched μ-calculi. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 540–551. Springer, Heidelberg (2006)

[BMP05] Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005)

[BP04] Bonatti, P.A., Peron, A.: On the undecidability of logics with converse, nominals, recursion and counting. Artif. Intelligence 158(1), 75–96 (2004)

[CE81] Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logics of Programs. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)

[CGP99] Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge, MA, USA (1999)

[FM07] Ferrante, A., Murano, A.: Enriched μ-calculus module checking. In: FOSSACS 2007. LNCS, vol. 4423, pp. 183–197 (2007)

[Hoa85] Hoare, C.A.R.: Communicating sequential processes. Prentice-Hall International, Upper Saddle River, NJ, USA (1985)

[HP85] Harel, D., Pnueli, A.: On the development of reactive systems. In: Logics and Models of Concurrent Systems. NATO Advanced Summer Institutes, vol. F-13, pp. 477–498. Springer, Heidelberg (1985)

[Koz83] Kozen, D.: Results on the propositional mu-calculus. Theoretical Computer Science 27, 333–354 (1983)

[KPV02] Kupferman, O., Piterman, N., Vardi, M.Y.: Pushdown specifications. In: Baaz, M., Voronkov, A. (eds.) LPAR 2002. LNCS (LNAI), vol. 2514, pp. 262–277. Springer, Heidelberg (2002)

[KSV02] Kupferman, O., Sattler, U., Vardi, M.Y.: The complexity of the graded μ-calculus. In: Voronkov, A. (ed.) CADE 2002. LNCS (LNAI), vol. 2392, pp. 423–437. Springer, Heidelberg (2002)

[KVW00] Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. of ACM 47(2), 312–360 (2000)

[KVW01] Kupferman, O., Vardi, M.Y., Wolper, P.: Module checking. Information & Computation 164, 322–344 (2001)

[QS81] Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: Dezani-Ciancaglini, M., Montanari, U. (eds.) International Symposium on Programming. LNCS, vol. 137, pp. 337–351. Springer, Heidelberg (1982)

[SV01] Sattler, U., Vardi, M.Y.: The hybrid mu-calculus. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 76–91. Springer, Heidelberg (2001)

[Var98] Vardi, M.Y.: Reasoning about the past with two-way automata. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 628–641. Springer, Heidelberg (1998)

Approved Models for Normal Logic Programs

Luís Moniz Pereira and Alexandre Miguel Pinto

Centro de Inteligência Artificial (CENTRIA)
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
{lmp,amp}@di.fct.unl.pt

Abstract. We introduce an original 2-valued semantics for Normal Logic Programs (NLPs) extending the well-known Argumentation work of Phan Minh Dung on Admissible Arguments and Preferred Extensions. In the 2-valued Approved Models Semantics set forth, an Approved Model (AM) correspond to the minimal positive strict consistent 2-valued completion of a Dung Preferred Extension. The AMs Semantics enjoys several non-trivial useful properties such as (1) Existence of a 2-valued Model for every NLP; (2) Relevancy, and (3) Cumulativity. Crucially, we show that the AMs Semantics is a conservative extension to the Stable Models (SMs) Semantics in the sense that every SM of a NLP is also an AM, thus providing every NLP with a model: a property not enjoyed by SMs. Integrity constraints, written in a simpler way, are introduced to identify undesired semantic scenarios, whilst permitting these to be produced nevertheless. We end the paper with some conclusions and mention of future work.

1 Introduction

This paper introduces a new 2-valued semantics for Normal Logic Programs (NLPs) based upon and inspired by the previous well-know Argumentation work of Phan Minh Dung on Admissible Arguments and Preferred Extensions [6].

After introducing in [14] and [12] the new Revised Stable Models semantics for NLPs (whose complexity has been studied in [11]) further work using the *Reductio ad Absurdum* (RAA) principle has been developed, namely the Revised Well-Founded Semantics [15]. Considering an argument-based view of NLPs, we define a new semantics which inherits the RAA principle studied in [14,12] and apply it to argumentation. Indeed, NLPs can be viewed as a collection of argumentative statements (rules) based on arguments (default negated literals) [2,6,4,7].

We start by presenting the general Motivation of this paper and, after introducing some needed Background Notation and Definitions, the more detailed problem description. We proceed by setting forth our proposal — the Approved Models Semantics — and show how it extends previous known results.

The Approved Models (AMs) Semantics enjoys several useful properties such as (1) Existence of a 2-valued Model for every Normal Logic Program; (2) Relevancy, which allows for the development of purely top-down query-driven proof procedures; and (3) Cumulativity. Our approach, in accordance with the previous work on Revision Complete Scenarios [13] and taking an argumentation point-of-view, allows one to obtain

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 454–468, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the Stable Models (SMs) as a special case with the advantage that every NLP has a model, a property not enjoyed by the SMs Semantics.

Indeed, we show that the Approved Models Semantics is a conservative extension to the Stable Models Semantics — in the sense that every Stable Model of a Normal Logic Program is also an Approved Model. This allows us to prove that, for certain specific types of NLPs, the Approved Models Semantics and the Stable Models Semantics coincide; and, therefore, for those NLPs the Stable Models Semantics also enjoys guarantee of Existence of a Model, Relevancy and Cumulativity. These important and useful results about the Stable Models Semantics are in line with those studied before for the Revised Stable Models Semantics [12,14].

In the quest for finding an Approved Model one can guess it and check its compliance with the properties that characterize Approved Models. Using an innovative alternative approach, suiting the purposes of a Collaborative Argumentation setting, we can start with an arbitrary set of hypotheses (default negated literals), which can be the result of joining together several different alternative sets of hypotheses, calculate its consequences, and make revisions to the initial assumptions if necessary (when inconsistencies between the hypotheses and the consequences arise) in order to achieve positive minimality respecting stratification, 2-valued Completeness and Consistency, whilst also working toward guaranteeing that the set of negative hypotheses includes a Preferred Extension.

The set of negative hypotheses is made conflict-free, admissible, and maximal; thus including a Preferred Extension [6]). This set is then 2-valued consistently completed with the remaining atoms ensuring the whole 2-valued complete model is approvable (a notion we will present formally in the sequel).

Finally, integrity constraints are introduced to identify undesired models, whilst permitting these to be produced nevertheless. Conclusions finish the paper.

1.1 Motivation

Ever since the beginning of Logic Programming the scientific community has formally defined, in several ways, the meaning or semantics of a Logic Program. Several were defined, some 2-valued, some 3-valued, and even multi-valued semantics. The current standard 2-valued semantics for NLPs — the Stable Models Semantics [9] — has been around for almost 20 years now, and it is generally accepted as the *de facto* standard for NLPs. This thoroughly studied semantics, however, lacks some important properties among which the guarantee of Existence of a Model for every NLP.

In [12] we defined a 2-valued semantics — the Revised Stable Models — which extends the Stable Models Semantics, guarantees Existence of a Model for every Normal Logic Program, enjoys Relevancy (allowing for top-down query-driven proof-procedu res to be built) and Cumulativity. Its main drawback is definitely that its definition is hard to grasp and understand.

Aiming to find a general perspective to seamlessly unify the Stable Models Semantics and the Revised Stable Models Semantics in a clear way, we drew our attention to Argumentation as a means to achieve it. This is the main motivation of the work we present in this paper: by taking the Argumentation perspective we intend to show methods of identifying and finding a 2-valued complete Model for any NLP.

In [8], François Fages proved that a NLP might have no Stable Models iff it contains Odd Loops Over Negation (OLONs)[1] and/or Infinite Chains Over Negation (ICONs)[2]. Both these notions of OLON and ICON were formally and thoroughly studied in [14,12] and the original Revised Stable Models Semantics was defined having also the OLONs and ICONs in mind.

For self-containment and to clarify the context, we now present a few informal motivating examples of OLONs and ICONs and how we solve them, so that every NLP has a semantics, employing a reasoning by contradiction argument. This kind of *reductio ad absurdum* reasoning takes place only when it is absolutely necessary in order to ensure 2-valued completeness of the resulting model, this "last resort" flavor of the RAA coming from the requirement of keeping the semantics maximally skeptical.

Example 1. **An invasion problem.** Some political leader thinks that "If Iran will have Weapons of Mass Destruction then we intend to invade Iran", also "If we do not intend to invade then surely they will have Weapons of Mass Destruction", rendered as the following OLON (where "not " denotes default negation)

$$\begin{array}{ll} intend\_to\_invade & \leftarrow iran\_will\_have\_WMD \\ iran\_will\_have\_WMD & \leftarrow not\ intend\_to\_invade \end{array}$$

The literal involved in the OLON is $intend\_to\_invade$ which is the literal that depends on itself through and odd number of default negations. The literal $iran\_will\_have\_WMD$ is just an in between stepping-stone for the OLON.

If we assume that "we do not intend to invade Iran" then, according to this program we will conclude that "Iran will have Weapons of Mass Destruction" and "we intend to invade Iran". These conclusions, in particular "we intend to invade Iran", contradict the initial hypothesis "we do not intend to invade Iran". So, reasoning by *Reductio ad Absurdum* in a 2-valued setting, we should "intend to invade Iran" in the first place.

This example gives a hint on how we resolve inconsistent arguments in the remainder of the paper, further exemplified below.

Example 2. **A going-out problem.** John likes Mary a lot so he asked her out: he said "We could go to the movies". Mary is more of a sports girl, so she replies "Either that, or we could go to the swimming pool". "Now, that's an interesting idea", John thought. The problem is that John cannot swim because he hasn't started learning to. He now thinks "Well, if I'm going to the swimming pool with Mary, and I haven't learned how to swim, I might risk drowning! And if I'm risking drowning then I really should start learning to swim". Here is the Normal Logic Program corresponding to these sentences:

$$\begin{array}{ll} start\_learning\_to\_swim & \leftarrow risk\_drowning \\ risk\_drowning & \leftarrow go\_to\_pool, not\ start\_learning\_to\_swim \\ go\_to\_pool & \leftarrow not\ go\_to\_movies \\ go\_to\_movies & \leftarrow not\ go\_to\_pool \end{array}$$

---

[1] Intuitively, an OLON is just a cycle in the dependency-graph induced by the NLP (an atom a depends on itself) where there is an odd number of default negations along the cycle.

[2] An example of an ICON with the explanation of the concept is presented below.

If John is not willing to go to the swimming pool — assuming $not\ go\_to\_pool$ — he just concludes go_to_movies and maybe he can convince Mary to join him. On the other hand, if the possibility of having a nice swim with Mary is more tempting, John assumes he is not going to the movies $not\ go\_to\_movies$ and therefore he concludes go_to_pool. In this case, since John does not know how to swim he could also assume $not\ start\_learning\_to\_swim$. But since John is going to the swimming pool, he concludes $risk\_drowning$. And because of $risk\_drowning$ he also concludes $start\_learning\_to\_swim$. That is, he must give up the hypothesis of $not\ start\_learning\_to\_swim$ in favour of $start\_learning\_to\_swim$ because he wants to go to the pool with Mary. As a nice side-effect he no longer risks drowning.

Example 3. **Middle region politics.** In a Middle Region two factions are at odds. One believes that if terrorism does not stop then oppression will do it and hence become unnecessary.

$oppression \leftarrow not\ end\_of\_terrorism \quad end\_of\_terrorism \leftarrow oppression$

The other faction believes that if oppression does not stop then terrorism will do it and hence become unnecessary.

$terrorism \leftarrow not\ end\_of\_oppression \quad end\_of\_oppression \leftarrow terrorism$

According to these rules, if we assume that $not\ end\_of\_terrorism$ we conclude that there is $oppression$ which in turn will cause the $end\_of\_terrorism$. So, the $end\_of\_terrorism$ should be true in the first place instead of $not\ end\_of\_terrorism$. The same happens with $end\_of\_oppression$. In spite of the peaceful resulting solution we propose, $\{end\_of\_oppression, end\_of\_terrorism\}$, there is no Stable Model for this program.

Example 4. **An infinite chain over negation.** The classical example of an ICON was first presented in [8] and is

$p(X) \leftarrow p(s(X)) \qquad p(X) \leftarrow not\ p(s(X))$

with a single constant 0 (zero), and so its ground version is

$$\begin{array}{ll} p(0) \leftarrow p(s(0)) & p(0) \leftarrow not\ p(s(0)) \\ p(s(0)) \leftarrow p(s(s(0))) & p(s(0)) \leftarrow not\ p(s(s(0))) \\ \vdots & \vdots \end{array}$$

As we can see, the ground version has an Infinitely long descending Chain in the dependency graph for every literal $p(X)$ Over default Negation. In [14] the author proved that every possible ICON can be reduced to this canonical case, i.e., other ICONs may exist with more rules besides these, but the main structure (relevant for its properties) of every ICON relies on the same as the one presented here.

According to the Stable Models semantics this program has no models. But there is one Approved Model where every $p(X)$ is true. To see this, by *reductio ad absurdum*, assume that $p(X)$ was false for some X; then the two bodies of each clause above would have to be false, meaning that $p(s(X))$ would be true by the second one; but then, by the first one, $p(X)$ would be true as well, thereby contradicting the default assumption. Hence, by *Reductio ad Absurdum* reasoning, $p(X)$ must be true, for arbitrary X.

1.2 Background Notation and Definitions

Definition 1. ***Logic Rule.*** *A Logic Rule r has the general form*
$L \leftarrow B_1, B_2, \ldots, B_n, not\ C_1, not\ C_2, \ldots, not\ C_m$
where L is an atom h, the B_i and C_j are atoms.

We call L the head of the rule — also denoted by $head(r)$. And $body(r)$ denotes the set $\{B_1, B_2, \ldots, B_n, not\ C_1, not\ C_2, \ldots, not\ C_m\}$ of all the literals in the body of r. Throughout this paper we will use 'not ' to denote default negation.

When the body of the rule is empty, we say the head of rule is a fact and we write the rule just as h or $not\ h$. Since we are considering only Normal Logic Programs facts will never be of the form $not\ h$.

Definition 2. ***Logic Program.*** *A Logic Program (LP for short) P is a (possibly infinite) set of ground Logic Rules of the form in definition 1.*

In this paper we focus solely on Normal LPs (NLPs), those whose heads of rules are positive literals, i.e., simple atoms; and there is just default negation in the bodies of the rules. Hence, when we write just "program" or "logic program" we mean a NLP.

We use the notation $Atoms(P)$ to denote the set of all atoms of P; and $not\ S$ — where S is a set of literals (both positive and/or default negated) — to denote the set resulting from default negating every literal of S. Note that the default negation of an already default negated literal $not\ a$ is just the positive literal, i.e., $not\ not\ a \equiv a$.

Definition 3. ***2-valued Interpretation.*** *A 2-valued interpretation $I = I^+ \cup I^-$ of P is a set of literals, both positive $I^+ \subseteq Atoms(P)$ and negative $I^- \subseteq not\ Atoms(P)$, such that*

- *$I^+ \cap not\ I^- = \emptyset$ — it is consistent, i.e., there is no atom a of $Atoms(P)$ such that both a and $not\ a$ are in I*
- *$I^+ \cup not\ I^- = Atoms(P)$ — it is 2-valued complete, i.e., there is no atom a of $Atoms(P)$ such that both a and $not\ a$ are not in I*

Definition 4. ***$\vdash$ operator.*** *Let P be a NLP and I a 2-valued interpretation of P. P' is the Horn theory obtained from P by replacing every default literal of the form $not\ L$ in P by the atom not_L. I' is likewise obtained from I using the same replacement rule. By definition, $P' \cup I'$ is a Horn theory, and so it has a least model $M' = least(P' \cup I')$. We define $\vdash$ in the following way, where a is any atom of P:*
$P \cup I \vdash a$ *iff* $a \in M'$ $\qquad$ $P \cup I \vdash not\ a$ *iff* $not\_a \in M'$

In the sequel, we will write $M = least(P \cup I)$ to mean $M = \{L : P \cup I \vdash L\}$, where L is any positive (a) or negative $(not\ a)$ literal derived from I in P. M corresponds thus to the result of applying the inverse substitution of literals not_a by $not\ a$ in M'. We will also sometimes refer to $least(P \cup I)$ as the consequences of I in P.

Definition 5. ***Internally Consistent 2-valued Interpretation.*** *Let P be a NLP and I a 2-valued interpretation I of P. I is Internally Consistent in P iff $least(P \cup I) \subseteq I$.*

Since we are considering only NLPs, where there are no negations in the heads of rules, we can never derive a negative literal by applications of the $least$ operator. Hence, the literals in $least(P \cup I)$ — besides including all the literals in I — include only positive literals (some heads of rules of P). By requiring $least(P \cup I) \subseteq I$ we ensure that all the literals in $least(P \cup I)$ do not contradict any of the literals in I^-; knowing beforehand that literals in I^+ cannot ever be contradicted by $least(P \cup I)$ because P is a Normal Logic Program.

In fact, since I is 2-valued complete, if $least(P \cup I) \subseteq I$ then necessarily $least(P \cup I) = I$. Moreover, in [10], the authors prove that if $least(P \cup I^-) = I$, then I is a Stable Model of P and vice-versa, with the appropriate language translation. We also write $\Gamma_P(I)$ as a shorthand notation for $least(P \cup M^-) \setminus M^-$, i.e., $\Gamma_P(M)$ is just the positive part of $least(P \cup M^-)$. Hence, M **is a Stable Model of** P **iff** $\Gamma_P(M) = M$. For any interpretation I, we consider $\Gamma_P^0(I) = I$, and $\Gamma_P^{n+1}(I) = \Gamma_P(\Gamma_P^n(I))$.

Definition 6. ***Approvable Interpretation.*** *Let P be a NLP and I an Internally Consistent 2-valued Interpretation of P. We say I is an Approvable Interpretation of P iff I is such that I^- is set maximal. I.e., there is no other Internally Consistent 2-valued Interpretation I' of P such that $I'^- \supset I^-$.*

2 The Approved Models Semantics for Normal Logic Programs

The Approved Models Semantics for Normal Logic Programs gets its inspiration from the Argumentation perspective and also from our previous works [14], [12], and [13].

In [6], the author shows that preferred maximal scenarios (with maximum default negated literals — the hypotheses) are always guaranteed to exist for NLPs; and that when these yield 2-valued complete (total), consistent, admissible scenarios, they coincide with the Stable Models of the program. However, preferred maximal scenarios are, in general, 3-valued. The problem we address now is how to define 2-valued complete models based on preferred maximal scenarios. In this paper we take a step further from what we achieved in [6], extending its results. We do so by completing a preferred set of hypotheses rendering it approvable (as presented above in definition 6), ensuring whole model consistency and 2-valued completeness.

The resulting semantics thus defined, dubbed Approved Models Semantics, is a conservative extension to the widely known Stable Models semantics [9] in the sense that every Stable Model is also an Approved Model. The Approved Models are guaranteed to exist for every NLP, whereas Stable Models are not. The concrete examples above show how NLPs with no Stable Models can usefully model knowledge, as well as produce additional models, as in Example 2. Moreover, this guarantee is crucial in program composition (say, from knowledge originating in divers sources) so that the result has a semantics. It is important too to warrant the existence of semantics after external updating, or in Stable Models based self-updating [1].

Moreover, the Approved Models Semantics enjoys the Relevancy property which allows for the development of purely top-down, program call-graph based, query driven methods to determine whether a literal belongs to some model or other. These methods can thus simply return a partial model, guaranteed extendable to a complete one, there being no need to compute all models or even to complete models in order to answer a query. Relevancy is crucial too for modeling abduction, it being query driven. Finally, the Approved Models Semantics enjoys Cumulativity, so lemmas may be stored and reused.

Before presenting the formal definition of an Approved Model we give a general intuitive idea to help the reader grasp the concept. For the formal definition of Approved Models Semantics we will also need some preliminary auxiliary definitions.

2.1 Intuition

In [4] the authors prove that every SM of a NLP corresponds to a stable set of hypotheses, and these correspond in turn to a 2-valued complete, consistent, admissible scenario. In order to guarantee the Existence of a 2-valued total Model for every NLP we allow in all the negative hypotheses which are self-conflict-free and admissible ([6]), and include a Preferred Extension [6]. The extra negative hypotheses approved beyond a Preferred Extension are criteriously allowed (only the approvable ones) to ensure that the resulting 2-valued completion is consistent.

2.2 Underlying Notions

Most of the ideas and notions underlying the work we now present come from the Argumentation field — mainly from the foundational work of Phan Minh Dung in [6] — plus the *Reductio ad Absurdum* reasoning studied in [14], [12], and [13]. For self-containment we now present the basic notions of argument (or set of hypotheses), attack, conflict-free set of arguments, acceptable argument, and admissible set of arguments (all original from [6]).

Definition 7. ***Argument.*** *In [6] the author presents an argument as*

> *"an abstract entity whose role is solely determined by its relations to other arguments. No special attention is paid to the internal structure of the arguments."*

In this paper, since we are focusing on NLPs, we will pay attention to the internal structure of an argument by considering an argument *(or set of hypotheses) as a set S of default negated literals of a NLP P, i.e., $S \subseteq not\ Atoms(P)$. Thus, a* simple argument *$not\ a$ of S (or simple hypothesis) is just an element of an* argument *S.*

Using these notions of argument *and* simple argument *we can define the set of* Arguments *of a NLP P — $Arguments(P)$ — as the set of all* arguments *of P, i.e., the set of all subsets of $not\ Atoms(P)$.*

Definition 8. ***Attack* — Argument** B ***Attacks* simple argument** $not\ a$ ***in*** P ***[6].*** *In [6] Dung does not specify what the* attacks *relationship concretely is, this way ensuring maximal generality of the argumentation framework. In our present work, since we are considering NLPs, and* arguments *as sets of default negated literals (each a* simple argument*), the* attacks *relationship corresponds to deriving a positive literal contradicting one* simple argument *of the attacked* argument.

Formally, if P is a NLP, $B \in Arguments(P)$, and $not\ a \in not\ Atoms(P)$, we say B attacks $not\ a$ in P iff $P \cup B \vdash a$, i.e., $a \in least(P \cup B)$. For simplicity, we just write $attacks_P(B, not\ a)$.

Abusing this notation, we also write $attacks_P(B, A)$*, where both* A *and* B *are* Arguments *of* P*, to mean that the* Argument B *attacks* Argument A *in* P*. This means that* $\exists_{not\ a \in A} attacks_P(B, not\ a)$*.*

Definition 9. ***Conflict-free argument*** A ***[6].*** *An* argument A *of* P *is said to be conflict-free iff there is no* simple argument $not\ a$ *in* A *such that* $attacks_P(A, not\ a)$*. I.e.,* A *does not attack itself.*

Definition 10. ***Acceptable argument [6].*** *An argument* $A \in AR$ *— where* AR *is a set of* arguments *— of* P *is said to be Acceptable with respect to a set* S *of arguments iff for each argument* $B \in AR$*: if* B *attacks* A *then* B *is attacked by* S*.*

Definition 11. ***Admissible Argument*** A ***of*** P***[6].*** *A conflict-free* argument A *is* admissible *in* P *iff* A *is acceptable with respect to* $Arguments(P)$*. Intuitively,* A *is* admissible *if it counter-attacks every argument* B *in* $Arguments(P)$ *attacking* A*. Formally,*

$$\forall_{B \in Arguments(P)} \forall_{not\ a \in A} attacks_P(B, not\ a) \Rightarrow \exists_{not\ b \in B} attacks_P(A, not\ b)$$

This definition corresponds to the definition of Admissible Set presented in [6]. Notice that it is not required an attacking set B *to be consistent with its consequences, i.e.,* $least(P \cup B)$ *is not required to be consistent.*

Definition 12. ***Preferred Extension[6].*** *A Preferred Extension is a maximal (with respect to set inclusion) admissible Argument of* P*.*

2.3 Definition of the Approved Models Semantics

Definition 13. ***Approved Models.*** *Let* P *be a NLP and* $M = M^+ \cup M^-$ *a 2-valued interpretation of* P*. We say* M *is an Approved Model of* P *iff:*

- M *is an Approvable Interpretation of* P*, and*
- M^- *contains a Preferred Extension of* P

We use the notation $AM_P(M)$ *to mean that* M *is an Approved Model of* P*. The Approved Models Semantics of an NLP* P *is just the intersection of the positive parts of all its Approved Models. We write*

- $AM^+(P)$ *to denote the set of atoms of* P *considered* true *by the Approved Models Semantics. This corresponds to the Approved Models Semantics of* P*, i.e.,* $AM^+(P) = \bigcap_{AM_P(M)} M^+$
- $AM^-(P)$ *to denote the set of atoms of* P *considered* false *by the Approved Models Semantics. I.e.,* $AM^-(P) = not\ \bigcap_{AM_P(M)} M^-$
- $AM^u(P)$ *to denote the set of atoms of* P *considered* undefined *by the Approved Models Semantics. I.e.,* $AM^u(P) = Atoms(P) - \big(AM^+(P) \cup AM^-(P)\big)$

2.4 Properties of the Approved Models Semantics

Stable Models Extension

Theorem 1. ***Every Stable Model is also an Approved Model.*** *If* P *is a NLP, and* SM *a Stable Model of* P*, then* $M = SM \cup M^-$*, where* $M^- = not\ (Atoms(P) \setminus SM)$*, is an Approved Model of* P*.*

Proof. Let P be a NLP, and SM a Stable Model of P. Since every Stable Model SM is a Minimal Herbrand Model of P it means that $not\ (Atoms(P) \setminus SM)$ is Maximal. Therefore, we conclude that $SM \cup not\ (Atoms(P) \setminus SM)$ is an Approvable Interpretation of P.

Moreover, since SM is a Stable Model of P we know, by [6], that $not\ (Atoms(P) \setminus SM)$ is Admissible in P. Since SM is a SM of P, it is minimal and therefore $not\ (Atoms(P) \setminus SM)$ is maximal. Hence, $not\ (Atoms(P) \setminus SM)$ is a Preferred Extension and $SM \cup not\ (Atoms(P) \setminus SM)$ is an Approved Model of P. □

Existence

Theorem 2. ***Every Normal Logic Program has at least one Approved Model.***

Proof. Let P be a NLP. In [6] the author proves that every NLP has at least one preferred extension, and that preferred extensions are Maximal Admissible Arguments. Consider $M'^{-} \in Arguments(P)$ one such preferred extension. It is always possible to construct one $M^{-} = M'^{-} \cup S^{-}$, where $S^{-} \in Arguments(P)$, such that $M = M^{-} \cup M^{+}$, and $M^{+} = Atoms(P) \setminus not\ M^{-}$, is an Approvable Interpretation. The limit case would be $S^{-} = \emptyset$ and $M^{+} = Atoms(P) \setminus (not\ M^{-})$.

The intuitive idea is that no more *simple arguments* can be added to a preferred extension M'^{-} because, for at least one $not\ s \in S^{-}$, $M'^{-} \cup \{not\ s\}$ would not be an Acceptable Argument in the sense of definition 10, so a corresponding positive atom s can be added instead, plus a new maximal addition of default atoms as a result of adding the positive one, for any such positive atom.

The addition of such positive atoms corresponds to resolving an inconsistency that would follow from adding the corresponding default negation literal, and computing the semantics according to definition 4. It is thus a form of reasoning by contradiction and the positive atom can be justified as a positive hypothesis introduced by that reasoning. Indeed, the reason a preferred extension is not complete is, according to Fage's result [8], because there may be some ICONs or OLONs that prevent turning it into an SM. □

In [13] we showed iterative and incremental methods for calculating the Revision Complete Scenarios. The same methods can be used to calculate the Approved Models for these coincide with the models yielded by the Revision Complete Scenarios. In a nutshell, to calculate a Revision Complete Scenario we start with a set of hypotheses which initially is the set of all default negated literals of the program $H^{-} = not\ Atoms(P)$. Next, we compute the least model of the program considering those hypotheses, i.e, $least(P \cup H^{-})$. Some inconsistencies (pairs $l, not\ l$) will be present in $least(P \cup H^{-})$: we select one such l and remove $not\ l$ from H^{-} thus reducing it. We repeat this process until there are no more inconsistencies in $least(P \cup H^{-})$. At that point we just add the remaining positive atoms needed in order to ensure 2-valued completeness and consistency. For greater clarity we show how this process develops with Example 2:

Example 5. **Iteratively calculating an Approved Model.** Recall the example 2 with John, Mary, going to the swimming pool or to the movies.

$$\begin{array}{ll} start\_learning\_to\_swim & \leftarrow risk\_drowning \\ risk\_drowning & \leftarrow go\_to\_pool, not\ start\_learning\_to\_swim \\ go\_to\_pool & \leftarrow not\ go\_to\_movies \\ go\_to\_movies & \leftarrow not\ go\_to\_pool \end{array}$$

For simplicity, we will use abbreviations for the literals. To iteratively calculate the Approved Models of this program let us do as explained above. First, we start with $H^- = not\ Atoms(P)$, i.e., $H^- = \{not\ sls, not\ rd, not\ gp, not\ gm\}$. Now we compute $least(P \cup H^-) = \{not\ sls, sls, not\ rd, rd, not\ gp, gp, not\ gm, gm\}$. There are several inconsistencies in $least(P \cup H^-)$, we can choose any one default negated literal participating in one inconsistency and remove it from H^-. Take notice that, since we are removing one *not* l from H^- and in the end we want a 2-valued complete model, we will necessarily have l as *true* in that model — we say l was revised from *false* (*not* l) to *true* (l). Let us choose to revise $not\ gp$. H^- thus becomes $\{not\ sls, not\ rd, not\ gm\}$. We recompute $least(P \cup H^-) = \{not\ sls, not\ rd, not\ gm, gp, rd, sls\}$. We now repeat the process of choosing any one assumption *not* l participating in an inconsistency to revise it, and update H^- accordingly. We now revise $not\ sls$. H^- is now $H^- = \{not\ rd, not\ gm\}$. The corresponding Approved Model is $M = \{not\ rd, not\ gm, sls, gp\}$ which is not a Stable Model. The only other Approved Model is
$M' = \{not\ rd, gm, not\ sls, not\ gp\}$ which is also a Stable Model.

Relevancy

Definition 14. $Rel_P(a)$ ***— Relevant part of NLP*** P ***for the positive literal*** a***.***
The Relevant part of a NLP P *for some positive literal* a*,* $Rel_P(a)$ *is defined as*
$Rel_P(a) =$
$\{r \in P : head(r) = a\} \cup \bigcup_{x:r\in P \wedge head(r)=a \wedge (x \in body(r) \vee not\ x \in body(r))} Rel_P(x)$

Intuitively, the relevant part of a program for some atom a is just the set of rules with head a and the rules relevant for each atom in the body of the rules for a. I.e., the relevant part is the set of rules in the call-graph for a.

Definition 15. ***Relevant Semantics.*** *This definition is taken from [5]. A Semantics* Sem *for NLPs is said to be Relevant iff for every program P and positive literal* a *of P* $a \in Sem(P) \Leftrightarrow a \in Sem(Rel_P(a))$

Theorem 3. ***The Approved Models Semantics is Relevant.***
$a \in AMS(P) \Leftrightarrow AMS(Rel_P(a))$

Proof. Let P be a NLP. By definition (see above), the semantics of P according to the Approved Models Semantics is the intersection of the positive parts of all the Approved Models of P. I.e., $AMS(P) = AM^+(P) = \bigcap_{AM_P(M)} M^+$

"⇒" Let a be an atom of P, i.e., a positive literal of P, belonging to $AMS(P)$. Then, necessarily $a \in M^+$ for any $AM_P(M)$. We know by definition that $a \in M^+$ iff $a \in least(P \cup M)$, and since the $least$ operator just computes the set of heads of rules whose body is satisfied — in this case by the interpretation M — a is in M because one of the rules with a as head, i.e. one of the Relevant Rules for a, has its body satisfied by M. Then, in this case, clearly, $a \in AMS(Rel_P(a))$.

"$\Leftarrow$" Consider now that $a \in AMS(Rel_P(a))$. For the same reason — knowing that $a \in M$ iff $a \in least(P \cup M)$ — it is easy to see that any rule $r \in P$ we might add to $Rel_P(a)$ which does not change $Rel_P(a)$, i.e. r is not Relevant for a in P, will not affect the fact that $a \in AMS(Rel_P(a))$ and hence that $a \in AMS(Rel_P(a) \cup \{r\})$. Therefore we can conclude that $a \in AMS(P)$, where $P \supseteq Rel_P(a)$. □

Cumulativity

Definition 16. ***Cumulative Semantics.*** *This definition is taken from [5]. Let P be an NLP, and Sem a semantics for NLPs. We say the semantics Sem is Cumulative (or that it enjoys Cumulativity) iff the Semantics of P remains unchanged when any atom considered* true *in the Semantics is added to P as a fact*

$$Cumulative(Sem) \Leftrightarrow \forall_P \forall_{a,b} a \in Sem(P) \wedge b \in Sem(P) \Rightarrow a \in Sem(P \cup \{b\})$$

Theorem 4. ***The Approved Models Semantics is Cumulative.***

$$\forall_P \forall_{a,b} a \in AMS(P) \wedge b \in AMS(P) \Rightarrow a \in AMS(P \cup \{b\})$$

Proof. Let P be a NLP. $a \in AMS(P) \wedge b \in AMS(P)$, i.e., both a and b are in all Approved Models of P. By definition of Approved Model we know that every Approved Model M of P satisfies $M = least(P \cup M)$, and since both $a \in M$ and $b \in M$ we know that $P \cup M = P \cup M \cup \{b\}$. Therefore, $least(P \cup M) = least(P \cup M \cup \{b\}) = M \ni a$. Since this hold for every literal a and b, and for every Approved Model M, it also holds for their intersection. □

2.5 Further Requirements: Respecting the Well-Founded Model

Although the Approved Models semantics enjoys already some nice properties as seen above, it still lacks a feature important for practical implementations and applicability: the respect for the program's natural stratification. This is the motivation for the further requirements we present next. These, when enforced, guarantee that respect which was already guaranteed by the Revised Stable Models semantics and this is the reason why we relate the Revised Stable Models semantics and the Approved Models semantics.

Definition 17. ***Well-Founded Model of a Normal Logic Program P.*** *According to [3] the* true *atoms of the Well-Founded Model of P (the irrefutably* true *atoms of P) can be calculated as the Least Fixed Point of the Gelfond-Lifschitz operator iterated twice — Γ_P^2. Formally, $WFM^+(P) = lfp(\Gamma_P^2 \uparrow^\omega (\emptyset))$. Also according to [3] the* true *or* undefined *literals of a program can be calculated as the consequences of assuming as true the* true *atoms of the Well-Founded Model. Formally, $WFM^{+u}(P) = \Gamma_P(WFM^+(P)) \supseteq WFM^+(P)$. So, the just* undefined *literals of P are $WFM^u(P) = WFM^{+u}(P) \setminus WFM^+(P)$. And finally, the* false *atoms of the Well-Founded Model of a Program P (the irrefutably* false *atoms of P) are, necessarily, the atoms of P which are neither* true *nor* undefined. *Formally, $WFM^-(P) = Atoms(P) \setminus WFM^{+u}(P)$.*

Definition 18. ***Atom a respects the subset S of Atoms of P.*** *Let P be a NLP, a an atom of P, and $S \subseteq Atoms(P)$ a subset of Atoms of P. We say a respects S in P iff a remains* true *or* undefined *in the Well-Founded Model of P when the context S is added to P as facts. Formally, $Respects_P(a, S) \Leftrightarrow a \in WFM^{+u}(P \cup S)$.*

Definition 19. ***Undefined part of the Model M — M^u.*** *Let P be a NLP, and M an Approvable Interpretation of P. We write M^u to denote the subset of M^+ of which all atoms are* undefined *in the Well-Founded Model of P. Formally, $M^u = M^+ \cap WFM^u(P)$. These are the only atoms that might come under* reductio ad absurdum.

The concept of $US_P(M, a)$, presented next, is based on the core idea of finding the subset of atoms of M^u which influence the truth value of a. To find such set we recur to the Well-Founded Model calculus because, as it is based upon the Γ_P^2 monotonic operator, it draws conclusions from premises according to the implication-based rules of the program. Therefore, when an atom a is concluded *true* by the Well-Founded Model we know that its truth value was calculated using only the part of the program in the call-graph for the atom a. We show this and the following concepts in example 6.

Definition 20. ***Undefined Support of Model M for atom a in P — $US_P(M, a)$.*** *Let P be an NLP, M an Approvable Interpretation of P and a an atom of M^+. We write $US_P(M, a)$ to denote the subset of atoms of M^u which support the truth value of a. Formally,*

$US_P(M, a) = \{b \in M^u : \exists_{S \subseteq Atoms(P)} \big(a \in WFM^u(P \cup S) \wedge a \notin WFM^u(P \cup S \cup \{b\})\big)\}$

By this definition we see b has critical influence on the undefinedness of the truth value of a under some context S: when b is not added as a fact a remains undefined*, when b is added as a fact a becomes defined (*true *or* false*, but no longer* undefined*).*

Next, we present the concept of $USS_P(M, a)$. The idea behind it is a stricter form of the previous one. We want $USS_P(M, a)$ to include all the atoms in $US_P(M, a)$ except for those which are in a loop (a call-graph loop) with a. So, roughly, the atoms in $USS_P(M, a)$ are all in strictly lower strata if we considered a kind of stratification where all the atoms in a loop belong to the same strata. To obtain such a set we remove from $US_P(M, a)$ all the atoms b for which a is an element of $US_P(M, b)$. If a depends on b and b depends on a then there is a loop between a and b.

Definition 21. ***Undefined Strict Support of Model M for atom a in P — $USSup_P(M, a)$.*** *Let P be an NLP, M an Approvable Interpretation of P and a an atom of M^+. We write $USS_P(M, a)$ to denote the subset of atoms of M^u which support the truth value of a and whose truth value is in turn not influenced by a. Formally,*

$USS_P(M, a) = US_P(M, a) \setminus \{b \in US_P(M, a) : a \in US_P(M, b)\}$

In a nutshell, the atoms in $USS_P(M, a)$ are guaranteed not to be in a loop with a — as happens with b in, for example, a program like $a \leftarrow not\ b \quad b \leftarrow not\ a$.

We wish to assume the literals in I to be *true*, hence we add them to P as facts obtaining $P \cup I$. Now we want to calculate what is necessarily *true* and necessarily *false* in that resulting program $P \cup I$. For that we calculate the Well-Founded Model of $P \cup I$, namely its Positive and Negative parts. We now say M Respects I iff all the atoms in the Positive part of the Well-Founded Model of $P \cup I$ are also considered *true* by M, and all the atoms in the Negative part of the Well-Founded Model of $P \cup I$ are also considered *false* by M. So, there are no contradictions between M and the Well-Founded Model of $P \cup I$. This is the rationale behind the next definition.

Definition 22. M ***Respects Interpretation*** I ***in*** P **—** $Respects_P(M, I)$**.** *Let P be an NLP, M an Approvable Interpretation of P, and I an arbitrary Interpretation in P. We say M Respects the Interpretation I in P iff the Positive Part of the Well-Founded Model of $P \cup I$ is totally contained in M^+, and the Negative Part of the Well-Founded Model of $P \cup I$ is totally contained in M^-. Formally,*

$Respects_P(M, I) \Leftrightarrow \left(M^+ \supseteq WFM^+(P \cup I)\right) \wedge \left(M^- \supseteq not\, WFM^-(P \cup I)\right)$

Since $M^+ \cap M^- = \emptyset$ it follows trivially that if $Respects_P(M, I)$ then $M^+ \cap WFM^-(P \cup I) = \emptyset$ and $(not\ M^-) \cap WFM^+(P \cup I) = \emptyset$

Example 6. **Respect and Dir-Respect for the** $USS_P(M, a)$**.** Consider NLP $P =$

$i \leftarrow not\ k \quad t \leftarrow a, b \quad a \leftarrow not\ b$
$k \leftarrow not\ t \qquad\qquad b \leftarrow not\ a$

There are four Approvable Interpretations for P: $M_1 = \{a, k, not\ b, not\ t, not\ i\}$, $M_2 = \{b, k, not\ a, not\ t, not\ i\}$, $M_3 = \{a, t, i, not\ b, not\ k\}$, and $M_4 = \{b, t, i, not\ a, not\ k\}$. Models M_1 and M_2 are symmetrical on a and b, and the same happens with M_3 and M_4. So, we will just examine M_1 and M_3 without any loss of information.

Let us calculate the $US_P(M_1, a)$. $M_1^u = \{a, k\} = M_1^+$. If we add k as a fact to P, a will remain *undefined*; however, when we add a as a fact to P, k becomes defined, in this case *true*. Since there are no other possible subsets of M_1^u to consider we have $US_P(M_1, a) = \{a\}$ and $US_P(M_1, k) = \{a\}$. It is now simple to calculate $USS_P(M_1, a) = \emptyset$ and $USS_P(M_1, k) = \{a\}$. Notice that the check for $Respects_P(M_1, USS_P(M_1, a))$ is trivial because $USS_P(M_1, a) = \emptyset$. Also, $Respects_P(M_1, USS_P(M_1, k))$ holds because, as we have already seen, adding $\{a\} = USS_P(M_1, k)$ to P as a fact renders a *true*. The case with M_3 is quite different. $M_3^u = \{a, t, i\}$. $US_P(M_3, a) = \{a\}$, $US_P(M_3, t) = \{a\}$, and $US_P(M_3, i) = \{a, t\}$. $USS_P(M_3, a) = \emptyset$, $USS_P(M_3, t) = \{a\}$, and $USS_P(M_3, i) = \{a, t\}$. Clearly, $Respects_P(M_3, USS_P(M_3, a))$ holds. However, $Respects_P(M_3, USS_P(M_3, t))$ does not hold. It suffices to check that adding $\{a\} \subseteq USS_P(M_3, t)$ to P as a fact, t, which was considered *true* in M_3, becomes now *false* in the $WFM(P \cup \{a\})$.

Definition 23. ***Strict Model*** M ***of a program*** P**.** *Let P be a NLP. An Approvable Interpretation M is Strict in P iff for any atom a in M^+, M Respects its own Undefined Strict Support, i.e. $Strict_P(M) \Leftrightarrow \forall_{a \in M^+} Respects_P(M, USS_P(M, a))$*

Conjecture 1. M^+ is a Revised Stable Model [12] of a NLP P iff M is an Approved Model and a Strict Interpretation of P.

Conjecture 2. $M = M^+ \cup M^-$ is an Approved Model of a NLP P iff there is some Revision Complete Scenario H of P, where $H = H^+ \cup H^-$ and $H^+ \subseteq M^+$ and $H^- = M^-$.

2.6 Integrity Constraints

Example 7. **Middle Region Politics Revisited.** Recall the Example 3 presented earlier. We are now going to add extra complexity to it.

We already know the two factions which are at odds and their thinking.

$$
\begin{array}{ll}
oppression \leftarrow not\ end\_of\_terrorism & end\_of\_terrorism \leftarrow oppression \\
terrorism \leftarrow not\ end\_of\_oppression & end\_of\_oppression \leftarrow terrorism
\end{array}
$$

We now combine these two sets of rules with the two following Integrity Constraints (ICs) which guarantee that $oppression$ and $end\_of\_oppression$ are never simultaneously true; and the same happens with terror:

$$
\begin{array}{c}
falsum \leftarrow oppression, end\_of\_oppression \\
falsum \leftarrow terrorism, end\_of\_terrorism
\end{array}
$$

As the reader can see, we write ICs not as OLONs of length 1 as it is usual in the Stable Models community, but we write them as simple, normal rules which have as head a special literal $falsum$. This literal is "special" not because the Approved Models Semantics treats it in any differently, but just because the programmer writing the rules uses it only for the purpose of writing ICs, i.e., $falsum$ is not to be used for any other purposes in the program. So far so good, there is still a single joint set of hypotheses resulting in a consistent scenario $\{end\_of\_oppression, end\_of\_terrorism\}$. Still, there is no SM for this program. But introducing either one or both of the next two rules, makes it impossible to satisfy the ICs:

$oppression \leftarrow not\ terrorism \quad terrorism \leftarrow not\ oppression$

In this case all the Approved Models contain the atom $falsum$. There are still no Stable Models for the resulting program. The semantics we propose allows two models for this program, corresponding to the 2-valued complete consistent models, both containing $falsum$. We can discard them or examine the failure to satisfy the ICs.

When posing a query to be solved in a top-down call-graph oriented manner, if the user wants to make sure $falsum$ is not part of the answer, (s)he just needs to conjoin $not\ falsum$ to the query conjunction. When using the SMs Semantics, typically ICs are written as OLONs of length 1 (e.g. $falsum \leftarrow IC\_Body, not\ falsum$). This is done to take advantage of the characteristics of that semantics, in particular to take advantage of the fact that the SMs do not deal with OLONs (in the sense that it does not provide a semantics to them) and, therefore, eliminate the interpretations which would "activate" the OLON by rendering the $IC\_Body\ true$. Since the Approved Models Semantics gives a semantics to all NLPs it has no special features to take advantage from in order to prune undesired interpretations. The programmer just writes the ICs as depicted in the example 7 above and, if (s)he wants to, asks for models without the $falsum$ atom.

3 Conclusions

We have presented a new 2-valued semantics for Normal Logic Programs, based upon the previous work on Argumentation by Phan Minh Dung. This new semantics conservatively extends the Stable Models Semantics with the advantage of enjoying three useful properties: guarantee of model Existence for every NLP; Relevancy, allowing for top-down query-driven proof-procedures; and Cumulativity. Using our semantics, we have showed how Integrity Constraints can be written in a simpler way and dealt with in a very intuitive manner. All previously known results about the Stable Models Semantics, when it exists, and their good properties are kept intact. The work presented is based upon valuable results of years of effort the scientific community has placed in studying the Stable Models Semantics and the Argumentation framework of Dung. We take another step forward, by adding *reductio ad absurdum* to argumentation in NLPs.

References

1. Alferes, J.J., Brogi, A., Leite, J.A., Pereira, L.M.: Evolving logic programs. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, pp. 50–61. Springer, Heidelberg (2002)
2. Alferes, J.J., Pereira, L.M.: An argumentation theoretic semantics based on non-refutable falsity. In: Dix, J., et al. (eds.) NMELP, pp. 3–22. Springer, Heidelberg (1994)
3. Baral, C., Subrahmanian, V.S.: Dualities between alternative semantics for logic programming and nonmonotonic reasoning. J. Autom. Reasoning 10(3), 399–420 (1993)
4. Bondarenko, A., Dung, P.M., Kowalski, R.A., Toni, F.: An abstract, argumentation-theoretic approach to default reasoning. Artif. Intell. 93, 63–101 (1997)
5. Dix, J.: A Classification-Theory of Semantics of Normal Logic Programs: I, II. Fundamenta Informaticae XXII(3), 227–255, 257–288 (1995)
6. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77(2), 321–358 (1995)
7. Dung, P.M., Kowalski, R.A., Toni, F.: Dialectic proof procedures for assumption-based, admissible argumentation. Artif. Intell. 170(2), 114–159 (2006)
8. Fages, F.: Consistency of Clark's completion and existence of stable models. Methods of Logic in Computer Science 1, 51–60 (1994)
9. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP, pp. 1070–1080. MIT Press, Cambridge (1988)
10. Kakas, A.C., Mancarella, P.: Negation as stable hypotheses. In: LPNMR, pp. 275–288. MIT Press, Cambridge (1991)
11. Malý, M.: Complexity of revised stable models. Master's thesis, Comenius University Bratislava (2006)
12. Pereira, L.M., Pinto, A.M.: Revised stable models - a semantics for logic programs. In: Bento, C., Cardoso, A., Dias, G. (eds.) EPIA 2005. LNCS (LNAI), vol. 3808, pp. 29–42. Springer, Heidelberg (2005)
13. Pereira, L.M., Pinto, A.M.: Reductio ad absurdum argumentation in normal logic programs. In: Argumentation and Non-monotonic Reasoning (ArgNMR 2007) workshop at LPNMR 2007, pp. 96–113 (2007)
14. Pinto, A.M.: Explorations in revised stable models — a new semantics for logic programs. Master's thesis, Universidade Nova de Lisboa (February 2005)
15. Soares, L.: Revising undefinedness in the well-founded semantics of logic programs. Master's thesis, Universidade Nova de Lisboa (2006)

Permutative Additives and Exponentials[⋆]

Gabriele Pulcini

Laboratoire d'Informatique de l'Université Paris-Nord
99, avenue Jean-Baptiste Clément – 93430 Villetaneuse
gabriele.pulcini@lipn.univ-paris13.fr

Abstract. Permutative logic (PL) is a noncommutative variant of multiplicative linear logic (MLL) arising from recent investigations concerning the topology of linear proofs. Permutative sequents are structured as oriented surfaces with boundary whose topological complexity is able to encode some information about the exchange in sequential proofs. In this paper we provide a complete permutative sequent calculus by extending that one of PL with rules for additives and exponentials. This extended system, here called *permutative linear logic* (PLL), is shown to be a conservative extension of linear logic and able to enjoy cut-elimination. Moreover, some basic isomorphisms are pointed out.

1 Introduction

Linear logic (LL) presents a remarkable skill in emphasizing geometrical features of logical proofs. LL comes in fact with a double syntax: the usual one in terms of sequential rules, and a more geometrical one, constituted by a set of links which allow to turn sequential proofs into graphs called *proof-nets*. Proof-nets quotient on the class of linear demonstrations enabling to avoid pointless syntactical bureaucracies [9], [7].

Studies on logical noncommutativity take advantage from this more geometrical approach due to the fact that the use of the exchange in a sequential proof affects the genus of its corresponding net. *Cyclic logic* (namely, linear logic in which only cyclic exchanges are allowed) [16] constitutes a limit case in which cut-free proofs always induce planar proof-nets [1]. This kind of results have been progressively generalized by topological investigations on linear proofs due to Bellin, Fleury [6], Melliès [12] and Métayer [13]. In particular, Métayer has proposed a way to translate any proof-net Π into a (compact and orientable) surface with boundary $\mathscr{S}(\Pi)$, such that:

- the rank of $\mathscr{S}(\Pi)$ constitutes a lower bound for the complexity of the exchange inside the proof π sequentialisation of Π [13];
- $\mathscr{S}(\Pi)$ represents the minimal surface on which Π can be drawn without crossings [8].

[⋆] Research supported by the Regional Council of *île-de-France*.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 469–483, 2007.
© Springer-Verlag Berlin Heidelberg 2007

By stressing the fact that topology tells about exchange, the above-mentioned works have induced the following non-commutative variants of MLL: *planar logic* [12], the *calculus of surfaces* [8] and, later, *permutative logic* [5]. Such calculi stand out for dealing with sequents structured as orientable surfaces with boundary. Being more precise, each sequent turns out indexed by a natural number (counting the handles) and with formulas grouped into disjoint cycles forming in this way a permutation (denoting, cycle by cycle, each boundary-component). Such structures (permutations with attached a natural number) have been called *q-permutations* [5] and separately studied in [14].

Unlike planar logic and the calculus of surfaces, PL comes with two explicit structural rules of *divide* and *merge* (topologically corresponding to an amalgamated sum) and it enjoys the two fundamental proof-theoretical properties of cut-elimination and focussing. Moreover, thanks to permutative modalities and constants, PL provides a specific mechanism able to manage (topological) resources during proof-construction [5].

This paper should be considered as the continuation of the first one in which the multiplicative fragment of permutative logic has been introduced [5]. We propose in fact an enrichment of the PL calculus with rules for additives and exponentials, here called *permutative linear logic* (PLL). On the one hand, additives pose the problem of establishing when two PL sequents can be considered as having same context with respect to a fixed formula in each one of them. There are in fact two ways to introduce the &-rule in the context of PL: either by requiring the two premises to share their permutative structure or by enabling two premises having different structures to be "mixed". In accordance with the fact that the &-rule should be negative in sense of Andreoli's property of focussing, we decide to "officially" adopt the first solution. Nevertheless, we also propose an alternative version of the &-rule in which structural rules are compacted and optimized: this version would be useful towards both a semantics and a theory of proof-nets for PL (remark that, in proof-nets, structural transformations should not explicitly appear). On the other hand, exponentials are treated in the standard way, i.e. as central elements essentially aside from the inner structure of sequents [15]. The inference system obtained in this way is shown to be a conservative extension of LL and able to enjoy cut-elimination. By stressing this latter property, we recall the notion of logical isomorphism and we point out some basic permutative isomorphisms.

The extension we propose in these pages is legitimate by the need of having a complete counterpart of *non-commutative logic* [2] in which, instead of the usual approach rooted in serial and parallel combinators, logical non-commutativity is approached from the more geometrical point of view afforded by topology. In our opinion, this change of viewpoint may offer an interesting framework in which reconsider some of the typical problems related with non-commutativity, for instance, problems arising in studies on linguistics and concurrency.

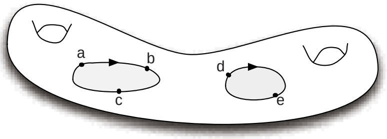
2 Multiplicative Permutative Logic

2.1 The Sequent Calculus

The well-known classification theorem for 2-dimensional surfaces says that any compact and connected orientable surface turns out to be homeomorphic to a sphere or to a connected sum of tori possibly with boundary [11]. If we consider an orientable surface $\mathscr{S}$ as the final result of identifying edges having same label in a set of polygons forming an its topological presentation, we have that each boundary-component will be formed by at least one edge. Let $\partial\mathscr{S}$ be the set of labels occurring on the boundary of $\mathscr{S}$: since fixed an orientation, we can notice that $\mathscr{S}$ induces a cyclic order on each one of the subsets of $\partial\mathscr{S}$ corresponding to boundary-components. In other words, we obtain nothing else but a permutation on $\partial\mathscr{S}$. The idea leading to the notion of q-permutation is that the basic information concerning any orientable connected surface $\mathscr{S}$ can always be encoded by a very easy mathematical structure consisting in a permutation σ (denoting cycle by cycle the boundary $\partial\mathscr{S}$) together with a natural number q counting the number of tori in the connected sum forming $\mathscr{S}$. Remark that the number of connected tori forming a surface $\mathscr{S}$ corresponds to the number of *handles* appearing on $\mathscr{S}$. In this way, q-permutations are able to characterize orientable connected surfaces modulo isomorphism, namely modulo homeomorphisms preserving the inner structure of the boundary together with an orientation.

Definition 1 (q-permutation, PL sequent). *A q-permutation is a triple* (X, σ, p)*, where* X *is a finite set,* σ *is a permutation on* X *and* $p \in \mathbb{N}$*. A PL sequent is nothing else, but a q-permutation in which the support* X *is a set of linear formulas.*

Example 1. It is easy to check that the surface proposed below is well-characterized by the q-permutation $\{(a, b, c), (d, e)\}, 2$.



Notation. – Capital Greek letters Γ, Δ, Λ, ... denote series of elements, whereas (Γ) means that the series Γ is taken modulo cyclic exchange; Σ, Ξ, Ψ, ... denote sets of cycles. q-permutations are indicated with small Greek letters α, β, γ, ... Moreover, $|\alpha|$ and $\alpha[a]$ respectively denote the support of α and that $a \in |\alpha|$. In the sequel of this paper, it will be useful to adopt a simplified notation for q-permutations obtained by omitting the support: (X, σ, p), in which $\sigma = (\Gamma_1)(\Gamma_2)\dots(\Gamma_q)$, will become $\{(\Gamma_1), (\Gamma_2), \dots, (\Gamma_q)\}, p$. For any permutation σ, $\sigma^\bullet$ denotes the number of its cycles.

- PL sequents will be denoted in two ways. We write $\vdash_p (\Gamma_1), (\Gamma_2), \dots, (\Gamma_q)$ for the PL sequent corresponding to the q-permutation $\{(\Gamma_1), (\Gamma_2), \dots, (\Gamma_q)\}, p$.

Otherwise, we directly write $\vdash \alpha$ for indicating the PL sequent corresponding to a certain q-permutation α.

Thanks to classification theorem, the topological complexity of an oriented surface $\mathscr{S}$ can be expressed by a couple of parameters (p, q), called the *genus* of the surface, such that $p \in \mathbb{N}$ is the number of handles of $\mathscr{S}$ and $q \in \mathbb{N}$ the number of pieces in which its boundary turns out to be decomposed. The rank can be straightforwardly obtained by the genus: $rk(\mathscr{S}) = 2p + q - 1$, if q is non-zero; $2p$, otherwise. Clearly we can associate genus and rank with q-permutations too.

Example 2. If $\alpha = \{(a, b), (c), (d, e)\}, 2$, then its genus is given by the couple $(2, 3)$ and $rk(\alpha) = 6$.

The multiplicative permutative calculus is recalled in Table 1; moreover, the involutive duality is given by De Morgan rules:

$$\begin{array}{llll} (A \parr B)^{\perp} = B^{\perp} \otimes A^{\perp} & (\flat A)^{\perp} = \# A^{\perp} & \hbar^{\perp} = h & \perp^{\perp} = 1 \\ (A \otimes B)^{\perp} = B^{\perp} \parr A^{\perp} & (\# A)^{\perp} = \flat A^{\perp} & h^{\perp} = \hbar & 1^{\perp} = \perp . \end{array}$$

By the fact that basic commutations are not provable keeping the lowest topological complexity, PL turns out to be an inference system able to deal with logical noncommutativity. As suggested by some of the next propositions, basic commutations can be recovered throughout the two permutative modalities $\flat$ and $\#$. It is easy to check in fact that formulas marked with permutative modalities behave as central elements, namely they can be freely moved inside sequents without any cost in terms of topological complexity.

Notation. $A_1, \ldots, A_n \vdash B$ denotes the sequent $\vdash_0 (A_1^{\perp}, \ldots, A_n^{\perp}, B)$ and $A \dashv\vdash B$ denotes the two sequents $A \vdash B$ and $B \vdash A$.

Proposition 1. [5] *The following sequents are provable in PL:*

$$\begin{array}{lll} A \vdash \flat A & \flat A \vdash A \parr \hbar & \flat A \dashv\vdash \flat \# A \\ (A \parr B) \parr C \dashv\vdash A \parr (B \parr C) & \flat\flat A \dashv\vdash \flat A & A \parr \flat B \dashv\vdash \flat B \parr A \\ A \parr \perp \dashv\vdash A & \flat \perp \dashv\vdash \perp & \flat(A \parr \flat B) \dashv\vdash \flat A \parr \flat B \\ \perp \parr A \dashv\vdash A & \flat \hbar \dashv\vdash \hbar & \flat(A \parr B) \dashv\vdash \flat(B \parr A). \end{array}$$

We can easily prove that PL without specific constants and modalities and with indexes fixed in 0, exactly corresponds to Mellies' planar logic; if we require in addition the rank of sequents to be null, we just obtain cyclic logic [5].

2.2 Relaxation

We call *relaxation* the relation induced on q-permutations by the two structural rules divide and merge. In particular, we say that a q-permutation β relaxes another q-permutation α, $\alpha \succeq \beta$, if α can be rewritten into β throughout a series of stuctural rules. A more formal definition is provided below.

Table 1. The sequent calculus of permutative logic

Identities

$$\text{ax.}\,\frac{}{\vdash_0 (A, A^\perp)} \qquad cut\,\frac{\vdash_d \Sigma, (\Gamma, A) \qquad \vdash_e \Theta, (\Delta, A^\perp)}{\vdash_{d+e} \Sigma, \Theta, (\Gamma, \Delta)}$$

Structural rules

$$\text{divide}\,\frac{\vdash_d \Sigma, (\Gamma, \Delta)}{\vdash_d \Sigma, (\Gamma), (\Delta)} \quad \text{merge}\,\frac{\vdash_d \Sigma, (\Gamma), (\Delta)}{\vdash_{d+1} \Sigma, (\Gamma, \Delta)}$$

Logical rules

$$\parr\,\frac{\vdash_d \Sigma, (\Gamma, A, B)}{\vdash_d \Sigma, (\Gamma, A \parr B)} \qquad \otimes\,\frac{\vdash_d \Sigma, (\Gamma, A) \qquad \vdash_e \Theta, (\Delta, B)}{\vdash_{d+e} \Sigma, \Theta, (\Delta, \Gamma, A \otimes B)}$$

$$\flat\,\frac{\vdash_d \Sigma, (\Gamma), (A)}{\vdash_d \Sigma, (\Gamma, \flat A)} \qquad \#\,\frac{\vdash_d \Sigma, (\Gamma, A)}{\vdash_d \Sigma, (\Gamma), (\# A)}$$

$$\hbar\,\frac{\vdash_{d+1} \Sigma, (\Gamma)}{\vdash_d \Sigma, (\Gamma, \hbar)} \qquad h\,\frac{}{\vdash_1 (h)}$$

$$\perp\,\frac{\vdash_d \Sigma, (\Gamma)}{\vdash_d \Sigma, (\Gamma, \perp)} \qquad 1\,\frac{}{\vdash_0 (1)}$$

Definition 2. [5] *Relaxation is the smallest reflexive transitive relation* $\succcurlyeq$ *on q-permutations such that:*

- *divide:* $(X, \sigma, p) \succcurlyeq (X, \sigma', p)$*, where* σ' *is obtained from* σ *dividing one cycle* (Γ, Δ) *of* σ *into two:* (Γ) *and* (Δ)*;*
- *merge:* $(X, \sigma, p) \succcurlyeq (X, \sigma', p+1)$*, where* σ' *is obtained by* σ *merging two cycles* (Γ) *and* (Δ) *of* σ *into one:* (Γ, Δ)*;*
- degenerate merge*:* $(X, \sigma, p) \succcurlyeq (X, \sigma, p+1)$*, namely we can always merge an empty cycle to a cycle of* σ *increasing of one the number* p*.*

Remark 1. The last point of the previous definition (degenerate merge) is based on the idea that a PL sequent may be presented in various different ways: $\vdash_p \Sigma, (\Gamma)$ as well as $\vdash_p \Sigma, (\Gamma), ()$.

Remark 2. Any application of a divide or merge (possibly degenerate) rule on a certain q-permutation increases its rank and this is the reason for which relaxation induces a partial order on the set of q-permutations [5].

Theorem 1 (decision of relaxation). [5],[14] *For any pair of q-permutations* $\alpha = (X, \sigma, p)$ *and* $\beta = (X, \tau, q)$*, we have:*

$$\alpha \succeq \beta \;\Leftrightarrow\; q - p \geqslant \frac{n - (\sigma^{-1}\tau)^\bullet + \sigma^\bullet - \tau^\bullet}{2}.$$

3 Permutative Additives and Exponentials

3.1 The Sequent Calculus

Negative (resp. positive) connectives are those ones having a reversible (resp. not reversible) introduction rule. These notions arise inside the framework of Andreoli's studies on the property of focussing which allow to eliminate redundant non-determinism during proof-construction, by imposing a rigid alternation between clusters of negative and positive connectives [3]. The introduction of the basic version of the &-rule in which premises share their permutative structure (Definition 3), allow to classify the &-connective as a negative one, without altering the fundamental symmetry between negative and positive connectives we have in linear logic. It is easy to see that structural rules commute with negative ones; in the perspective of a focussed calculus, this aspect bears out our choice: it means that structural rules can be relegated between generalized positive and negative connectives, as a sort of shift rule changing the polarity.

Definition 3 (permutative additives). *Rules for additives are introduced as follows.*

$$\frac{\vdash_p \Sigma, (\Gamma, A)}{\vdash_p \Sigma, (\Gamma, A \oplus B)}\,\oplus_L \qquad\qquad \frac{\vdash_p \Sigma, (\Gamma, B)}{\vdash_p \Sigma, (\Gamma, A \oplus B)}\,\oplus_R$$

$$\frac{\vdash_p \Sigma, (\Gamma, A) \qquad \vdash_p \Sigma, (\Gamma, B)}{\vdash_p \Sigma, (\Gamma, A \& B)}\,\&$$

$$\frac{}{\vdash_p \Sigma, (\Gamma, \top)}\,\mathit{true} \qquad\qquad \textit{(no rule for zero).}$$

Definition 4 (permutative exponentials). *Rules for permutative exponentials are introduced as follows.*

$$\frac{\vdash_p \Sigma, (\Gamma, A)}{\vdash_p \Sigma, (\Gamma, ?A)}\,\text{dereliction} \qquad\qquad \frac{\vdash_p \Sigma, (\Gamma)}{\vdash_p \Sigma, (\Gamma, ?A)}\,\text{weakening}$$

$$\frac{\vdash_p \Sigma, (\Gamma, ?A), (?A, \Delta)}{\vdash_p \Sigma, (\Gamma, ?A), (\Delta)}\,\text{contraction} \qquad\qquad \frac{\vdash_0 ?\Sigma, (?\Gamma, A)}{\vdash_0 ?\Sigma, (?\Gamma, !A)}\,\text{promotion}$$

Remark 3. By making the following two rules of *center* derivable, contraction rule induces the following two rules of *center*.

$$\frac{\vdash_p \Sigma, (\Gamma, ?A, \Delta)}{\vdash_p \Sigma, (\Gamma, \Delta, ?A)}\,\text{center}(1) \quad \cong \quad \dfrac{\dfrac{\dfrac{\vdash_p \Sigma, (\Gamma, ?A, \Delta)}{\vdash_p \Sigma, (\Gamma, ?A, \Delta, ?A)}\,\text{weak.}}{\vdash_p \Sigma, (\Gamma, \Delta, ?A), (?A)}\,\text{divide}}{\vdash_p \Sigma, (\Gamma, \Delta, ?A)}\,\text{contr.}$$

$$\frac{\vdash_p \Sigma, (\Gamma, ?A), (\Delta)}{\vdash_p \Sigma, (\Gamma), (?A, \Delta)}\,\text{center}(2) \quad \cong \quad \dfrac{\dfrac{\vdash_p \Sigma, (\Gamma, ?A), (\Delta)}{\vdash_p \Sigma, (\Gamma, ?A), (?A, \Delta)}\,\text{weak.}}{\vdash_p \Sigma, (\Gamma), (?A, \Delta)}\,\text{contr.}$$

As for the two permutative modalities $\flat$ and $\#$, formulas marked with exponentials behave as central elements, in other words they turn out to be essentially aside from the inner permutative structure of sequents. This is consistent with the standard treatment of exponentials in non-commutative systems, for instance in non-commutative logic [15]. In spite of their centrality, permutative modalities and exponentials remain two distinct logical objects. In fact, unlike permutative modalities, permutative exponentials allow to recover the basic properties concerning exponentials we have linear logic. In particular, as we will show in the next theorem, we can provide a proof for $\flat A \vdash ?A$, but the converse does not hold.

Theorem 2. *The following propositions are provable in PLL.*

- *Commutations:* $A\&B \dashv\vdash B\&A$; $!A \otimes B \dashv\vdash B\otimes !A$; $!(A \otimes B) \dashv\vdash !(B \otimes A)$.
- *Associativity:* $A\&(B\&C) \dashv\vdash (A\&B)\&C$.
- *Distributivity:* $A \otimes (B \oplus C) \dashv\vdash (A \otimes B) \oplus (A \otimes C)$.
- *Constants:* $!\top \dashv\vdash 1$; $A\&\top \dashv\vdash A$; $A \parr \top \dashv\vdash \top$.
- *Exponentials:* $!!A \dashv\vdash !A$; $!(A\&B) \dashv\vdash (!A) \otimes (!B)$; $?\flat A \vdash \flat ?A$; $\flat A \vdash ?A$.

Proof. We respectively report the proofs of the sequents $A\&B \vdash B\&A$, $!A \vdash A \otimes A$, $?\flat A \vdash \flat ?A$, $!A \otimes B \vdash B\otimes !A$ and $!A \vdash \flat A$.

$$\dfrac{\otimes_R\,\dfrac{\text{ax.}\,\dfrac{}{\vdash_0 (B^\perp, B)}}{\vdash_0 (B^\perp \oplus A^\perp, B)} \qquad \dfrac{\dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (B^\perp \oplus A^\perp, A)}\,\otimes_L}{\vdash_0 (B^\perp \oplus A^\perp, B\&A)}\,\&$$

$$\dfrac{\dfrac{\text{der.}\,\dfrac{\text{ax.}\,\dfrac{}{\vdash_0 (A^\perp, A)}}{\vdash_0 (?A^\perp, A)} \qquad \dfrac{\dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (A, ?A^\perp)}\,\text{der.}}{\vdash_0 (A), (?A^\perp)}\,\text{div.}}{\vdash_0 (?A^\perp, A \otimes A), (?A^\perp)}\,\otimes}{\vdash_0 (?A^\perp, A \otimes A)}\,\text{contr.}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (A^\perp, ?A)}\,\text{der.}}{\vdash_0 (\#A^\perp), (?A)}\,\#}{\vdash_0 (!\#A^\perp), (?A)}\,!}{\vdash_0 (!\#A^\perp, \flat ?A)}\,\flat$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{ax.}\,\dfrac{}{\vdash_0 (B^\perp, B)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (A, ?A^\perp)}\,\text{der.}}{\vdash_0 (!A, ?A^\perp)}\,!}{\vdash_0 (!A), (?A^\perp)}\,\text{div.}}{\vdash_0 (B^\perp, B\otimes !A), (?A^\perp)}\,\otimes}{\vdash_0 (B^\perp, ?A^\perp, B\otimes !A), (?A^\perp)}\,\text{weak.}}{\vdash_0 (B^\perp, ?A^\perp, B\otimes !A)}\,\text{contr.}}{\vdash_0 (B^\perp \parr ?A^\perp, B\otimes !A)}\,\parr$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (A^\perp, ?A)}\,\text{der.}}{\vdash_0 (\#A^\perp), (?A)}\,\#}{\vdash_0 (\#A^\perp, ?A)}\,\text{center(2)}$$

3.2 Embedding Linear Logic

Definition 5. *We define the function "$p\ell$" from LL to PL formulas in the following way. If p is an atom or a constant, then $p^{p\ell} = p$; moreover:*

$$\begin{array}{ll}
(A^\perp)^{p\ell} = (A^{p\ell})^\perp & \\
(A \parr B)^{p\ell} = A^{p\ell} \parr \flat B^{p\ell} & (A \otimes B)^{p\ell} = \# A^{p\ell} \otimes B^{p\ell} \\
(A \& B)^{p\ell} = A^{p\ell} \& B^{p\ell} & (A \oplus B)^{p\ell} = A^{p\ell} \oplus B^{p\ell} \\
(?A)^{p\ell} = ?A^{p\ell} & (!A)^{p\ell} = !A^{p\ell}
\end{array}$$

This function can be extended to sequents by mapping any set of formulas Σ into the identical permutation, namely: if $\Sigma = A_1, A_2, \dots, A_n$, then $\Sigma^{p\ell} = (A_1^{p\ell}), (A_2^{p\ell}), \dots, (A_n^{p\ell})$.

Theorem 3. *A sequent $\vdash \Sigma$ is provable in LL if, and only if, $\vdash_0 \Sigma^{p\ell}$ is provable in PLL.*

Proof. ($\Rightarrow$) We proceed by induction on the length of the LL proof $\pi \vdash \Sigma$. The base is easily verified as follows:

$$\dfrac{}{\vdash A, A^\perp}\,\text{ax.} \quad \xmapsto{p\ell} \quad \dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (A), (A^\perp)}\,\text{divide}$$

Then we consider some induction steps; the missing cases are immediate.

$$\dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \quad \xmapsto{p\ell} \quad \dfrac{\dfrac{\vdash \Gamma^{p\ell}, (A^{p\ell}), (B^{p\ell})}{\vdash \Gamma^{p\ell}, (A^{p\ell}, \flat B^{p\ell})}\,\flat}{\vdash \Gamma^{p\ell}, (A^{p\ell} \parr \flat B^{p\ell})}\,\parr$$

$$\dfrac{\vdash \Gamma, A \qquad \vdash B, \Delta}{\vdash \Gamma, A \otimes B, \Delta}\,\otimes \quad \xmapsto{p\ell} \quad \dfrac{\#\,\dfrac{\vdash_0 \Gamma^{p\ell}, (A^{p\ell})}{\vdash_0 \Gamma^{p\ell}, (\# A^{p\ell})} \qquad \vdash_0 (B^{p\ell}), \Delta^{p\ell}}{\vdash_0 \Gamma^{p\ell}, (\# A^{p\ell} \otimes B^{p\ell}), \Delta^{p\ell}}\,\otimes$$

$$\dfrac{\vdash \Gamma, A \qquad \vdash \Gamma, B}{\vdash \Gamma, A \& B}\,\& \quad \xmapsto{p\ell} \quad \dfrac{\vdash_0 \Gamma^{p\ell}, (A^{p\ell}) \qquad \vdash_0 \Gamma^{p\ell}, (B^{p\ell})}{\vdash_0 \Gamma^{p\ell}, (A^{p\ell} \& B^{p\ell})}\,\&$$

$$\dfrac{\vdash \Gamma, A}{\vdash \Gamma, A, ?B}\,\text{weak.} \quad \xmapsto{p\ell} \quad \dfrac{\dfrac{\vdash_0 \Gamma^{p\ell}, (A^{p\ell})}{\vdash_0 \Gamma^{p\ell}, (?A^{p\ell}, ?B^{p\ell})}\,\text{weak.}}{\vdash_0 \Gamma^{p\ell}, (?A^{p\ell}), (?B^{p\ell})}\,\text{divide}$$

($\Leftarrow$) It is sufficient to remark that any PLL proof $\pi \vdash_0 \Gamma^{p\ell}$ can be turned into an LL proof $\pi' \vdash \Gamma$ simply by removing all the superfluous information: structural rules together with permutative decorations.

3.3 An Alternative Approach to Additives

In order to perform a &-rule involving two premises having different structures, we have to relax sequents since we arrive to a compromise, a common form allowing the application of the basic &-rule. This process of "approaching" sequents throughout structural rules is formalized by the set of the *nearest common stops* introduced in Definition 7. Before introducing this notion some technics concerning chains of structural transformations are required.

Notation. Let α and β be two q-permutations such that $\alpha \succeq \beta$. With $\mathscr{C} : \alpha \rightsquigarrow_{d/m} \beta$ we denote a chain of q-permutations rewriting α into β, such that each step of $\mathscr{C}$ corresponds to an application of either divide or merge rule.

Definition 6 (minimal chain). *A chain is said to be minimal, if it consists in a minimal number of steps.*

Procedure 4 (computing chains) [14] *Let α and β be two q-permutations such that $\alpha \succeq \beta$ We can obtain a chain $\mathscr{C} : \alpha \rightsquigarrow_{d/m} \beta$ simply by arbitrarily applying the following three specific versions of divide and merge rules. τ denotes the permutation of β.*

$$\text{If } \tau(a) = b : \qquad \frac{\{\Sigma, (a, \Gamma, b, \Delta)\}, p}{\{\Sigma, (a, b, \Delta), (\Gamma)\}, p}\ \textit{divide(1)};$$

$$\text{if } \Gamma \text{ is a cycle of } \tau\text{:} \qquad \frac{\{\Sigma, (\Gamma, \Delta)\}, p}{\{\Sigma, (\Gamma), (\Delta)\}, p}\ \textit{divide(2)};$$

$$\text{if } \tau(a) = b\text{:} \qquad \frac{\{\Sigma, (\Gamma, a), (b, \Delta)\}, p}{\{\Sigma, (\Gamma, a, b, \Delta)\}, p+1}\ \textit{merge}.$$

Example 3. Procedure 4 is here applied in order to produce a chain $\mathscr{C} : \alpha \rightsquigarrow_{d/m} \beta$.

$$\cfrac{\cfrac{\cfrac{\cfrac{\alpha = \{(a, b, c, d, e)\}, 0}{\{(a, d, e), (b, c)\}, 0}\ \text{divide(1)}}{\{(a, d), (e), (b, c)\}, 0}\ \text{divide(2)}}{\{(a, d), (e, b, c)\}, 1}\ \text{merge}}{\beta = \{(a, d), (e, b), (c)\}, 1}\ \text{divide(2)}$$

Theorem 5. [14] *If $\mathscr{C}$ is a chain afforded by Procedure 4, then it is minimal.*

If we ignore the superfluous information concerning indexes, the divide\merge rewriting system can be seen as directly working on permutations. In this way, any chain of q-permutations $\mathscr{C} : \alpha \rightsquigarrow_{d/m} \beta$, where $\alpha = (X, \sigma, p)$ and $\beta =$

(X, τ, q), is implicitly a chain of permutations $\sigma \rightsquigarrow_{d/m} \tau$ too. Moreover, remark that, unlike chains of q-permutations, any chain of permutations $\sigma \rightsquigarrow_{d/m} \tau$ can be reversed into a chain $\tau \rightsquigarrow_{d/m} \sigma$ such that, if $\sigma \rightsquigarrow_{d/m} \tau$ is minimal, then $\tau \rightsquigarrow_{d/m} \sigma$ is minimal too.

Theorem 6. [14] *Any chain of permutations implicit into a minimal chain of q-permutations, is minimal too.*

Definition 7 (nearest common stops). *Let α and β be two q-permutations sharing the support. A q-permutation ξ belongs to the set of the* nearest common stops *of α and β, denoted with $\mathtt{ncs}(\alpha, \beta)$, if, and only if, $\alpha, \beta \succeq \xi$ and $rk(\xi)$ is minimal.*

Proposition 2. *For any pair of q-permutations α and β sharing the support, we have:*

1. $\mathtt{ncs}(\alpha, \beta) = \mathtt{ncs}(\beta, \alpha)$*;*
2. $\mathtt{ncs}(\alpha, \beta) \neq \varnothing$*;*
3. *if ξ, $\xi' \in \mathtt{ncs}(\alpha, \beta)$, then they are incomparable;*
4. *if $\alpha \succeq \beta$, then $\mathtt{ncs}(\alpha, \beta) = \{\beta\}$.*

Now we aim to provide an effective procedure able to reach elements in any set $\mathtt{ncs}(\alpha, \beta)$. For α and β such that $\alpha \succeq \beta$, we know that $\mathtt{ncs}(\alpha, \beta) = \{\beta\}$. The next theorem deals with the case in which α and β are incomparable.

Theorem 7. *Consider two incomparable q-permutations $\alpha = (X, \sigma, p)$ and $\beta = (X, \tau, q)$, and a third one ξ obtained as follows.*

> *According to Procedure 4, we start rewriting α in order to reconstruct the permutation τ expressed by β: we call ξ the first q-permutation we meet such that it relaxes β.*

We have that $\xi \in \mathtt{ncs}(\alpha, \beta)$.

Proof. Consider three q-permutations $\alpha = (X, \sigma, p)$, $\beta = (X, \tau, q)$ and ξ obtained from α and β according to the claim of the theorem. Suppose by absurd that $\xi \notin \mathtt{ncs}(\alpha, \beta)$ and consider any $\theta \in \mathtt{ncs}(\alpha, \beta)$ (by Proposition 2.2, we know that $\mathtt{ncs}(\alpha, \beta) \neq \varnothing$). Now consider the chain $\mathscr{C} : \alpha \rightsquigarrow_{d/m} \beta'$, where $\beta' = (X, \tau, q+k)$, computed in order to obtain ξ. By the fact that $\theta \in \mathtt{ncs}(\alpha, \beta)$ and $rk(\theta) < rk(\xi)$, there exist two chains $\mathscr{C}_1 : \alpha \rightsquigarrow_{d/m} \theta$ and $\mathscr{C}_2 : \beta \rightsquigarrow_{d/m} \theta$ respectively shorter than $\mathscr{C}'_1 : \alpha \rightsquigarrow_{d/m} \xi$ and $\mathscr{C}'_2 : \beta \rightsquigarrow_{d/m} \xi$. So, we have a chain of permutations $\sigma \rightsquigarrow_{d/m} \tau$ shorter than that one implicit in $\mathscr{C}$ which is, by Theorem 6, absurd.

Example 4. Consider the following chain performed in order to compute an element $\xi \in \mathtt{ncs}(\alpha, \beta)$, where $\alpha = \{(a, b, c, d)\}, 0$ and $\beta = \{(a, d, c), (b)\}, 0$. The first line we meet such that it relaxes β is the third one and so $\xi = \{(a, d, c, b)\}, 1$.

$$\begin{array}{cl} \alpha = \{(a,b,c,d)\}, 0 & \\ \hline \{(a,d),(b,c)\}, 0 & \text{divide} \\ \hline \xi = \{(a,d,c,b)\}, 1 & \text{merge} \\ \hline \beta' = \{(a,d,c),(b)\}, 1 & \text{divide} \end{array}$$

At this point, we have at disposal a complete technical background for providing a version of the &-rule, denoted with [&], which enables to mix two premises having different structures by compacting and optimizing structural rules.

Definition 8. *We write $\alpha\,[a'/a]$ for the q-permutation obtained from α by replacing an element a of its support with another one a'. The* [&]*-rules is here introduced by indicating sequents as q-permutations.*

$$\dfrac{\vdash \alpha \qquad\qquad \vdash \beta}{\vdash \xi \in \texttt{ncs}(\alpha\,[A\&B/A], \beta\,[A\&B/B])}\,[\&],\ \textit{where}\ |\alpha| \backslash \{A\} = |\beta| \backslash \{B\}.$$

Example 5. Below we propose a concrete application of the [&]-rule together with an its "extracted" version.

$$\dfrac{\vdash_0 (a,b,c,f_1) \qquad \vdash_0 (b),(c,a,f_2)}{\vdash_1 (c,b,a,f_1\&f_2)}\,[\&] \quad \cong$$

$$\cong \quad \dfrac{\text{merge}\,\dfrac{\text{divide}\,\dfrac{\vdash_0 (a,b,c,f_1)}{\vdash_0 (a,f_1),(b,c)}}{\vdash_1 (c,b,a,f_1)} \qquad \dfrac{\vdash_0 (b),(c,a,f_2)}{\vdash_1 (c,b,a,f_2)}\,\text{merge}}{\vdash_1 (c,b,a,f_1\&f_2)}\,\&.$$

Thanks to the main result provided in the next section (Theorem 8), we can easily notice that cut-elimination is preserved by replacing the basic version of the &-rule with that one just provided in Definition 8. Remark that, unlike the basic &, the [&] connective cannot be catalogued as a negative one because of the fact that different conclusions may be in accordance with the same pair of premises.

4 Cut Elimination and Isomorphisms

4.1 Cut Elimination

Theorem 8. *Any PL proof $\pi \vdash_p \Sigma$ can be rewritten into a PL proof $\pi' \vdash_p \Sigma$ without cuts.*

Proof. Here we extend the proof already provided in [5] for the limited case of multiplicatives. Our proof is organized in two steps. At first we remark that cut-elimination for LL [9] implies that any PLL proof $\pi \vdash \alpha$ can be reduced into a PLL proof $\pi' \vdash \beta$ without cuts, such that $|\alpha| = |\beta|$. In other words, cut-elimination preserves multisets of formulas. The second step consists in showing that cut-elimination preserves permutative structures too. It is easy to check; we illustrate below just some key cases of symmetric reductions.

- Contraction/promotion.

$$\dfrac{\text{contr.}\,\dfrac{\vdash_p \Sigma,(\Gamma,?A),(\Delta,?A)}{\vdash_p \Sigma,(\Gamma,?A),(\Delta)} \qquad \dfrac{\vdash_0 ?\Xi,(?\Lambda,A^{\perp})}{\vdash_0 ?\Xi,(?\Lambda,!A^{\perp})}\,!}{\vdash_p \Sigma,?\Xi,(\Gamma,?\Lambda),(\Delta)}\,\text{cut} \qquad \rightsquigarrow$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\vdash_p \Sigma,(\Gamma,?A),(\Delta,?A) \qquad \dfrac{\vdash_0 ?\Xi,(?\Lambda,A^\perp)}{\vdash_0 ?\Xi,(?\Lambda,!A^\perp)}\,!}{\vdash_p \Sigma,?\Xi,(\Gamma,?\Lambda),(\Delta,?A)}\,\text{cut} \qquad \dfrac{\vdash_0 ?\Xi,(?\Lambda,A^\perp)}{\vdash_0 ?\Xi,(?\Lambda,!A^\perp)}\,!}{\dfrac{\vdash_p \Sigma,?\Xi,?\Xi,(\Gamma,?\Lambda),(\Delta,?\Lambda)}{\vdash_p \Sigma,?\Xi,(\Gamma,?\Lambda),(\Delta)}\,\text{contr.}}\,\text{cut}$$

- Weakening/promotion.

$$\text{weak.}\,\dfrac{\dfrac{\vdash_p \Sigma,(\Gamma)}{\vdash_p \Sigma,(\Gamma,?A)} \qquad \dfrac{\vdash_0 ?\Xi,(?\Delta,A^\perp)}{\vdash_0 ?\Xi,(?\Delta,!A^\perp)}\,!}{\vdash_p \Sigma,?\Xi,(\Gamma,?\Delta)}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\vdash_p \Sigma,(\Gamma)}{\vdash_p \Sigma,?\Xi,(\Gamma,?\Delta)}\,\text{weak.}$$

- &/⊕.

$$\&\,\dfrac{\dfrac{\vdash_p \Sigma,(\Gamma,A) \qquad \vdash_p \Sigma,(\Gamma,B)}{\vdash_p \Sigma,(\Gamma,A\&B)} \qquad \dfrac{\vdash_q \Xi,(\Delta,B^\perp)}{\vdash_q \Xi,(\Delta,B^\perp\oplus A^\perp)}\,\oplus_R}{\vdash_{p+q} \Sigma,\Xi,(\Gamma,\Delta)}\,\text{cut} \quad \rightsquigarrow$$

$$\rightsquigarrow \quad \dfrac{\vdash_p \Sigma,(\Gamma,B) \qquad \vdash_q \Xi,(\Delta,B^\perp)}{\vdash_{p+q} \Sigma,\Xi,(\Gamma,\Delta)}\,\text{cut.}$$

4.2 Some Isomorphisms

Definition 9 (η-expansion). *We inductively define the function η which associates to each PL formula F a PL proof $\eta(F)$, η-expansion of F.*

- *For every atom A:* $\eta(A)=\eta(A^\perp)=\dfrac{}{\vdash_0 (A,A^\perp)}\,ax.$
- $\eta(1)=\dfrac{}{\vdash_0 (1)}\,1$
- $\eta(h)=\dfrac{}{\vdash_1 (h)}\,h$
- $\eta(\top)=\dfrac{}{\vdash_p \Sigma,(\Gamma,\top)}\,\top$
- *For every formula F: $\eta(F)=\eta(F^\perp)$ and, if $F=\psi(F_1,\dots,F_n)$ with ψ positive n-ary connective, then:*

$$\eta(F):\quad \dfrac{\dfrac{\eta(F_1) \qquad \dots \qquad \eta(F_n)}{\psi(F_1,\dots,F_n),F_1^\perp,\dots,F_n^\perp}\,\psi}{\psi(F_1,\dots,F_n),\psi^\perp(F_1^\perp,\dots,F_n^\perp)}\,\psi^\perp$$

Definition 10 (isomorphism). *Consider the proofs $\lambda\circ\pi \vdash_0 (A^\perp,A)$ and $\pi\circ\lambda \vdash_0 (B^\perp,B)$ obtained from the two proofs $\pi \vdash_0 (A^\perp,B)$ and $\lambda \vdash_0 (B^\perp,A)$ respectively by cutting B with $B^\perp$ and A with $A^\perp$. $A \dashv\vdash B$ is said to be an isomorphism if, and only if, $\pi\circ\lambda=\eta(A)$ and $\lambda\circ\pi=\eta(B)$.*

Theorem 9 (multiplicative isomorphisms). *The following equivalences are isomorphisms:*

$$(A \parr B) \parr C \dashv\vdash A \parr (B \parr C) \qquad \flat\flat A \dashv\vdash \flat A \qquad A \parr \flat B \dashv\vdash \flat B \parr A$$
$$A \parr \bot \dashv\vdash A \qquad \flat\bot \dashv\vdash \bot \qquad \flat(A \parr \flat B) \dashv\vdash \flat A \parr \flat B$$
$$\bot \parr A \dashv\vdash A \qquad \flat\hbar \dashv\vdash \hbar \qquad \flat(A \parr B) \dashv\vdash \flat(B \parr A)$$

Proof. We detail below just the case of $A \parr \bot \dashv\vdash A$.

$$\dfrac{\dfrac{1\,\dfrac{}{\vdash_0 (1)} \qquad \dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (1 \otimes A^\perp, A)}\,\otimes \qquad \dfrac{\dfrac{\dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (A^\perp, A, \bot)}\,\bot}{\vdash_0 (A^\perp, A \parr \bot)}\,\parr}{\vdash_0 (1 \otimes A^\perp, A \parr \bot)}\,\text{cut} \quad \rightsquigarrow$$

$$\rightsquigarrow \quad \dfrac{\dfrac{\dfrac{1\,\dfrac{}{\vdash_0 (1)} \qquad \dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (1 \otimes A^\perp, A)}\,\otimes}{\vdash_0 (1 \otimes A^\perp, A, \bot)}\,\bot}{\vdash_0 (1 \otimes A^\perp, A \parr \bot)}\,\parr$$

$$\dfrac{\dfrac{1\,\dfrac{}{\vdash_0 (1)} \qquad \dfrac{}{\vdash_0 (A^\perp, A)}\,\text{ax.}}{\vdash_0 (A, 1 \otimes A^\perp)}\,\otimes \qquad \dfrac{\dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (A, \bot, A^\perp)}\,\bot}{\vdash_0 (A \parr \bot, A^\perp)}\,\parr}{\vdash_0 (A, A^\perp)}\,\text{cut} \quad \rightsquigarrow$$

$$\rightsquigarrow \quad \dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}$$

Example 6. $\flat\#A \dashv\vdash \flat A$ constitutes an example of an equivalence which is not an isomorphism too. The reduced proof on the right is not an η-expansion of $\flat\#A$, in fact it presents an application of the $\flat$ rule which interrupts a block of positive rules.

$$\dfrac{\flat\,\dfrac{\#\,\dfrac{\#\,\dfrac{\text{ax.}\,\dfrac{}{\vdash_0 (A, A^\perp)}}{\vdash_0 (\#A), (A^\perp)}}{\vdash_0 (\#A), (\#A^\perp)}}{\vdash_0 (\flat\#A, \#A^\perp)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (A), (A^\perp)}\,\text{divide}}{\vdash_0 (A), (\flat A^\perp)}\,\flat}{\vdash_0 (A), (\#\flat A^\perp)}\,\#}{\vdash_0 (\flat A, \#\flat A^\perp)}\,\flat}{\vdash_0 (\flat\#A, \#\flat A^\perp)}\,\text{cut} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash_0 (A, A^\perp)}\,\text{ax.}}{\vdash_0 (\#A), (A^\perp)}\,\#}{\vdash_0 (\#A), (\flat A^\perp)}\,\flat}{\vdash_0 (\#A), (\#\flat A^\perp)}\,\#}{\vdash_0 (\flat\#A, \#\flat A^\perp)}\,\flat$$

Theorem 10 (additive and exponential isomorphisms). *All the equivalences listed in Theorem 2 are isomorphisms too.*

5 Future Work

A focussed version of the PLL calculus should be defined by extending that one already existing for PL [5].

Semantical issues (phase and denotational semantics) together with possible topological interpretations of proof-nets with additives are still waiting to be explored. The alternative approach to additives outlined in Subsection 3.3 would be useful in both these directions. In particular, in order to translate proof-nets into topological surfaces [13], we should be able to associate with each link of the net, its corresponding cell. Because of in proof-nets structural rules do not explicitly appear, the problem of associating a cell with a &-link requires to take in account the involved structural rules, exactly what the [&]-rule makes.

Concerning exponentials, the solution proposed in these pages should be considered as a "minimalist" one, i.e. in Section 3.2 we have showed that, in order to embed LL into PLL, it is sufficient to consider exponential formulas as essentially gathered into multisets associated with ordinary permutative sequents. However, it would be worthwhile to define exponentials in a genuine permutative way, really sharing the permutative structure with other formulas: a deductive system in which, for instance, rules for duplicating and absorbing formulas work taking in account their position. In this direction, the main obstacle to overcome consists in defining a deductive system which is still a conservative extension of LL. In our opinion, an in-deep investigation on the relations between permutative modalities and exponentials could be useful. Moreover, in terms of geometry, an extension of our structures including also non-orientable surfaces might offer a wider framework in which this kind of problems could be more properly placed.

Acknowledgements

The author wish to thank Christophe Fouqueré, Virgile Mogbil and Paul Ruet for their support and suggestions.

References

1. Abrusci, V.M.: Lambek calculus, cyclic linear logic, noncommutative linear logic: language and sequent calculus. In: Proofs and Linguistic Categories. Cooperativa Libraria Universitaria Editrice Bologna, pp. 21–48 (1997)
2. Abrusci, V.M., Ruet, P.: Non-commutative logic I: the multiplicative fragment. Annals of Pure and Applied Logic 101(1), 29–64 (2000)
3. Andreoli, J.-M.: Focussing and proof construction. Annals of Pure and Applied Logic 107, 131–163 (2001)
4. Andreoli, J.-M.: An axiomatic approach to structural rules for locative linear logic. In: Linear logic in computer science. London Mathematical Society Lecture Notes Series, vol. 316, Cambridge University Press, Cambridge (2004)

5. Andreoli, J.-M., Pulcini, G., Ruet, P.: Permutative Logic. In: Ong, L. (ed.) CSL 2005. LNCS, vol. 3634, pp. 184–199. Springer, Heidelberg (2005)
6. Bellin, G., Fleury, A.: Planar and braided proof-nets for multiplicative linear logic with mix. Archive for Mathematical Logic 37(5-6), 309–325 (1998)
7. Danos, V., Regnier, L.: The structure of multiplicatives. Archive for Mathematical Logic 28, 181–203 (1989)
8. Gaubert, C.: Two-dimensional proof-structures and the exchange rule. Mathematical Structures in Computer Science 14(1), 73–96 (2004)
9. Girard, J.-Y.: Linear Logic: its syntax and semantics. In: Advances in Linear Logic. London Mathematical Society Lecture Note Series, vol. 222, pp. 1–42. Cambridge University Press, Cambridge (1995)
10. Lambek, J.: The mathematics of sentence structure. American Mathematical Monthly 65(3), 154–170 (1958)
11. Massey, W.S.: A basic course in algebraic topology. Springer, Heidelberg (1991)
12. Melliés, P.-A.: A topological correctness criterion for multiplicative non-commutative logic. In: Linear logic in computer science. London Mathematical Society Lecture Notes Series, vol. 316, Cambridge University Press, Cambridge (2004)
13. Métayer, F.: Implicit exchange in multiplicative proofnets. Mathematical Structures in Computer Science 11(2), 261–272 (2001)
14. Pulcini, G.: Computing Relaxation in Permutative Logic. Preprint downloadable from: `http://logica.uniroma3.it/?q=pubblicazioni/Pulcini%2C+Gabriele`
15. Ruet, P.: Non-commutative logic II: sequent calculus and phase semantics. Mathematical Structures in Computer Science 10(2), 277–312 (2000)
16. Yetter, D.N.: Quantales and (non-commutative) linear logic. Journal of Symbolic Logic 55(1) (1990)

Algorithms for Propositional Model Counting*

Marko Samer and Stefan Szeider

Department of Computer Science
Durham University, UK
`{marko.samer,stefan.szeider}@durham.ac.uk`

Abstract. We present algorithms for the propositional model counting problem #SAT. The algorithms are based on tree-decompositions of graphs associated with the given CNF formula, in particular primal, dual, and incidence graphs. We describe the algorithms in a coherent fashion that admits a direct comparison of their algorithmic advantages. We analyze and discuss several aspects of the algorithms including worst-case time and space requirements and simplicity of implementation. The algorithms are described in sufficient detail for making an implementation reasonably easy.

1 Introduction

Propositional model counting (#SAT) is the problem of determining the number of satisfying truth assignments (models) of a given propositional formula in conjunctive normal form (CNF). This problem arises in several areas of artificial intelligence, in particular in the context of probabilistic reasoning [3,25]. However, since the problem is #P-complete [28], it is very unlikely that it can be solved in polynomial time. #SAT remains #P-hard even for monotone 2CNF formulas and Horn 2CNF formulas, and it is NP-hard to approximate the number of models of a formula with n variables within $2^{n^{1-\varepsilon}}$ for $\varepsilon > 0$. This approximation hardness holds also for monotone 2CNF formulas and Horn 2CNF formulas [25]. Thus, restricting the syntax of the instances does not lead to tractability.

An alternative to restricting the syntax is to impose *structural restrictions* on the input formulas. Structural restrictions can be applied in terms of certain parameters (invariants) of graphs or hypergraphs associated with formulas. In this paper we will mainly consider the following graphs (more exact definitions are given in Section 2.3, examples are shown in Figure 1). The *primal graph* has as vertices the variables of the given formula, two variables are joined by an edge if they occur together in a clause. Symmetrically, the *dual graph* has as vertices the clauses of the formula, two clauses are joined by an edge if they share a variable. Finally, the *incidence graph* is a bipartite graph where one vertex class consists of the clauses of the given formula, and the other vertex class consists of the variables; a clause and a variable are joined by an edge if the variable occurs in the clause. Primal and incidence graphs have been widely studied in the literature on satisfiability and constraint satisfaction, whereas dual graphs have received less attention.

---

* Research supported by the EPSRC, project EP/E001394/1.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 484–498, 2007.
© Springer-Verlag Berlin Heidelberg 2007

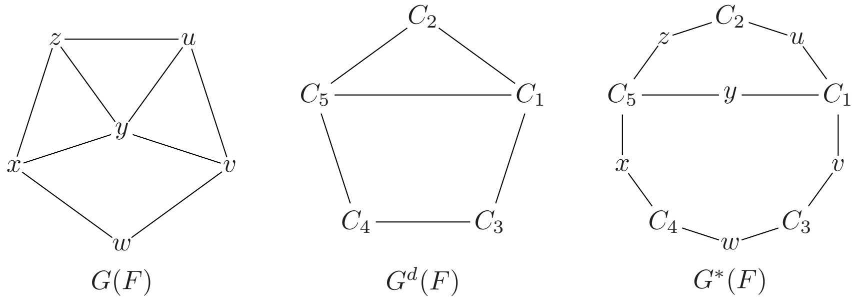


Fig. 1. Graphs associated with the CNF formula $F = \{C_1, \ldots, C_5\}$ with $C_1 = \{u, \neg v, \neg y\}$, $C_2 = \{\neg u, z\}$, $C_3 = \{v, \neg w\}$, $C_4 = \{w, \neg x\}$, $C_5 = \{x, y, \neg z\}$; the primal graph $G(F)$, the dual graph $G^d(F)$, and the incidence graph $G^*(F)$.

We apply structural restrictions on CNF formulas by bounding the graph parameter treewidth of the associated graphs. Treewidth, a graph parameter introduced by Robertson and Seymour in their Graph Minors Project, measures in a certain sense the "tree-likeness" of a graph. Many otherwise NP-hard graph problems such as Hamiltonicity and 3-colorability are solvable in polynomial time for graphs of bounded treewidth. It is generally believed that many practically relevant problems actually do have low treewidth [4]. Treewidth is based on certain decompositions of graphs, called tree-decompositions, where sets of vertices ("bags") of a graph are arranged at the nodes of a tree such that certain conditions are satisfied (see Section 2.1). If a graph has treewidth k then it admits a tree-decomposition of *width* k, i.e., a tree-decomposition where all bags have size at most $k+1$. Depending on whether we consider the treewidth of the primal, dual, or incidence graph of a given CNF formula, we speak of the primal, dual, or incidence treewidth of the formula, respectively.

Owing to a general result on Monadic Second Order Logic of Courcelle, Makowsky, and Rotics [9], the model counting problem can be solved in polynomial time for formulas of bounded primal, dual, and incidence treewidth, respectively. However, the algorithms obtained via this general method are impractical. For getting practical results, one needs to design tailor-suited algorithms for the particular problem domain. As the algorithms under consideration are typically exponential in the treewidth, small improvements can have a strong impact on the practicability.

Contributions of this paper

We propose three efficient model counting algorithms that utilize small primal, dual, and incidence treewidth of instances, respectively. We present the three algorithms in a coherent fashion that provides an insight into theoretical advantages and disadvantages of the three parameters. Furthermore, we describe the algorithms at a level of detail that makes an implementation reasonably straightforward.

Our three algorithms follow the principle of *dynamic programming*: we start at leaf nodes of the tree-decomposition and work our way up in the tree, computing at each node some information (stored in a table) on the subgraph thus far encountered. More details on the dynamic programming process and an analysis of space requirements is given in Section 3.4.

The following table summarizes worst-case runtimes of the three algorithms. Here k and N denote width and number of nodes of the given tree-decomposition of the primal, dual, and incidence graph, respectively; d and l denote the maximum number of occurrences over all variables and the cardinality of a largest clause of the given CNF formula, respectively. For the bounds on the runtimes we assume arithmetic operations to have constant runtime; in Section 3 we actually provide a refined analysis based on bit complexity.

primal graph	dual graph	incidence graph
$\mathcal{O}(2^k kdN)$	$\mathcal{O}(2^k klN)$	$\mathcal{O}(2^k k\,(l+2^k)\,N)$

Note that all three algorithms are *fixed-parameter algorithms* with respect to the corresponding treewidth parameter. A fixed-parameter algorithm solves instances of size n and parameter k in time $\mathcal{O}(f(k)n^c)$ where f denotes a computable function and c denotes a constant that is independent of the parameter k [10,12,21]. The main advantage of fixed-parameter algorithms is that the runtime increases moderately when n becomes large, in contrast to algorithms with runtime of, say, $\mathcal{O}(n^k)$.

The *incidence treewidth algorithm* is superior to the other two algorithms if the input formula has large clauses and contains variables that occur in many clauses. Such instances have large primal and large dual treewidth since a clause containing more than n literals causes the primal treewidth to be at least n, a variable occurring in more than n clauses causes the dual treewidth to be at least n (this follows from the fact that if a graph contains a complete subgraph on more than n vertices then the treewidth of the graph is at least n [17]).

However, our results indicate that the *primal treewidth algorithm* as well as the *dual treewidth algorithm* are exponentially faster then the incidence treewidth algorithm, imposing an exponential factor of 2^k instead of 4^k. Thus, although one can simulate the primal and dual treewidth algorithms by the incidence treewidth algorithm (a CNF formula of primal or dual treewidth k has incidence treewidth at most $k+1$ [19]), such a simulation increases the runtime exponentially.

We also study space requirements of the three algorithms in terms of maximum number of tables that need to be kept simultaneously in memory during the dynamic programming process. We analyze the table requirements and explain how optimal bottom-up traversals can be computed efficiently.

In summary, our analysis indicates that each of the three algorithms has its advantages and disadvantages. One needs to choose the right algorithm depending on context and area of the application under consideration.

Related work

Fischer, Makowsky, and Ravve [11] propose a fixed-parameter algorithm for #SAT with respect to the incidence treewidth. Their algorithm is based on a recursive splitting of the given formula according to a tree-decomposition of the incidence graph, making use of the inclusion-exclusion principle.

Branchwidth is a graph parameter that is related to treewidth by a constant factor [24]. Bacchus, Dalmao, and Pitassi [3] propose an algorithm that solves #SAT in time $n^{O(1)}2^{O(k)}$ for formulas with n variables whose formula hypergraphs have branchwidth k. The algorithm is based on the DPLL procedure and uses caching techniques for an efficient reuse of solutions for sub-problems; the branch-decomposition provides an ordering of the variables as processed by the DPLL procedure. A fixed-parameter algorithm for SAT with respect to primal treewidth has previously been proposed by Gottlob, Scarcello, and Sideri [15].

A different approach for solving #SAT was presented by Nishimura, Ragde, and Szeider [22]. They developed a fixed-parameter algorithm for computing strong backdoor sets with respect to cluster formulas, which yields a fixed-parameter algorithm for #SAT. In terms of generality, the corresponding parameter *clustering-width* is incomparable with incidence treewidth.

The *clique-width of directed incidence graphs* of CNF formulas provides a parameterization that is strictly more general than the treewidth parameters considered above. The directed (or signed) incidence graph is obtained from the incidence graph by indicating positive or negative occurrences of variables by the orientation of the corresponding edge. Fixed-parameter tractability of #SAT follows via the meta-theorem of Courcelle, Makowsky, and Rotics [8] on counting problems expressible in a certain fragment of Monadic Second Order Logic (MSO_1), yielding an algorithm that is double-exponential in the width of the clique-width decomposition. A single-exponential algorithm is due to Fisher, Makowsky, and Ravve [11]. However, both algorithms rely on Oum and Seymour's approximation algorithm for clique-width [23] which is of limited practical value in view of its runtime of $\mathcal{O}(n^9 \log n)$ and the exponential approximation error of $2^{3k+2} - 1$.

The various treewidth parameters can be defined analogously for instances of the constraint satisfaction problem (CSP), considering constraints (i.e., relations) instead of clauses. From the work of Gottlob et al. [15] it follows that the Boolean CSP is fixed-parameter tractable with respect to the parameter primal treewidth. In contrast to SAT and #SAT, this result cannot be generalized to the more general parameter incidence treewidth (subject to a complexity theoretic assumption): Samer and Szeider [26] have shown that the Boolean CSP (also known as Generalized Satisfiability) parameterized by the incidence treewidth is W[1]-hard. W[1] is a complexity class in parameterized complexity theory; there is strong theoretical evidence that W[1]-hard problems are not fixed-parameter tractable [10].

In the context of constraint satisfaction several hypergraph parameters have been considered, such as hypertree-width [14], spread-cut width [7], and fractional hypertree-width [16]. For instances of unbounded arity (i.e., the associated hypergraphs have hyperedges of arbitrary size) these parameters are strictly more general than incidence treewidth. In the following paragraph we provide arguments that indicate that these hypergraph parameters have no apparent significance for CNF satisfiability.

A hypergraph is *acyclic* if there is a tree-decomposition (of its primal graph) whose number of nodes equals the number of hyperedges and for each hyperedge there is a tree-node that contains exactly the vertices of the hyperedge in its bag (cf. Gottlob et al. [14]). Note that if a hyperedge contains all the vertices of a hypergraph, then the hypergraph is acyclic and all the above mentioned hypergraph parameters equal 1. In the following we show that satisfiability remains NP-hard for CNF formulas with acyclic associated hypergraphs. In particular, let F be an arbitrary CNF formula and let x be a new variable not occurring in F. Now consider the CNF formula F' obtained from F by adding the clause $C = var(F) \cup \{x\}$. Since x is a pure literal, F and F' are equivalent with respect to satisfiability. The *primal hypergraph* of F', obtained by dropping negations and considering clauses as hyperedges, is acyclic. Hence satisfiability remains NP-hard for instances with acyclic primal hypergraphs. A similar construction can be applied with respect to the *dual hypergraph* whose vertices are the clauses and which contains for every variable y a hyperedge consisting of all the clauses that contain y or $\neg y$. Again, let F be an arbitrary CNF formula. Take a new variable x and obtain from F the formula F' by replacing every clause C with $C' = C \cup \{x\}$ and by adding the unit clause $\{\neg x\}$. Clearly F and F' are equivalent with respect to satisfiability. The dual hypergraph of F' is acyclic. Hence satisfiability remains NP-hard for instances with acyclic dual hypergraphs.

2 Preliminaries

2.1 Tree-Decompositions

Let G be a graph, T a tree, and χ a labeling of the vertices of T by sets of vertices of G. We refer to the vertices of T as "nodes" to avoid confusion with the vertices of G. The tuple (T, χ) is a *tree-decomposition* of G if the following three conditions hold:

1. For every vertex $v \in V(G)$ there exists a node $t \in V(T)$ such that $v \in \chi(t)$.
2. For every edge $vw \in E(G)$ there exists a node $t \in V(T)$ such that $v, w \in \chi(t)$.
3. For any three nodes $t_1, t_2, t_3 \in V(T)$, if t_2 lies on the path from t_1 to t_3, then $\chi(t_1) \cap \chi(t_3) \subseteq \chi(t_2)$ ("Connectedness Condition").

The *width* of a tree-decomposition (T, χ) is defined by $\max_{t \in V(T)} |\chi(t)| - 1$. The *treewidth* $tw(G)$ of a graph G is the minimum width over all its tree-decompositions. For constant k, there exists a linear-time algorithm that checks whether a given graph has treewidth at most k and, if so, outputs a tree-decomposition of minimum width [5]. However, the huge constant factor in the runtime of this algorithm makes it practically infeasible. For our purposes, however, it suffices to obtain tree-decompositions of small but not necessarily minimal width. There exist several powerful tree-decomposition heuristics that construct tree-decompositions of small width for many cases that are relevant in practice [6,20].

In this paper we also consider a special type of tree-decompositions. The triple (T, χ, r) is a *nice tree-decomposition* of G if (T, χ) is a tree-decomposition, the tree T is rooted at node r, and the following three conditions hold [17]:

1. Every node of T has at most two children.
2. If a node t of T has two children t_1 and t_2, then $\chi(t) = \chi(t_1) = \chi(t_2)$; in that case we call t a *join node*.
3. If a node t of T has exactly one child t', then exactly one of the following prevails:

(a) $|\chi(t)| = |\chi(t')| + 1$ and $\chi(t') \subset \chi(t)$; in that case we call t an *introduce node*.
(b) $|\chi(t)| = |\chi(t')| - 1$ and $\chi(t) \subset \chi(t')$; in that case we call t a *forget node*.

It is known that one can transform efficiently any tree-decomposition of width k of a graph with n vertices into a nice tree-decomposition of width at most k and at most $4n$ nodes [17].

Let (T, χ, r) be a nice tree-decomposition of a graph G. For each node t of T let T_t denote the subtree of T rooted at t; furthermore, let G_t denote the subgraph of G that is induced by the set $V_t = \bigcup_{t' \in V(T_t)} \chi(t')$ of vertices. Observe that $(T_t, \chi|_{V(T_t)}, t)$ is a nice tree-decomposition of G_t.

2.2 Propositional Formulas

We consider propositional formulas F in conjunctive normal form (CNF) represented as set of clauses. Each clause in F is a finite set of *literals*, and a literal is a negated or unnegated propositional *variable*. For example,

$$F = \{\{\neg x, y, z\}, \{\neg y, \neg z\}, \{x, \neg y\}\}$$

represents the propositional formula $(\neg x \vee y \vee z) \wedge (\neg y \vee \neg z) \wedge (x \vee \neg y)$. For a clause C we denote by $var(C)$ the set of variables that occur (negated or unnegated) in C; for a formula F we put $var(F) = \bigcup_{C \in F} var(C)$. The *size* of a clause is its cardinality.

A *truth assignment* is a mapping $\tau : X \to \{0, 1\}$ defined on some set X of variables. We extend τ to literals by setting $\tau(\neg x) = 1 - \tau(x)$ for $x \in X$. A truth assignment $\tau : X \to \{0, 1\}$ *satisfies* a clause C if for some variable $x \in var(C) \cap X$ we have $x \in C$ and $\tau(x) = 1$, or $\neg x \in C$ and $\tau(x) = 0$. A truth assignment $\tau : X \to \{0, 1\}$ *falsifies* a clause C if $var(C) \subseteq X$ and for every variable $x \in var(C)$ we have $x \in C$ and $\tau(x) = 0$, or $\neg x \in C$ and $\tau(x) = 1$. An assignment satisfies (resp. falsifies) a set A of clauses if it satisfies (resp. falsifies) every clause in A. A set A of clauses is *satisfiable* (resp. *falsifiable*) if there exists a truth assignment that satisfies (resp. falsifies) A; otherwise F is *unsatisfiable* (resp. *unfalsifiable*). Note that a set A of clauses is unfalsifiable if and only if the union of A contains a complementary pair of literals. For a formula F, we call a truth assignment $\tau : var(F) \to \{0, 1\}$ a *model* of F if τ satisfies F. We denote by $\#(F)$ the number of models of F. Thus, F is satisfiable if and only if $\#(F) \geq 1$. The *propositional satisfiability problem* SAT is the problem of deciding whether a given propositional formula in CNF is satisfiable. The *propositional model counting problem* #SAT is the problem of computing $\#(F)$ of a given propositional formula F in CNF.

2.3 Primal, Dual, and Incidence Treewidth

The *primal graph* $G(F)$ of a CNF formula F is the graph with vertex set $var(F)$; two variables x, y are joined by an edge if and only if $x, y \in var(C)$ for some clause $C \in F$. The *primal treewidth* (or *treewidth*, for short) $tw(F)$ of a CNF formula F is the treewidth of its primal graph, that is $tw(F) = tw(G(F))$.

The *dual graph* $G^d(F)$ of a CNF formula F is the graph with vertex set F; two clauses C, C' are joined by an edge if and only if $var(C) \cap var(C') \neq \emptyset$. The *dual treewidth* $tw^d(F)$ of a CNF formula F is the treewidth of its dual graph, that is $tw^d(F) = tw(G^d(F))$.

The *incidence graph* $G^*(F)$ of a CNF formula F is the bipartite graph with vertex set $F \cup var(F)$; a variable x and a clause C are joined by an edge if and only if $x \in var(C)$. The *incidence treewidth* $tw^*(F)$ of a CNF formula F is the treewidth of its incidence graph, that is $tw^*(F) = tw(G^*(F))$.

3 The Fixed-Parameter Algorithms

Since the number of models of a CNF formula can be exponential in the number of its variables (and thus may become too large to be stored in a single data word), we consider in the following the bit complexity of our algorithms, i.e., instead of assuming that arithmetic operations have constant runtime we bound their runtime by the number of bit operations (cf. Aho, Hopcroft, and Ullman [1], pages 22–23). To this aim, we introduce δ to denote the runtime of multiplying two n-bit integers, the computationally most expensive arithmetic operation in our algorithms. In the literature there exist several algorithms for multiplying two n-bit integers; we refer the interested reader to Knuth's in-depth overview [18]. One of the most prominent of these algorithms is due to Schönhage and Strassen [18,27] and runs in time $\mathcal{O}(n \log n \log \log n)$. Thus, we can assume that $\delta = \mathcal{O}(n \log n \log \log n)$, where n is the number of variables of the given CNF formula. Recently, Fürer [13] presented an even faster algorithm for integer multiplication. If arithmetic operations are assumed to have constant runtime, that is, $\delta = \mathcal{O}(1)$, we easily obtain the runtimes listed in the introduction.

Due to space restrictions, proofs of the lemmas in this section have been omitted.

3.1 Parameter Primal Treewidth

For this section, let (T, χ, r) be a nice tree-decomposition of the primal graph $G(F)$ of a CNF formula F. Let k denote the width of (T, χ, r) and let t be a node of T. For each truth assignment $\alpha : \chi(t) \rightarrow \{0,1\}$ we define $N(t, \alpha)$ as the set of truth assignments $\tau : V_t \rightarrow \{0,1\}$ for which the following two conditions hold:

1. $\tau(x) = \alpha(x)$ for all variables $x \in \chi(t)$.
2. There is no clause in F that is falsified by τ.

We represent the values of $n(t, \alpha) = |N(t, \alpha)|$ for all $\alpha : \chi(t) \rightarrow \{0,1\}$ by a table M_t with $|\chi(t)| + 1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding $\alpha(x)$ for variables $x \in \chi(t)$. The last entry of each row contains the integer $n(t, \alpha)$.

Lemma 1. *Let t be a* join node *of T with children t_1, t_2. Then, for each truth assignment $\alpha : \chi(t) \rightarrow \{0,1\}$, we have*

$$n(t, \alpha) = n(t_1, \alpha) \cdot n(t_2, \alpha).$$

Lemma 2. *Let t be an* introduce node *with child t' and $\chi(t) = \chi(t') \cup \{x\}$ for a variable x. Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) = \begin{cases} 0 & \text{if } \alpha \text{ falsifies some } C \in F; \\ n(t', \alpha|_{\chi(t')}) & \text{otherwise.} \end{cases}$$

Lemma 3. *Let t be a* forget node *with child t' and $\chi(t) = \chi(t') \setminus \{x\}$ for a variable x. Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) = n(t', \alpha \cup \{(x,0)\}) + n(t', \alpha \cup \{(x,1)\}).$$

Lemma 4. *Let t be a* leaf node. *Then, for each truth assignment $\alpha : \chi(t) \to \{0,1\}$, we have*

$$n(t,\alpha) = \begin{cases} 0 & \text{if } \alpha \text{ falsifies some } C \in F; \\ 1 & \text{otherwise.} \end{cases}$$

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

Lemma 5. *Let t be a node of T. Given the tables of the children of t, we can compute the table M_t in time $\mathcal{O}(2^k(kd + \delta))$, where d is the maximum number of occurrences over all variables.*

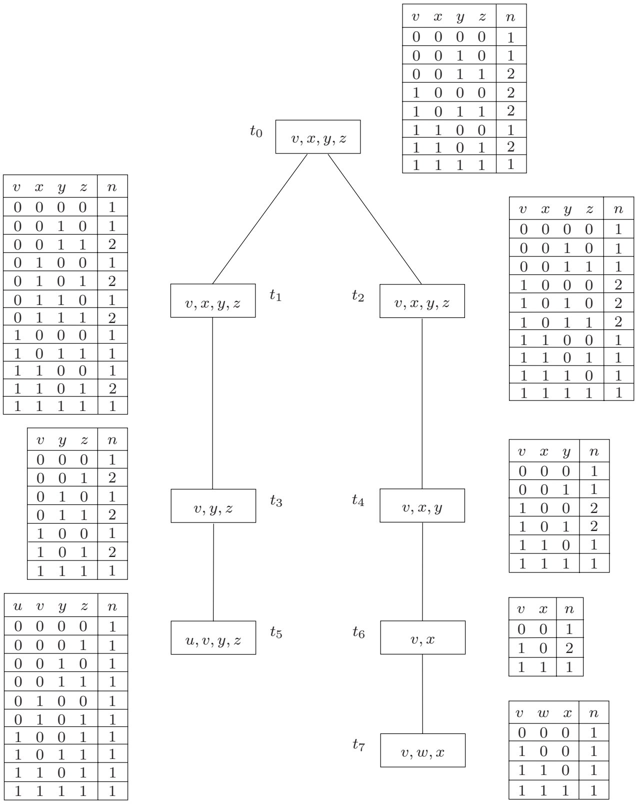
Theorem 1. *Given a nice tree-decomposition of the primal graph of a CNF formula F, we can compute $\#(F)$ in time $\mathcal{O}(2^k(kd + \delta)\,N)$; d denotes the maximum number of occurrences over all variables in F, k denotes the width and N the number of nodes of the tree-decomposition.*

Proof. Let (T, χ, r) be a nice tree-decomposition of the primal graph of F; let k and N be the width and number of nodes of (T, χ, r) respectively. Starting from the leaf nodes of T, we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 5 in time $\mathcal{O}(2^k(kd + \delta))$. Since we have

$$\#(F) = \sum_{\alpha:\chi(r)\to\{0,1\}} n(r,\alpha),$$

we can read off $\#(F)$ from the table M_r at the root r. □

An example of this algorithm on the tree-decomposition of the primal graph in Figure 1 is shown in Figure 2. Note that, for simplicity, we have omitted those rows from the tables where $n(t,\alpha) = 0$. From table M_{t_0} we can read off that there are exactly $1+1+2+2+2+1+2+1 = 12$ models of the corresponding CNF formula. Let us remark that our above algorithm is related to Yannakakis' algorithm [29] for deciding whether an acyclic constraint satisfaction instance has a solution.



v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	2
1	0	0	0	2
1	0	1	1	2
1	1	0	0	1
1	1	0	1	2
1	1	1	1	1

v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	2
0	1	0	0	1
0	1	0	1	2
0	1	1	0	1
0	1	1	1	2
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	2
1	1	1	1	1

v	x	y	z	n
0	0	0	0	1
0	0	1	0	1
0	0	1	1	1
1	0	0	0	2
1	0	1	0	2
1	0	1	1	2
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

v	y	z	n
0	0	0	1
0	0	1	2
0	1	0	1
0	1	1	2
1	0	0	1
1	0	1	2
1	1	1	1

v	x	y	n
0	0	0	1
0	0	1	1
1	0	0	2
1	0	1	2
1	1	0	1
1	1	1	1

u	v	y	z	n
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

v	x	n
0	0	1
1	0	2
1	1	1

v	w	x	n
0	0	0	1
1	0	0	1
1	1	0	1
1	1	1	1

Fig. 2. Solving #SAT on a nice tree-decomposition of the primal graph

3.2 Parameter Dual Treewidth

For this section, let (T, χ, r) be a nice tree-decomposition of the dual graph $G^d(F)$ of a CNF formula F. Let k denote the width of (T, χ, r) and let t be a node of T. For each subset $A \subseteq \chi(t)$ we define $N(t, A)$ as the set of truth assignments $\tau : \mathit{var}(V_t) \rightarrow \{0, 1\}$ for which the following two conditions hold:

1. Every clause in A is falsified by τ.
2. Every clause in $V_t \setminus \chi(t)$ is satisfied by τ.

We represent the values of $n(t, A) = |N(t, A)|$ for all $A \subseteq \chi(t)$ by a table M_t with $|\chi(t)| + 1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding membership of C in A for clauses $C \in \chi(t)$. The last entry of each row contains the integer $n(t, A)$.

Lemma 6. *Let t be a* join node *of T with children t_1, t_2. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t, A) = \frac{n(t_1, A) \cdot n(t_2, A)}{2^{|\mathit{var}(\chi(t)) \setminus \mathit{var}(A)|}}.$$

Lemma 7. *Let t be an* introduce node *with child t' and $\chi(t) = \chi(t') \cup \{C\}$ for a clause C. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t,A) = \begin{cases} 0 & \text{if } A \text{ is unfalsifiable;} \\ n(t',A) \cdot 2^{|var(C) \setminus var(\chi(t'))|} & \text{otherwise, if } C \notin A; \\ \dfrac{n(t', A \setminus \{C\})}{2^{|var(C) \cap (var(\chi(t')) \setminus var(A \setminus \{C\}))|}} & \text{otherwise, if } C \in A. \end{cases}$$

Lemma 8. *Let t be a* forget node *with child t' and $\chi(t) = \chi(t') \setminus \{C\}$ for a clause C. Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t,A) = n(t',A) - n(t', A \cup \{C\}).$$

Lemma 9. *Let t be a* leaf node. *Then, for each set $A \subseteq \chi(t)$, we have*

$$n(t,A) = \begin{cases} 0 & \text{if } A \text{ is unfalsifiable;} \\ 2^{|var(\chi(t)) \setminus var(A)|} & \text{otherwise.} \end{cases}$$

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

Lemma 10. *Let t be a node of T. Given the tables of the children of t, we can compute the table M_t in time $\mathcal{O}(2^k(kl+\delta))$, where l is the size of a largest clause.*

Theorem 2. *Given a nice tree-decomposition of the dual graph of a CNF formula F, we can compute $\#(F)$ in time $\mathcal{O}(2^k(kl+\delta)\,N)$; l denotes the size of a largest clause, k denotes the width and N the number of nodes of the tree-decomposition.*

Proof. Let (T,χ,r) be a nice tree-decomposition of the dual graph of F; let k and N be the width and number of nodes of (T,χ,r) respectively. Starting from the leaf nodes of T we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 10 in time $\mathcal{O}(2^k(kl+\delta))$. Now we show how to compute $\#(F)$ from table M_r at the root r. To this aim, recall that $n(r,A)$ is the number of truth assignments $\tau : var(F) \to \{0,1\}$ such that every clause in A is falsified by τ and every clause in $F \setminus \chi(r)$ is satisfied by τ. Thus, it is easy to see that we can compute the number of models of F from the entries of M_r in the following way:

$$\#(F) = \sum_{i=0}^{|\chi(r)|} \Big((-1)^i \sum_{A \subseteq \chi(r),\, |A|=i} n(r,A) \Big)$$

We can do this by going through all at most 2^{k+1} choices of $A \subseteq \chi(r)$: Starting with an initial value of 0, we add or subtract $n(r,A)$, depending on whether the cardinality of A is even or odd. This can be done in time $\mathcal{O}(2^k(k+\delta))$. □

An example of this algorithm on the tree-decomposition of the dual graph in Figure 1 is shown in Figure 3. Note that, for simplicity, we have omitted those rows from the tables where $n(t,A) = 0$. From table M_{t_0} we can read off that there are exactly $36 - 6 - 12 - 8 + 2 = 12$ models of the corresponding CNF formula.

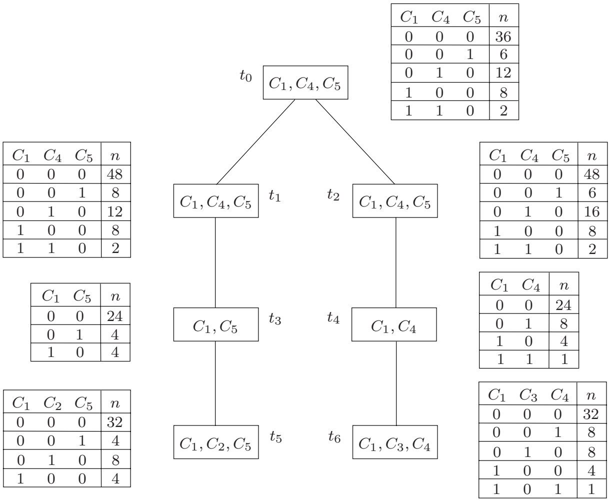

t_0 C_1, C_4, C_5
t_1 C_1, C_4, C_5
t_2 C_1, C_4, C_5
t_3 C_1, C_5
t_4 C_1, C_4
t_5 C_1, C_2, C_5
t_6 C_1, C_3, C_4
C_1 C_4 C_5 n: 0 0 0 36; 0 0 1 6; 0 1 0 12; 1 0 0 8; 1 1 0 2
C_1 C_4 C_5 n: 0 0 0 48; 0 0 1 8; 0 1 0 12; 1 0 0 8; 1 1 0 2
C_1 C_4 C_5 n: 0 0 0 48; 0 0 1 6; 0 1 0 16; 1 0 0 8; 1 1 0 2
C_1 C_5 n: 0 0 24; 0 1 4; 1 0 4
C_1 C_4 n: 0 0 24; 0 1 8; 1 0 4; 1 1 1
C_1 C_2 C_5 n: 0 0 0 32; 0 0 1 4; 0 1 0 8; 1 0 0 4
C_1 C_3 C_4 n: 0 0 0 32; 0 0 1 8; 0 1 0 8; 1 0 0 4; 1 0 1 1

Fig. 3. Solving #SAT on a nice tree-decomposition of the dual graph

3.3 Parameter Incidence Treewidth

For this section, let (T, χ, r) be a nice tree-decomposition of the incidence graph $G^*(F)$ of a CNF formula F. Let k denote the width of (T, χ, r).

For each node t of T, let F_t denote the set consisting of all the clauses in V_t, and let X_t denote the set of all variables in V_t, i.e., $F_t = V_t \cap F$ and $X_t = V_t \cap var(F)$. We also use the shorthands $\chi_c(t) = \chi(t) \cap F$ and $\chi_v(t) = \chi(t) \cap var(F)$ for the set of variables and the set of clauses in $\chi(t)$, respectively.

Let t be a node of T. For each truth assignment $\alpha : \chi_v(t) \to \{0,1\}$ and subset $A \subseteq \chi_c(t)$ we define $N(t, \alpha, A)$ as the set of truth assignments $\tau : X_t \to \{0,1\}$ for which the following two conditions hold:

1. $\tau(x) = \alpha(x)$ for all variables $x \in \chi_v(t)$.
2. A is exactly the set of clauses in F_t that are not satisfied by τ.

We represent the values of $n(t, \alpha, A) = |N(t, \alpha, A)|$ for all $\alpha : \chi_v(t) \to \{0,1\}$ and $A \subseteq \chi_c(t)$ by a table M_t with $|\chi(t)| + 1$ columns and $2^{|\chi(t)|}$ rows. The first $|\chi(t)|$ columns of M_t contain Boolean values encoding $\alpha(x)$ for variables $x \in \chi_v(t)$, and membership of C in A for clauses $C \in \chi_c(t)$. The last entry of each row contains the integer $n(t, \alpha, A)$.

Lemma 11. *Let t be a* join node *of T with children t_1, t_2. Then, for each truth assignment $\alpha : \chi_v(t) \to \{0,1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t, \alpha, A) = \sum_{A_1, A_2 \subseteq \chi_c(t),\ A_1 \cap A_2 = A} n(t_1, \alpha, A_1) \cdot n(t_2, \alpha, A_2).$$

Lemma 12. *Let t be an* introduce node *with child t'.*
(a) *If $\chi(t) = \chi(t') \cup \{x\}$ for a variable x, then, for each truth assignment $\alpha : \chi_v(t') \to \{0,1\}$ and set $A \subseteq \chi_c(t)$, we have*

$$n(t,\alpha\cup\{(x,0)\},A)=\begin{cases}0 & \textit{if } \neg x\in C \textit{ for some clause } C\in A;\\ \sum_{B'\subseteq B} n(t',\alpha,A\cup B') & \textit{otherwise, where}\\ & B=\{C\in\chi_c(t)\mid \neg x\in C\};\end{cases}$$

$$n(t,\alpha\cup\{(x,1)\},A)=\begin{cases}0 & \textit{if } x\in C \textit{ for some clause } C\in A;\\ \sum_{B'\subseteq B} n(t',\alpha,A\cup B') & \textit{otherwise, where}\\ & B=\{C\in\chi_c(t)\mid x\in C\}.\end{cases}$$

(b) *If* $\chi(t)=\chi(t')\cup\{C\}$ *for a clause* C*, then, for each truth assignment* $\alpha:\chi_v(t)\to\{0,1\}$ *and set* $A\subseteq\chi_c(t)$*, we have*

$$n(t,\alpha,A)=\begin{cases}n(t',\alpha,A) & \textit{if } C\notin A \textit{ and } \alpha \textit{ satisfies } C;\\ n(t',\alpha,A\setminus\{C\}) & \textit{if } C\in A \textit{ and } \alpha \textit{ does not satisfy } C;\\ 0 & \textit{otherwise.}\end{cases}$$

Lemma 13. *Let* t *be a* forget node *with child* t'.
(a) *If* $\chi(t)=\chi(t')\setminus\{x\}$ *for a variable* x*, then, for each truth assignment* $\alpha:\chi_v(t)\to\{0,1\}$ *and set* $A\subseteq\chi_c(t)$*, we have*

$$n(t,\alpha,A)=n(t',\alpha\cup\{(x,0)\},A)+n(t',\alpha\cup\{(x,1)\},A).$$

(b) *If* $\chi(t)=\chi(t')\setminus\{C\}$ *for a clause* C*, then, for each truth assignment* $\alpha:\chi_v(t)\to\{0,1\}$ *and set* $A\subseteq\chi_c(t)$*, we have*

$$n(t,\alpha,A)=n(t',\alpha,A).$$

Lemma 14. *Let* t *be a* leaf node*. Then, for each truth assignment* $\alpha:\chi_v(t)\to\{0,1\}$ *and set* $A\subseteq\chi_c(t)$*, we have*

$$n(t,\alpha,A)=\begin{cases}1 & \textit{if } A=\{\,C\in\chi_c(t)\mid \alpha \textit{ does not satisfy } C\,\};\\ 0 & \textit{otherwise.}\end{cases}$$

By using these equalities, we can now construct the tables M_t from the leaves to the root according to the following lemma.

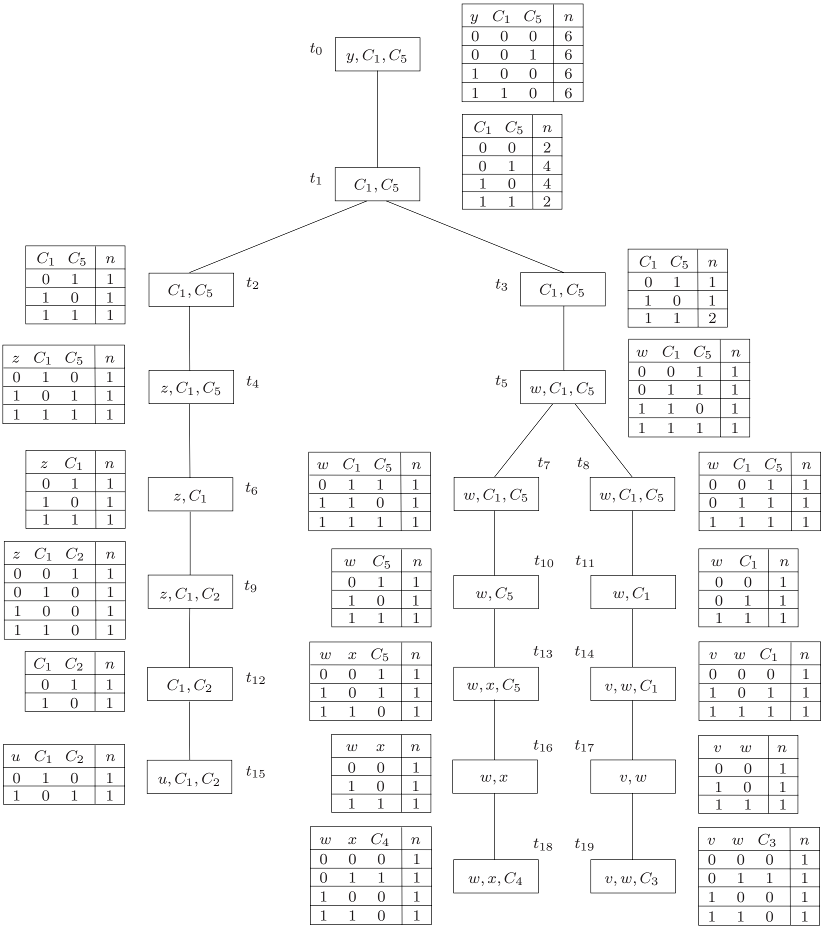
Lemma 15. *Let* t *be a node of* T*. Given the tables of the children of* t*, we can compute the table* M_t *in time* $\mathcal{O}(2^k(kl+2^k(k+\delta)))$*, where* l *is the size of a largest clause.*

Theorem 3. *Given a nice tree-decomposition of the incidence graph of a CNF formula* F*, we can compute* $\#(F)$ *in time* $\mathcal{O}(2^k(kl+2^k(k+\delta))N)$*;* l *denotes the size of a largest clause,* k *denotes the width and* N *the number of nodes of the tree-decomposition.*

Proof. Let (T,χ,r) be a nice tree-decomposition of the incidence graph of F; let k and n be the width and number of nodes of (T,χ,r) respectively. Starting from the leaf nodes of T we compute the tables M_t for all nodes t of T in a bottom-up ordering. Each table can be computed by Lemma 15 in time $\mathcal{O}(2^k(kl+2^k(k+\delta)))$. Since we have

$$\#(F)=\sum_{\alpha:\chi_v(r)\to\{0,1\}} n(r,\alpha,\emptyset),$$

we can read off $\#(F)$ from the table M_r at the root r. ☐


t_0 y, C_1, C_5
t_1 C_1, C_5
t_2 C_1, C_5
t_3 C_1, C_5
t_4 z, C_1, C_5
t_5 w, C_1, C_5
t_6 z, C_1
t_7 w, C_1, C_5
t_8 w, C_1, C_5
t_9 z, C_1, C_2
t_{10} w, C_5
t_{11} w, C_1
t_{12} C_1, C_2
t_{13} w, x, C_5
t_{14} v, w, C_1
t_{15} u, C_1, C_2
t_{16} w, x
t_{17} v, w
t_{18} w, x, C_4
t_{19} v, w, C_3

y	C_1	C_5	n
0	0	0	6
0	0	1	6
1	0	0	6
1	1	0	6

C_1	C_5	n
0	0	2
0	1	4
1	0	4
1	1	2

C_1	C_5	n
0	1	1
1	0	1
1	1	1

C_1	C_5	n
0	1	1
1	0	1
1	1	2

z	C_1	C_5	n
0	1	0	1
1	0	1	1
1	1	1	1

w	C_1	C_5	n
0	0	1	1
0	1	1	1
1	1	0	1
1	1	1	1

z	C_1	n
0	1	1
1	0	1
1	1	1

w	C_1	C_5	n
0	1	1	1
1	1	0	1
1	1	1	1

w	C_1	C_5	n
0	0	1	1
0	1	1	1
1	1	1	1

z	C_1	C_2	n
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	1

w	C_5	n
0	1	1
1	0	1
1	1	1

w	C_1	n
0	0	1
0	1	1
1	1	1

C_1	C_2	n
0	1	1
1	0	1

w	x	C_5	n
0	0	1	1
1	0	1	1
1	1	0	1

v	w	C_1	n
0	0	0	1
1	0	1	1
1	1	1	1

u	C_1	C_2	n
0	1	0	1
1	0	1	1

w	x	n
0	0	1
1	0	1
1	1	1

v	w	n
0	0	1
1	0	1
1	1	1

w	x	C_4	n
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	1

v	w	C_3	n
0	0	0	1
0	1	1	1
1	0	0	1
1	1	0	1

Fig. 4. Solving #SAT on a nice tree-decomposition of the incidence graph

An example of this algorithm on the tree-decomposition of the incidence graph in Figure 1 is shown in Figure 4. Note that, for simplicity, we have omitted those rows from the tables where $n(t, \alpha, A) = 0$. From table M_{t_0} we can read off that there are exactly $6 + 6 = 12$ models of the corresponding CNF formula.

3.4 Space Requirements

When we perform dynamic programming on a nice tree-decomposition we traverse the nodes of the tree in an arbitrary bottom-up ordering. When we compute the table of a node we can assume that the tables at its children are already computed and are currently kept in memory. Once the table of a node is computed, the tables of its children can be discarded. Thus, at some point, when the table of a node is computed, all tables of its children are simultaneously in memory; we will refer to this scheme of table computation as the "simultaneous updating scheme."

A variant of this scheme was considered by Aspvall, Proskurowski, and Telle [2], not requiring that the tables of children of a node are present simultaneously; the parent table is updated whenever a child table becomes available. We will refer to the scheme of Aspvall et al. as the "sequential updating scheme." In view of the updating functions for join nodes as defined in Lemmas 1, 6, and 11, respectively, one can use the sequential updating scheme for the primal and dual treewidth algorithms. The incidence treewidth algorithm, however, requires the simultaneous updating scheme.

The following algorithm carries out the simultaneous updating scheme on a nice tree-decomposition (T, χ, r); the algorithm also computes for every node t the number $\rho(t)$ of tables required simultaneously to compute the table M_t. The algorithm is recursive, initially $t = r$.

1. Clearly $\rho(t) = 1$ if t is a leaf; M_t can be computed independently.
2. If t has only one child t', then recurse on the subtree $T_{t'}$ rooted at t' and compute the table $M_{t'}$ and the number $\rho(t')$. Now discard all tables of nodes below t' and compute the table M_t; then discard $M_{t'}$. This gives $\rho(t) = \max(2, \rho(t'))$.
3. If t has two children t' and t'', then compute $\rho(t')$ and $\rho(t'')$; w.l.o.g., assume $\rho(t') \geq \rho(t'')$. First recurse on $T_{t'}$ and compute the table $M_{t'}$; discard all tables below t' and keep $M_{t'}$ in memory. Next recurse on $T_{t''}$ to compute the table $M_{t''}$; discard all tables below t'' and keep $M_{t''}$ in memory. Now compute the table M_t using the tables $M_{t'}$ and $M_{t''}$; afterwards discard the tables $M_{t'}$ and $M_{t''}$. This gives $\rho(t) = \max(3, \rho(t'), \rho(t'') + 1)$.

Aspvall et al. [2] show that if T has N nodes than the sequential updating scheme requires not more than $\lfloor \log_2 \frac{4}{3}(N + 1) \rfloor$ tables at any point of the computation. Using a similar reasoning one can easily show that the simultaneous updating scheme requires not more than $\lfloor 1 + \log_2(N + 1) \rfloor$ tables at any point of the computation, in particular when the algorithm outlined above is applied. Thus the simultaneous updating scheme requires at most one more table than the sequential one. This result suggests the use of the simultaneous updating scheme for all three algorithms as it is slightly more convenient to implement without requiring significantly more space.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms chapter 1, Models of computation, pp. 1–41. Addison-Wesley, Reading (1974)
2. Aspvall, B., Proskurowski, A., Telle, J.A.: Memory requirements for table computations in partial k-tree algorithms. Algorithmica 27(3-4), 382–394 (2000)
3. Bacchus, F., Dalmao, S., Pitassi, T.: Algorithms and complexity results for #SAT and Bayesian inference. In: FOCS 2003, pp. 340–351. IEEE Computer Society Press, Los Alamitos (2003)
4. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–21 (1993)
5. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
6. Bodlaender, H.L.: Discovering treewidth. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 1–16. Springer, Heidelberg (2005)

7. Cohen, D., Jeavons, P., Gyssens, M.: A unified theory of structural tractability for constraint satisfaction and spread cut decomposition. In: IJCAI 2005, pp. 72–77. Professional Book Center (2005)
8. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory of Computing Systems 33(2), 125–150 (2000)
9. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics 108(1-2), 23–52 (2001)
10. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
11. Fischer, E., Makowsky, J.A., Ravve, E.R.: Counting truth assignments of formulas of bounded tree-width or clique-width. Discrete Applied Mathematics (in press)
12. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
13. Fürer, M.: Faster integer multiplication. In: STOC 2007, pp. 57–66. ACM Press, New York (2007)
14. Gottlob, G., Leone, N., Scarcello, F.: Hypertree decompositions and tractable queries. Journal of Computer and System Sciences 64(3), 579–627 (2002)
15. Gottlob, G., Scarcello, F., Sideri, M.: Fixed-parameter complexity in AI and nonmonotonic reasoning. Artificial Intelligence 138(1-2), 55–86 (2002)
16. Grohe, M., Marx, D.: Constraint solving via fractional edge covers. In: SODA 2006, pp. 289–298. ACM Press, New York (2006)
17. Kloks, T.: Treewidth: Computations and Approximations. Springer, Heidelberg (1994)
18. Knuth, D.E.: The Art of Computer Programming. In: Seminumerical Algorithms, chapter 4.3.3 How fast can we multiply? 3rd edn., vol. 2, pp. 294–318. Addison-Wesley, Reading (1998)
19. Kolaitis, P.G., Vardi, M.Y.: Conjunctive-query containment and constraint satisfaction. Journal of Computer and System Sciences 61(2), 302–332 (2000)
20. Koster, A.M.C.A., Bodlaender, H.L., van Hoesel, S.P.M.: Treewidth: Computational experiments. Electronic Notes in Discrete Mathematics, 8 (2001)
21. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
22. Nishimura, N., Ragde, P., Szeider, S.: Solving #SAT using vertex covers. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 396–409. Springer, Heidelberg (2006)
23. Oum, S., Seymour, P.: Approximating clique-width and branch-width. Journal of Combinatorial Theory, Series B 96(4), 514–528 (2006)
24. Robertson, N., Seymour, P.D.: Graph minors X. Obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52(2), 153–190 (1991)
25. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82(1-2), 273–302 (1996)
26. Samer, M., Szeider, S.: Constraint satisfaction with bounded treewidth revisited. In: Benhamou, F. (ed.) CP 2006. LNCS, vol. 4204, pp. 499–513. Springer, Heidelberg (2006)
27. Schönhage, A., Strassen, V.: Schnelle Multiplikation ganzer Zahlen. Computing 7, 281–292 (1971)
28. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)
29. Yannakakis, M.: Algorithms for acyclic database schemes. In: VLDB 1981, pp. 81–94. IEEE Computer Society Press, Los Alamitos (1981)

Completeness for Flat Modal Fixpoint Logics

(Extended Abstract)

Luigi Santocanale[1,⋆] and Yde Venema[2,⋆⋆]

[1] Laboratoire d'Informatique Fondamentale de Marseille, Université de Provence,
`luigi.santocanale@lif.univ-mrs.fr`
[2] Institute for Logic, Language and Computation, Universiteit van Amsterdam
`yde@science.uva.nl`

Abstract. Given a set Γ of modal formulas of the form $\gamma(x, \boldsymbol{p})$, where x occurs positively in γ, the language $\mathcal{L}_\sharp(\Gamma)$ is obtained by adding to the language of polymodal logic $\mathbf{K}$ connectives $\sharp_\gamma$, $\gamma \in \Gamma$. Each term $\sharp_\gamma$ is meant to be interpreted as the parametrized least fixed point of the functional interpretation of the term $\gamma(x)$. Given such a Γ, we construct an axiom system $\mathbf{K}_\sharp(\Gamma)$ which is sound and complete w.r.t. the concrete interpretation of the language $\mathcal{L}_\sharp(\Gamma)$ on Kripke frames. If Γ is finite, then $\mathbf{K}_\sharp(\Gamma)$ is a finite set of axioms and inference rules.

Keywords: fixpoint logic, modal algebra, completeness.

1 Introduction

Suppose that we extend the language of basic (poly-)modal logic with a set $\{\sharp_\gamma \mid \gamma \in \Gamma\}$ of so-called *fixpoint connectives*, which are defined as follows. Each connective $\sharp_\gamma$ is indexed by a modal formula $\gamma(x, \boldsymbol{p})$ in which x occurs only positively, and the intended meaning of the formula $\sharp_\gamma(\boldsymbol{p})$ in a labelled transition system (Kripke model) is the least fixpoint of the formula $\gamma(x, \boldsymbol{p})$,

$$\sharp_\gamma(\boldsymbol{p}) \equiv \mu x.\gamma(x, \boldsymbol{p}).$$

Many logics of interest in computer science are of this kind, such fixpoint connectives can be found for instance in **PDL** [6]: $\langle a^* \rangle = \mu x.p \vee \langle a \rangle x$, in **CTL** [4]: $EU(p, q) = \mu x.p \vee (q \wedge \Diamond x)$ and $AFp = \mu x.p \vee \Box x$, and in **LTL**. Generalizing these examples we arrive at the notion of flat modal fixpoint logic. Let $\mathcal{L}_\sharp(\Gamma)$ denote the language we obtain if we extend the syntax of (poly-)modal logic with a connective $\sharp_\gamma$ for every $\gamma \in \Gamma$. The flat modal fixpoint logic induced by Γ is the set of $\mathcal{L}_\sharp(\Gamma)$-validities, i.e., the collection of formulas in the language $\mathcal{L}_\sharp(\Gamma)$ that are true at every state of every Kripke model.

Clearly, every fixpoint connective of this kind can be seen as a macro over the language of the modal μ-calculus. Because the associated formula γ of a

---

[⋆] The research of this author supported by the ANR project SOAPDC.

[⋆⋆] The research of this author has been made possible by VICI grant 639.073.501 of the Netherlands Organization for Scientific Research (NWO).

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 499–513, 2007.
© Springer-Verlag Berlin Heidelberg 2007

fixpoint connective is itself a basic modal formula (which explains our name *flat*), it is easy to see that every flat modal fixpoint language corresponds to a relatively simple alternation-free fragment of the modal μ-calculus [7]. Despite this restrictive expressive power, flat modal fixpoint logics such as **CTL** and **LTL** are often preferred by end users, because of their transparency and simpler semantics. In fact, most verification tools implement some flat fixpoint logic rather than the full μ-calculus.

Up to now however, general investigations of flat modal fixpoint logics have not been pursued. Our research is driven by the wish to understanding the combinatorics of fixpoint logics in their wider algebraic and order theoretic setting. As such it continues earlier work by the first author [9,10]. In this paper we move on in this direction by addressing the problem of *uniformly* axiomatizing flat fixpoint logics. Concretely, our main contribution concerns an algorithm that, when given as input a (finite) set of positive formulas Γ, produces a (finite) axiom system $\mathbf{K}_\sharp(\Gamma)$ which is sound and complete w.r.t. the standard interpretation of the language $\mathcal{L}_\sharp(\Gamma)$ in Kripke frames. Note that this result does not follow from Walukiewicz' completeness result for the modal μ-calculus [11]. Rather, it should be interpreted by saying that we add to Walukiewicz' theorem the observation that, modulo a better choice of axioms, proofs of validities in a given flat fragments of the modal μ-calculus can be carried out *inside* this fragment.

Let us summarize the construction of and the ideas behind the axiom system $\mathbf{K}_\sharp(\Gamma)$. Mimicking Kozen's axiomatization of the modal μ-calculus, an intuitive axiomatization would be to add to a standard axiomatization $\mathbf{K}$ for (poly-)modal logic, the axiom and the derivation rule

$$\vdash \gamma(\sharp_\gamma(\boldsymbol{p}), \boldsymbol{p}) \to \sharp_\gamma(\boldsymbol{p})\,, \qquad (\sharp_\gamma\text{-prefix})$$

$$\text{from } \vdash \gamma(\varphi, \boldsymbol{p}) \to \varphi \text{ infer } \vdash \sharp_\gamma(\boldsymbol{p}) \to \varphi\,, \qquad (\sharp_\gamma\text{-least})$$

for each $\gamma \in \Gamma$. These axioms and rules express that $\sharp_\gamma(\boldsymbol{p})$ is the least prefixpoint of $\gamma(-, \boldsymbol{p})$. The proof we present reveals that this is already a complete axiomatization if all the formulas in Γ are *disjunctive* or *aconjunctive* in the sense of [11,7]. However, as soon as arbitrary formulas are considered, the usual problems on the use of conjunction within fixpoints arise obstructing the way to completeness.

The intuitive axiomatization – which we may well call *Kozen's* or Park's [5] axiomatization – may however be modified, and the Subset Construction [1, §9.5] suggests how to successfully do it. Roughly speaking, this is a procedure that transforms a $\gamma \in \Gamma$ into a *disjunctive* system of equations – called here $\mathcal{P}_+(T_\sharp^\gamma)$. It is shown in [1] that on *complete* lattices, the least solution of $\mathcal{P}_+(T_\sharp^\gamma)$ is constructible from the least fixed point of γ. The key idea of *our* axiomatization $\mathbf{K}_\sharp(\Gamma)$ is to *force* this relation to hold on arbitrary algebraic models, by imposing $\mathcal{P}_+(T_\sharp^\gamma)$ to have a least solution, constructible from $\sharp_\gamma$.

While our methodology is based on earlier work by the first author, we extend the results of [9] in two significant ways. First, the idea to use the subset construction of Arnold & Niwiński to *define an axiom system* for flat modal fixpoint logics, is novel. And second, the Representation Theorem presented in Section 6 strengthens the main result of [9], which applies to complete algebras only, to a completeness result for *Kripke frames*.

2 Preliminaries

We first give a formal definition of the syntax and semantics of flat modal fixpoint logics. We then discuss the reformulation of modal logic in terms of the cover modalities ∇_i. Finally, we introduce modal $\sharp$-algebras as the key structures of the algebraic setting in which we shall prove our completeness result. For background in the algebraic perspective on modal logic, see [2].

Flat Modal Fixpoint Logic. Throughout this paper we fix a set Γ of (poly-) modal formulas/terms $\gamma(x, \boldsymbol{p})$ where x occurs only positively, i.e. under no negation. The vector $\boldsymbol{p}$ may be different for each γ, but we decided not to make this explicit in the syntax, in order not to clutter up notation. We also fix a finite set I of atomic actions.

Definition 1. *The set $\mathcal{L}_\sharp(\Gamma)$ of flat modal fixpoint formulas associated with Γ is defined by the following grammar:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \Diamond_i \varphi \mid \sharp_\gamma(\boldsymbol{\varphi})$$

where i and γ range over I and Γ, respectively.

We move on to the intended semantics of this language. A labeled transition system of type I, or equivalently a Kripke model, is a structure $\mathbb{S} = \langle S, \{ R_i \mid i \in I \}\rangle$, where S is a set of states and $R_i \subseteq S \times S$ is, for each $i \in I$, a transition relation. Given a valuation $\boldsymbol{v} : P \longrightarrow \mathcal{P}(S)$ of propositional variables as subsets of states, we inductively define the semantics of flat modal fixed point formulas. Most of the inductive clauses are standard, for instance:

$$\|\Diamond_i \varphi\|_{\boldsymbol{v}} = \{ x \in S \mid \exists y \in S \text{ s.t. } xR_i y \text{ and } y \in \|\varphi\|_{\boldsymbol{v}} \}$$

In order to define $\|\sharp_\gamma(\boldsymbol{\varphi})\|_{\boldsymbol{v}}$, let x be a variable which is not free in $\boldsymbol{\varphi}$ and, for $Y \subseteq S$, let $(\boldsymbol{v}, x \to Y)$ be the valuation sending x to Y and every other variable y to $\boldsymbol{v}(y)$. We let

$$\|\sharp_\gamma(\boldsymbol{\varphi})\|_{\boldsymbol{v}} = \bigcap \{ Y \mid \|\gamma(x, \boldsymbol{\varphi})\|_{(\boldsymbol{v}, x \to Y)} \subseteq Y \}\,. \tag{1}$$

By the Knaster-Tarski theorem, Definition (1) just says that the interpretation of $\sharp_\gamma(\boldsymbol{\varphi})$ is the least fixed point of the order preserving function sending Y to $\|\gamma(x, \boldsymbol{\varphi})\|_{(\boldsymbol{v}, x \to Y)}$.

The Cover Modalities ∇_i. We will frequently work in a reformulation of the modal language based on the *cover modalities* ∇_i, $i \in I$. These connectives, taking a *set* of formulas as their argument, can be defined in terms of the box and diamond operators:

$$\nabla_i \Phi := \Box_i \bigvee \Phi \wedge \bigwedge \Diamond_i \Phi,$$

where $\Diamond_i \Phi$ denotes the set $\{\Diamond_i \varphi \mid \varphi \in \Phi\}$. Conversely, the standard diamond and box modalities can be defined in terms of the cover modality:

$$\Diamond_i \varphi \equiv \nabla_i \{\varphi, \top\} \qquad\qquad \Box_i \varphi \equiv \nabla_i \varnothing \vee \nabla_i \{\varphi\},$$

from which it follows that we may equivalently base our modal language on ∇_i as a primitive symbol. What makes the cover modality so useful is the distributive law:

$$\nabla_i \Phi \wedge \nabla_i \Phi' \equiv \bigvee_{Z \in \Phi \bowtie \Phi'} \nabla_i \{\, \varphi \wedge \varphi' \mid (\varphi, \varphi') \in Z \,\}, \tag{2}$$

where $\Phi \bowtie \Phi'$ denotes the set of relations $R \subseteq \Phi \times \Phi'$ that are *full* in the sense that for all $\varphi \in \Phi$ there is a $\varphi' \in \Phi'$ with $(\varphi, \varphi') \in R$, and vice versa. We mention two key corollaries of (2), but first we need some definitions.

Definition 2. *Let X, Y be sets of variables. Then we define the following sets of formulas/terms:*

1. *$Lit(X)$ is the set $\{x, \neg x \mid x \in X\}$ of literals over X,*
2. *$SC(X; Y)$ is the set of special conjunctions of the form $\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j \Phi_j$, where $\Lambda \subseteq Lit(X)$, $J \subseteq I$, and $\Phi_j \subseteq Y$ for each $j \in J$.*
3. *$D(X)$ is the set of disjunctive terms over X given by the following grammar:*

$$\varphi ::= x \mid \bot \mid \varphi \vee \varphi$$

4. *$DT(X)$ is the set of distributive terms over X given by the following grammar:*

$$\varphi ::= x \mid \bot \mid \varphi \vee \varphi \mid \top \mid \varphi \wedge \varphi$$

5. *$MT(X)$ is the set of modal terms over X given by the following grammar:*

$$\varphi ::= x \mid \neg x \mid \bot \mid \varphi \vee \varphi \mid \top \mid \varphi \wedge \varphi \mid \nabla_i \Phi$$

   where $i \in I$ and $\Phi \subseteq MT(X)$.

6. *$MT_\nabla(X)$ is the set of terms in* ∇-normal form *given by the following grammar:*

$$\varphi ::= \bot \mid \varphi \vee \varphi \mid \bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j \Phi_j,$$

   where $\Lambda \subseteq Lit(X)$, $J \subseteq I$, and $\Phi_j \subseteq MT_\nabla(X)$ for each $j \in J$.

Note the restricted use of conjunction in terms in ∇-normal form.

Proposition 3. *Let P and Φ be sets of proposition letters, and define $Y := \{y_\Psi \mid \Psi \subseteq \Phi\}$. There is an effective procedure associating with each modal formula $\varphi \in DT(SC(P; \Phi))$ a formula $\varphi^\vee \in D(SC(P; Y))$ such that φ is equivalent to the formula obtained from $\varphi^\vee$ by uniformly substituting each variable y_Ψ by the conjunction $\bigwedge \Psi$.*

Proposition 4. *Let X be a set of proposition letters. There is an effective procedure associating with each modal formula $\varphi \in MT(X)$ an equivalent formula $\varphi^- \in MT_\nabla(X)$.*

Modal Algebras. We now move on to the algebraic perspective on flat modal fixpoint logic. Recall that a *modal algebra* (of type I) is a structure of the form $A = \langle A, \bot, \top, \neg, \wedge, \vee, \{\Diamond_i^A \mid i \in I\}\rangle$, where each $\Diamond_i : A \to A$ preserves all finite joins of the Boolean algebra $\langle A, \bot, \top, \neg, \wedge, \vee\rangle$.

Definition 5. *Let $A = \langle A, \leq\rangle$ and $B = \langle B, \leq\rangle$ be two partial orders. Suppose that $f : A \to B$ and $g : B \to A$ are order-preserving maps such that $fa \leq b$ iff $a \leq gb$, for all $a \in A$ and $b \in B$. Then we call (f, g) an* adjoint pair*, and say that f is the* left adjoint *of, or* residuated *by, g, and that g is the* right adjoint*, or residual, of f. We say that f is an $\mathcal{O}$-*adjoint *if it satisfies the weaker property that for every $b \in B$ there is a finite set $G_f(b) \subseteq A$ such that*

$$fa \leq b \text{ iff } a \leq a' \text{ for some } a' \in G_f(b),$$

for all $a \in A$ and $b \in B$.

It is well known that left adjoint maps preserve all existing joins of a poset. Similarly, one may prove that $\mathcal{O}$-adjoints preserve all existing joins of *directed* sets.

Modal $\sharp$-Algebras. Given a modal algebra A, a modal formula $\gamma(x, p_1, \ldots, p_n)$ is interpreted as a map $\gamma^A : A \times A^n \to A$, its *term function*. Given a vector $\boldsymbol{b} = (a_1, \ldots, a_n) \in A^n$, we let $\gamma_{\boldsymbol{b}}^A : A \to A$ denote the map given by

$$\gamma_{\boldsymbol{b}}^A(a) := \gamma^A(b, \boldsymbol{a}). \tag{3}$$

Definition 6. *A modal $\sharp$-algebra is a modal algebra A endowed with an operation $\sharp_\gamma^A$ for each $\gamma \in \Gamma$ such that for each $\boldsymbol{b}$, $\sharp_\gamma^A(\boldsymbol{b})$ is the least fixpoint of $\gamma_{\boldsymbol{b}}^A$ as defined in (3).*

In this paper we will be mainly interested in two kinds of modal $\sharp$-algebras: the "concrete" or "semantic" ones that encode a Kripke frame, and the "axiomatic" ones that can be seen as algebraic versions of the axiom system $\mathbf{K}_\sharp$ to be defined in the next section. We first consider the concrete ones.

Definition 7. *Let $\mathbb{S} = \langle S, \{ R_i \mid i \in I \}\rangle$ be a transition system. Define, for each $i \in I$, the operation $\langle R_i\rangle$ by putting, for each $X \subseteq S$, $\langle R_i\rangle X = \{ y \in S \mid \exists x \in X \text{ s.t. } yR_i x \}$. The $\sharp$-complex algebra $\mathbb{S}^\sharp$ is given as the structure*

$$\langle \mathcal{P}(S), \varnothing, S, \overline{(\,\cdot\,)}, \cup, \cap, \{ \langle R_i\rangle \mid i \in I \}\rangle.$$

We will also call these structures Kripke $\sharp$-algebras*.*

Definition 8. *Let $A = \langle A, \leq\rangle$ be a partial order with least element $\bot$, and let $f : A \to A$ be an order-preserving map on A. For $k \in \omega$ and $a \in A$, we inductively define $f^k a$ by putting $f^0 a := a$ and $f^{k+1} a := f(f^k a)$. If f has a least fixpoint $\mu.f$, then we say that this least fixpoint is* constructive *if $\mu.f = \bigvee_{k \in \omega} f^k(\bot)$. A modal $\sharp$-algebra is called constructive if $\sharp_\gamma^A(\boldsymbol{b})$ is a constructive least fixpoint, for each $\gamma \in \Gamma$ and each $\boldsymbol{b}$ in A.*

We explain now why $\mathcal{O}$-adjoints are relevant for the theory of the least fixed point. If $f : A^n \longrightarrow A$ is an $\mathcal{O}$-adjoint, say that $V \subseteq A$ is f-closed if $y \in V$ and $\boldsymbol{a} = (a_1, \ldots, a_n) \in G_f(y)$ implies $a_i \in V$ for $i = 1, \ldots, n$. If $\mathcal{F}$ is a family of $\mathcal{O}$-adjoints $f : A^n \longrightarrow A$, say that V is $\mathcal{F}$-closed if it is f-closed for each $f \in \mathcal{F}$.

Definition 9. *A family of $\mathcal{O}$-adjoints $\mathcal{F} = \{ f_i : A^{n_i} \longrightarrow A \mid i \in I \}$ is said to be* finitary *if, for each $x \in A$, the least set $\mathcal{F}$-closed set containing x is finite. The $\mathcal{O}$-adjoint $f : A \longrightarrow A$ is finitary if the singleton $\{ f \}$ is finitary.*

Clearly, if f belongs to a finitary family, then it is finitary.

Proposition 10. *If $f : A \longrightarrow A$ is a finitary $\mathcal{O}$-adjoint, then its least prefixed point, whenever it exists, is constructive.*

3 The Axiomatization

The axiomatization we shall propose depends on what is informally called the *subset construction* [1, Theorem 9.3.4]. This transformation takes as input a set of modal terms and produces a set of modal terms in ∇-normal form that are equivalent – w.r.t. the respective least prefixed points – to the terms given in input. Since the transformation plays an essential role both in the proposed axiomatization as well as in the proof of its completeness, we recall it and, at the same time, we adapt it to the setting of flat fixpoint logic.

Before carrying on, let us fix some notation. If $t \in MT(Y \cup P)$ and $\{ s_y \mid y \in Y \} \subseteq MT(X)$ is a collection of terms indexed by Y, then we shall denote by $\boldsymbol{s}$ such a collection, and denote by $t[\boldsymbol{s}/\boldsymbol{y}]$ the result of simultaneously substituting every variable $y \in Y$ with the term s_y.

In order to obtain the axiomatization, the following steps must be performed, for each $\gamma(x, p_1, \ldots, p_n) \in \Gamma$.

(i) *Transform γ into an equivalent guarded formula.* We can assume that each occurrence of x is guarded in γ, that is, each occurrence of x is in the scope of some modal operator. As a matter of fact, our goal is to axiomatize the least prefixed point of $\gamma(x)$. If x is not guarded in γ, then we can find terms γ_1, γ_2 such that the equation

$$\gamma(x, \boldsymbol{p}) = (x \wedge \gamma_1(x, \boldsymbol{p})) \vee \gamma_2(x, \boldsymbol{p}) \,,$$

holds on every modal algebra, and x is guarded in both γ_1 and γ_2. It is easily seen that, on every modal algebra, γ and γ_2 have the same set of prefixed points. Thus, instead of axiomatizing $\sharp_\gamma$, we can equivalently axiomatize $\sharp_{\gamma_2}$.

(ii) *Transform γ into an equivalent system of equations T^γ.* By Proposition 4, we can assume that $\gamma \in MT_\nabla(\{ x \} \cup \{ P \})$, where $P = \{ p_1, \ldots, p_n \}$. Let SC_γ denote the set of subformulas of γ that are special conjunctions, i.e., that are of the form $\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j \Phi_j$ where $\Lambda \subseteq Lit(\{ x \} \cup P)$ and $J \subseteq I$. If $\psi \in SC_\gamma$, then we modify it as follows: (a) if its set of literals Λ does not contain x, $\Lambda \subseteq Lit(P)$, then we let $\widetilde{\psi} = \psi$, (b) otherwise we can write $\psi = x \wedge \psi'$, where ψ' is a special conjunction whose set of literals does not contain x, and, in this case, we let $\widetilde{\psi} = \psi'$. Moreover, we let $\widetilde{\gamma} = \gamma$. Let $Z = \{ z_\psi \mid \psi \in \{ \gamma \} \cup SC_\gamma \}$ be a set of variables, disjoint from $\{ x \}$ and P, in bijection with $\{ \gamma \} \cup SC_\gamma$. Express each $\widetilde{\psi}$, $\psi \in \{ \gamma \} \cup SC_\gamma$, as the result of substituting the modified version of the special conjunctions into a modal term t_{z_ψ}, of modal depth one, whose variables are among Z and x:

$$\widetilde{\psi} = t_{z_\psi}[\, \widetilde{\varphi}/z_\varphi \mid \varphi \in \{ \gamma \} \cup SC_\gamma \text{ and } \varphi \text{ is a proper subformula of } \psi \,] \,.$$

The reader will have no difficulties verifying that each term t_z is a disjunction of special conjunctions $\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j T_j$, where $\Lambda \subseteq \mathit{Lit}(P)$ and $t \in DT(\{ x \} \cup Z)$ whenever $j \in J$ and $t \in T_j$. We call $T^\gamma = \langle Z, \{ t_z \mid z \in Z \} \rangle$ the *system representation of* γ.

(iii) *Construct the system* $T^\gamma_\sharp$. This system is obtained from T^γ by substituting each occurrence of x with z_γ. That is, if we let $r_z = t_z[z_\gamma/x]$, $z \in Z$, then $T^\gamma_\sharp = \langle Z, \{ r_z \mid z \in Z \} \rangle$. Observe that each term r_z is a disjunction of special conjunctions $\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j T_j$ with now $t \in DT(Z)$ (instead of $t \in DT(\{ x \} \cup Z)$), for $j \in J$ and $t \in T_j$, and, as before, $\Lambda \subseteq \mathit{Lit}(P)$.

(iv) *Construct* $\mathcal{P}_{+}(T^\gamma_\sharp)$, *the powerset system of* $T^\gamma_\sharp$. Let $Y = \{ y_S \mid S \in \mathcal{P}_{+}(Z) \}$ be a set of new variables in bijection with $\mathcal{P}_{+}(Z)$, the set of non empty subsets of Z. For $S \in \mathcal{P}_{+}(Z)$, let

$$z_{y_S} = \bigwedge_{z \in S} z \,,$$

and denote by $\boldsymbol{z}$ the vector of terms $\{ z_y \mid y \in Y \}$.

Lemma 11. *A collection of terms* $\{ q_y \,|\, y \in Y \}$ *can be constructed such that*

- *each term* q_y *is a disjunction of special conjunctions* $\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j D_j$*, where* $\Lambda \subseteq \mathit{Lit}(P)$ *and* $d \in D(Y)$ *for each* $d \in D_j$*,*
- *the equations*

$$\bigwedge_{z \in S} r_z = q_{y_S}[\boldsymbol{z}/\boldsymbol{y}] \tag{4}$$

hold on every modal algebra.

The pair $\mathcal{P}_{+}(T^\gamma_\sharp) = \langle Y, \{ q_y \,|\, y \in Y \} \rangle$ is what we call the powerset system of $T^\gamma_\sharp$. We remark that the terms q_y validating the equations (4) may be constructed by iteratively applying distributive laws of distributive lattices as well as the distributive law (2) of the cover modalities.

(v) *Produce the axiom system for* γ. Recall that $\widetilde{\psi}$, $\psi \in \{ \gamma \} \cup SC_\gamma$, are the modified special conjunctions of γ. Let

$$\widetilde{\psi}^\sharp_{y_S} = \bigwedge_{z_\psi \in S} \widetilde{\psi}[\sharp_\gamma/x] \,, \qquad\qquad S \in \mathcal{P}_{+}(Z) \,,$$

and, as usual, let $\widetilde{\boldsymbol{\psi}}^\sharp$ be the vector of terms $\{ \widetilde{\psi}^\sharp_y \mid y \in Y \}$.

Definition 12. *Let* $\mathbf{K}$ *be a standard axiomatization which completely axiomatizes the set of polymodal validities. The axiom system* $\mathbf{K}_\sharp(\gamma)$ *is obtained by adding to* $\mathbf{K}$ *the axiom* ($\sharp_\gamma$-prefix)*, the derivation rules* ($\sharp_\gamma$-least)*, the axioms:*

$$\vdash q_{y_S}[\widetilde{\boldsymbol{\psi}}^\sharp/\boldsymbol{y}] \rightarrow \widetilde{\psi}^\sharp_{y_S}, \qquad S \in \mathcal{P}_{+}(Z)\,,$$

as well as the following derivation rule:

$$\begin{aligned} &\textit{from } \{ \vdash q_{y_T}[\boldsymbol{\varphi}/\boldsymbol{y}] \rightarrow \varphi_{y_T} \ \mid T \in \mathcal{P}_{+}(Z) \, \} \\ &\textit{infer } \vdash q_{y_S}[\widetilde{\boldsymbol{\psi}}^\sharp/\boldsymbol{y}] \rightarrow \varphi_{y_S}\,, \qquad S \in \mathcal{P}_{+}(Z)\,. \end{aligned}$$

Finally, the axiom system $\mathbf{K}_\sharp(\Gamma)$ *is obtained as the union of all the axioms and inferences rules of the axiom systems* $\mathbf{K}_\sharp(\gamma)$*,* $\gamma \in \Gamma$*.*

The axioms and derivation rules of the form ($\sharp_\gamma$-prefix) and ($\sharp_\gamma$-least) may be eliminated from the axiom system. We include them mainly for clarity of exposition. On the other hand, if γ itself is already in ∇-normal form, then the simpler axiomatization, adding ($\sharp_\gamma$-prefix) and ($\sharp_\gamma$-least) to $\mathbf{K}$, already suffices. We can now formulate the main result of this paper:

Theorem 13. *The axiom system* $\mathbf{K}_\sharp(\Gamma)$ *is sound and complete with respect to the Kripke semantics of* $\mathcal{L}_\sharp(\Gamma)$*.*

Soundness will be discussed in the remainder of this section, an overview of the completeness proof will be given in the next.

Algebraic Interpretation. We elucidate now the algebraic meaning of the proposed axiomatization. To begin with, let us formally define a *modal system* (of equations) as a pair $T = \langle Z, \{\, t_z \,\}_{z \in Z} \rangle$ where Z is a finite set of variables and $t_z \in MT(Z \cup P)$ for each $z \in Z$. We say that a modal system is pointed if it comes with a specified variable $z_0 \in Z$. Given a modal system T and a modal algebra A, there exists a unique function $T^A : A^Z \times A^P \longrightarrow A^Z$ such that, for each projection $\pi_z : A^Z \longrightarrow A$, $\pi_z \circ T^A = t_z^A$. We shall say that T^A is the interpretation of T in A. Whenever it exists, we shall denote by $\mu_Z.T^A : A^P \longrightarrow A^Z$ the parametrized least prefixed point of T^A.

The (directed) graph of a system T has as vertices the variables Z and edges are of the form $z \rightarrow z'$ whenever z' occurs in t_z. We say that T is *acyclic* if the graph of T contains no cycle. Let us define the iterates of a modal system by $T^1 = T$, and $T^{n+1} = \langle Z, \{\, t_z[T^n/\boldsymbol{z}] \mid z \in Z \,\} \rangle$. If T is acyclic, then $T^{n+1} = T^n$ for some $n \geq 1$. Let n_0 be the least such integer and define $S = T^{n_0} = \langle Z, \{\, s_z \mid z \in Z \,\} \rangle$. If the terms in T are not themselves variables, then each term s_z of S is a term in $MT(P)$. If A is an arbitrary modal algebra, let $S^A : A^P \longrightarrow A^Z$ be determined by $\pi_z \circ S^A = s_z^A$, $z \in Z$. It is easily seen that $T^A(S^A(\boldsymbol{v}), \boldsymbol{v}) = S^A(\boldsymbol{v})$, and that S^A is the parameterized least prefixed point of T^A, $S^A = \mu_Z.T^A$. Let us call S the solution of T.

Let us now fix $\gamma \in \Gamma$. When presenting the axiomatization we have introduced three modal systems T^γ, $T^\gamma_\sharp$, and $\mathcal{P}_{+}(T^\gamma_\sharp)$. Since γ is fixed we shall from now on and later – whenever γ is understood – omit the superscript. T, the system representation of γ is acyclic. Also, T and $T_\sharp$ are modal systems pointed by z_γ. Let in the following A be a fixed but arbitrary modal algebra. Since T is acyclic let $S = \{ s_z \mid z \in Z \}$ be its solution. It is easily argued that $s^A_{z_\psi} = \widetilde{\psi}^A$, hence, in particular, $\pi_{z_\gamma} \circ S^A = s^A_{z_\gamma} = \gamma^A$. Recall that $T_\sharp$ is obtained from $T = \langle Z, z_\gamma, \{ t_z \}_{z \in Z} \rangle$ by substituting z_γ for x in every term t_z. This means that $T^A_\sharp$ is the following compose:

$$T^A_\sharp : \; A^Z \times A^P \xrightarrow{\langle \pi_Z, \pi_{z_\gamma} \rangle \times A^P} A^Z \times A^x \times A^P \xrightarrow{T^A} A^Z \,.$$

The following Lemma shows that, in view of our axiomatizing purposes, it is equivalent to axiomatize $\mu_x.\gamma$ or to axiomatize $\mu_Z.T_\sharp$.

Lemma 14. *For every modal algebra A and every vector $\boldsymbol{v} \in A^P$, the least prefixed point $\mu_Z.T^A_\sharp(Z, \boldsymbol{v})$ exists if and only if the least prefixed point $\mu_x.\gamma^A(x, \boldsymbol{v})$ exists. If existing, they are related as follows:*

$$\mu_Z.T^A_\sharp(Z, \boldsymbol{v}) = S^A(\mu_x.\gamma^A(x, \boldsymbol{v}), \boldsymbol{v}) \,, \qquad \mu_x.\gamma^A(x, \boldsymbol{v}) = \pi_{z_\gamma}(\mu_Z.T^A_\sharp(Z, \boldsymbol{v})) \,.$$

The proof of the Lemma is an application of the Bekič and Rolling rules, see for example [8, §2.3] and [3, §8.29]. It follows that, if A is a modal $\sharp$-algebra, then $S^A(\sharp^A(\boldsymbol{v}), \boldsymbol{v})$ is necessarily the least fixed point of $T^A_\sharp$.

Let us analyse now the role of the system $\mathcal{P}_+(T_\sharp)$. If $S \in \mathcal{P}_+(Z)$ and $\boldsymbol{v} \in A^Z$, let $\iota^A_{y_S}(\boldsymbol{v}) = \bigwedge_{z \in S} \boldsymbol{v}_z$ and $\iota^A : A^Z \longrightarrow A^{\mathcal{P}_+(Z)}$ be defined by $\pi_{y_S} \circ \iota^A = \iota^A_{y_S}(z)$, $S \in \mathcal{P}_+(Z)$. The meaning of the equations (4) is that the diagram

$$\begin{tikzcd}
A^Z \times A^P \arrow[r, "T^A_\sharp"] \arrow[d, "\iota^A \times A^P"] & A^Z \arrow[d, "\iota^A"] \\
A^Y \times A^P \arrow[r, "\mathcal{P}_+(T_\sharp)^A"] & A^Y
\end{tikzcd} \tag{5}$$

commutes. The main statement proved in [1, §9] is the following Proposition.

Proposition 15. *If A is a complete modal algebra and $\boldsymbol{v} \in A^P$, then*

$$\iota^A(\mu_Z.T^A_\sharp(\boldsymbol{v})) = \mu_Y.\mathcal{P}_+(T_\sharp)^A(\boldsymbol{v}) \,. \tag{6}$$

The proof presented in [1, §1.2.15] that equation (6) holds crucially depends on A being complete. It is not clear that this equation is derivable only on the basis that $\mu_Z.T^A_\sharp$ is the least prefixed point of $T^A_\sharp$. However equation (6) holds in every Kripke model, and if our goal is to collect the formulas valid in every Kripke model, then we can freely add to a formal system axioms and inference rules stating that the least solution of $\mathcal{P}_+(T_\sharp)$ is the image by ι^A of the least solution of $T^A_\sharp$, that is, $\iota^A(S^A(\sharp^A(\boldsymbol{v}), \boldsymbol{v}))$. This is precisely the goal of the axiom system $\mathbf{K}_\sharp(\gamma)$ as well as of the next Definition.

Definition 16. *A modal $\sharp$-algebra A is* regular *if, for each $\gamma \in \Gamma$ and each $\boldsymbol{v} \in A^P$, $\iota^A(S^{\gamma A}(\sharp_\gamma^A(\boldsymbol{v}), \boldsymbol{v}))$ is the least prefixed point of $\mathcal{P}_{+}(T_\sharp^\gamma)_{\boldsymbol{v}}^A$:*

$$\iota^A(S^{\gamma A}(\sharp_\gamma^A(\boldsymbol{v}), \boldsymbol{v})) = \mu_Y . \mathcal{P}_{+}(T_\sharp^\gamma)^A(\boldsymbol{v}) .$$

From Proposition 15, we obtain the following corollary, implying *soundness.*

Corollary 17. *Every Kripke $\sharp$-algebra is regular.*

4 Overview of the Completeness Proof

Let us recall that $f : A \longrightarrow B$ is a modal algebra morphism if the operations $\langle \bot, \top, \neg$
$\wedge, \{ \Diamond_i \mid i \in I \} \rangle$ are preserved by f. If A and B are also modal $\sharp$-algebras then f is a *modal $\sharp$-algebra morphism* if moreover each $\sharp_\gamma$, $\gamma \in \Gamma$, is preserved by f. This means that

$$f(\sharp_\gamma^A(\boldsymbol{v})) = \sharp_\gamma^B(f \circ \boldsymbol{v}) ,$$

for each $\boldsymbol{v} \in A^P$ and $\gamma \in \Gamma$. A $\sharp$-algebra morphism is an embedding if it is injective. We say that A embeds into B if there exists an embedding $f : A \longrightarrow B$.

Let X be a set of variables. The elements of the *Lindenbaum algebra $\mathcal{L}(X)$* are equivalence classes of terms whose variables are contained in X, where two terms t, s are declared to be equivalent if $\vdash t \leftrightarrow s$ is derivable in the system $\mathbf{K}_\sharp(\Gamma)$. This is a standard construction of an algebra from the syntax of the logic [2], for example we shall have $\sharp_\gamma^{\mathcal{L}(X)}([t_1], \dots, [t_n]) = [\sharp_\gamma(t_1, \dots, t_n)]$. By construction, $\mathcal{L}(X)$ is a regular modal $\sharp$-algebra and there is a canonical interpretation of the variables in X as elements of $\mathcal{L}(X)$, sending the variable x to the equivalence class $[x]$ of the term x. Moreover, $\mathcal{L}(X)$ has the following property: whenever A is a regular modal $\sharp$-algebra and $\boldsymbol{v} : X \longrightarrow A$ is a valuation of the variables in x as elements of A, then there exists a unique modal $\sharp$-algebra morphism $f : \mathcal{L}(X) \longrightarrow A$ such that $f[x] = \boldsymbol{v}(x)$ for all $x \in X$. In universal algebraic, or categorical terms, $\mathcal{L}(X)$ is the *free regular $\sharp$-algebra over X*. We recall that this property, freeness, determines $\mathcal{L}(X)$ up to isomorphism of modal $\sharp$-algebras. In the sequel we shall use the words 'free regular $\sharp$-algebra' as a synonym of the Lindenbaum algebra.

The key to the completeness of the system $\mathbf{K}_\sharp(\Gamma)$ is the following Theorem:

Theorem 18. *If X is countable, then $\mathcal{L}(X)$ embeds in a Kripke $\sharp$-algebra.*

The theorem implies completeness as follows. Let X be the set of variables of a term/
formula t. If the formula t is valid in every Kripke frame, then the equation $t = \top$ holds in every Kripke $\sharp$-algebra, and thus certainly in the one that $\mathcal{L}(X)$ embeds into. Consequently, the equation $t = \top$ holds in the Lindenbaum algebra $\mathcal{L}(X)$. This in particular implies $[\top] = [t]$, that is $\vdash \top \leftrightarrow t$ is derivable in $\mathbf{K}_\sharp(\Gamma)$. As usual, this implies that $\vdash t$ is derivable in $\mathbf{K}_\sharp(\Gamma)$.

In turn, the proof of Theorem 18 is subdivided in many steps, which we here collect into two main results, to be proved successively in the next two sections.

Theorem 19. *The modal operators* $\Diamond_i^{\mathcal{L}(X)}$, $i \in I$, *of a Lindenbaum algebra* $\mathcal{L}(X)$ *are residuated. Moreover,* $\mathcal{L}(X)$ *is constructive.*

Theorem 20. *If a countable* $\sharp$*-algebra* A *is constructive and its modal operators* $\Diamond_i^A$, $i \in I$, *are residuated, then* A *has an embedding into a Kripke* $\sharp$*-algebra.*

Since $\mathcal{L}(X)$ is countable whenever X is countable, Theorem 18 follows.

5 Properties of the Lindenbaum Algebra

The goal of this section is to prove that the Lindenbaum algebra $\mathcal{L}(X)$ is constructive, cf. Definition 8. We shall obtain this result by subsequently analyzing properties of this algebra. Let us first say that a modal algebra A generated by a set X is *rigid* w.r.t. X if

$$\bigwedge \Lambda \wedge \bigwedge_{j \in J} \nabla_j Y_j \leq \bot \quad \text{implies} \quad \bigwedge \Lambda \leq \bot \text{ or } \exists j \in J, y \in Y_j \text{ s.t. } y \leq \bot$$

holds in A, where Λ is a finite set of literals, $J \subseteq I$, and, for each $j \in J$, Y_j is a finite possibly empty set of elements of A.

Theorem 21. *The free regular modal* $\sharp$*-algebra* $\mathcal{L}(X)$ *is rigid w.r.t.* X.

The proof of the Theorem depends on the following construction. If A is any modal algebra, let us call a pair $\mathfrak{f} = \langle J, \{ Y_j \mid j \in J \} \rangle$ – where $J \subseteq I$ and for each $j \in J$ Y_j is a finite subset of A – a *candidate failure* for rigidness if $y \not\leq \bot$ whenever $j \in J$ and $y \in Y_j$. Given a candidate failure $\mathfrak{f}$, a *repair* for $\mathfrak{f}$ is a collection $\chi = \{ \chi_j^y : A \longrightarrow 2 \mid j \in J, y \in Y_j \}$ of Boolean algebra morphisms such that $\chi_j^y(y) = \top$ for each $j \in J$ and $y \in Y_j$. Observe that, by the prime filter theorem, such a repair always exists. Define $\chi_j(z) = \bigvee_{y \in Y_j} \chi_j^y(z)$ if $j \in J$ and, otherwise, $\chi_j(z) = \bot$.

Definition 22. *The modal algebra* $A_{\mathfrak{f},\chi}$ *has as Boolean algebra reduct the product Boolean algebra* $A \times 2$. *For* $i \in I$, *the modal operators* $\Diamond_i$ *are defined by:*

$$\Diamond_i^{A_{\mathfrak{f},\chi}}(z, w) = (\Diamond_i^A z, \chi_i(z)) .$$

Observe that $\Diamond_i$ are indeed modal operators, since the functions χ_i preserve finite joins. The point of considering this construction is the following statement. If $\mathcal{K}$ is a category of modal algebras and modal algebra morphisms, such that whenever $\mathfrak{f}$ is a candidate failure in A and χ is some repair of $\mathfrak{f}$, then $A_{\mathfrak{f},\chi}$ (as well as the projection to A) belongs to $\mathcal{K}$, then a modal algebra $\mathcal{F}_{\mathcal{K}}(X)$, which is free within $\mathcal{K}$, is rigid. Thus we prove:

Proposition 23. *If* A *is a* $\sharp$*-algebra, then* $A_{\mathfrak{f},\chi}$ *is also a* $\sharp$*-algebra and the projection is a* $\sharp$*-algebra morphism. If moreover* A *is regular, then* $A_{\mathfrak{f},\chi}$ *is regular.*

Recall the definition of the cover modalities ∇_i, $i \in I$: $\nabla_i Y = \bigwedge \Diamond_i Y \wedge \Box_i \bigvee Y$, for some set of variables Y. This implies that in order to consider the interpretation of ∇_i in a modal algebra A we need to fix an indexing Y_0. Hence, we shall write $\nabla_{i\,Y_0}^{A} : A^{Y_0} \longrightarrow A$ and observe that, using this notation, it is not the case that, for $Y \subseteq Y_0$, $\nabla_{i\,Y_0}^{A}$ is obtained from $\nabla_{i\,Y}^{A}$ by precomposing with the projections.

Proposition 24. *The Lindenbaum algebra $\mathcal{L}(X)$ is such that, for each finite set $\{ k_i \in \mathcal{L}(X) \mid i = 1, \dots, n \}$, the collection*

$$\mathcal{F} = \{ k_i \wedge \bigwedge_{j \in J} \nabla_{j\,Y}^{\mathcal{L}(X)} \mid i = 1, \dots, n,\ J \subseteq I,\ Y \subseteq Y_0 \}$$

is a family of finitary $\mathcal{O}$-adjoints. Moreover the modal operators $\Diamond_i^{\mathcal{L}(X)}$, $i \in I$, are residuated.

The proof, crucially involving Theorem 21, is along the same lines as in [10], see Propositions 5.1, 6.7, and 7.2. We are ready to state and prove the main goal of this section.

Proposition 25. *The Lindenbaum algebra $\mathcal{L}(X)$ is constructive.*

Proof. Let us remark first that, for each fixed vector $\boldsymbol{k} \in \mathcal{L}(X)^P$, the least prefixed point of $\mathcal{P}_{+}(T_\sharp)_{\boldsymbol{k}}^{\mathcal{L}(X)} : \mathcal{L}(X)^Y \longrightarrow \mathcal{L}(X)^Y$ exists by the definition of a regular modal $\sharp$-algebra. Let us verify that such a least prefixed point is constructive. For each $S \in \mathcal{P}_{+}(Z)$, $\pi_{y_S} \circ \mathcal{P}_{+}(T_\sharp)_{\boldsymbol{k}}^{\mathcal{L}(X)}$ is of the form

$$\bigvee_{j \in J_S} k_j \wedge \bigwedge_{i \in I_j} \nabla_{i\,W_i}^{\mathcal{L}(X)} \boldsymbol{f}^i$$

with k_j constant and, for each $w \in W_i$, $\boldsymbol{f}_w^i$ is a join of elements in Y. Since families of finitary $\mathcal{O}$-adjoints can be closed under joins and substitution, it follows from Proposition 24 that $\{ \pi_{y_S} \circ \mathcal{P}_{+}(T_\sharp)_{\boldsymbol{k}}^{\mathcal{L}(X)} \mid S \subseteq \mathcal{P}_{+}(Z) \}$ is a family of finitary $\mathcal{O}$-adjoints. By [10, Proposition 6.3.4], $\mathcal{P}_{+}(T_\sharp)_{\boldsymbol{k}}^{\mathcal{L}(X)}$ is itself a finitary $\mathcal{O}$-adjoint and its least prefixed point, which exists, is constructive. By [10, Lemma 7.4], it follows that the least prefixed point of $(T_\sharp^{\mathcal{L}(X)})_{\boldsymbol{k}}$ is constructive. Finally, it follows from Lemma 26 below that the least prefixed point of $\gamma_{\boldsymbol{k}}^{\mathcal{L}(X)}$ is itself constructive. □

We end this section stating the mentioned Lemma, which is an analogous of Lemma 14 for continuous functions and constructive fixed points.

Lemma 26. *Let us suppose that the operations of the $\sharp$-algebra A are continuous. Let $\boldsymbol{v} \in A^P$ be arbitrary. The least prefixed point of $(T_\sharp^A)_{\boldsymbol{v}}$ exists and is constructive if and only if the least prefixed point of $\gamma_{\boldsymbol{v}}^A$ exists and is constructive.*

6 A Representation Theorem

In this section we shall prove Theorem 20. Let us fix a modal $\sharp$-algebra A as in the statement of the Theorem. For simplicity we restrict attention to a language with a single diamond $\Diamond$, and a single fixpoint connective $\sharp$. We let $\gamma(x, \boldsymbol{p})$ denote the associated formula of $\sharp$, where $\boldsymbol{p} = (p_1, \ldots, p_n)$. The main lemma in the proof of Theorem 20 is the following.

Lemma 27. *For each $a \in A$ there is a Kripke frame S_a and a modal $\sharp$-homomorphism $\rho_a : A \to S_a^\sharp$ such that $\rho_a(a) > \bot$.*

We shall prove Lemma 27 by a step-by-step approximation process involving the notion of a *network* [2]. Let ω^* denote the set of finite sequences of natural numbers. We denote concatenation of such sequences by juxtaposition, and write ϵ for the empty sequence. If $s = tk$ for some $k \in \omega$ we say that s is the parent of t and write either $s = t^-$ or $s \lhd t$. A *tree* is a subset T of ω^* which is both downward and leftward closed; that is, if $t \neq \epsilon$ belongs to T, then so does t^-, and if $sm \in T$ then $sk \in T$ for all $k < m$. Obviously, a tree T, together with the relation $\lhd$, forms a Kripke frame; this frame will simply be denoted as T, and its complex $\sharp$-algebra, as $T^\sharp$.

An *A-network* is a pair $N = \langle T, L\rangle$ such that T is a tree, and $L : T \to \mathcal{P}(A)$ is some labelling. Such a network N induces a map $r_N : A \to \mathcal{P}(T)$, given by

$$r_N(a) := \{t \in T \mid a \in L(t)\}. \tag{7}$$

The aim of the proof will be to construct, for an arbitrary nonzero $a \in A$, a network $N = \langle T, L\rangle$, with $a \in L(\epsilon)$, and such that r_N is a modal $\sharp$-homomorphism from A to $T^\sharp$. We need some definitions.

A network $N = \langle T, L\rangle$ is called *locally coherent* if $\bigwedge X > \bot$, whenever X is a finite subset of $L(t)$ for some $t \in T$; *modally coherent* if $\bigwedge X \wedge \Diamond \bigwedge Y > \bot$, for all $s, t \in T$ such that $s \lhd t$ and all finite subsets X and Y of respectively $L(s)$ and $L(t)$; and *coherent* if it satisfies both coherence conditions. N is *prophetic* if for every $s \in T$, and for every $\Diamond a \in L(s)$, there is a *witness* $t \rhd s$ such that $a \in L(t)$; *decisive* if either $a \in L(t)$ or $-a \in L(t)$, for every $t \in T$ and $a \in A$; and *$\sharp$-constructive* if, for every $t \in T$, and every sequence $\boldsymbol{a}$ in A such that $\sharp\boldsymbol{a} \in L(t)$, there is a natural number n such that $(\gamma^A)^n_{\boldsymbol{a}}(\bot) \in L(t)$. A network is *perfect* if it has all of the above properties.

Lemma 28. *If N is a perfect A-network, then r_N is a modal $\sharp$-homomorphism.*

Clearly, we shall have that $r_N(a) \neq \varnothing$ for all $a \in A$ for which there is a $t \in T$ with $a \in L(t)$. From the above proposition it follows that in order to prove Lemma 27 it suffices to construct, for an arbitrary nonzero $a \in A$, a perfect network with $a \in L(\epsilon)$. Our construction will be carried out in a step-by-step process, where at each stage we are dealing with a finite *approximation* of the final network. Since these approximations are not perfect themselves, they will suffer from certain *defects*. We will only be interested in those defects that can be *repaired* in the sense that the network can be extended to a bigger version that is lacking the defect.

Formally we define a *defect* of a network $N = \langle T, L \rangle$ to be an object d of one of the following three kinds:

1. $d = (t, a, -)$, with $t \in T$ and $a \in A$ such that neither a nor $-a$ belongs to $L(t)$,
2. $d = (t, a, \Diamond)$, with $t \in T$ and $a \in A$ such that $\Diamond a \in L(t)$, but there is no witness $s \rhd t$ such that $a \in L(s)$,
3. $d = (t, \boldsymbol{a}, \sharp)$, with $t \in T$ and $\boldsymbol{a} \in A^n$ such that $\sharp\boldsymbol{a} \in L(t)$, but there is no $n \in \omega$ such that $(\gamma^A)^n_{\boldsymbol{a}}(\bot) \in L(t)$.

While in principle we could construct a perfect network as a limit of coherent networks, the networks that we will actually use will in fact satisfy a much stronger version of coherency. In order to define this notion, we first need to extend the *local* labelling function L of the network to a global one. Recall that the operator of the algebra A is *residuated*, and hence, *conjugated*. That is, there is an operation $\blacklozenge : A \to A$ such that

$$a \wedge \Diamond b > \bot \text{ iff } \blacklozenge a \wedge b > \bot, \tag{8}$$

for all $a, b \in A$. Using this operation $\blacklozenge$, we can in fact define the global labelling map $\widetilde{L}$ as follows:

$$\begin{aligned}
\Delta_{\downarrow}(t) &:= \bigwedge L(t) \wedge \bigwedge_{t \lhd s} \Diamond \Delta_{\downarrow}(s)\,, \qquad \Delta_{\downarrow,-u}(t) := \bigwedge L(t) \wedge \bigwedge_{t \lhd s, s \neq u} \Diamond \Delta_{\downarrow}(s)\,,\\
\Delta_{\uparrow}(t) &:= \begin{cases} \top\,, & \text{if } t = \epsilon\,,\\ \blacklozenge(\,\Delta_{\uparrow}(t^-) \wedge \Delta_{\downarrow,-t}(t^-)\,)\,, & \text{otherwise}\,, \end{cases}\\
\widetilde{L}(t) &:= \Delta_{\downarrow}(t) \wedge \Delta_{\uparrow}(t)\,.
\end{aligned}$$

The following observation is a consequence of the conjugacy relation (8) and of the fact that the tree is connected.

Lemma 29. *If N is a finite network and $s, t \in N$, then $\widetilde{L}(s) > \bot$ iff $\widetilde{L}(t) > \bot$.*

Call a finite network $N = \langle T, L \rangle$ *globally coherent* if $\widetilde{L}(t) > \bot$ for all $t \in T$. We can now prove our *repair lemma*. We say that N' *extends* N, notation: $N \leq N'$, if $T \subseteq T'$ and $L(t) \subseteq L'(t)$ for every $t \in T$.

Lemma 30 (Repair Lemma). *Let $N = \langle T, L \rangle$ be a globally coherent A-network. Then for any defect d of N there is a globally coherent extension N^d of N which lacks the defect d.*

Proof. We have to take action depending on the type of the defect d. In each case we will make heavily use of the global extension $\widetilde{L}$ of L. For instance, suppose $d = (t, \boldsymbol{a}, \sharp)$ is a defect of the third kind. By strong coherency, $\widetilde{L}^N(t) > \bot$. Suppose for contradiction that $\widetilde{L}^N(t) \wedge (\gamma^A)^n_{\boldsymbol{a}}(\bot) = \bot$ for all numbers n. Then for all n we have $(\gamma^A)^n_{\boldsymbol{a}}(\bot) \leq -\widetilde{L}^N(t)$, and so by constructiveness of $\sharp$ on A it follows that $\sharp^A \boldsymbol{a} \leq -\widetilde{L}^N(t)$. But this contradicts the fact that N is coherent.

It follows that $\widetilde{L}^N(t) \wedge (\gamma^A)^n_{\boldsymbol{a}}(\bot) > \bot$ for some natural number n. Now define $N' := \langle T, L' \rangle$, where $L'(s) := L(s)$ for $s \neq t$, while $L'(t) := L(t) \cup \{(\gamma^A)^n_{\boldsymbol{a}}(\bot)\}$. It is not difficult to check that N' satisfies all the requirements stated in the Lemma. □

Lemma 31. *Every globally coherent A-network can be extended to a perfect network.*

Proof. On the basis of successive applications of Lemma 30, properly scheduled, one may define a sequence of networks $N = N_0 \leq N_1 \leq N_2 \leq \ldots$ such that for each $i \in \omega$ and each defect d of N_i there is a $j > i$ such that d is not a defect of N_j. Then define $N' := \langle T', L' \rangle$, with $T' := \bigcup_{i<\omega} T_i$ and for each $t \in T'$, $L'(t) := \bigcup_{i<\omega} L_i(t)$. It is then straightforward to verify that N' is a perfect extension of N. □

Proof of Lemma 27. Consider an arbitrary nonzero element $a \in A$, and let N_a be the network $\langle \{\epsilon\}, L_a \rangle$, L_a given by $L_a(\epsilon) := \{a\}$. It is obvious that N_a is globally coherent, so Lemma 27 follows by a direct application of the Lemmas 31 and 28.
Proof of Theorem 20. Let S be the disjoint union of the family $\{S_a \mid \bot \neq a \in A\}$, where the S_as are given by Lemma 27. It is straightforward to verify that A can be embedded into the product $\prod_{a\neq\bot} S^\sharp_a$, and that this latter product is isomorphic to $S^\sharp$, the complex $\sharp$-algebra of S.

References

1. Arnold, A., Niwiński, D.: *Rudiments of μ-calculus.* Studies in Logic and the Foundations of Mathematics, vol. 146. North-Holland Publishing Co., Amsterdam (2001)
2. Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)
3. Davey, B.A., Priestley, H.A.: Introduction to lattices and order, 2nd edn. Cambridge University Press, New York (2002)
4. Emerson, E.A.: Temporal and modal logic. In: Handbook of theoretical computer science, vol. B, pp. 995–1072. Elsevier, Amsterdam (1990)
5. Ésik, Z.: Completeness of Park induction. Theoret. Comput. Sci. 177(1), 217–283 (1997)
6. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. MIT Press, Cambridge (2000)
7. Kozen, D.: Results on the propositional μ-calculus. Theoret. Comput. Sci. 27(3), 333–354 (1983)
8. Santocanale, L.: μ-bicomplete categories and parity games. Theoretical Informatics and Applications 36, 195–227 (2002)
9. Santocanale, L.: Completions of μ-algebras. In: LICS 2005, pp. 219–228. IEEE Computer Society, Los Alamitos (2005)
10. Santocanale, L.: Completions of μ-algebras. arXiv:math.RA/0508412 (August 2005)
11. Walukiewicz, I.: Completeness of Kozen's axiomatisation of the propositional μ-calculus. Inform. and Comput. 157(1-2), 142–182 (2000) LICS 1995 (San Diego, CA)

$\mathbb{FDNC}$: Decidable Non-monotonic Disjunctive Logic Programs with Function Symbols[⋆]

Mantas Šimkus and Thomas Eiter

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
`(simkus,eiter)@kr.tuwien.ac.at`

Abstract. Current Answer Set Programming systems are built on non-monotonic logic programs without function symbols; as well-known, they lead to high undecidability in general. However, function symbols are highly desirable for various applications, which challenges to find meaningful and decidable fragments of this setting. We present the class $\mathbb{FDNC}$ of logic programs which allows for function symbols, disjunction, non-monotonic negation under answer set semantics, and constraints, while still retaining the decidability of the standard reasoning tasks. Thanks to these features, they are a powerful formalism for rule-based modeling of applications with potentially infinite processes and objects, which allows also for common-sense reasoning. We show that consistency checking and brave reasoning are ExpTime-complete in general, but have lower complexity for restricted fragments, and outline worst-case optimal reasoning procedures for these tasks. Furthermore, we present a finite representation of the possibly infinitely many infinite stable models of an $\mathbb{FDNC}$ program, which may be exploited for knowledge compilation purposes.

1 Introduction

Answer Set Programming (ASP) is a declarative programming paradigm which has its roots in Logic Programming and Non-monotonic Reasoning. It is well-suited for modeling and solving problems which involve common sense reasoning, and has been fruitfully applied to a range of applications including data integration, configuration, reasoning about actions and change, etc.; see [16].

While Answer Set Semantics, which underlies ASP, was defined in the setting of a general first-order language, current ASP frameworks and implementations, like DLV [10], Smodels [15], and other efficient solvers are based on function-free languages and resort to Datalog with negation and its extensions. However, it is widely acknowledged that this leads to drawbacks related to expressiveness, and also to inconvenience in knowledge representation, cf. [2]. Since one is forced to work with finite domains, potentially infinite processes cannot be represented naturally in ASP. Additional tools must be used, which may incur high space requirements.

[⋆] Work partially supported by the Austrian Science Funds (FWF) project P17212 and the EC project REWERSE (IST-2003-506779).

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 514–530, 2007.
© Springer-Verlag Berlin Heidelberg 2007

Function symbols, in turn, are a very convenient means for generating infinite domains and objects, and allow for more natural representation of problems on such domains. However, they have been banned in ASP for a good reason, since they quickly lead to undecidability even for Horn programs, and with negation under the answer set semantics, they lead to high undecidability, cf. [11,12]. This raises the challenge to single out meaningful fragments of ASP with function symbols which allow to model infinite domains while still retaining the decidability of the standard reasoning tasks. Important work on this is by Bonatti and his colleagues on their *finitary programs* and *finitely recursive programs*[2,1], which impose syntactic conditions on the groundings of logic programs. However, the hardness of verifying the satisfaction of the conditions limits the applicability of the results; see Section 5 for more discussion.

In this paper, we pursue an approach to obtain decidable logic programs with function symbols by merely constraining the syntax in a way that can be effectively checked. To this end, we take inspiration from results in automated deduction and other areas of knowledge representation, where many procedures, like tableaux algorithms with blocking, or hyper-resolution, have been developed for deciding satisfiability in various fragments of first-order logic. When function symbols (or existential quantification) may occur, these procedures are often sophisticated since they must deal with possibly infinite models. However, because of the peculiarities of Answer Set Semantics, transferring these results to logic programs is not straightforward. Reasoning with logic programs needs to be more refined since only minimal (or stable) models count as models of a given program. Our main contributions are briefly summarized as follows.

- We introduce the class 𝔽𝔻ℕℂ of logic programs, which allow for function symbols, disjunction, constraints, and non-monotonic negation under the answer set semantics [6]. The restrictions we apply are syntactic and ensure that programs have a *forest-shaped model* property. 𝔽𝔻ℕℂ programs are a convenient tool for knowledge representation. They allow, e.g., the representation of an evolving action domain (see Section 3).
- We show that standard reasoning tasks are decidable for 𝔽𝔻ℕℂ, and are ExpTime-complete; this includes checking the consistency (i.e., the existence of a stable model), and brave entailment of ground atomic or existential atomic queries. Disallowing disjunction and constraints (𝔽ℕ) or non-monotonic negation (𝔽𝔻ℂ) does not lead to lower complexity, i.e., the problems considered remain ExpTime-complete. Depending on the reasoning task, reasoning is at most PSpace-complete for further restricted classes.
- Noticeably, the hardness proofs for consistency checking in 𝔽ℕ, 𝔽𝔻ℂ, and 𝔽𝔻ℕℂ, are by a reduction from satisfiability testing in the ExpTime-complete Description Logic $\mathcal{ALC}$. Thus, as a side result we obtain a novel polynomial time mapping of a well-known Description Logic to logic programming.
- 𝔽𝔻ℕℂ programs can have infinitely many and infinitely large stable models, which therefore can not be explicitly represented. We provide a method to finitely represent all the stable models of a given 𝔽𝔻ℕℂ program. This is

achieved by a composition technique that allows to reconstruct stable models as forests, i.e., sets of trees, from *knots*, which are instances of generic labeled trees of depth 1. The finite representation technique allows us to define an elegant decision procedure for brave reasoning in $\mathbb{FDNC}$, and may also be exploited for offline knowledge compilation to speed up online reasoning, by precomputing and storing the knots of a program.

Thanks to their features, $\mathbb{FDNC}$ programs are a powerful formalism for rule-based modeling of applications with a potentially infinite domain, which also accommodates common-sense reasoning through non-monotonic negation. From a complexity perspective, $\mathbb{FDNC}$ and its subclasses offer *effective syntax* for encoding problems in PSpace and ExpTime to logic programs with function symbols.

2 Preliminaries

A *disjunctive rule* (briefly, *rule*) is an expression of the form

$$A_1 \vee \ldots \vee A_n \leftarrow L_1, \ldots, L_m,$$

where $n{+}m > 0$, $A_1, \ldots, A_n$ are *atoms* and $L_1, \ldots, L_m$ are *literals*. The atoms are from a standard first-order language with countably infinite sets of *variables*, *constant symbols*, and *function* and *predicate symbols* of positive arity. A literal is either an atom A (*positive literal*), or an expression *not* A (*negative literal*). The atoms $A_1, \ldots, A_n$ are *head atoms*, while $L_1, \ldots, L_m$ are *body literals*. For a rule r, let $\mathsf{head}(r)$, $\mathsf{body}^+(r)$, and $\mathsf{body}^-(r)$ respectively denote the set of its head atoms, positive body literals, and negative body literals. We say r is a *fact*, if $n = 0$; a *constraint*, if $m = 0$; and *positive*, if $\mathsf{body}^-(r) = \emptyset$.

A *disjunctive logic program* (briefly, *program*) is an arbitrary set of rules. It is *positive* (resp., *ground*), if contains only positive (resp., ground) rules. For a program P, its *Herbrand universe*, *Herbrand base* and its ground instantiation are defined in the standard way, and are respectively denoted by $\mathcal{HU}^P$, $\mathcal{HB}^P$ and $\mathsf{Ground}(P)$; see [13]. Furthermore, by $MM(P)$ we denote the set of *(Herbrand) interpretations* that are (set-inclusion) minimal models of a ground positive program P.

An interpretation I of a program P is a *stable model* of P iff $I \in MM(P^I)$, where P^I is the *Gelfond-Lifschitz reduct* [6] of P, obtained from $\mathsf{Ground}(P)$ by removing (i) each rule r such that $\mathsf{body}^-(r) \cap I \neq \emptyset$, and (ii) all the negative literals from the remaining rules. The set of stable models of a program P is denoted by $SM(P)$.

A program P is *consistent*, if $SM(P) \neq \emptyset$. A program P *bravely entails* a ground (variable-free) atom A (in symbols, $P \models_b A$), if some stable model I of P contains A. An *existential atomic query* is an expression $\exists \boldsymbol{x}.A(\boldsymbol{x})$, where $\boldsymbol{x}$ is a n-tuple of variables and A is a predicate symbol of arity n. A program P *bravely entails* $\exists \boldsymbol{x}.A(\boldsymbol{x})$ (in symbols, $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$), if some stable model I of P contains a ground atom $A(\boldsymbol{t})$.

(1) $Change(x, grow(x)) \leftarrow Young(x), Warm(x)$

(2) $Change(x, cell_1(x)) \leftarrow Mature(x), Warm(x)$

(3) $Change(x, cell_2(x)) \leftarrow Mature(x), Warm(x)$

(4) $Change(x, die(x)) \leftarrow Cold(x)$

(5) $Young(cell_1(x)) \leftarrow Change(x, cell_1(x))$

(6) $Young(cell_2(x)) \leftarrow Change(x, cell_2(x))$

(7) $Mature(grow(x)) \leftarrow Change(x, grow(x))$

(8) $Warm(grow(x)) \vee Cold(grow(x)) \leftarrow Change(x, grow(x))$

(9) $Warm(y) \leftarrow Warm(x), Change(x, y), not\ Cold(y)$

(10) $Cold(y) \leftarrow Cold(x), Change(x, y), not\ Warm(y)$

(11) $\leftarrow Cold(x), Warm(x)$

(12) $Young(b) \leftarrow$

(13) $Warm(b) \leftarrow$

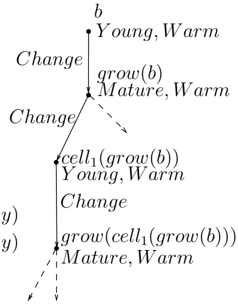


Fig. 1. Example: Evolution of a Cell

3 $\mathbb{FDNC}$ Programs

We now introduce the class $\mathbb{FDNC}$ of logic programs with function symbols. The syntactic restrictions that are applied are to ensure the decidability of the formalism. As we shall see, $\mathbb{FDNC}$ programs can have infinitely many possibly infinite stable models. In this section we analyze the model-theoretic properties of the formalism and introduce a method for finite representation of those possibly infinite stable models of a program.

Definition 1. *An $\mathbb{FDNC}$ program is a finite disjunctive logic program whose rules are of the following forms:*

$$
\begin{array}{lrcl}
(R1) & A_1(x) \vee \ldots \vee A_n(x) & \leftarrow & (not)B_0(x), \ldots, (not)B_l(x) \\
(R2) & R_1(x,y) \vee \ldots \vee R_n(x,y) & \leftarrow & (not)P_0(x,y), \ldots, (not)P_l(x,y) \\
(R3) & R_1(x,f_1(x)) \vee \ldots \vee R_n(x,f_n(x)) & \leftarrow & (not)P_0(x,g_0(x)), \ldots, (not)P_l(x,g_l(x)) \\
(R4) & A_1(y) \vee \ldots \vee A_n(y) & \leftarrow & (not)B_0(Z_0), \ldots, (not)B_l(Z_l), R(x,y) \\
(R5) & A_1(f(x)) \vee \ldots \vee A_n(f(x)) & \leftarrow & (not)B_0(W_0), \ldots, (not)B_l(W_l), R(x,f(x)) \\
(R6) & R_1(x,f_1(x)) \vee \ldots \vee R_n(x,f_k(x)) & \leftarrow & (not)B_0(x), \ldots, (not)B_l(x) \\
(R7) & C_1(\boldsymbol{c_1}) \vee \ldots \vee C_n(\boldsymbol{c_n}) & \leftarrow & (not)D_1(\boldsymbol{d_1}), \ldots, (not)D_l(\boldsymbol{d_l}),
\end{array}
$$

where $n, l \geq 0$, each $Z_i \in \{x, y\}$, $W_i \in \{x, f(x)\}$, and each $\boldsymbol{c_i}$, $\boldsymbol{d_i}$ is a tuple of constants of arity ≤ 2. Each rule r is safe*, i.e., each of its variables occurs in* $\mathsf{body}^+(r)$*. Moreover, at least one rule is of type (R7) and is a fact.*

The fragments obtained from $\mathbb{FDNC}$ by disallowing disjunction, constraints or negative literals are denoted by respectively removing $\mathbb{D}$, $\mathbb{C}$ or $\mathbb{N}$ from "$\mathbb{FDNC}$".

The structure of the rules in $\mathbb{FDNC}$ syntax, the availability of non-monotonic negation and function symbols allow us to represent possibly infinite processes in a rather natural way. We provide here an example from the biology domain.

Example 1. The $\mathbb{FDNC}$ program P in Figure 1 represents the evolution of a cell; its growth and splitting into two cells. The rules (1)-(4) describe changes of a cell. If it is warm, a young cell will grow and a mature cell will split into two cells; any cell dies if it is cold. The rules (5)-(7) determine whether a cell is young or mature. The rules (8)-(11) state the knowledge about the temperature. During the growth (which takes longer time), it might alter, while in the other changes (which take short time), it stays the same; the latter is expressed by inertia rules (9) and (10). Finally, (12) and (13) are the initialization facts.

It is easy to see that P is consistent. In fact, it has infinitely many stable models, corresponding to the possible evolutions of the initial situation. It might have finite and infinite stable models, as cell splitting might go on forever. The piece of the stable model that is depicted represents a development where the temperature does not change during the growth of b and its child. Another stable model is $\{Young(b), Warm(b), Change(b, grow(b)), Cold(grow(b)), Mature(grow(b)),$ $Change(grow(b),\ die(grow(b)))\}$ which corresponds to the situation that the temperature changes and the bacterium dies.

The brave query $\exists x.Cold(x)$ evaluates to true, while it is not the case for the brave query $Change(b, die(b))$. Note that the query whether there is some evolution in which bacteria never die is expressed by adding the constraint $\leftarrow$ $Change(x, die(x))$ and asking whether the resulting program is consistent.

Example 2. $\mathbb{FDNC}$ is well-suited to encode action domain descriptions in transition-based action formalisms which support incomplete states and nondeterministic action effects, like $\mathcal{C}$ [7], $\mathcal{K}$ [3], or (propositional) situation calculus.

We outline the elements for a possible such encoding. A unary predicate $Sit(x)$ encodes the situation in which the domain is, where the initial situation is given by $Sit(init) \leftarrow$. State descriptions are in terms of unary fluent predicates $F(x)$, which intuitively means that F is true in the state associated with x.

A predicate $Trans(x, y)$ describes the transition from situation x to the next situation y; for this, the rule $Sit(y) \leftarrow Trans(x, y)$ is included. Transitions are due to actions α_1, ..., α_k, which can be represented using function symbols f_{α_1}, ..., f_{α_k}. A rule $Trans(x, f_{\alpha_1}(x)) \vee \cdots \vee Trans(x, f_{\alpha_k}(x)) \leftarrow Sit(x)$ may describe the action execution, while the constraints $\leftarrow Trans(x, f_{\alpha_i}(x)), Trans(x, f_{\alpha_j}(x))$, for all different i and j ensure that actions are not concurrent.

Action effects during a transition can be stated by rules, while executability conditions for actions can be stated by constraints; in particular, inertia for fluent F can be expressed using the rule $F(y) \leftarrow F(x), Trans(x, y), not\ neg\_F(y)$ where $neg\_F(x)$ is a predicate for the complement of F.

Using these elements, $\mathbb{FDNC}$ may be used to represent a number of actions domains from the literature, e.g., the Yale Shooting, Bomb in the Toilet, and others cf. [3]; in fact, usually $\mathbb{FN}$ is already convenient.

Example 1 shows that in presence of function symbols, an $\mathbb{FDNC}$ program may have infinite stable models. We present in the sequel a method to finitely represent the possibly infinite stable models. To this end, we first provide a semantic characterization of the stable models of an $\mathbb{FDNC}$ program.

3.1 Semantic Characterization of Stable Models

Like many decidable logics, including Description Logics, $\mathbb{FDNC}$ programs enjoy a *forest-shaped model property.* A stable model of an $\mathbb{FDNC}$ program can be viewed as a graph and a set of trees rooted at each of the nodes in the graph.

Proposition 1. *An interpretation I is* forest-shaped*, if the following hold:*

(a) Each atom in I is either unary or binary. Additionally, each binary atom is of the form $R(c, d)$ or $R(t, f(t))$, where c, d are constants, and t is a term.
(b) If $A \in I$ is an atom with a term of the form $f(t)$ occurring as an argument, then for some binary predicate symbol R, $R(t, f(t)) \in I$.

If H is an arbitrary interpretation for an $\mathbb{FDNC}$ program P and $J \in MM(P^H)$, then J is forest-shaped. Therefore, every $J \in SM(P)$ is forest-shaped.

The methods that we present in this paper are aimed at providing the decidability results together with the worst-case optimal algorithms for $\mathbb{FDNC}$. We note, however, that the decidability of the reasoning tasks discussed in this paper can be inferred from the results in [5]. The technique in [5] shows how the stable model semantics for the disjunctive logic programs with functions symbols can be expressed by formulae in second-order logic, where the minimality of models is enforced by second-order quantifiers. Due to the forest-shaped model property one can express the semantics of $\mathbb{FDNC}$ programs in monadic second-order logic over trees *SkS* which is know to be decidable (see [14] for a related encoding). Unfortunately, optimal algorithms for such encodings are not apparent.[1]

The semantic characterization, and the reasoning methods later on, follow an intuition that stable models for an $\mathbb{FDNC}$ program P can be constructed by the iterative computation of stable models of *local programs.* During the construction, local programs are obtained "on the fly" by taking certain finite subsets of $\mathsf{Ground}(P)$ and adding facts (*states*) obtained in the previous iteration.

For the rest of Section 3, we assume that P is an arbitrary $\mathbb{FDNC}$ program. For the convenience of presentation, for a term t and a set of atoms I, we write $t \hat{\in} I$, if there exists an atom in I with t as its argument.

Definition 2. *Let t be a term. A* state *of t is an arbitrary set U^t containing only unary atoms ground with t (i.e., with t as the argument); the superscript t will be dropped if t is not of particular interest. For a set of atoms I and a term $t \hat{\in} I$, we denote by $\mathsf{st}(I, t)$ the state of t in I, i.e., the set $\{A(t) \mid A(t) \in I\}$.*

For a one-variable rule r in $\mathbb{FDNC}$ syntax, let $r_{\downarrow t}$ denote the rule obtained by substituting every occurrence of the variable in r with a term t. Without loss of generality, we assume that in a two-variable rule, i.e., a rule of type (R2) or (R4), the tuple of variables in binary atoms is always $\langle x, y \rangle$. For such a rule r, let $r_{\downarrow s,t}$ denote the rule obtained by substituting every occurrence of x with a term s and every occurrence of y with a term t.

---

[1] Via an encoding into *SkS*, one can show the decidability of $\mathbb{FDNC}$ extended with *inverse* rules of the form $R(y, x) \leftarrow P(x, y)$, which, together with the rules of type (R4), allow for bidirectionality of information-passing. Due to more involved minimality-testing, our techniques cannot be extended easily to handle inverse rules.

Definition 3. *Let* U^t *be a state. The* local program $P(U^t)$ *is the smallest program containing the following rules:*

- $A(t) \leftarrow$*, for each* $A(t) \in U^t$*,*
- $r_{\downarrow t}$*, for each* $r \in P$ *of type (R3), (R5), or (R6),*
- $r_{\downarrow t,f(t)}$*, for each* $r \in P$ *of type (R2) or (R4) and function symbol* f *of* P*, and*
- $r_{\downarrow f(t)}$*, for each* $r \in P$ *of type (R1) and function symbol* f *of* P*.*

Suppose I is a forest-shaped interpretation for P, $t\hat{\in}I$, and U is the state of t in I, i.e., $U = \mathsf{st}(I,t)$. Intuitively, the stable models of $P(U)$ define the set of possible immediate successor structures for t in I. In other words, if I is a stable model of P, then I must contain a stable model of $P(U)$. Stable models of local programs have a simple structural property, captured by the notion of *knots*.

Definition 4. *(Knots) A* knot *with a root term* t *is a set of atoms* K *such that (i) each atom in* K *has form* $A(t)$*,* $R(t, f(t))$*, or* $A(f(t))$ *where* A*,* R*, and* f *are arbitrary, and (ii) for each term* $f(t)\hat{\in}K$*, there exists* $R(t, f(t)) \in K$ *(connectedness). Let* $\mathsf{succ}(K)$ *denote the set of all terms* $f(t)\hat{\in}K$*.*

A knot with a root term t can be viewed as a labeled tree of depth at most 1, where $\mathsf{succ}(K)$ are the leaf nodes. The nodes are labeled with unary predicate symbols, while the edges are labeled with binary predicate symbols. Note that $\emptyset$ is a knot whose root term can be arbitrary.

It is easy to see that due to the structure of local programs, their stable models satisfy the conditions in the definition of knots, and therefore are knots. On the other hand, knots are also the structures that appear in the trees of the forest-shaped interpretations. To "extract" a knot occurring in a forest-shaped interpretation, the following will be helpful.

For a term t, let $\mathcal{HB}_t$ denote the set of all atoms that can be built from unary and binary predicate symbols using t and terms of the form $f(t)$. For any forest-shaped interpretation I of P and $t\hat{\in}I$, the set $K := I \cap \mathcal{HB}_t$ is a knot.

A knot K with a root term t is *over (the signature of) P*, if each predicate and function symbol occurring in K also occurs in P (t need not be from $\mathcal{HU}^P$).

The following notion is central. The introduced *stable* knots are self-contained model building blocks for $\mathbb{FDNC}$ programs.

Definition 5. *(Stable Knot) Let* K *be a knot with a root term* t *and* $U^t = \mathsf{st}(K,t)$*. Then* K *is* stable *w.r.t. the program* P *iff* $K \in SM(P(U^t))$*.*

Intuitively, stable knots encode an assumption and a solution. Suppose a knot K with a root term t is stable w.r.t. P. Moreover, suppose t occurs in a forest-shaped interpretation I for P, as a "leaf node", i.e., there are no atoms of the form $R(t, f(t))$ in I. Intuitively, if the states of t in I and K coincide, i.e., $\mathsf{st}(I,t) = \mathsf{st}(K,t)$, then K becomes an eligible set of atoms that can be introduced in I to give t the necessary successors.

After introducing the necessary notions for dealing with the tree-part of forest-shaped interpretations, we deal with the graph part.

Definition 6. *By* P^{G} *we denote the program* $\mathsf{Ground}(P')$*, where* P' *is obtained from* P *by removing all the rules containing function symbols.*[2]

The following theorem characterizes the stable models of P. For an interpretation I, let I^c be the set of all atoms $A(\boldsymbol{c}) \in I$ such that $\boldsymbol{c}$ is a tuple of constants.

Theorem 1. *Let* I *be an interpretation for* P*. Then* I *is a stable model of* P *iff* I *is a forest-shaped interpretation such that (i)* I^c *is a stable model of* P^{G}*, and (ii) for each term* $t \hat{\in} I$*,* $I \cap \mathcal{HB}_t$ *is a knot that is stable w.r.t.* P*.*

3.2 Finite Representation of Stable Models

The semantic characterization of stable models for $\mathbb{FDNC}$ programs allows to view stable models as being constructed of knots, where each of them is stable. Next, we show that Theorem 1 allows us to provide a finite representation of those stable models. Roughly, it is based on the observation that although infinitely many knots might occur in some stable model of a program, only finitely many of them are non-isomorphic modulo the root term.

Definition 7. *Let* K *be a knot with a root term* t*. By* $K_{\downarrow u}$ *we denote the knot obtained from* K *by replacing each occurrence of* t *in* K *with a term* u*.*

Indeed, if the program P has an infinite stable model I, then set of knots $L := \{(I \cap \mathcal{HB}_t) \mid t \hat{\in} I\}$ is infinite. However, for a fixed term t, the set $L' := \{K_{\downarrow t} \mid K \in L\}$ is finite due to the fact that there are only finitely many knots with the root term t over the signature of P. Intuitively, if we view t as a variable, then each $K \in L$ can be viewed as an instance of some knot in L'.

To talk about sets of knots with common root term, we assume a special constant $\mathbf{x}$ not occurring in $\mathbb{FDNC}$ programs. We say a set of knots is $\mathbf{x}$-*grounded*, if it contains only knots with root term $\mathbf{x}$. The following notion lets us collect the knots occurring in a stable model and abstract them by substituting with $\mathbf{x}$.

Definition 8. *(Scanning) Let* I *be a forest-shaped interpretation for* P*. We define the set of* $\mathbf{x}$*-grounded knots* $\mathbb{K}(I) := \{(I \cap \mathcal{HB}_t)_{\downarrow \mathbf{x}} \mid t \hat{\in} I\}$*.*

In the following we show that $\mathbf{x}$-grounded sets of knots can be used to represent the stable models of an $\mathbb{FDNC}$ program. First, we observe that the stability of a knot is preserved under substitutions.

Proposition 2. *If* K *is a knot that is stable w.r.t.* P*, and* u *is an arbitrary term, then* $K_{\downarrow u}$ *is stable w.r.t.* P*.*

We introduce the notion of *founded* sets of $\mathbf{x}$-grounded knots. The intention is to capture the properties of the set $\mathbb{K}(I)$ when I is a stable model of P. To this end, we need a notion of *state equivalence* as a counterpart for substitutions in knots. Formally, states U^t and V^s are *equivalent* (in symbols, $U^t \approx V^s$), if $U^t = \{A(t) \mid A(s) \in V^s\}$, i.e., in both states terms satisfy the same unary predicates.

---

[2] Note that P^{G} is finite since its Herbrand universe contains only the constants of P.

Definition 9. *Let $S \neq \emptyset$ be a set of states. A set L of $\mathbf{x}$-grounded knots that are stable w.r.t. the program P is* founded *w.r.t. P and S, if the following hold:*

1. *For each $U \in S$, there exists $K \in L$ such that $U \approx \mathsf{st}(K, \mathbf{x})$.*
2. *For each $K \in L$, the following hold:*
   a. *for each $s \in \mathsf{succ}(K)$, there exists $K' \in L$ s.t. $\mathsf{st}(K, s) \approx \mathsf{st}(K', \mathbf{x})$, and*
   b. *there exists a sequence $\langle K_0, \ldots, K_n \rangle$ of knots in L such that:*
      - *$K_n = K$,*
      - *K_0 is such that $\mathsf{st}(K, \mathbf{x}) \approx U$ for some $U \in S$, and*
      - *for each $0 \leq i < n$, there exists $s \in \mathsf{succ}(K_i)$ s.t. $\mathsf{st}(K_i, s) \approx \mathsf{st}(K_{i+1}, \mathbf{x})$.*

For an interpretation I, let $S(I)$ denote the set of states of constants occurring in I, i.e., $S(I) := \{\mathsf{st}(I, c) \mid c \hat{\in} I \text{ is a constant}\}$. The following is easy to verify.

Proposition 3. *If I is a stable model of P, then $\mathbb{K}(I)$ is a set of knots that is founded w.r.t. P and $S(I^c)$.*

In what follows we provide a construction of stable models out of knots in a founded set. Moreover, we show that for a given consistent program there exists a founded set of knots that captures all the stable models.

Generating Stable Models out of Knots. Before describing the construction of forest-shaped interpretations, we first state the construction of trees, which are represented in the standard way by prefix-closed sets of words. For a sequence of elements $p = [e_1, \ldots, e_n]$, let $\tau(p)$ denote the last element e_n, and $[p|e_{n+1}]$ denote the sequence $[e_1, \ldots, e_n, e_{n+1}]$.

Definition 10. *(Tree Construction) Let L be a set of knots that is founded w.r.t. P and a set of states S, and let U^t be a state such that $U^t \approx V$, for some $V \in S$. A set T of sequences, where each element in a sequence is a tuple of a knot and a term, is called a* tree induced by L starting at U^t, *if the following hold:*

(a) $[\langle K, t \rangle] \in T$, where $K \in L$ is s.t. $\mathsf{st}(K, \mathbf{x}) \approx U^t$.
(b) If there exists $p \in T$ with $\tau(p) = \langle K, t \rangle$ and $f(\mathbf{x}) \in \mathsf{succ}(K)$, then there exists $[p|\langle K', f(t) \rangle] \in T$, where K' is a knot in L s.t. $\mathsf{st}(K, f(\mathbf{x})) \approx \mathsf{st}(K', \mathbf{x})$.
(c) T is minimal, i.e., each $T' \subset T$ violates (a) or (b).

We state the transformation of trees into Herbrand interpretations.

Definition 11. *Let T be a tree induced by a founded set of knots L starting at some state. We define the set of atoms $T_{\downarrow} := \{K_{\downarrow t} \mid p \in T \text{ with } \tau(p) = \langle K, t \rangle\}$.*

We generalize the construction of trees to forest-shaped interpretations.

Definition 12. *(Forest Construction) Let G be a set of atoms ground with the constants of P only, and L be a set of knots founded w.r.t. P and a set of states $S \supseteq S(G)$. Then $\mathcal{F}(G, L)$ is the largest set of forest-shaped interpretations*

$$I = G \cup (T^{c_1})_{\downarrow} \cup \ldots \cup (T^{c_n})_{\downarrow},$$

where $\{c_1, \ldots, c_n\}$ is the set of all constants occurring in G and each T^{c_i} a tree induced by L starting at $\mathsf{st}(G, c_i)$.

$\mathcal{F}(G, L)$ represents all the forest-shaped interpretations that can be build from G by attaching, for each of the constants, a tree induced by L.

Theorem 2. *If $G \in SM(P^{\mathsf{G}})$, L is a set of knots that is founded w.r.t. P and some $S \supseteq S(G)$, then $\mathcal{F}(G, L) \neq \emptyset$ and each $I \in \mathcal{F}(G, L)$ is a stable model of P.*

Proof. Indeed, $\mathcal{F}(G, L) \neq \emptyset$ due to foundedness of L. Assume some $I \in \mathcal{F}(G, L)$. Each $K \in L$ is stable w.r.t. P. Then due to Proposition 2, for each term $t \hat{\in} I$, $I \cap \mathcal{HB}_t$ is a knot that is stable w.r.t. P. Keeping in mind that $G \in SM(P^{\mathsf{G}})$, Theorem 1 implies that I is a stable model of P.

We showed that stable model existence can be proved by checking that some founded set of knots exists. As we see next, the properties of founded sets of knots imply that we can obtain a set capturing all the stable models of a program.

Capturing Stable Models. We define the set of states that occur in the stable models of P^{G} as $S(P) := \{\mathsf{st}(G, c) \mid G \in SM(P^{\mathsf{G}}) \wedge c \hat{\in} G\}$.

Definition 13. *By $\mathbb{K}_P$ we denote the smallest set of knots which contains every set of knots L that is founded w.r.t. P and some $S \subseteq S(P)$.*

Proposition 4. *For the program P, the following hold:*

(a) If $\mathbb{K}_P \neq \emptyset$, then $\mathbb{K}_P$ is founded w.r.t. P and some $S \subseteq S(P)$.
(b) If L is a set of knots that is founded w.r.t. P and some $S \subseteq S(P)$, then $\mathbb{K}_P$ is founded w.r.t. P and some $S' \supseteq S$.
(c) Each $L \supset \mathbb{K}_P$ is not founded w.r.t. P and any $S \subseteq S(P)$.

Proof. The claim follows from the following property: if L_1 and L_2 are two sets of knots founded w.r.t. P and, respectively, sets of states S_1 and S_2, then $L_1 \cup L_2$ is founded w.r.t. P and $S_1 \cup S_2$.

It is easy to verify that a stable model I can be reconstructed out of knots in $\mathbb{K}(I)$. Naturally, the same holds for any superset of $\mathbb{K}(I)$ satisfying Definition 9.

Proposition 5. *If I is a stable model of P, then $I \in \mathcal{F}(I^c, L)$ for each set of knots $L \supseteq \mathbb{K}(I)$ s.t. L is founded w.r.t. P and some set of states $S \supseteq S(I^c)$.*

The following will be helpful.

Definition 14. *We say $\mathbb{K}_P$ is* compatible *with a set of states S, if for each state $U \in S$, there exists $K \in \mathbb{K}_P$ s.t. $U \approx \mathsf{st}(K, \mathbf{x})$.*

The crucial property of $\mathbb{K}_P$ is that it captures the tree-structures of all the stable models of P. Together with the stable models of P^{G}, it represents the stable models of P.

Theorem 3. *Let I be an interpretation for P. Then, $I \in SM(P)$ iff $I \in \mathcal{F}(G, \mathbb{K}_P)$, for some $G \in SM(P^{\mathsf{G}})$ s.t. $\mathbb{K}_P$ is compatible with $S(G)$.*

Table 1. Complexity of $\mathbb{FDNC}$ and Fragments (Completeness Results)

Fragments	Consistency	$P \models_b A(\boldsymbol{t})$	$P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$
$\mathbb{F}$	Trivial	P	PSpace
$\mathbb{FD}$	Trivial	Σ_2^P	PSpace
$\mathbb{FC}$	PSpace	PSpace	PSpace
$\mathbb{FDC}$, $\mathbb{FN}$, $\mathbb{FDNC}$	ExpTime	ExpTime	ExpTime

Proof. If $I \in SM(P)$, then, by Prop. 3, $\mathbb{K}(I)$ is founded w.r.t. P and $S(I^c)$. By definition, $\mathbb{K}(I) \subseteq \mathbb{K}_P$. By Prop. 4, $\mathbb{K}_P$ is founded w.r.t. P and some $S \supseteq S(I^c)$. By Proposition 5, $I \in \mathcal{F}(I^c, \mathbb{K}_P)$. The other direction is proved by Theorem 2.

We have obtained a finite representation of stable model of an $\mathbb{FDNC}$ program. Indeed, all of its stable models can be generated out of some stable model of P^{G} and a set of knots $\mathbb{K}_P$.

4 Reasoning and Complexity

The algorithms for consistency check and brave entailment are based on computing $\mathbb{K}_P$. The complexity of these tasks for $\mathbb{FDNC}$ and its fragments is compactly summarized in Table 1. For space reasons, we focus here on $\mathbb{FDNC}$ and briefly discuss the other fragments at the end of this section.

Deriving the Set $\mathbb{K}_P$*.* To derive $\mathbb{K}_P$, we proceed in two phases. In the first phase, we generate the set of knots $\mathsf{All}(P)$ that surely contains $\mathbb{K}_P$. In the second phase, we remove knots from it to ensure that it satisfies Definition 13.

To ease presentation, for a knot set L, let $\mathsf{states}(L) := \{\mathsf{st}(K, s) \mid K \in L, s \in \mathsf{succ}(K)\}$ be the set of all states of the successor terms of knots in L.

Definition 15. *For an* $\mathbb{FDNC}$ *program* P*, let* $\mathsf{All}(P)$ *be the smallest set of* $\mathbf{x}$*-grounded knots obeying the following conditions:*

a) If $U \in S(P)$ *and* $K \in SM(P(U))$*, then* $K_{\downarrow\mathbf{x}} \in \mathsf{All}(P)$*.*
b) If $U \in \mathsf{states}(\mathsf{All}(P))$ *and* $K \in SM(P(U))$*, then* $K_{\downarrow\mathbf{x}} \in \mathsf{All}(P)$*.*

By construction, $\mathsf{All}(P)$ contains each set of knots which is founded w.r.t. P and some set of states $S \subseteq S(P)$. The problem is that $\mathsf{All}(P)$ might contain a knot K such that some $s \in \mathsf{succ}(K)$ has no potential successor knot (see (2.a) in Definition 9). Such knots should be removed from $\mathsf{All}(P)$. In turn, such removal might leave some knots in $\mathsf{All}(P)$ without a potential predecessor (see (2.b) in Definition 9). The second phase deals with this problem.

Definition 16. *For any set of* $\mathbf{x}$*-grounded knots* L *and set of states* S*,* $\mathsf{reach}(L, S)$ *is the smallest set of knots such that:*

a) if $U \in S$*,* $K \in L$ *and* $U \approx \mathsf{st}(K, \mathbf{x})$*, then* $K \in \mathsf{reach}(L, S)$*, and*
b) if $U \in \mathsf{states}(\mathsf{reach}(L, S))$*,* $K \in L$ *and* $U \approx \mathsf{st}(K, \mathbf{x})$*, then* $K \in \mathsf{reach}(L, S)$*.*

```
Algorithm Knots
Input: $\mathbb{FDNC}$ program $P$
Output: $\mathbb{K}_P$
repeat
  $L := \mathsf{All}(P)$; $S := S(P)$; $L^{aux} := L$;
  for each $K \in L$ and $s \in \mathsf{succ}(K)$ do
      if not $\exists K' \in L$ s.t. $\mathsf{st}(K, s) \approx \mathsf{st}(K', \mathbf{x})$ then $L := L \setminus \{K\}$;
  $L := \mathsf{reach}(L, S)$
until $L^{aux} = L$
return $L$
```

Fig. 2. Algorithm for computing $\mathbb{K}_P$ of an $\mathbb{FDNC}$ program

Intuitively, $\mathsf{reach}(L, S)$ are the knots in L reachable from the states in S. Thus, if $\mathsf{reach}(L, S) = L$, then L fulfills the condition (2.b) of Definition 9 to be founded w.r.t. S.

The cleaning of $\mathsf{All}(P)$ involves removing each knot violating (2.a) or (2.b) in Definition 9. Figure 2 shows an algorithm for computing $\mathbb{K}_P$ which elaborates on this.

Soundness and Completeness. We must verify that *Knots(P)* satisfies the condition in Definition 13, i.e., *Knots(P)* is the single (set-inclusion) minimal set which contains each set L of knots that is founded w.r.t. P and some $S \subseteq S(P)$.

Indeed, $L \subseteq \mathsf{All}(P)$ by construction. In the computation of *Knots(P)* no knot in L can be removed from $\mathsf{All}(P)$, i.e., $L \subseteq Knots(P)$. Suppose *Knots(P)* is not minimal. Hence, some $N \subset Knots(P)$ contains every knot set L which is founded w.r.t. P and some $S \subseteq S(P)$. Then $Knots(P)$ must be nonempty, and it holds that $Knots(P)$ is founded w.r.t. P and some $S \subseteq S(P)$. Roughly, this is because the algorithm ensures that every knot in $Knots(P)$ is stable w.r.t. P, has proper successors to satisfy (2.a) in Definition 9, and has a proper sequence of predecessors to satisfy (2.b) reaching a state in $S(P)$. By assumption on N and foundedness of $Knots(P)$, we have $Knots(P) \subseteq N$. This, however, contradicts $N \subset Knots(P)$. Thus $Knots(P)$ satisfies Definition 13, i.e., $Knots(P) = \mathbb{K}_P$.

Complexity. The procedure $Knots(P)$ runs in time single exponential in the size of P. The claim follows from the following observations:

- The number of $\mathbf{x}$-grounded knots over P, max, is bounded by single exponential in the size of P; more precisely, $max \leq 2^{n+k\cdot(n+m)}$, when P has k function, n unary, and m binary predicate symbols.
- Computing $\mathsf{All}(P)$ requires adding at most max $\mathbf{x}$-grounded knots. Each such knot has polynomial size and its stability is verifiable using an NP^{NP} oracle. Thus, $\mathsf{All}(P)$ is computable in time single exponential in the size of P.
- Computing $\mathsf{reach}(L, S)$ is polynomial in the combined size of L and S.
- The size of $S(P)$ is bounded by a single exponential in the size of P.
- $Knots(P)$ runs in time that is polynomial in the size of $\mathsf{All}(P)$ and $S(P)$.

Consistency Check. Theorems 2 and 3 imply the following characterization.

Theorem 4. *An* $\mathbb{FDNC}$ *program* P *is consistent iff* $\mathbb{K}_P$ *is compatible w.r.t.* $S(G)$*, for some* $G \in SM(P^{\mathsf{G}})$*.*

Compatibility of $\mathbb{K}_P$ w.r.t. $S(G)$, for $G \in SM(P^{\mathsf{G}})$, is decidable in time polynomial in $n+m$, where m is the size of $\mathbb{K}_P$ and n is the size of $SM(P^{\mathsf{G}})$. This is single exponential in the size of P, since both m and n are single exponential in the size of P. Since $SM(P^{\mathsf{G}})$ is computable in single exponential time, Theorem 4 implies that consistency checking in $\mathbb{FDNC}$ is feasible in single exponential time. In the full paper, by a reduction of satisfiability testing in the ExpTime-hard DL $\mathcal{ALC}$, we show that consistency check is ExpTime-hard already for $\mathbb{FDC}$. Thus, the algorithm emerging from Theorem 4 is worst-case optimal.

Theorem 5. *Deciding whether a given* $\mathbb{FDNC}$ *program is consistent, i.e., has some stable model, is* ExpTime*-complete.*

Brave Entailment. We can also exploit $\mathbb{K}_P$ for brave reasoning in $\mathbb{FDNC}$. We focus here on unary atomic queries; binary queries are easily reduced to this case. The idea is to perform "back-propagation" of unary predicate symbols in a founded set of knots. For a set of $\mathbf{x}$-grounded knots L, we call $K' \in L$ a *possible successor* of $K \in L$ if $\mathsf{st}(K', \mathbf{x}) \approx \mathsf{st}(K, s)$ for some $s \in \mathsf{succ}(K)$.

Definition 17. *Let* L *be a set of knots founded w.r.t. an* $\mathbb{FDNC}$ *program* P *and a set of states* S*. Let* C *be the set of unary predicate symbols occurring in* P*. By* $\mathcal{E}_L$ *we denote the smallest relation over* $L \times C$ *closed under the following rules:*

(a) if $K \in L$ *and some* $A(\mathbf{x}) \in K$*, then* $\langle K, A\rangle \in \mathcal{E}_L$*, and*
(b) if $K' \in L$ *is a possible successor of* $K \in L$ *s.t.* $\langle K', A\rangle \in \mathcal{E}_L$*, then* $\langle K, A\rangle \in \mathcal{E}_L$*.*

Intuitively, $\langle K, A\rangle \in \mathcal{E}_L$ means that starting from K a sequence of possible successor knots will eventually reach a knot containing $A(\mathbf{x})$. We have the following:

Theorem 6. *Let* P *be an* $\mathbb{FDNC}$ *program. Then,* $P \models_b \exists x.A(x)$ *iff* $(\star)$ *for some* $G \in SM(P^{\mathsf{G}})$*, (a)* $\mathbb{K}_P$ *is compatible w.r.t.* $S(G)$*, and (b) there exist a constant* c *and* $K \in \mathbb{K}_P$ *such that* $\mathsf{st}(G, c) \approx \mathsf{st}(K, \mathbf{x})$ *and* $\langle K, A\rangle \in \mathcal{E}_{\mathbb{K}_P}$*.*

Condition $(\star)$ is verifiable in time (single) exponential in the size of P. Indeed, computing $\mathcal{E}_{\mathbb{K}_P}$ requires time quadratic in the size of $\mathbb{K}_P$, or exponential in the size of P. Once $\mathbb{K}_P$, $\mathcal{E}_{\mathbb{K}_P}$, and $SM(P^{\mathsf{G}})$ are computed, the conditions in $(\star)$ are verifiable in time polynomial in the combined size of $\mathbb{K}_P$, $\mathcal{E}_{\mathbb{K}_P}$, and $SM(P^{\mathsf{G}})$.

Via this algorithm, we obtain that brave reasoning for existential unary queries in $\mathbb{FDNC}$ is in ExpTime. In the full paper we show that the algorithm is worst-case optimal (by reducing consistency to brave entailment in $\mathbb{FDNC}$), and that this result extends to binary existential queries.

Theorem 7. *Deciding* $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ *is* ExpTime*-complete for* $\mathbb{FDNC}$*.*

The method for deciding brave entailment of ground unary queries is based on an adaptation of the algorithm for the existential queries.

Definition 18. *Let $q = A(t)$ be a ground atom and L be a set of knots founded w.r.t. an $\mathbb{FDNC}$ program P and a set of states S. Let T be the set of subterms of the term t. Then $\mathcal{G}_L^q$ is the smallest relation over $L \times T$ such that:*

(a) if $K \in L$ and $A(\mathbf{x}) \in K$, then $\langle K, t\rangle \in \mathcal{G}_L^q$, and
(b) if there exist (i) $K \in L$ with $f(\mathbf{x}) \in \mathsf{succ}(K)$ and (ii) $K' \in L$ s.t. $\mathsf{st}(K, f(\mathbf{x})) \approx \mathsf{st}(K', \mathbf{x})$ and $\langle K', f(v)\rangle \in \mathcal{G}_L^q$, then $\langle K, v\rangle \in \mathcal{G}_L^q$.

Suppose we have a ground query $q = A(f(g(f(c))))$ and a knot K in L such that $\langle K, c\rangle \in \mathcal{G}_L^q$. Roughly, it means that we can construct a tree with root term c containing a node $f(g(f(c)))$ labeled with A. The following is easily verified.

Theorem 8. *For any $\mathbb{FDNC}$ program P and ground query q, $P \models_b q$ iff ($\star\star$) for some $G \in SM(P^{\mathsf{G}})$, (a) $\mathbb{K}_P$ is compatible w.r.t. $S(G)$, and (b) some $K \in \mathbb{K}_P$ exists s.t. $\mathsf{st}(G, c) \approx \mathsf{st}(K, \mathbf{x})$ and $\langle K, c\rangle \in \mathcal{G}_{\mathbb{K}_P}^q$, where c is the constant in q.*

By similar arguments as for existential queries, we can see that checking condition ($\star\star$) is feasible in time single exponential in the size of P. Note that computing $\mathcal{G}_{\mathbb{K}_P}^q$ requires time that is polynomial in the size of $\mathbb{K}_P$, or single exponential in the size of P. Once $\mathbb{K}_P$, $\mathcal{G}_{\mathbb{K}_P}^q$, and $SM(P^{\mathsf{G}})$ are computed, the conditions in ($\star\star$) can be verified in time polynomial in the combined size of $\mathbb{K}_P$, $\mathcal{G}_{\mathbb{K}_P}^q$, and $SM(P^{\mathsf{G}})$, each of which is single exponential in the size of P.

We thus have an algorithm for deciding $P \models_b A(t)$ in exponential time. The full paper shows that it is worst-case optimal, by providing an ExpTime-hardness result, and extends the result to binary ground queries.

Theorem 9. *Deciding $P \models_b A(\boldsymbol{t})$, for ground $\boldsymbol{t}$, is* ExpTime*-complete for $\mathbb{FDNC}$.*

The remaining entries in Table 1 are briefly explained as follows. $\mathbb{F}$ and $\mathbb{FD}$ programs are trivially consistent. For an $\mathbb{FC}$ program P, consistency can be decided by checking if each constant c in the single stable model G of P^{G} (if it exists) has a set of knots founded w.r.t. P and $\{\mathsf{st}(G, c)\}$. This can be refuted by nondeterministically constructing stepwise a sequence of at most exponentially many knots which leads to inconsistency. This is feasible in polynomial space, and since NPSpace = PSpace, consistency checking is in PSpace. Matching PSpace-hardness is shown by a generic Turing machine reduction, where a simple constraint of form $\leftarrow A(x)$ is sufficient to show the hardness.

The PSpace-hardness of $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ for $\mathbb{F}$ is immediate from the fact $P \models_b \exists \boldsymbol{x}.A(\boldsymbol{x})$ holds iff $P \cup \{\leftarrow A(\boldsymbol{x})\}$ is inconsistent. The problem is in PSpace, since we can nondeterministically construct a sequence of at most exponentially many knots until A occurs. For $\mathbb{FD}$, this can be decided similarly; for $\mathbb{FC}$, an additional consistency check has to be made, but polynomial space is sufficient.

In case of ground entailment $P \models_b A(\boldsymbol{t})$, membership of $A(\boldsymbol{t})$ in the single stable model of an $\mathbb{F}$ or $\mathbb{FC}$ program is witnessed by a sequence of knots that is fully determined by $\boldsymbol{t}$ and computable in polynomial time. For $\mathbb{FC}$, the consistency check of P remains to be done. In case of $\mathbb{FD}$, we still need to guess knots, guided by $\boldsymbol{t}$, and verify their stability; this is feasible in Σ_2^P. Matching Σ_2^P-hardness follows from propositional logic programs [4].

Finally, ExpTime-completeness of the reasoning tasks for $\mathbb{FN}$ is proved by a polynomial reduction of consistency check in $\mathbb{FDC}$ to consistency check in $\mathbb{FN}$.

5 Related Work

Our $\mathbb{FN}$ programs are decidable *Finitely Recursive Programs (FRPs)* [2,1], which are normal logic programs P with function symbols where in the grounding of P, each atom depends only on finitely many atoms; disjunction and constraints are not allowed. For FRPs, inconsistency checking is r.e.-complete and brave ground entailment is co-r.e.-complete [1]; for $\mathbb{FN}$ and our full class $\mathbb{FDNC}$, which implicitly obeys the condition of FRPs, these problems are ExpTime-complete. On the other hand, $\mathbb{FN}$ is not a subclass of the *Finitary Programs (FPs)* [2], which are those RFPs in whose grounding only finitely many atoms occur in odd cycles. For FPs, consistency checking is decidable, and brave and cautious entailment are decidable for ground queries but r.e.-complete for existential atomic queries. Note that for $\mathbb{FN}$, all these problems are decidable in exponential time. Finally, the explicit syntax of $\mathbb{FN}$ and our other fragments of $\mathbb{FDNC}$ allows to effectively recognize such programs. FRPs and FPs, instead, suffer the undecidability of the conditions that define them, i.e., FRPs and FPs cannot be effectively recognized.

Related to our work are *Local Extended Conceptual Logic Programs (LECLPs)* [9], which evolved from [8]. These programs are function-free but have answer sets over *open domains*, i.e., of the grounding of a program with any superset of its constants. LECLPs are syntactically restricted to ensure the forest-shape model property of answer sets. Deciding consistency of an LECLP P is feasible in nondeterministic triple exponential time, as one can ground P with double exponentially many constants in the size of P, and then use standard ASP. For $\mathbb{FDNC}$, deciding the consistency is ExpTime-complete and thus less complex.

Comparing the expressiveness of LECLPs and $\mathbb{FDNC}$ is intricated due the different settings. At least, both formalisms can encode certain description logics (e.g., $\mathcal{ALC}$). LECLPs may be more expressive than $\mathbb{FDNC}$ programs, since the expressive DL $\mathcal{ALCHOQ}$ is reducible to satisfiability in LECLPs. On the other hand, LECLPs undermine the general intuition behind minimal model semantics of logic programs. So-called *free rules* of the form $p(x) \vee \mathit{not}\ p(x) \leftarrow$; allow to unfoundedly add atoms in an answer set. $\mathbb{FDNC}$, instead, has no free rules, and each atom in a stable model of P must be justified from the very facts of P.

6 Discussion and Conclusion

In line with efforts to pave the way for effective Answer Set Programming engines with function symbols [2,1], we presented $\mathbb{FDNC}$ programs as a decidable class of disjunctive logic programs with function symbols under stable model semantics. They are a tool for knowledge representation and reasoning for some applications involving infinite processes and objects, like evolving action domains. From our results on consistency checking and brave entailment of ground and existential atomic queries q, one can easily determine the complexity of cautious entailment

$P \models_c q$, i.e., whether q is true in all stable models of P. The results in Table 1 for brave entailment carry over to cautious entailment except for $\mathbb{FD}$; here, $P \models_c A(\boldsymbol{t})$ is coNP-complete and $P \models_c \exists \boldsymbol{x}.A(\boldsymbol{x})$ is ExpTime-complete. Intuitively, the former is because minimality of models is irrelevant for inference of an atom $A(\boldsymbol{t})$, and the latter because consistency checking with constraints $\leftarrow B(\boldsymbol{x})$ can be reduced to cautious inference with rules $A(\boldsymbol{x}) \leftarrow B(\boldsymbol{x})$.

$\mathbb{FDNC}$ programs can be easily extended with strong negation $\neg p(\boldsymbol{x})$ [6], which can be expressed in the language as usual (view $\neg p$ as a predicate symbol and add constraints $\leftarrow p(\boldsymbol{x}), \neg p(\boldsymbol{x})$). Implementation of $\mathbb{FDNC}$ programs is another subject of future work. To this aim, recent extensions of the DLV system like DLVHEX (`http://con.fusion.at/dlvhex/`) might be exploited.

Acknowledgments. We thank a reviewer for pointing out the decidability of $\mathbb{FDNC}$ (even with inverse rules) via monadic second-order logic over trees.

References

1. Baselice, S., Bonatti, P.A., Criscuolo, G.: On Finitely Recursive Programs. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007, September 8-13. LNCS, vol. 4670, pp. 89–103. Springer, Heidelberg (2007) `http://dx.doi.org/10.1007/978-3-540-74610-2_7`
2. Bonatti, P.A.: Reasoning with infinite stable models. Artif. Intell. 156(1), 75–111 (2004)
3. Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity. ACM Transactions on Computational Logic 5(2), 206–263 (2004)
4. Eiter, T., Gottlob, G.: On the Computational Cost of Disjunctive Logic Programming: Propositional Case. Annals of Mathematics and Artificial Intelligence 15(3/4), 289–323 (1995)
5. Eiter, T., Gottlob, G.: Expressiveness of stable model semantics for disjuncitve logic programs with functions. J. Log. Program. 33(2), 167–178 (1997)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Comput. 9(3/4), 365–386 (1991)
7. Giunchiglia, E., Lifschitz, V.: An Action Language Based on Causal Explanation: Preliminary Report. In: AAAI 1998. Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 623–630 (1998)
8. Heymans, S.: Decidable Open Answer Set Programming. PhD thesis, Theoretical Computer Science Lab (TINF), Department of Computer Science, Vrije Universiteit Brussel, Pleinlaan 2, B1050 Brussel, Belgium (February 2006)
9. Heymans, S., Nieuwenborgh, D.V., Vermeir, D.: Nonmonotonic ontological and rule-based reasoning with extended conceptual logic programs. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 392–407. Springer, Heidelberg (2005)
10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Transactions on Computational Logic 7(3), 499–562 (2006)
11. Marek, V.W., Remmel, J.B.: On the expressibility of stable logic programming. In: LPNMR, pp. 107–120 (2001)

12. Marek, W., Nerode, A., Remmel, J.: How Complicated is the Set of Stable Models of a Recursive Logic Program? Annals of Pure and Applied Logic 56, 119–135 (1992)
13. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann, San Francisco (1988)
14. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. In: Proc. of WWW 2007, pp. 807–816 (2007)
15. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence 138, 181–234 (2002)
16. Woltran, S.: Answer Set Programming: Model Applications and Proofs-of-Concept. Technical Report WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004) (July 2005), available at `http://www.kr.tuwien.ac.at/projects/WASP/report.html`

The Complexity of Temporal Logic with Until and Since over Ordinals*

Stéphane Demri[1] and Alexander Rabinovich[2]

[1] LSV, ENS Cachan, CNRS, INRIA
`demri@lsv.ens-cachan.fr`
[2] Tel Aviv University, Ramat Aviv
`rabinoa@post.tau.ac.il`

Abstract. We consider the temporal logic with since and until modalities. This temporal logic is expressively equivalent over the class of ordinals to first-order logic thanks to Kamp's theorem. We show that it has a PSPACE-complete satisfiability problem over the class of ordinals. Among the consequences of our proof, we show that given the code of some countable ordinal α and a formula, we can decide in PSPACE whether the formula has a model over α. In order to show these results, we introduce a class of simple ordinal automata, as expressive as Büchi ordinal automata. The PSPACE upper bound for the satisfiability problem of the temporal logic is obtained through a reduction to the nonemptiness problem for the simple ordinal automata.

1 Introduction

The main models for time are $\langle \mathbb{N}, < \rangle$, the natural numbers as a model of *discrete time* and $\langle \mathbb{R}, < \rangle$, the real line as the model for *continuous time*. These two models are called the *canonical models of time*. A major result concerning linear-time temporal logics is Kamp's theorem [Kam68, GHR94] which says that $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$, the temporal logic having "*Until*" and "*Since*" as only modalities, is expressively complete for first-order monadic logic of order over the class of Dedekind complete linear orders. The canonical models of time are indeed Dedekind-complete. Another important class of Dedekind-complete orders is the class of ordinals, see e.g. an axiomatization of $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ over ordinals in [Ven93].

In this paper the satisfiability problem for the temporal logic with until and since modalities over the class of ordinals is investigated. Our main results are the following: the satisfiability problem for $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ over the class of ordinals is PSPACE-complete and a formula ϕ in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ has some α-model for some ordinal α iff it has an β-model for some $\beta < \omega^{|\phi|+2}$ where $|\phi|$ is the size of ϕ.

In order to prove these results we use an automata-based approach [VW94]. In Section 3, we introduce a new class of ordinal automata which we call simple ordinal automata. These automata are expressive equivalent to Büchi automata

---

* Partially supported by an invited professorship from ENS de Cachan and project AutoMathA (ESF).

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 531–545, 2007.
© Springer-Verlag Berlin Heidelberg 2007

over ordinals [BS73]. However, the locations and the transition relations of these automata have additional structures as in [VW94, Roh97]. In particular, a location is a subset of a base set X. Herein, we provide a translation from formulae in LTL(U, S) into simple ordinal automata that allows to characterize the complexity of the satisfiability problem for LTL(U, S). However, the translation of the formula ϕ into the automaton $\mathcal{A}_\phi$ provides an automaton of exponential size in $|\phi|$ but the base of $\mathcal{A}_\phi$ has a cardinality linear in $|\phi|$.
Section 4 contains our main technical lemmas. We show there that every run in a simple ordinal automaton is equivalent to a short run. Consequently, we establish that a formula $\phi \in$ LTL(U, S) has an α-model iff it has a model of length $\mathrm{trunc}_{|\phi|+2}(\alpha)$ where $\mathrm{trunc}_{|\phi|+2}(\alpha)$ is a truncated part of α strictly less than $\omega^{|\phi|+2} \times 2$ (see the definition of truncation in Section 4).
In Section 5 we present two algorithms to solve the nonemptiness problem for simple ordinal automata. The first one runs in (simple) exponential time and does not take advantage of the short run property. The second algorithm runs in polynomial space and the short run property plays the main role in its design and its correctness proof.

In Section 6 we investigate several variants of the satisfiability problem and show that all of them are PSPACE-complete. Section 7 compares our results with related works. The satisfiability problem for LTL(U, S) over ω-models is PSPACE-complete [SC85]. Reynolds [Rey03, Rey] proved that the satisfiability problem for LTL(U, S) over the reals is PSPACE-complete. The proofs in [Rey03, Rey] are non trivial and difficult to grasp and it is therefore difficult to compare our proof technique with those of [Rey03, Rey] even though we believe cross-fertilization would be fruitful. We provide uniform proofs and we improve upper bounds for decision problems considered in [Cac06, DN07, Roh97]. We also compare our results and techniques with Rohde's thesis [Roh97]. Finally we show how our results entail most of the results from [DN07] and we solve some open problems stated there.

2 Temporal Logic with Until and Since

The formulae of LTL(U, S) are defined as follows:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathsf{U} \phi_2 \mid \phi_1 \mathsf{S} \phi_2$$

where $p \in$ PROP for some set PROP of atomic propositions. Given a formula ϕ in LTL(U, S), we write $sub(\phi)$ to denote the set of subformulae of ϕ or their negation assuming that $\neg\neg\psi$ is identified with ψ. The size of ϕ is defined as the cardinality of $sub(\phi)$ and therefore implicitly we encode formulae as DAGs. This feature will be helpful for defining translations that increase polynomially the number of subformulae but for which the tree representation might suffer an exponential blow-up. We use the following abbreviations $\mathsf{G}\phi = \phi \wedge \neg(\top \mathsf{U} \neg\phi)$ and $\mathsf{F}\phi = \neg\mathsf{G}\neg\phi$ that do cause only a polynomial increase in size.

The satisfaction relation is inductively defined below where σ is an α-model of the form $\alpha \rightarrow \mathcal{P}(\mathrm{PROP})$ for some ordinal $\alpha > 0$ ($\beta < \alpha$):

- $\sigma, \beta \models p$ iff $p \in \sigma(\beta)$,
- $\sigma, \beta \models \neg\phi$ iff not $\sigma, \beta \models \phi$, $\sigma, \beta \models \phi_1 \wedge \phi_2$ iff $\sigma, \beta \models \phi_1$ and $\sigma, \beta \models \phi_2$,
- $\sigma, \beta \models \phi_1 \mathsf{U} \phi_2$ iff there is $\gamma \in (\beta, \alpha)$ such that $\sigma, \gamma \models \phi_2$ and for every $\gamma' \in (\beta, \gamma)$, we have $\sigma, \gamma' \models \phi_1$,
- $\sigma, \beta \models \phi_1 \mathsf{S} \phi_2$ iff there is $\gamma \in [0, \beta)$ such that $\sigma, \gamma \models \phi_2$ and for every $\gamma' \in (\gamma, \beta)$, we have $\sigma, \gamma' \models \phi_1$.

The satisfiability problem for LTL(U, S) consists in determining, given a formula ϕ, whether there is a model σ such that $\sigma, 0 \models \phi$.

We recall that well orders are particular cases of Dedekind complete linear orders. Indeed, a chain is Dedekind complete iff every non-empty bounded subset has a least upper bound. Kamp's theorem applies herein.

Theorem 1. *(I) [Kam68]* LTL(U, S) *over the class of ordinals is as expressive as the first-order logic over the class of structures* $\langle \alpha, < \rangle$ *where* α *is an ordinal. (II) [BS73] The satisfiability problem for* LTL(U, S) *over the class of ordinals is decidable.*

Hence, LTL(U, S) is a fundamental logic to be studied. Moreover, another key result is the PSPACE-completeness of LTL(U, S) restricted to ω-models [SC85].

We recall below a definability result that will be used in Sections 6 and 7. Ordinals strictly below ω^ω can be defined in LTL(U, S) with the truth constant $\top$ (no propositional variable).

Lemma 2. *Given an ordinal* $0 < \alpha = \omega^{k_1} a_{k_1} + \cdots \omega^{k_m} a_{k_m} < \omega^\omega$ *with* $k_1 > \ldots > k_m \geq 0$, $a_{k_1}, \ldots a_{k_m} > 0$, *there is a formula* def_α *in* LTL(U, S) *of linear size in* $\Sigma_i (k_i \times a_{k_i})$ *such that for any model* σ, *we have* $\sigma, 0 \models \mathrm{def}_\alpha$ *iff* σ *is of length* α.

3 Translation from Formulae to Simple Ordinal Automata

In Section 3.1, we introduce a new class of ordinal automata which we call simple ordinal automata. These automata are expressive equivalent to Büchi automata over ordinals [BS73]. However, the locations and the transition relations of these automata have additional structures. In Section 3.2, we provide a translation from LTL(U, S) into simple ordinal automata which assigns to every formula in LTL(U, S) an automaton that recognizes exactly its models. We borrow the automata-based approach for temporal logics from [VW94, KVW00].

3.1 Simple Ordinal Automata

Definition 3. *A simple ordinal automaton* $\mathcal{A}$ *is a structure* $\langle X, Q, \delta_{next}, \delta_{lim} \rangle$ *such that*

- X *is a finite set (the basis of* $\mathcal{A}$*),* $Q \subseteq \mathcal{P}(X)$ *(the set of locations),*
- $\delta_{next} \subseteq Q \times Q$ *is the next-step transition relation,*
- $\delta_{lim} \subseteq \mathcal{P}(X) \times Q$ *is the limit transition relation.*

$\mathcal{A}$ can be viewed as a finite directed graph whose set of nodes is structured. Given a simple ordinal automaton $\mathcal{A}$, an α-path (or simply a path) is a map $r : \alpha \rightarrow Q$ for some $\alpha > 0$ such that

- for every $\beta + 1 < \alpha$, $\langle r(\beta), r(\beta + 1)\rangle \in \delta_{next}$,
- for every limit ordinal $\beta < \alpha$, $\langle \text{always}(r, \beta), r(\beta)\rangle \in \delta_{lim}$ where

$$\text{always}(r, \beta) \stackrel{\text{def}}{=} \{a \in X : \exists\, \gamma < \beta \text{ such that } a \in \cap_{\gamma' \in (\gamma,\beta)}\, r(\gamma')\}.$$

The set always(r, β) contains exactly the elements of the basis that belong to every location from some $\gamma < \beta$ until β. We sometimes write always(r) instead of always(r, α) when α is a limit ordinal or always(r) instead of always$(r, \alpha - 1)$ when α is a successor ordinal and $\alpha - 1$ is a limit ordinal.

Given an α-path r, for $\beta, \beta' < \alpha$ we write

- $r_{\geq\beta}$ to denote the restriction of r to positions greater or equal to β,
- $r_{\leq\beta}$ to denote the restriction of r to positions less or equal to β,
- $r_{[\beta,\beta')}$ to denote the restriction of r to positions in $[\beta, \beta')$ (half-open interval).

A simple ordinal automaton with acceptance conditions is a structure of the form $\langle X, Q, I, F, \mathcal{F}, \delta_{next}, \delta_{lim}\rangle$ where

- $I \subseteq Q$ is the set of initial locations,
- $F \subseteq Q$ is the set of final locations for accepting runs whose length is some successor ordinal,
- $\mathcal{F} \subseteq \mathcal{P}(X)$ encodes the accepting condition for runs whose length is some limit ordinal.

Given a simple ordinal automaton with acceptance conditions, an accepting run is a path $r : \alpha \rightarrow Q$ such that $r(0) \in I$ and

- if α is a successor ordinal, then $r(\alpha - 1) \in F$,
- otherwise $\{a \in X : \exists\, \gamma < \alpha \text{ such that } a \in \cap_{\gamma' \in (\gamma,\alpha)}\, r(\gamma')\} \in \mathcal{F}$.

The nonemptiness problem for simple ordinal automata consists in checking whether $\mathcal{A}$ has an accepting run.

Our current definition for simple ordinal automata does not make them language acceptors since they have no alphabet. It is possible to add in the definition a finite alphabet Σ and to define the next-step transition relation as a subset of $Q \times \Sigma \times Q$. If we do so, our model of automata can recognize the same languages as the usual ordinal automata with Muller acceptance conditions in the limit transitions. The proof is not very difficult. Additionally, the current definition can be viewed as the case either when the alphabet is a singleton or when the read letter is encoded in the locations through the dedicated elements of the basis. This second reading will be in fact used implicitly in the sequel.

We also write $\mathcal{A}$ to denote either a simple ordinal automaton or its extension with acceptance conditions.

3.2 Translation from LTL(U,S) Formulae to Simple Ordinal Automata

As usual, a set Y is a maximally Boolean consistent subset of $sub(\phi)$ when the following conditions are satisfied: for every $\psi \in sub(\phi)$, $\neg\psi \in Y$ iff $\psi \not\in Y$ and for every $\psi_1 \wedge \psi_2 \in sub(\phi)$, $\psi_1 \wedge \psi_2 \in Y$ iff $\psi_1, \psi_2 \in Y$. Given a formula ϕ, the simple ordinal automaton $\mathcal{A}_\phi = \langle X, Q, I, F, \mathcal{F}, \delta_{next}, \delta_{lim} \rangle$ is defined as follows:

- $X = sub(\phi)$.
- Q is the set of maximally Boolean consistent subsets of $sub(\phi)$.
- I is the set of locations that contain ϕ and no since formulae.
- F is the set of locations with no elements of the form $\psi_1 \mathsf{U} \psi_2$.
- $\mathcal{F}$ is the set of sets Y such that $\{\psi_1, \neg\psi_2, \psi_1\mathsf{U}\psi_2\} \not\subseteq Y$, for every $\psi_1\mathsf{U}\psi_2 \in X$.
- For all $q, q' \in Q$, $\langle q, q' \rangle \in \delta_{next}$ iff the conditions below are satisfied:
  (next$_\mathsf{U}$) for $\psi_1\mathsf{U}\psi_2 \in sub(\phi)$, $\psi_1\mathsf{U}\psi_2 \in q$ iff either $\psi_2 \in q'$ or $\psi_1, \psi_1\mathsf{U}\psi_2 \in q'$,
  (next$_\mathsf{S}$) for $\psi_1\mathsf{S}\psi_2 \in sub(\phi)$, $\psi_1\mathsf{S}\psi_2 \in q'$ iff either $\psi_2 \in q$ or $\psi_1, \psi_1\mathsf{S}\psi_2 \in q$.
- For all $Y \subseteq X$ and $q \in Q$, $\langle Y, q \rangle \in \delta_{lim}$ iff the conditions below are satisfied:
  (lim$_\mathsf{U}$1) if $\psi_1, \neg\psi_2, \psi_1\mathsf{U}\psi_2 \in Y$, then either $\psi_2 \in q$ or $\psi_1, \psi_1\mathsf{U}\psi_2 \in q$,
  (lim$_\mathsf{U}$2) if $\psi_1, \psi_1\mathsf{U}\psi_2 \in q$ and $\psi_1 \in Y$, then $\psi_1\mathsf{U}\psi_2 \in Y$,
  (lim$_\mathsf{U}$3) if $\psi_1 \in Y$, $\psi_2 \in q$ and $\psi_1\mathsf{U}\psi_2$ is in the basis X, then $\psi_1\mathsf{U}\psi_2 \in Y$,
  (lim$_\mathsf{S}$) for every $\psi_1\mathsf{S}\psi_2 \in sub(\phi)$, $\psi_1\mathsf{S}\psi_2 \in q$ iff ($\psi_1 \in Y$ and $\psi_1\mathsf{S}\psi_2 \in Y$).

Even though the conditions above can easily be shown correct, at this stage it might sound mysterious how they have been made up. For some of them, their justification comes with the proof of Lemma 4.

Let σ be an α-model and ϕ be a formula in LTL(U,S). The Hintikka sequence for σ and ϕ is an α-sequence $H^{\sigma,\phi}$ defined as follows: for every $\beta < \alpha$,

$$H^{\sigma,\phi}(\beta) \stackrel{\text{def}}{=} \{\psi \in sub(\phi) \ : \ \sigma, \beta \models \psi\}.$$

Now we can state the correctness lemma.

Lemma 4

(I) *If $\sigma, 0 \models \phi$, then $H^{\sigma,\phi}$ is an accepting run of $\mathcal{A}_\phi$.*
(II) *If r is an accepting run of $\mathcal{A}_\phi$, then there is a model σ such that $\sigma, 0 \models \phi$ and r is the Hintikka sequence for σ and ϕ.*
(III) *ϕ is satisfiable iff $\mathcal{A}_\phi$ has an accepting run.*

4 Short Run Properties

Let $\mathcal{A}$ be a simple ordinal automaton and Y be a subset of its basis. Y is said to be present in $\mathcal{A}$ iff there is a limit transition of the form $\langle Y, q \rangle$ in $\mathcal{A}$. Given a set Y present in $\mathcal{A}$, its weight, noted weight(Y), is the maximal l such that $Y_1 \subset Y_2 \subset \cdots \subset Y_l$ is a sequence of present subsets in $\mathcal{A}$ and $Y_1 = Y$. Obviously, weight(Y) $\leq |X| + 1$.

Given a path $r : \alpha \rightarrow Q$ in $\mathcal{A}$ with $\alpha \geq \omega + 1$, its weight, noted weight(r), is the maximal value in the set $\{\text{weight}(\text{always}(r, \beta)) \ : \ \beta < \alpha, \ \beta \text{ is a limit ordinal}\}$.

By convention, if a path is of length strictly less than $\omega + 1$, then its weight is zero (no limit transition is fired). Furthermore, we write exists(r) to denote the set $\bigcup_{\beta<\alpha} r(\beta)$ and all(r) to denote the set $\bigcap_{\beta<\alpha} r(\beta)$. For example, all$(r)$ corresponds to the set of elements from the basis that are present in all locations of the run r. Let r, r' be two paths of respective length $\alpha+1$ and $\alpha'+1$, we say that they are congruent (noted $r \sim r'$) iff the conditions below are meet: $r(0) = r'(0)$, $r(\alpha) = r'(\alpha')$, exists(r) = exists(r') and all(r) = all(r'). We can easily adapt the congruence relation to runs r and r' of length some limit ordinal by requiring that always(r) = always(r') instead of the condition on final locations for runs of length some successor ordinal.

Let r_1 be a path of length $\alpha + 1$ and r_2 be a path of length β such that $r_1(\alpha) = r_2(0)$. The concatenation $r_1 \cdot r_2$ is the path r of length $\alpha + \beta$ such that for $\gamma \in [0, \alpha]$, $r(\gamma) = r_1(\gamma)$ and for $\gamma \in [0, \beta)$, $r(\alpha+\gamma) = r_2(\gamma)$. For every ordinal α, the concatenation of α-sequences of paths is defined similarly. The relation $\sim$ can be viewed as a congruence for the concatenation operation on paths.

Lemma 5

(I) *Let $r \cdot r_0 \cdot r'$, r_1 be two paths such that $r_0 \sim r_1$. Then, $r \cdot r_1 \cdot r'$ is a path that is congruent to $r \cdot r_0 \cdot r'$.*

(II) *Let $r_0^0, r_0^1, r_0^2, \ldots$ and $r_1^0, r_1^1, r_1^2, \ldots$ be two ω-sequences of pairwise consecutive paths such that for $i \geq 0$, $r_0^i \sim r_1^i$ and their length is a successor ordinal. If $r \cdot r_0^0 \cdot r_0^1 \cdot r_0^2 \cdot \ldots \cdot r'$ is a path, then it is congruent to $r \cdot (r_1^0 \cdot r_1^1 \cdot r_1^2 \cdot \ldots) \cdot r'$.*

The proof of the above lemma is by an easy verification.

Lemma 6. *Let $r : \alpha \rightarrow Q$ be a path in $\mathcal{A}$. Then, there is a path $r' : \alpha' \rightarrow Q$ such that $\alpha' < \omega^{\mathrm{weight}(r)+1}$ and $r \sim r'$.*

Lemma 6 states a crucial property for most of complexity results established in the sequel. Indeed, for usual ordinal automata, it is not possible to get this polynomial bound as an exponent of ω for the length of the short paths. Actually, the exponent is linear in the cardinal of its basis and can be logarithmic in the number of locations for large automata. By combination of Lemma 4 and Lemma 6, we obtain the following interesting result.

Corollary 7. *If ϕ is satisfiable, then ϕ has an α-model with $\alpha < \omega^{|\phi|+2}$.*

This can be still be refined a little more by observing that for each $\omega^i \times a$ occurring in the Cantor normal form of the length of a small model of ϕ (strictly less than $\omega^{|\phi|+2}$), a is bounded by $2^{|\phi|-1}$ since the cardinal of the set of locations of $\mathcal{A}_\phi$ is bounded by $2^{|\phi|-1}$.

For $n \in \mathbb{N}$, let trunc$_n$ be the function that assigns to every ordinal $\alpha > 0$ an ordinal in $(0, \omega^n 2)$ as follows. α can be written in the form $\alpha = \omega^n \gamma + \beta$ with $\beta \in [0, \omega^n)$. Then trunc$_n(\alpha) = \omega^n \times min(\gamma, 1) + \beta$.

Lemma 8. *Let $\mathcal{A}$ be a simple ordinal automaton.*

(I) *If r is a path of length $\omega^{\mathrm{weight}(r)+1} \times \alpha$ for some countable ordinal $\alpha > 0$, then there is a path r' of length $\omega^{\mathrm{weight}(r)+1}$ such that $r \sim r'$.*

(II) *If a path r has length $\omega^{\mathrm{weight}(r)+1}$, then for every ordinal $\alpha > 0$, there is a path r' of length $\omega^{\mathrm{weight}(r)+1} \times \alpha$ such that $r \sim r'$.*

(III) *If r is a path of length α and $\beta \approx_{|X|+1} \alpha$, then there is a path r' of length β such that $r \sim r'$.*

Only in (I), α is supposed to be countable. Because of the translation from formulae to automata, we can also establish a pumping lemma at the level of formulae.

Lemma 9

(I) *Let $\mathcal{A}$ be a simple ordinal automaton with acceptance conditions and α, β be ordinals such that $\alpha \approx_{|X|+1} \beta$. Then, $\mathcal{A}$ has an accepting run of length α iff $\mathcal{A}$ has an accepting run of length β.*

(II) *Let ϕ be a formula in* $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ *and α, β be ordinals such that $\alpha \approx_{|\phi|+2} \beta$. Then ϕ has an α-model iff ϕ has a β-model.*

Proof. (I) Direct consequence of Lemma 6 and Lemma 8 since accepting runs can be viewed as paths.

(II) By Lemma 4, ϕ has an α-model iff $\mathcal{A}_\phi$ has an accepting run r of length α. Since $|\phi| + 1$ bounds the weight of any path in $\mathcal{A}_\phi$ and by (I), we get that $\mathcal{A}_\phi$ has an accepting run r of length α iff $\mathcal{A}_\phi$ has an accepting run r of length β. Equivalently, ϕ has a β-model. □

5 Checking Nonemptiness of Simple Ordinal Automata

In this section, we provide algorithms to check whether a simple ordinal automata admits accepting runs. The first one is in EXPTIME. Our optimal algorithm runs in polynomial space in the size of the basis.

Let $\mathcal{A}$ be a simple ordinal automaton $\langle X, Q, I, F, \mathcal{F}, \delta_{next}, \delta_{lim} \rangle$. We provide below an algorithm to check given $q, q' \in Q$ and $n \in \mathbb{N}$ whether there is path $r : \alpha + 1 \rightarrow Q$ such that $r(0) = q$, $r(\alpha) = q'$ and $\alpha < \omega^n$. Given an $(\alpha + 1)$-path we write $\mathrm{abs}(r)$ to denote the quadruple $\langle r(0), \mathrm{exists}(r), \mathrm{all}(r), r(\alpha) \rangle$. We define a family of relations containing the quadruples of the form $\mathrm{abs}(r)$. Each relation R_i below is therefore a subset of $R_i \subseteq Q \times \mathcal{P}(X)^2 \times Q$.

- $R_0 = \{\langle q, q \cup q', q \cap q', q' \rangle : \langle q, q' \rangle \in \delta_{next}\}$,
- For $i \in \mathbb{N}$,

$$R'_i = \{\langle q_0, \bigcup_{i=0}^{m} E_i, \bigcap_{i=0}^{m} A_i, q_{m+1} \rangle :$$

$$\exists \langle q_0, E_0, A_0, q_1 \rangle R_i \langle q_1, E_1, A_1, q_2 \rangle R_i \cdots R_i \langle q_m, E_m, A_m, q_{m+1} \rangle\}$$

- For $i \in \mathbb{N}$, R_{i+1} is defined from R'_i as follows: $\langle q, E, A, q' \rangle \in R_{i+1}$ iff
  - either $\langle q, E, A, q' \rangle \in R'_i$
  - or there exist a limit transition $\langle Y, q' \rangle \in \delta_{lim}$ and a path

$$\langle q_0, E_0, A_0, q_1 \rangle R_i \langle q_1, E_1, A_1, q_2 \rangle R_i \cdots R_i \langle q_m, E_m, A_m, q_{m+1} \rangle$$

such that
(a) $q_0 = q_{m+1}$, (b) $\bigcap_{i=0}^{m} A_i = Y$, (c) $\langle q, E', A', q_0 \rangle \in R'_i$ for some E', A',
(d) $E = (E' \cup q') \cup \bigcup_{i=0}^{m} E_i$, $A = (A' \cap q') \cap \bigcap_{i=0}^{m} A_i$.

Because $R_i \subseteq R_{i+1}$ for all i, for some $N \leq 2^{4\times|X|} + 1$, $R_{N+1} = R_N$. The bound $2^{4\times|X|} + 1$ takes simply into account that $Q \subseteq \mathcal{P}(X)$.

Lemma 10. *(I) If* $\langle q, E, A, q' \rangle \in R_n$, *then there is an* $(\alpha + 1)$*-path such that* $\mathrm{abs}(r) = \langle q, E, A, q' \rangle$ *and* $\alpha < \omega^n$. *(II) Conversely, let* $r : \alpha + 1 \rightarrow Q$ *be a path such that* $\alpha < \omega^n$. *Then* $\mathrm{abs}(r) \in R'_n$.

We provide below a first complexity result.

Lemma 11. *The nonemptiness problem for simple ordinal automata with acceptance conditions can be checked in exponential time in* $|X|$.

Proof. Let $\mathcal{A}$ be of the form $\langle X, Q, I, F, \mathcal{F}, \delta_{next}, \delta_{lim} \rangle$. $\mathcal{A}$ has an accepting run iff either (A) there are $q_0 \in I$, $q_f \in F$ and $E, A \subseteq X$ such that $\langle q_0, E, A, q_f \rangle \in R'_n$ for some n or (B) there are $q_0 \in I$, and a run r from q_0 such that always$(r) \in \mathcal{F}$. (A) deals with accepting runs of length some successor ordinal, whereas (B) deals with accepting runs of length some limit ordinal.

By Lemma 6 and Lemma 10(II), in order to check (A), it is sufficient to test for $\langle q_0, E, A, q_f \rangle \in I \times \mathcal{P}(X)^2 \times F$ whether $\langle q_0, E, A, q_f \rangle \in R'_{|X|+2} \subseteq R_{|X|+3}$. Since $|Q|$ is in $\mathcal{O}(2^{|X|})$, computing $R_{|X|+3}$ takes $|X|+3$ steps that requires polynomial time in $|\mathcal{A}|$ and exponential time in $|X|$, we obtain the desired result. Observe that we can take advantage of the fact that computing the transitive closure of a relation and the maximal strongly connected components can be done in polynomial time in the size of the relations.

By Ramsey theorem, (B) is equivalent to the following condition: there are $q \in Q$, $E, E', A \subseteq X$, $A' \in \mathcal{F}$ and runs r_1 and r_2 such that $\mathrm{abs}(r_1) = \langle q_0, E, A, q \rangle$ and $\mathrm{abs}(r_2) = \langle q, E', A', q \rangle$.

Hence. in order to check these, it is enough to check whether there are $q_0 \in I$, $q \in Q$ and $E, A \subseteq X$ such that $\langle q_0, E, A, q \rangle \in R'_{|X|+2}$, $\langle q, E', A', q \rangle \in R'_{|X|+2}$ and $A' \in \mathcal{F}$. This can be done in exponential time as for (A). □

The proof of Lemma 11 mentions Lemma 6 but the exponential time upper bound can be obtained by observing that an exponential number of steps, such as $2^{4\times|X|} + 1$ would provide the same bound in the worst case. As a corollary of Lemma 11, satisfiability for LTL(U, S) is in EXPTIME. Moreover, this can be improved as shown in the proof of Theorem 13 presented in Section 6.

We improve below the bound in Lemma 11.

Theorem 12. *The nonemptiness problem for simple ordinal automata can be checked in polynomial space in* $|X|$.

Proof. Following the proof of Lemma 11, $\mathcal{A}$ has an accepting run iff (A) there are $q_0 \in I$, $q_f \in F$ and $E, A \subseteq X$ such that $\langle q_0, E, A, q_f \rangle \in R_{|X|+3}$ or (B) there are $q_0 \in I$, $q \in Q$ and $E', A' \subseteq X$ such that $\langle q_0, E', A', q \rangle \in R_{|X|+3}$, $\langle q, E', A', q \rangle \in$

$R_{|X|+3}$ and $A' \in \mathcal{F}$. X denotes the basis of $\mathcal{A}$. In order to check (A), the non-deterministic algorithm guesses $q_0 \in I$, $q_f \in F$ and $E, A \subseteq X$ (encoded in polynomial space in $\mathcal{O}(|X|)$ and test whether $\mathrm{PATH}(\mathcal{A}, \langle q_0, E, A, q_f \rangle, |X| + 3)$ returns true. Condition (B) admits a similar treatment. The non-deterministic algorithm PATH defined below works in polynomial space in $|X|$ assuming that the last argument is polynomial in $|X|$ which is the case with $|X| + 3$. Figure 1 contains the definition of the function PATH (some details are omitted).

In (2.), guessing on-the-fly a long sequence means that only two consecutive quadruples are kept in memory at any time. We need a counter to guarantee that $m < 2^{4\times|X|+1}$ and it requires only space in $\mathcal{O}(|X|)$. Moreover, in order to check $E = \bigcup_j E_j$ and $A = \bigcap_j A_j$ we need two auxiliary variables that bookkeep the E_j and A_j so far respectively. Similar techniques are used in (3.) to guarantee that this non-deterministic algorithm requires only polynomial space in $\mathcal{O}(|X| + N)$ (we only need more variables and steps). It is straightforward to show that

```
PATH($\mathcal{A}, \langle q, E, A, q' \rangle, N$)

– If $N = 0$ then (if (either $E \neq q \cup q'$ or $A \neq q \cap q'$ or $\langle q, q' \rangle \notin \delta_{next}$) then abort else
  return $\top$);
– If $N > 0$ then go non-deterministically to 1.,2. or 3.
  (1.) Return PATH($\mathcal{A}, \langle q, E, A, q' \rangle, N - 1$)
  (2.) Guess on-the-fly a sequence

           $\langle q_0, E_0, A_0, q_1 \rangle, \langle q_1, E_1, A_1, q_2 \rangle, \ldots, \langle q_m, E_m, A_m, q_{m+1} \rangle$

       such that
         • $m < 2^{4\times|X|+1} + 1$,
         • for $0 \leq i \leq m$, PATH($\mathcal{A}, \langle q_i, E_i, A_i, q_{i+1} \rangle, N - 1$) returns $\top$,
         • $E = \bigcup_j E_j$ and $A = \bigcap_j A_j$
         • $q = q_0$, $q' = q_{m+1}$;
  (3.) We guess here two long sequences:
       (3.1) Guess on-the-fly a sequence

               $\langle q_0, E_0, A_0, q_1 \rangle, \langle q_1, E_1, A_1, q_2 \rangle, \ldots, \langle q_m, E_m, A_m, q_{m+1} \rangle$

             such that
               • $m < 2^{4\times|X|+1} + 1$,
               • for $0 \leq i \leq m$, PATH($\mathcal{A}, \langle q_i, E_i, A_i, q_{i+1} \rangle, N - 1$) returns $\top$,
               • $E' = \bigcup_j E_j$ and $A' = \bigcap_j A_j$;
               • $q_0 = q$;
       (3.2) Guess a limit transition $\langle Y, q' \rangle \in \delta_{lim}$ and on-the-fly a sequence
             $\langle q'_0, E'_0, A'_0, q'_1 \rangle, \langle q'_1, E'_1, A'_1, q'_2 \rangle, \ldots, \langle q'_m, E'_{m'}, A'_{m'}, q'_{m'+1} \rangle$ such that
               • $m' < 2^{4\times|X|+1}$,
               • for $0 \leq i \leq m'$, PATH($\mathcal{A}, \langle q'_i, E'_i, A'_i, q'_{i+1} \rangle, N - 1$) returns $\top$,
               • $E = (E' \cup q'_{m'+1}) \cup \bigcup_j E'_j$,
               • $A = (A' \cap q'_{m'+1}) \cap \bigcap_j A'_j$, $Y = \bigcap_j A'_j$, $q'_0 = q_{m+1}$;
– Return $\top$.
```

Fig. 1. Algorithm PATH

$\mathrm{PATH}(\mathcal{A}, \langle q, E, A, q' \rangle, N)$ has a computation that returns $\top$ (all the guesses were correct) iff $\langle q, E, A, q' \rangle \in R_N$. Finally Savitch's Theorem allows to conclude that nonemptiness can be checked in deterministic polynomial space in $|X|$. $\Box$

6 Complexity of Satisfiability Problems

We establish new complexity results for problems related to $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ satisfiability thanks to the intermediate results we have established so far.

6.1 Complexity of $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$

Here is the main result of the paper.

Theorem 13. *The satisfiability problem for* $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ *over the class of ordinals is* PSPACE*-complete.*

Proof. By Lemma 4, given a formula ϕ in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$, there is an automaton $\mathcal{A}_\phi$ whose accepting runs correspond exactly to models of ϕ. In order to check nonemptiness of $\mathcal{A}_\phi$, we do not build it explicitly (as usual) but we run the algorithm from the proof of Theorem 12 and we compute the locations, and transition relations of $\mathcal{A}_\phi$ on demand. Hence, we obtain a polynomial space non-deterministic algorithm since the basis of $\mathcal{A}_\phi$ has a cardinality in $\mathcal{O}(|\phi|)$ and checking whether a subset of X is a location of $\mathcal{A}_\phi$ or $\langle q, q' \rangle \in \delta_{next}$ or $\langle Y, q \rangle \in \delta_{lim}$ can be done in polynomial space in $\mathcal{O}(|\phi|)$. Again by Savitch's Theorem, we get that the satisfiability problem for $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ is in PSPACE. The PSPACE lower bound can be easily shown inherited from LTL. $\Box$

Thanks to Kamp's theorem, we get the following corollary.

Corollary 14. *Let* $\mathrm{LTL}(\mathsf{U}, \mathsf{S}, \mathsf{O}_1, \ldots, \mathsf{O}_k)$ *be an extension of* $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ *with* k *first-order definable temporal operators. Then the satisfiability problem for the logic* $\mathrm{LTL}(\mathsf{U}, \mathsf{S}, \mathsf{O}_1, \ldots, \mathsf{O}_k)$ *over the class of ordinals is in* PSPACE.

Indeed, every formula $\mathsf{O}_i(p_1, \ldots, p_{n_i})$ encoded as a DAG can be translated into an equivalent formula in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ encoded as a DAG over the propositional variables $p_1, \ldots, p_{n_i}$. Since $\mathsf{O}_1, \ldots, \mathsf{O}_k$ and their definition in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ are constant of $\mathrm{LTL}(\mathsf{U}, \mathsf{S}, \mathsf{O}_1, \ldots, \mathsf{O}_k)$ we obtain a translation in polynomial-time (with our definition for the size of formulae).

6.2 A Family of Satisfiability Problems

The satisfiability problem for $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ asks for the existence of a model for a given formula. A natural variant of this problem consists in fixing the length of the models in advance as for LTL. The satisfiability problem for $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ over α-models, noted $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$, is defined as follows: given a formula ϕ in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$, is ϕ satisfiable over an α-model? In this subsection we prove that $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$ is in PSPACE for every countable ordinal α. First we consider the case of ordinals strictly less than ω^ω. Recall that for every $\alpha < \omega^\omega$ there is a formula def_α in $\mathrm{LTL}(\mathsf{U}, \mathsf{S})$ such that for every β-model σ, we have $\sigma, 0 \models \mathrm{def}_\alpha$ iff $\beta = \alpha$.

Corollary 15. *For every* $\alpha < \omega^\omega$*, the problem* $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$ *is in* PSPACE.

Proof. ϕ has a α-model iff $\psi = \phi \wedge \mathrm{def}_\alpha$ is satisfiable over the class of ordinals. Thanks to Lemma 2 and Theorem 13, we obtain the PSPACE upper bound. □

Now we consider the case of a countable ordinal $\alpha \geq \omega^\omega$. Let α' be the unique ordinal strictly less than ω^ω such that $\alpha = \omega^\omega \times \gamma + \alpha'$ for some ordinal γ. Note that for every k, $\mathrm{trunc}_k(\alpha) = \mathrm{trunc}_k(\omega^k + \alpha') < \omega^\omega$. By Lemma 9, ϕ has an α-model iff ϕ has a $\alpha_{|\phi|}$-model with $\alpha_{|\phi|} = \mathrm{trunc}_{|\phi|+2}(\alpha) = \mathrm{trunc}_{|\phi|+2}(\omega^{|\phi|+2} + \alpha')$. Hence, ϕ has an α-model iff $\phi \wedge \mathrm{def}_{\alpha_{|\phi|}}$ is satisfiable (over the class of countable ordinals). Since the size of $\mathrm{def}_{\alpha_{|\phi|}}$ is polynomial in the size of ϕ, we derive from Theorem 13 the following result.

Corollary 16. *For every countable* $\alpha \geq \omega^\omega$*, the problem* $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$ *is in* PSPACE.

Corollaries 15, 16 and the arguments similar to the arguments in the proof of Corollary 14 imply the result below.

Theorem 17. *For every finite set* $\{\mathsf{O}_1, \ldots, \mathsf{O}_k\}$ *of first-order definable temporal operators and every countable ordinal* α*, the satisfiability problem for the logic* $\mathrm{LTL}(\mathsf{O}_1, \ldots, \mathsf{O}_k)$ *restricted to* α*-models is in* PSPACE.

Observe that (1) if α is finite, then $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{O}_1, \ldots, \mathsf{O}_k))$ is NP-complete whereas (2) if α is infinite, then PSPACE-hardness for $\mathrm{SAT}(\alpha, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$ follows from the PSPACE-completeness of $\mathrm{SAT}(\omega, \mathrm{LTL}(\mathsf{U}, \mathsf{S}))$.

6.3 Uniform Satisfiability

Büchi (see, e.g., [BS73]) has shown that there is a *finite* amount of data concerning any countable ordinal that determines its monadic theory.

Definition 18 (Code of an ordinal). *Let* α *be a countable ordinal and let* m *be in* $[1, \omega]$.

1. *Write* $\alpha = \omega^m \alpha' + \zeta$ *with* $\zeta < \omega^m$ *(this can be done in a unique way), and let*
$$p_m(\alpha) := \begin{cases} -2 \text{ if } \alpha' = 0 \\ -1 \text{ if } 0 < \alpha' < \omega_1 \end{cases}.$$
2. *If* $\zeta \neq 0$*, write* $\zeta = \sum_{i \leq n} \omega^{n-i} \cdot a_{n-i}$ *where* $a_i \in \omega$ *for* $i \leq n$ *and* $a_n \neq 0$ *(this can be done in a unique way), and let* $t_m(\alpha) := (a_n, \ldots, a_0)$*. If* $\zeta = 0$*, let* $t(\alpha) = -3$.
3. *The* m-code *of* α *is the pair* $(p_m(\alpha), t_m(\alpha))$.

The following is implicit in [BS73].

Theorem 19 (Code Theorem). *There is an algorithm that, given a monadic second-order sentence* ϕ *and the* ω-code *of a countable ordinal* α*, determines whether* $\langle \alpha, < \rangle \models \phi$.

Lemma 9 can be rephrased as "the $(|\phi| + 2)$-code of an ordinal α determines whether ϕ has a model of length α".

Let $C = (b, a_n, \ldots a_0)$ be an m-code. Its size is defined as $n+a_0+a_1+\cdots+a_n$. It is clear that for $m_1 < m_2$ the m_2-code of an ordinal determines its m_1-code and there is a linear time algorithm, that given m_2-code of an ordinal and $m_1 < m_2$ computes the m_1-code of the ordinal.

The arguments used in the proof of Corollary 16 show the following theorem.

Theorem 20 (Uniform Satisfiability)

(I) *There is a polynomial-space algorithm that, given an* LTL(U, S) *formula* ϕ *and the* ω-code *of a countable ordinal* α*, determines whether* ϕ *has an* α*-model.*

(II) *There is a polynomial-space algorithm that, given an* LTL(U, S) *formula* ϕ *and the* $(|\phi|+2)$*-code of a countable ordinal* α*, determines whether* ϕ *has an* α*-model.*

7 Related Work

In this section, we compare our results with those from the literature. Because of lack of place, we omit to cite works in which models of length higher than ω are considered for formal verification of computer systems, see e.g. [GW94].

7.1 Comparison with Rohde's Thesis

In [Roh97], it is shown that an uniform satisfiability problem for temporal logic with until (and without since) can be solved in exponential-time. The inputs of this problem are a formula in LTL(U) and the representation of an ordinal. The satisfiability problem is also shown in EXPTIME. In order to obtain this upper bound, formulae are shown equivalent to alternating automata and a reduction from alternating automata into a specific subclass of non-deterministic automata is given. Finally, a procedure for testing nonemptiness is provided. Here are the similarities between [Roh97] and our results.

1. We follow an automata-based approach and the class of non-deterministic automata in [Roh97] and ours have a structured set of locations and limit transitions use elements that are true from some position.
2. Existence of α-paths in the automata depends on some truncation of α.
3. The logical decision problems can be solved in exponential-time.

However, our work improves considerably some results from [Roh97].

1. Our temporal logic includes the until and since operators (instead of until only) and it is therefore as expressive as first-order logic.
2. We establish a tight PSPACE upper bound (instead of EXPTIME) thanks to the introduction of a class of simple ordinal automata.
3. Our proofs are much shorter and transparent (instead of the lengthy developments from [Roh97]).

Consequently, the developments from [Roh97] and ours follow the same approach with different definitions for automata, different intermediate lemmas and distinct final complexity bounds. On the other hand, the structure of the whole proof to obtain the main complexity bounds is similar.

7.2 Comparison with Reynolds' Results

Even though the results for linear-time temporal logics from [Rey03, Rey] involve distinct models, our automata-based approach has similarities with these works that uses a different proof method, namely mosaics. Indeed, equivalence classes of the relation $\sim$ between runs of length a successor ordinal roughly correspond to mosaics from [Rey03]. We recall the main results below.

Theorem 21. *(I) The satisfiability problem for the temporal logic with until and since over the reals is* PSPACE*-complete. (II) The satisfiability problem for* LTL(U) *over the class of all linear orders is* PSPACE*-complete.*

The proofs in [Rey03, Rey] are much more involved than our proofs since the orders are more complex than the class of ordinals. Unfortunately, we do not understand these proofs fully and find it difficult to compare to our proof.

7.3 Quantitative Temporal Operators

In this section, we show that the main results from [DN07] are subsumed by the current paper. We also solve an open problem from [Cac06, DN07]. For every fixed countable ordinal $\alpha \leq \omega$, let us introduce the logic LTL($\mathcal{O}_\alpha$) where the set of temporal operators $\mathcal{O}_\alpha$ is defined as follows: $\{\mathsf{X}^\beta : \beta < \omega^\alpha\} \cup \{\mathsf{U}^\beta : \beta \leq \omega^\alpha\}$. The models of LTL($\mathcal{O}_\alpha$) as those of LTL($\mathsf{U}, \mathsf{S}$) and the formulae of LTL($\mathcal{O}_\alpha$) are precisely defined by: $\phi ::= p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathsf{X}^\beta\phi \mid \phi_1\mathsf{U}^{\beta'}\phi_2$. The size of a formula ϕ is the number of subformulae occurring in ϕ plus the sum of all the natural numbers occurring in ϕ either as a coefficient or as an exponent of ω. The satisfaction relation is inductively defined below where σ is a model for LTL($\mathcal{O}_\alpha$) (we omit the obvious clauses):

- $\sigma, \beta \models \mathsf{X}^{\beta'}\phi$ iff $\beta + \beta'$ is a position of σ and $\sigma, \beta + \beta' \models \phi$,
- $\sigma, \beta \models \phi_1\mathsf{U}^{\beta'}\phi_2$ iff there is $0 < \gamma < \beta'$ such that $\beta + \gamma$ is a position of σ, $\sigma, \beta + \gamma \models \phi_2$ and for every $0 < \gamma' < \gamma$, we have $\sigma, \beta + \gamma' \models \phi_1$.

The satisfiability problem for LTL($\mathcal{O}_\alpha$) consists in determining, given a formula ϕ, whether there is a model σ such that $\sigma, 0 \models \phi$. The main results of [Cac06, DN07] are the following: for every $k \geq 1$, the satisfiability problem for LTL($\mathcal{O}_k$) restricted to models of length ω^k is PSPACE-complete and LTL($\mathcal{O}_\omega$) restricted to models of length ω^ω is decidable.

Observe that LTL($\mathcal{O}_k$) cannot express the temporal operator U over the class of countable ordinals but it can do it on models of length ω^k. Hence, each logic LTL($\mathcal{O}_k$) is less expressive than LTL(U, S).

Moreover, it is easy to show that for every $\alpha \leq \omega$, the logic LTL($\mathcal{O}_\alpha$) is expressive equivalent (over the class of countable ordinals) to its sublogic over the following set $\mathcal{O}'_\alpha$ of temporal operators:

$$\mathcal{O}'_\alpha = \{\mathsf{X}^{\omega^i} : \omega^i < \omega^\alpha, i \in \mathbb{N}\} \cup \{\mathsf{U}^{\omega^\beta} : \omega^\beta \leq \omega^\alpha, \beta \leq \omega\}.$$

This set is finite when α is finite. Moreover, there is a linear time (and logarithmic space) meaning preserving translation from LTL($\mathcal{O}_\alpha$) into LTL($\mathcal{O}'_\alpha$).

We obtain alternative proofs for known results and we get new results.

Theorem 22. *For every $k \geq 1$,*

- **(I)** *the satisfiability problem for* LTL($\mathcal{O}_k$) *over ω^k-models is in* PSPACE,
- **(II)** *the satisfiability problem for* LTL($\mathcal{O}'_k$) *restricted to ω^k-models is* PSPACE-*complete,*
- **(III)** *for every countable infinite ordinal α, the satisfiability problem for* LTL($\mathcal{O}'_k$) *restricted to α-models is* PSPACE-*complete.*

(III) is an instance of Theorem 17. (II) is an instance of (III). (I) can be shown by observing that there is logarithmic space meaning preserving translation from LTL($\mathcal{O}_k$) to LTL($\mathcal{O}'_k$). (I) is the main result of [DN07] with the unary encoding of natural numbers occurring in ordinal expressions.

Finally, the corollary below improves the non-elementary bounds obtained in [Cac06, DN07] for LTL($\mathcal{O}_\omega$) by reducing this temporal logic to the monadic second-order logic, and then to Büchi ordinal automata.

Corollary 23. *Satisfiability for* LTL($\mathcal{O}_\omega$) *over the class of ω^ω-models is* PSPACE-*complete.*

8 Conclusion

In the paper, we have shown that the linear-time temporal logic with until and since over the class of ordinals, namely LTL(U, S) has a PSPACE-complete satisfiability problem. Thanks to Kamp's theorem [Kam68], we know that LTL(U, S) is a fundamental temporal logic since it is as expressive as first-order logic over the class of ordinals. In order to establish this tight complexity characterization, we have introduced the class of simple ordinal automata. This class of automata is more structured than usual ordinal automata and the sets of locations have some structural properties, typically it is a subset of the powerset of some set (herein called the basis). As a consequence, we are also able to improve some results from [Roh97, DN07]. For instance the uniform satisfiability problem is PSPACE-complete and we obtain alternative proofs for results in [DN07].

Extensions of our results include that the satisfiability problem for the language LTL($\mathsf{O}_1, \ldots, \mathsf{O}_k$) where the O_is form a finite set of MSO definable temporal operators is in PSPACE by adapting the developments from [VW94] and showing that our simple ordinal automata augmented with alphabet has the expressive power of standard ordinal automata. Furthermore, our results can be extended to scattered linear orderings, see e.g. [BC07]. Indeed, one should add right limit transitions, using the terminology from [BC07] and adapt the developments herein.

Acknowledgments. We would like to thank the anonymous referees for helpful suggestions and remarks.

References

[BC07] Bruyère, V., Carton, O.: Automata on linear orderings. Journal of Computer and System Sciences 73, 1–24 (2007)

[BS73] Büchi, J.R., Siefkes, D.: *The monadic second order theory of all countable ordinals*. Lecture Notes in Mathematics, vol. 328. Springer, Heidelberg (1973)

[Cac06] Cachat, T.: Controller synthesis and ordinal automata. In: Graf, S., Zhang, W. (eds.) ATVA 2006. LNCS, vol. 4218, pp. 215–228. Springer, Heidelberg (2006)

[DN07] Demri, S., Nowak, D.: Reasoning about transfinite sequences. International Journal of Foundations of Computer Science 18(1), 87–112 (2007)

[GHR94] Gabbay, D., Hodkinson, I., Reynolds, M.: Temporal Logic - Mathematical Foundations and Computational Aspects, vol. 1. OUP, Oxford (1994)

[GW94] Godefroid, P., Wolper, P.: A partial approach to model checking. I & C 110(2), 305–326 (1994)

[Kam68] Kamp, J.: Tense Logic and the theory of linear order. PhD thesis, UCLA, USA (1968)

[KVW00] Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. J. ACM 47(2), 312–360 (2000)

[Rey] Reynolds, M.: The complexity of the temporal logic over the reals. Under submission

[Rey03] Reynolds, M.: The complexity of the temporal logic with until over general linear time. Journal of Computer and System Sciences 66(2), 393–426 (2003)

[Roh97] Rohde, S.: Alternating Automata and The Temporal Logic of Ordinals. PhD thesis, University of Illinois (1997)

[SC85] Sistla, A., Clarke, E.: The complexity of propositional linear temporal logic. J. ACM 32(3), 733–749 (1985)

[Ven93] Venema, Y.: Completeness via completeness: Since and until. In: de Rijke, M. (ed.) Diamonds and Defaults, pp. 279–286. Kluwer Academic Publishers, Dordrecht (1993)

[VW94] Vardi, M., Wolper, P.: Reasoning about infinite computations. I & C 115, 1–37 (1994)

ATP Cross-Verification of the Mizar MPTP Challenge Problems

Josef Urban[1,⋆] and Geoff Sutcliffe[2]

[1] Dep't of Theoretical Computer Science, Charles University in Prague
[2] Dep't of Computer Science, University of Miami

Abstract. Mizar is a proof assistant used for formalization and mechanical verification of mathematics. The main use of Mizar is in the development of the Mizar Mathematical Library (MML), in which proofs are verified by the Mizar proof checker. The Mizar proof checker has a quite complex implementation, and also lacks the ability to print out detailed atomic proof steps in a format that is easy to verify by an independent proof-checking tool. This can raise concerns about the correctness of the MML. This paper describes how a Mizar-to-ATP translation (the MPTP system), ATP verification tools (the GDV system), and Automated Theorem Proving (ATP) systems, have been used for an independent cross-verification of a part of the MML.

1 Introduction, Motivation, and Related Work

Mizar [Rud92, RT99] is a proof checker based on first-order classical logic and set theory. The main use of Mizar is in the development of the Mizar Mathematical Library (MML)[1], a large library of formally verified mathematics, allowing the formalization of more and more advanced mathematical theories.

1.1 Motivation: Verification of Mizar Proofs

Starting in the 1990s with John Harrison's Mizar Mode for HOL [Har96], various versions of the Mizar declarative proof formalism have been developed [Zam99, Sym99, Wen99]. However, so far, there has been no independent verification of Mizar proofs. This (among other effects) prevents trusted semantic communication between Mizar and other systems. Such inter-system communication is useful for very large formalization efforts, e.g., the verification of the proof of Kepler Conjecture [Hal04]. In this context, communication between LCF-based systems has recently become popular [OS06, McL06].

While Mizar's reasoning formalism is quite simple and intuitive, it is mainly Mizar's implementation of its "atomic inference step"[2] that makes Mizar emulation complicated. Although the basic idea motivated by the notion of obviousness [Rud87, Dav81] is relatively easy to explain [Wie00], and has even been

---

[⋆] Supported by a Marie Curie International Fellowship within the 6th European Community Framework Programme.

[1] `http://mizar.uwb.edu.pl/library/`

[2] Denoted with the keyword **by** in Mizar proofs - see Section 2.2.

N. Dershowitz and A. Voronkov (Eds.): LPAR 2007, LNAI 4790, pp. 546–560, 2007.
© Springer-Verlag Berlin Heidelberg 2007

re-implemented in Harrison's Mizar Mode for HOL, there have been a number of additions in Mizar (e.g., interaction with Mizar's type system, built-in computer algebra [NB02, NB04], etc.) that make even a Mizar implementation of a detailed documentation mode for this procedure quite expensive. On the other hand, Mizar's classical first-order foundations are very close to the formalism used in today's efficient Automated Theorem Proving (ATP) systems. Translation of the MML to ATP formats (like TPTP [SS98]) is the goal of the MPTP (Mizar Problems for Theorem Proving) project. First experiments [Urb06a, Urb04] indicated that current ATP systems are able to verify an overwhelming majority of Mizar's atomic inference steps – above 99% of 18,429 problems generated from 48 initial Mizar articles [Urb06a] – without any low-level guidance. Given this capability for verification of atomic inference steps, the next logical step (described in this paper) is to develop techniques extending this to ATP-based verification of whole (sometimes quite long) Mizar proofs, with the potential target of ATP-based verification of the whole MML.

This paper describes how the Mizar-to-ATP translation has been enhanced (Section 2.2) so that the translated Mizar atomic inference steps of a Mizar proof are linked together to form a TPTP format derivation that reflects the the higher-level (natural deduction) Mizar reasoning structure. The derivation can then be verified using tools for verification of ATP systems' derivations, to provide verification of the Mizar proof. As necessary byproducts, the TPTP language has been extended for encoding derivations containing *assumptions* (Section 2.1), correctness conditions for such derivations have been designed and implemented in the GDV verification tool (Section 3), and an explanation of the Mizar natural deduction constructs has been provided in terms of this simpler formalism.

1.2 Related Verification Work

A list of previous mutual proof assistant verifications would probably be quite long (the translations between HOL Light and Isabelle are cited above). However, they are usually done on systems with low-level "proof object" capabilities, or systems in which addition of such capability is relatively easy, thanks to their "minimal logical kernel" design. The Otter ATP system has the "detailed proof object" option, and the generated Otter proof objects can be verified by the IVY system [MS00]. A recent notable use of ATP systems for certification of safety properties of NASA software is described in [DFS06].

1.3 Target: Verification of the MPTP Challenges

The methods developed for the ATP verification of Mizar theorems have one immediate application and nontrivial target: The set of ATP problems serving as the MPTP Challenges.[3] This is a set of 252 related mathematical problems translated from the Mizar library by the MPTP system, to create a benchmark for measuring the capability of ATP systems to work in large theories. Some of these problems have quite long Mizar proofs, making their direct solution by ATP systems quite difficult. As of May 2007, only 199 of the 252 problems (79%) have been solved by any ATP system,[4] leaving open questions about the provability of the remaining 53 problems. Since errors have been encountered before, both in the Mizar verifier and in the MPTP translation, an independent verification of the problems would be useful. This has been carried out and is discussed in Section 4.

---

[3] `http://www.cs.miami.edu/~tptp/MPTPChallenge/`

2 Translation of Mizar Proofs to TPTP Derivations

2.1 Extending the TPTP Format for Assumptions

Assumptions (*suppositions* in the original Jaskowski's [Jas34] terminology) are unproved propositions introduced (assumed to be true) locally at some part of a proof, coupled with an elimination (discharge) rule for making the results of such subproofs unconditionally valid. A common implicit use of these introduction/discharge rules in refutational ATP systems is the introduction of the negated conjecture, the derivation of a contradiction, and the discharge of the negated conjecture at that point, i.e., the argument that "`~conjecture |- $false`" is equivalent to "`|- ~conjecture => $false`", and hence to "`|- conjecture`". Other examples from the ATP world include explicit (SPASS-like) splitting [WBH+02], and the formalisms used in tableaux systems.

In order to record information about the use of assumptions in derivations, a new formulae role `assumption` has been added to the TPTP language, and a semantic grammar rule has been added for the source information recorded with every inferred formula. The semantic grammar rule specifies how an arbitrary list (treated as set by the GDV checks, see Section 3) of assumptions can be recorded with a formula, to keep track of the assumptions upon which the formula depends. The existing semantic grammar rules were already sufficient for recording the discharge of an arbitrary list of assumptions. These semantic grammar rules make it possible to record and verify that a derivation is independent of any assumptions introduced in it. These changes are part of the TPTP language specification in TPTP v3.3.0. An example of an assumption introduced into a derivation, an inferred formula that depends on recorded assumptions, and an inference involving the discharge of assumptions (modified from an MPTP challenge problem), are:

```
fof(e1,assumption, subset(c1,c2), introduced(assumption)).

fof(i3,plain, c2 = set_union2(c1,set_diff(c2,c1)),
    inference(conclusion,[status(thm),assumptions([dt_c1,e1])],[e3,i4])).

fof(i2,plain, (subset(c1,c2) => c2 = set_union2(c1,set_diff(c2,c1))),
    inference(discharge_asm,[status(thm),assumptions([dt_c1]),
        discharge_asm(discharge,[e1])], [e1,i3])),
```

---

[4] See `http://www.cs.miami.edu/~tptp/MPTPChallenge/Results/SVGResults.html`

This “recording” approach is more effective than adding assumptions as antecedents of implications to their descendants. Adding assumptions as antecedents would significantly increase the difficulty of the ATP problems built from the Mizar atomic inference steps, possibly preventing completion of the cross-verification. These syntactic additions are sufficient for translation of Mizar proofs into efficiently verifiable TPTP derivations (see Section 2.2). The consequence of this decision is a need to ensure that recorded assumptions are correctly propagated and discharged. This is done by GDV, as described in Section 3.

2.2 Mizar Natural Deduction

Quite often it is claimed that Mizar uses Jaskowski-style natural deduction. Even though Jaskowski’s formalism is certainly related to the Mizar formalism, there are too many differences to take Jaskowski’s original paper [Jas34] (or its more accessible version [Pel99]) as an accurate account of the Mizar formalism that could be used for explaining its translation to classical first-order logic form. Quite a precise overview of the Mizar formalism is given in [Har96], for the purpose of its implementation in HOL. In what follows the use of Gentzen-like introduction/elimination terminology, and comparison with various Gentzen-style calculi, is avoided. This is mainly to keep the exposition simple, and to avoid possible confusion caused by different flavors and semantics of those calculi used in different systems (the interested reader is referred to the detailed overview of these issues in [Pel99]).

The Mizar parser translates Mizar articles into a semantic XML format [Urb06b], which is used internally by the Mizar proof checker, and also by various external tools such as the Mizar-to-TPTP (MPTP) translator. The XML format is specified in the RELAX NG schema language.[5] The XML format specification is always up-to-date, because it is generated in a semi-automated way from the Mizar source code, and because XML validation can be used to check the XML representation of articles. For these reasons, the XML format specification has been used in this work as a complete up-to-date abstract syntax for the Mizar formalism.[6] The part of the XML format specification describing Mizar natural deduction proofs is (with minor simplification) as follows:

```
## Theorem as a proposition with justification.
JustifiedTheorem = element JustifiedTheorem {Proposition, Justification}
Justification = ( Inference | Proof | SkippedProof )
## By and From encode the atomic inference steps done by Mizar.
Inference = ( By | From | element ErrorInf { empty } )
## By encodes one simple justification from zero or more references.
By = element By { Ref* }
## From encodes a simple justification involving Mizar schemes
From = element From { Ref* }
## Proofs (of BlockThesis) encode ND reasonings consisting of many steps.
Proof = element Proof {BlockThesis, Reasoning }
## Reasoning is a series of skeleton and auxiliary items (steps)
```

---

[5] `http://relaxng.org/`

[6] Available at `ftp://mizar.uwb.edu.pl/pub/version/doc/xml/Mizar.html`

```
## optionally finished by reasoning per cases.
Reasoning = ( ( SkeletonItem | AuxiliaryItem )*, PerCasesReasoning? )
## Skeleton items are the steps that change the current thesis, the
## new Thesis is printed explicitly after them.
SkeletonItem = ((Let|Conclusion|Assume|Given|Take|TakeAsVar), Thesis)
## Auxiliary items are items which do not change thesis.
AuxiliaryItem = (JustifiedProposition|Consider|Set|Reconsider|
                 DefFunc|DefPred)
```

The `By` and `From` justifications denote atomic inference steps that are simple enough for the Mizar proof checker. There is no substructure to atomic inference steps, and they are translated directly as first-order inferences, e.g.,

```
A1: p;             fof(a1,plain,p).
A2: p implies q;   fof(a2,plain,p => q).
A3: q by A1, A2;   fof(a3,plain,q,inference(by,[status(thm)],[a1,a2])).
```

The Mizar inferences on the left translate into the TPTP format derivation on the right. (The translation can also add parent formulae that encode “background” information used implicitly by the Mizar proof checker, e.g., the Mizar type hierarchy.) The interesting case is when a full `Proof` is used in Mizar. This is done when the Mizar proof checker (used for the `By` and `From` constructs) is not strong enough, and the goal (`thesis` in Mizar terminology) either needs to be simplified by various natural deduction steps (`Let, Conclusion, Assume, Given, Take, TakeAsVar, PerCasesReasoning`), or the proof context needs to be augmented by a new lemma (`JustifiedProposition`), a new constant satisfying some property (`Consider, Reconsider`), or a useful macro (`Set, DefFunc, DefPred`). The following subsections explain these parts of Mizar reasoning in more detail.

Explanation of the Mizar Assumption and Conclusion steps

```
;; Example 1 (Mizar code)
Lemma1: p implies q;       ;; this has been proved already
Lemma2: q implies r;       ;; this has been proved already
Goal1: p implies q & r
proof                      ;; (Proof) thesis (Goal1) is: p implies q & r
  assume E1: p;            ;; (Assumption) thesis (T1) is: q & r
  thus E2: q by Lemma1,E1; ;; (Conclusion) thesis (T2) is: r
  thus E3: r by Lemma2,E2; ;; (Conclusion) thesis (T3) is: verum
end;
```

An `Assume` (keyword `assume`) step is used when the thesis can be considered to be an implication. Very much like in the original Jaskowski’s formalism, it adds the antecedent (`p` in Example 1) of the `thesis` (`p implies q & r`) to the list of valid propositions available for further use in this block, and changes the `thesis` of the block to its consequent (`q & r`). As can be seen from Example 2, the Mizar notion of *normalization* (see, e.g., [Wie00] for details) is quite strong. It identifies the proposition “`not p`” with the proposition “`p implies contradiction`”. In this feature Mizar certainly differs from Jaskowski’s original treatment. The advantage is that it turns a proof by contradiction into just a special case of proof by supposition.

```
;; Example 2 (Mizar code)
Lemma3: p implies q;       ;; this has been proved already
Lemma4: not q;             ;; this has been proved already
Goal2: not p;
proof                    ;; (Proof) thesis (Goal2) is: not p
  assume E1: p;          ;; (Assumption) thesis (T1) is: contradiction
  E2: q by E1, Lemma3;       ;; (JustifiedProposition) thesis unchanged
  thus E3: contradiction     ;; (Conclusion) thesis (T2) is: verum
             by E2,Lemma4;
end;
```

The `Conclusion` (keyword `thus`) step is used to prove (a part of) the current thesis. Mizar checks that the proposition that is proved is either equal (in a normalized form) to the current thesis (this is the E3 case in the first example), or a conjunct of the current thesis (this is the case E2 in the first example). This way, the thesis is reduced by each `Conclusion` step, and upon encountering the `end` keyword, Mizar checks that the thesis is just `verum` (*true*). It should be noted that here again the Mizar normalization is responsible for seemingly unrelated deconstructions of the thesis. For example, if the thesis is "`p iff q`", a `Conclusion` step proving "`p implies q`" is legal, transforming the thesis to "`q implies p`".

Translation of the Mizar Assumption and Conclusion steps

```
%% Translation of Example 1 into extended TPTP derivation
fof(lemma1,plain, p => q).
fof(lemma2,plain, q => r).
fof(e1,assumption,p,introduced(assumption)).
fof(e2,plain,q,
    inference(mizar_by,[status(thm),assumptions([e1])],[lemma1,e1])).
fof(e3,plain,r,
    inference(mizar_by,[status(thm),assumptions([e1])],[lemma2,e2])).
fof(t3,plain,$true,introduced(tautology)).
fof(t2,plain,r,
    inference(conclusion,[status(thm),assumptions([e1])],[t3,e3])).
fof(t1,plain,q & r,
    inference(conclusion,[status(thm),assumptions([e1])],[t2,e2])).
fof(goal1,plain,p => q & r,
    inference(discharge_asm,[status(thm),discharge_asm(discharge,[e1])],
              [e1,t1])).
```

The translation of Example 1 to a TPTP derivation is quite straightforward. Note that the `theses` are collected in a backwards manner and used to justify their `thesis`-predecessors, either by a `Conclusion` inference (usually involving some other lemmas), or by a `discharge_asm` inference involving a assumption.

This ends at the top-most thesis, i.e., the goal of the proof. The bottom-most `$true` thesis could be trivially removed, but retaining it makes the translated proofs more faithfully correspond to their Mizar counterparts. Note how the assumptions are propagated and finally discharged.

Explanation of the Mizar Let, Consider, and Take Steps

```
;; Example 3 (Mizar code)
Lemma1: for x being set holds p(x)         ;; this has been proved already
Lemma2: for x being set holds p(x) => q(x) ;; this has been proved already
Goal1: for x being set holds q(x)
proof                          ;; (Proof) thesis (Goal1) is: for x being set holds q(x)
  let new_constant be set;                ;; (Let) thesis (T1) is: q(new_constant)
  E1: p(new_constant) by Lemma1;          ;; (JustifiedProposition) thesis unchanged
  thus E2: q(new_constant) by Lemma2,E1;  ;; (Conclusion) thesis (T2) is: verum ($true)
end;

;; Example 4 (Mizar code)
Lemma3: ex x being set st p(f(x))          ;; this has been proved already
Lemma4: for x being set holds p(x) => q(x) ;; this has been proved already
Goal2: ex x being set st q(x);
proof                             ;; (Proof) thesis (Goal1) is: ex x being set st q(x)
  consider new_constant being set such       ;; (Consider) thesis unchanged, adds new_constant
    that E1: p(f(new_constant)) by Lemma3;   ;; of type set and proposition p(f(new_constant))
  take f(new_constant);                      ;; (Take) thesis (T1) is: q(f(new_constant))
  thus E2: q(f(new_constant)) by E1, Lemma4; ;; (Conclusion) thesis (T2) is: verum ($true)
end;
```

A `Let` (keyword `let`) step is used when the normalized form of thesis is a universally quantified formula. This step introduces a new constant of the proper type (`set` is used above), which is available from then on in the proof block. (In Mizar the new constants are numbered serially, so instead of the `new_constant` used in Example 3, it would be just, e.g., `c1`) The thesis is instantiated with this new constant (to `q(new_constant)` above). This corresponds to the common mathematical expression “let's have an arbitrary but fixed x”, and provided that the constant is fresh (which is guaranteed in Mizar by the serial numbering), the original universally quantified thesis follows from the instantiated thesis. (This is the standard theorem about constants in Hilbert's predicate calculus.)

The `Consider` (keyword `consider`) step has no effect on the proof thesis, although it is in some sense dual to the `Let` construct. Given a normalized existentially quantified statement (like `Lemma3` in Example 4), Mizar creates a fresh constant of the appropriate type and instantiates with it the existential statement. The new constant and the instance are available for further use in the proof block. Provided that the constant is fresh, this is again just a conservative extension of the original theory.

The `Take` (keyword `take`) step is used to select a suitable term for instantiating an existentially quantified thesis (`f(new_constant)` is taken in Example 4). The type of the term has to correspond to the type of the existential variable. The original thesis then follows from the instantiated one. A minor variant of this syntax is the `TakeAsVar` step (in Example 4 it would be “`take another_new_constant = f(new_constant);`”), which additionally creates a fresh constant of the same type as the selected term, adds their equality to the proof context, and instantiates the thesis with this fresh constant.

Translation of the Mizar Let, Consider, and Take Steps

```
%% Translation of Example~3 into extended TPTP derivation
fof(lemma1,plain, ![X]: (set(X) => p(X))).
fof(lemma2,plain, ![X]: (set(X) => (p(X) => q(X)))).
fof(henkin_ax1, plain, (set(new_constant) => q(new_constant)) =>
                       ![X]: (set(X) => q(X)),
    introduced(definition, [new_symbol(new_constant)])).
fof(type_ass1, assumption, set(new_constant), introduced(assumption)).
fof(e1,plain, p(new_constant), inference(mizar_by,[status(thm),
                             assumptions([type_ass1])],[type_ass1,lemma1])).
fof(e2,plain, q(new_constant), inference(mizar_by,[status(thm),
                          assumptions([type_ass1])],[type_ass1,e1,lemma2])).
fof(t2,plain, $true,introduced(tautology)).
fof(t1,plain, q(new_constant),
    inference(conclusion,[status(thm),assumptions([type_ass1])],[t2,e2])).
fof(t1_1,plain, set(new_constant) => q(new_constant),
    inference(discharge_asm, [status(thm),
               discharge_asm(discharge, [type_ass1])], [type_ass1, t1])).
fof(goal1,plain, ![X]: (set(X) => q(X)),
    inference(let, [status(thm)], [t1_1, henkin_ax1])).
```

This translation of Example 3 demonstrates that it is necessary to deal with Mizar's typed language, which is done through relativization to the untyped formalism used by ATP systems. The method of translating the `Let` inference step is the introduction of a Henkin (sometimes also called Skolem) axiom (`henkin_ax1` above) partially defining the new constant. These additional axioms are generated by the MPTP translation system from the known (instantiated) thesis (say "`T(constant)`") as the corresponding implications ("`T(constant) => ![X]:T(X)`"). Axioms of this form (as well as of the dual form used for translation of the `Consider` constructs, i.e., '`?[X]:T(X) => T(constant)`") extend the existing theory conservatively, provided that the constant is really new. To check this (a bit metalogical) fact (i.e., that Mizar always generates a fresh constant name here), the "`introduced(definition, [new_symbol(...)])`" TPTP status is used for these axioms. This tells GDV to carry out this check independently (see the description of leaf checks in Section 3.2). GDV does not yet implement a structural check verifying that these are really Henkin axioms, i.e., that they have the required form of an instance implying the universal formula (or of the dual existential form). Their correctness is "by construction" in the MPTP translation system, which is quite obvious from the following core of the Prolog code used for this:

```
%% given a Const-ant, and the Instance, generate a Henkin
%% axiom introducing Const
create_henkin_axiom_let(+Const,+Instance,-Henkin_Ax):-
        apply_const_subst((Const/NewVar), Instance, GenInst),
        Henkin_Ax = (Instance  => ( ! [NewVar] : GenInst )).

%% replace atom Const everywhere in Fla with Var, yielding Fla1
apply_const_subst(+(Const/Var),+Fla,-Fla1):-
... [code skipped] ...
```

Given that the relativization method is used for type translation, the `Let` steps additionally have to be complemented by the assumption and corresponding discharge of the new constant's type (see the translation example above).

The translation of Example 4 is not shown here because of space restrictions. The translation of the `Consider` construct is very similar to `Let` – again a corresponding Henkin axiom is added by the MPTP translation system, but the constant's type is derived instead of assumed. The translation of the `Take` (or `TakeAsVar`) steps is straightforward, as they only justify an existential thesis from its instance (which is a standard first-order inference).

Translation of the other Mizar reasoning constructs

The remaining Mizar reasoning constructs not explained so far are: `Given`, `PerCasesReasoning`, `JustifiedProposition`, `Reconsider`, `Set`, `DefFunc`, and `DefPred`. The `Set`, `DefFunc`, and `DefPred` constructs define macros (shorthands for terms and formulae) that are automatically expanded during the translation of formulae, and therefore the macros themselves do not have to be translated.[7] A `JustifiedProposition` is an arbitrary lemma introduced in the reasoning, and it (similarly to `JustifiedTheorem`) can be justified either by atomic inference steps, or by its own subproof. The `Given` step is a compressed syntax for an `Assume` step followed by a `Consider` step. `PerCasesReasoning` allows several logically complementary alternatives to be explored in Mizar, and it is translatable using the TPTP assumption format. `Reconsider` is used to justify that a term has a certain type, and it introduces a new constant (with the new type) as a shortcut for the term. This is translated as a corresponding equality introducing the new constant (its freshness is again checked by GDV), and a (provable) proposition stating the typing of the new constant. The translation of all these constructs does not require any more extensions of the TPTP formalism, or any new kinds of axioms, and given the explanations in the previous sections, their translation becomes quite routine. Due to the space restrictions, we therefore do not provide detailed examples of translations of these constructs here. Interested readers can consult the MizarTPTP web page (explained in more detail in [UTSP07]), where all (ca. 40000) Mizar theorems are linked to their corresponding TPTP derivations.[8]

3 The GDV Derivation Verifier

GDV [Sut06] is a tool for verifying derivations in the TPTP format. GDV uses structural and semantic techniques. Structural verification checks that inferences have been done correctly in the context of the derivation, e.g., checking that the derivation is acyclic. Semantic verification checks the required semantic properties of inference steps, e.g., checking that an inferred formula is a logical consequence of its parents. As [Sut06] already provides comprehensive coverage of the design and implementation of many aspects of GDV, this section provides an overview of the GDV process, and details of new aspects of GDV that were used in this work.

---

[7] ATP systems' clausifiers often use similar shortcuts to improve the resulting CNF.

[8] `http://www.cs.miami.edu/~tptp/MizarTPTP/`

3.1 Structural Verification

The structural verifications done by GDV are, for the most part, quite simple. The most basic checks are made first: checking that all the nodes in the derivation DAG are distinctly named, checking that all inference steps are adequately documented and the named parents exist, checking that the derivation is acyclic, i.e., the derivation is a DAG, and checking that refutations have (only) $false$ roots. More complex structural checks are made after that: checking that proofs by contradiction are well formed, checking that split refutations are not mutually dependent, and checking that inference chains with assumptions have been recorded correctly.

Other than the checking of assumptions, all the structural checks listed are described in [Sut06]. The checking of inference chains with assumptions was added to GDV for this work - it was not needed previously because GDV had been used only to check derivations from refutation based ATP systems, which do not use assumptions[9], while the natural deduction style of MML proofs makes extensive use of assumptions. The structural checking of inference chains with assumptions has two aspects. First it is necessary to check that assumptions are propagated from parents to inferred formulae, modulo discharged assumptions. This is done by ensuring that the set of assumptions of the parents, i.e., the parents that are assumptions plus the recorded assumptions of the parents, are a subset of the union of the discharged and recorded assumptions of the inferred formula. Second it is necessary to check that all assumptions are discharged. This is done by checking that no root of the DAG has any remaining recorded assumptions.

Note that these structural checks on assumptions make no examination of the logical formulae involved, so that they do not ensure that the discharges do correctly remove dependence on the assumptions. Semantic verification is used to check that, as is described in Section 3.2.

3.2 Semantic Verification

The technique used in semantic verification is to encode the required semantic relationship between each inferred formula and its parent formulae into logical obligations, in the form of ATP problems. The obligations are discharged by having trusted ATP systems solve the ATP problems. The required semantic relationship between an inferred formula and its parent formulae depends on the intent of the inference rule used. Most commonly an inferred formula is intended to be a theorem (logical consequence) of its parent formulae, but in other cases, e.g., Skolemization and splitting, the inferred formula has a weaker relationship with its parents. This intent is recorded as an SZS [SZS04] status annotation in the inference record of each inferred formula in a TPTP format derivation, e.g., in the second formulae in Section 2.1 the SZS status is `thm`, which indicates that the inferred formula is required to be a theorem of its parents. GDV's semantic verification has three phases: leaf verification, rule specific verification, and general semantic verification.

---

[9] However, it is planned to convert the checking of the negated conjectures in refutations to the checking of assumptions form.

There are two types of leaves in a derivation - leaves that come from the problem solved by the derivation, and leaves that were introduced by the ATP system. Verification of leaves that come from the problem is described fully in [Sut06], based on the idea that leaf axioms must be derivable from problem axioms, and a leaf conjecture must be able to derive the problem conjecture. Leaves are introduced into derivations by ATP systems for a variety of (legitimate) reasons. For this work the reasons that occur are the introduction of tautologies, the introduction of definitions, and the introduction of assumptions. Tautologies are verified by proving (using a trusted ATP system, as explained above) them. Definitions and assumptions do not produce a verification obligation, but definitions are checked to ensure that the defined symbol does not occur in any other non-introduced leaf.

Rule specific verification deals with explicit splitting, as implemented in the ATP system SPASS [WBH$^+$02], pseudo-splitting, as implemented in Vampire [RV02] and E [Sch02], application of definitions as implemented in E, and discharging assumptions as found in the MPTP derivations that are the focus of this work. The verification of splitting is dealt with in [Sut06], and the application of definitions does not arise in this work. To verify that assumptions have been discharged, the assumptions must be added as conditions on the non-assumption parents, forming an implication from the conjunction of the assumptions to the parents. The inferred formula must then be proved (the SZS status must be `thm`) from this implication. In GDV this overall verification obligation is broken down into an equivalent set of small verification obligations, to prove the inferred formula from each of the negated assumptions, and from the non-assumption parents. The benefit of breaking down this verification into the individual components is that a failure to discharge any of these verification obligations provides precise information regarding the fault in the derivation.

General semantic verification has not been modified from that described in [Sut06]. The most common type of verification obligation that must be discharged is, as explained above, to prove that the inferred formula is a theorem of its parents (SZS status `thm`). While this can always be directly attempted by a trusted ATP system, depending on the nature of the inference step the obligation problem might be quite difficult, and the discharge attempt might fail due to resource limits. In these cases GDV tries to disprove the obligation using model finding ATP systems, to establish that the inference is really faulty. Two other types of obligations that arise are to prove that the inferred formula is a countertheorem of it's parents (SZS status `cth`), e.g., in the case of a negated conjecture, and to prove a satisfiability-bijection between the parents and inferred formula, e.g., in Skolemization steps. At this stage GDV does only incomplete verification of satisfiability-bijection inferences. Such steps are therefore avoided in the translation of Mizar proofs, and the additional Henkin axioms are used instead for Mizar steps (`Let, Consider`) analogous to ATP Skolemization.

4 Verification of the MPTP Challenge Problems

The MPTP system was used to generate the derivations corresponding to the 252 problems in the *bushy* division of the MPTP challenge (the *chainy* division problems simply add redundant lemmas to the bushy versions). The derivations, together with the corresponding MPTP challenge problems, were given to GDV for verification. GDV was configured to use E 0.99 [Sch02] for finding proofs, and Paradox 2.0b [CS03] for disproofs. A 10s time limit was imposed on each E and Paradox run. All the structural checks (notably the propagation of assumptions) were successful, verification of leaves (checking that they either correspond to the problem, or are introduced definitions, tautologies, or assumptions) was successful, and all the (1303) ATP problems originating from checking the correctness of discharging assumptions were successfully proved.

From the 6765 ATP tasks corresponding to the Mizar atomic inference steps, 60 could not be solved in the 10s time limit by E, but none were found to be countersatisfiable by Paradox. Such timeouts can often be resolved by heuristic pruning of the (potentially superfluous) background formulae added as axioms by the MPTP translation. Therefore the MaLARea system, developed for doing such heuristic pruning along with time limit manipulations in a systematic and automated way [Urb07], was used. The 60 unsolved ATP tasks came from 27 derivations, i.e., at this point 225 of the problems had been verified. To create a wide training set for MaLARea's machine learning based heuristic pruning of axioms, all 759 of the ATP tasks generated by GDV during verification of the 27 derivations were put into the MaLARea's initial set of problems. MaLARea was then run with the maximal time limit set to 64 seconds, and the maximal axiom limit set to 64 axioms (these values were estimated empirically). The result of the MaLARea run was the solution of 742 tasks (this took about 4 hours), solving 43 of the 60 previously unsolved tasks, and leaving 17 tasks unsolved. This was a sufficiently low number to inspect the remaining problems manually.

One task contained quite large formulae, and could be solved easily by running E with more aggressive introduction of definitions during clausification. Thirteen tasks were solved by inspecting the original Mizar proofs and pruning unnecessary axioms by hand. It turns out that the 3 remaining ATP tasks (all coming from checking the derivation of Theorem 29 in Mizar article `YELLOW_0`) might be countersatisfiable. This was traced back to a very rare case when the MPTP algorithm for addition of background formulae was incomplete. Once one additional background formula was added manually to these three ATP tasks, all of them became easily provable. Also when this additional background formula was added to the original MPTP challenge problem, it became solvable directly by some ATP systems. This means that the original challenge problem is very likely countersatisfiable – this would have to be confirmed by finding a model. The original Mizar proof cannot be used to guide a proof due to the missing background formula, and the problem does not seem to have alternative solution without it. The remaining 251 MPTP Challenge problems were thus shown to be solvable, i.e., it has been verified that the problem conjectures really are theorems of the problem axioms. Detailed documentation of all the steps of the verification process described here is available online.[10]

5 Conclusions and Future Work

We have defined and implemented a translation of the Mizar natural deduction formalism to a simpler sequent-like formalism, encoded in the TPTP language (which required a mild extension to deal with assumptions). Proofs of all MML theorems have been exported into this format. The GDV verification tool has been extended to fully verify these derivations. Together with GDV's checking of the introduced derivation leaves, and a simple "correct by construction" argument about the introduced Henkin axioms, this provides a framework for full independent verification of the translated MML proofs. The framework was tested on 252 problems selected for the MPTP challenge, verifying the structural correctness of all the translated derivations, and using the E 0.99 system to automatically verify over 99% of the proof obligations coming from the atomic inference steps. 72% (43) of the remaining obligations were solved by the MaLARea system, 23% (14) were solved by expert advice, and 5% (3) were found to be very likely to be countersatisfiable, uncovering a rare case where the MPTP translation did not add enough Mizar background axioms for solution of the corresponding ATP problems. The result is a verification of 251 of the MPTP challenge problems, and a verification of the remaining problem corrected by addition of the missing background axiom.

An obvious future step is to verify the whole MML from its axioms. This will require some extensions of the MPTP system, so that the background theories are correctly introduced (most of them have to be proved in Mizar before they can be used), and not treated as just additional axioms. It would be generally hard for other proof assistants (such as HOL Light and Isabelle) to import and verify the translated derivations, because their automated theorem proving is quite weak. However, it is likely that these systems will be able to check the detailed proof objects created by ATP systems such as E and Otter. So other future work is to instruct ATP systems to create detailed proof objects during the GDV verification, and use them afterwards for refining the translated derivations to the point where other proof assistants will be able to import them. A similar import of the ATP proofs back to Mizar would also be useful (a basic translator for detailed Otter proof objects has been around since 2003). The imported proofs could be cross-verified by the Mizar proof checker to strengthen the confidence (which is necessarily empirical) in the translation and verification processes. Finally, it is planned to make this this verification process available as a push-button tool for Mizar authors.

---

[10] `http://lipa.ms.mff.cuni.cz/~urban/MPTPChallengeVerification.tar.gz`

References

[BDH$^+$99] Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.): TPHOLs 1999. LNCS, vol. 1690. Springer, Heidelberg (1999)

[CS03] Claessen, K., Sorensson, N.: New Techniques that Improve MACE-style Finite Model Finding. In: Baumgartner, P., Fermueller, C. (eds.) Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications (2003)

[Dav81] Davis, M.: Obvious logical inferences. In: Hayes, P.J. (ed.) IJCAI, pp. 530–531. William Kaufmann (1981)

[DFS06] Denney, E., Fischer, B., Schumann, J.: An empirical evaluation of automated theorem provers in software certification. International Journal on Artificial Intelligence Tools 15(1), 81–108 (2006)

[FS06] Furbach, U., Shankar, N. (eds.): IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 17–20. Springer, Heidelberg (2006)

[Hal04] Hales, T.C.: Formalizing the proof of the kepler conjecture. In: Slind, K., Bunker, A., Gopalakrishnan, G.C. (eds.) TPHOLs 2004. LNCS, vol. 3223, p. 117. Springer, Heidelberg (2004)

[Har96] Harrison, J.: A Mizar Mode for HOL. In: von Wright, J., Harrison, J., Grundy, J. (eds.) TPHOLs 1996. LNCS, vol. 1125, pp. 203–220. Springer, Heidelberg (1996)

[Jas34] Jaskowski, S.: On the rules of suppositions. Studia Logica, 1 (1934)

[McL06] McLaughlin, S.: An interpretation of Isabelle/HOL in HOL Light. In: Furbach and Shankar [FS06], pp. 192–204

[MS00] McCune, W., Shumsky, O.: System description: IVY. In: McAllester, D. (ed.) Automated Deduction - CADE-17. LNCS, vol. 1831, pp. 401–405. Springer, Heidelberg (2000)

[NB02] Naumowicz, A., Bylinski, C.: Basic elements of computer algebra in MIZAR. Mechanized Mathematics and Its Applications 2 (2002)

[NB04] Naumowicz, A., Bylinski, C.: Improving mizar texts with properties and requirements. In: Asperti, A., Bancerek, G., Trybulec, A. (eds.) MKM 2004. LNCS, vol. 3119, pp. 290–301. Springer, Heidelberg (2004)

[OS06] Obua, S., Skalberg, S.: Importing HOL into Isabelle/HOL. In: Furbach and Shankar [FS06], pp. 298–302

[Pel99] Pelletier, F.J.: A brief history of natural deduction. History and Philosophy of Logic 20, 1–31 (1999)

[RT99] Rudnicki, P., Trybulec, A.: On equivalents of well-foundedness. J. Autom. Reasoning 23(3-4), 197–234 (1999)

[Rud87] Rudnicki, P.: Obvious inferences. J. Autom. Reasoning 3(4), 383–393 (1987)

[Rud92] Rudnicki, P.: An overview of the Mizar project. In: 1992 Workshop on Types for Proofs and Programs, Chalmers University of Technology, Bastad, pp. 311–332 (1992)

[RV02] Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. Journal of AI Communications 15(2-3), 91–110 (2002)

[Sch02] Schulz, S.: E – a brainiac theorem prover. Journal of AI Communications 15(2-3), 111–126 (2002)

[SS98] Sutcliffe, G., Suttner, C.B.: The TPTP problem library: CNF release v1.2.1. Journal of Automated Reasoning 21(2), 177–203 (1998)

[Sut06] Sutcliffe, G.: Semantic Derivation Verification. International Journal on Artificial Intelligence Tools 15(6), 1053–1070 (2006)

[Sym99] Syme, D.: Three tactic theorem proving. In: Bertot et al. [BDH+99], pp. 203–220

[SZS04] Sutcliffe, G., Zimmer, J., Schulz, S.: TSTP Data-Exchange Formats for Automated Theorem Proving Tools. In: Sorge, V., Zhang, W. (eds.) Distributed and Multi-Agent Reasoning. Frontiers in Artificial Intelligence and Applications, IOS Press, Amsterdam (2004)

[Urb04] Urban, J.: MPTP - motivation, implementation, first experiments. Journal of Automated Reasoning 33(3-4), 319–339 (2004)

[Urb06a] Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)

[Urb06b] Urban, J.: XML-izing Mizar: making semantic processing and presentation of MML easy. In: Kohlhase, M. (ed.) MKM 2005. LNCS (LNAI), vol. 3863, pp. 346–360. Springer, Heidelberg (2006)

[Urb07] Urban, J.: MaLARea: a metasystem for automated reasoning in large theories. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) ESARLT: Empirically Successful Automated Reasoning in Large Theories. CEUR Workshop Proceedings. CEUR, vol. 257, pp. 45–58 (2007)

[UTSP07] Urban, J., Trac, S., Sutcliffe, G., Puzis, Y.: Combining Mizar and TPTP semantic presentation tools. In: Proceedings of the Mathematical User-Interfaces Workshop 2007 (2007), `http://www.activemath.org/workshops/MathUI/07/proceedings/Urban-etal-MizarIDV.html`

[WBH$^+$02] Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topic, D.: SPASS version 2.0. In: CADE, pp. 275–279 (2002)

[Wen99] Wenzel, M.: Isar - a generic interpretative approach to readable formal proof documents. In: Bertot et al. [BDH+99], pp. 167–184

[Wie00] Wiedijk, F.: CHECKER - notes on the basic inference step in Mizar (2000), available at `http://www.cs.kun.nl/~freek/mizar/by.dvi`

[Zam99] Zammit, V.: On the implementation of an extensible declarative proof language. In: Bertot et al. [BDH+99], pp. 185–202

Author Index

Abadi, Aharon 17
Abate, Pietro 32
Akbarpour, Behzad 47
Axelsson, Roland 62

Baaz, Matthias 77
Baelde, David 92
Bertossi, Leopoldo 107
Bezem, Marc 123
Blanqui, Frédéric 138
Bonichon, Richard 151
Brandt, Sebastian 166
Bravo, Loreto 107

Chevalier, Yannick 181
Ciabattoni, Agata 77
Cortier, Véronique 196

Dargaye, Zaynah 211
de Groote, Philippe 273
Dechesne, Francien 226
Delahaye, David 151
Delaune, Stéphanie 196, 242
Demri, Stéphane 531
Denecker, Marc 378
Doligez, Damien 151
Dubois, Catherine 288

Eiter, Thomas 514

Fermüller, Christian G. 77
Ferrante, Alessandro 438

Giordano, Laura 257
Gliozzi, Valentina 257
Goré, Rajeev 32

Hach, Faraz 423

Jaeger, Éric 288
Jouannaud, Jean-Pierre 138

Kazakov, Yevgeny 303
Kieroński, Emanuel 318
Krisnadhi, Adila 333

Lange, Martin 62
Langholm, Tore 123
Leroy, Xavier 211
Lin, Hai 242
Ludwig, Michel 348
Lugiez, Denis 181
Lutz, Carsten 333
Lynch, Christopher 242

Maarek, Sarah 273
Maieli, Roberto 363
Makowsky, Johann A. 1
Mariën, Maarten 378
Mazza, Damiano 393
Miličić, Maja 408
Miller, Dale 92
Mitchell, David G. 423
Mohebali, Raheleh 423
Mousavi, MohammadReza 226
Murano, Aniello 438

Olivetti, Nicola 257
Orzan, Simona 226

Pagani, Michele 393
Parente, Mimmo 438
Paulson, Lawrence C. 47
Pereira, Luís Moniz 454
Pinto, Alexandre Miguel 454
Pozzato, Gian Luca 257
Pulcini, Gabriele 469

Rabinovich, Alexander 17, 531
Rubio, Albert 138
Rusinowitch, Michaël 181

Sagiv, Mooly 17
Samer, Marko 2, 484
Santocanale, Luigi 499
Sattler, Ulrike 303
Šimkus, Mantas 514
Sutcliffe, Geoff 546
Szeider, Stefan 484

Tendera, Lidia 318

Urban, Josef 546

Veith, Helmut 2
Venema, Yde 499

Waldinger, Richard 15
Waldmann, Uwe 348
Walicki, Michał 123
Widmann, Florian 32
Wittocx, Johan 378

Yoshinaka, Ryo 273

Zolin, Evgeny 303