Preface

The annual conference of the European Association for Computer Science Logic, CSL 2003, was held jointly with the 8th Kurt Gödel Colloquium, 8. KGC, at the Vienna University of Technology on 25–30 August 2003.

The conference series CSL started as a program of international workshops on Computer Science Logic, and then in its sixth meeting became the annual conference of the EACSL. This conference was the 17th meeting and 12th EACSL conference.

The KGC is the biennial conference of the Kurt Gödel Society. It has taken place in various formats and has been devoted to special topics in logic, such as computational logic, set theory, algebraic logic, and history of logic.

The CSL 2003 and 8. KGC were organized by the Kurt Gödel Society jointly with the Institut für Algebra und Computermathematik and the Institut für Computersprachen, Vienna University of Technology.

The CSL 2003 and 8. KGC joint program committee had the difficult task of choosing among 112 mostly high-quality submissions. Each paper was refereed by at least two reviewers and then discussed by the whole program committee. In the final stage, during a three day electronic discussion, the program committee selected 39 papers for presentation at the conference and publication in these proceedings. Unfortunately, many high-quality papers had to be rejected due to the limitations set by the conference duration.

The program committee chose as invited speakers Bruno Buchberger, Dov Gabbay, Helmut Veith, Nikolai Vorobjov, and Andrei Voronkov. Additionally, Sergei Artemov was invited jointly by CSL 2003 and 8. KGC and the European Summer School in Logic Language and Information (ESSLLI 2003) for the keynote address of both events.

In addition to the main conference program, CSL 2003 and 8. KGC featured four tutorials given by Ahmed Bouajjani (Verification of Infinite State Systems), Georg Moser and Richard Zach (The Epsilon Calculus), Nikolai Vorobjov (Effective Quantifier Elimination over Real Closed Fields), Igor Walukiewicz (Winning Strategies and Synthesis of Controllers).

The last day of CSL 2003 and 8. KGC was jointly held with the 2nd annual workshop of the European Research Training Network GAMES (Games and Automata for Synthesis and Validation), organized by Erich Grädel.

We thank the program committee and all the referees for the work in reviewing the papers. We are grateful to Norbert Preining for taking care of the main organizational tasks and we thank the remaining members of the local organizing team, Arnold Beckmann, Agata Ciabattoni, Christian Fermüller, Rosalie Iemhoff, and Sebastiaan Terwijn.
We would like to thank the following institutions for supporting the meeting: the European Association for Computer Science Logic (EACSL), the Austrian Federal Ministry for Education, Science and Culture, the City Council of Vienna, the Vienna University of Technology, and the companies IBM and Siemens.

June 2003

Matthias Baaz and Johann Makowsky
Program Committee

Matthias Baaz Vienna University of Technology (Chair)
Arnold Beckmann Vienna University of Technology
Lev Beklemishev Steklov Institute, Moscow/Utrecht Univ.
Agata Ciabattoni Vienna University of Technology
Kousha Etessami University of Edinburgh
Chris Fermüller Vienna University of Technology
Didier Galmiche LORIA - UHP Nancy
Harald Ganzinger Max Plank Institut Saarbrücken
Erich Grädel Aachen University
Petr Hajek Czech Academy of Science
Martin Hyland University of Cambridge
Reinhard Kahle Universidade Nova de Lisboa
Helene Kirchner LORIA CNRS, Nancy
Daniel Leivant Indiana University Bloomington
Johann Makowsky Technion-IIT Haifa (Co-chair)
Jerzy Marcinkowski Wroclaw University
Franco Montagna University of Siena
Robert Nieuwenhuis Tech. Univ. Catalonia Barcelona
Michel Parigot CNRS University of Paris 7
Jeff Paris Manchester University
Helmut Schwichtenberg Munich University
Jerzy Tiuryn Warsaw University

Additional Referees

Klaus Aehlig Veronique Cortier Hugo Herbelin
José Alferes Victor Dalmau M. Hermann
Rajeev Alur Rod Downey William Hesse
Carlos Areces Jacques Duparc Wim Hesselinx
Albert Atserias Roy Dyckhoff Pascal Hitzler
Ralph Back Mauro Ferrari Michael Huth
Ulrich Berger N. France Rosalie Iemhoff
Dietmar Berwanger Carsten Führmann Neill Immerman
Lars Birkedal Brunella Gerla Mark Jerrum
Bruno Blanchet Silvio Ghilardi Felix Joachimski
Steve Bloch Patrice Godefroid Marcin Jurdzinski
Stefano Borgo Rajeev Gore Hans Jürgen Ohlbach
Glen Bruns Bernhard Gramlich Lukasz Kaiser
Marco Cadoli Martin Grohe Michael Kohlhase
Witold Charatonik O. Grumberg Boris Konev
Petr Cintula Therese Hardin Jan Krajicek
Horatiu Cirstea Jerry den Hartog Lars Kristiansen
Jean-Louis Krivine  Virgile Mogbil  Harold Simmons
Orna Kupferman  Tobias Niopkow  Alex Simpson
Dietrich Kuske  Hans de Nivelle  V. Sofronie-Stokkermans
Oliver Kutz  Damian Niwinski  V. Stoltenberg-Hansen
Olivier Laurent  Gethin Norman  Martin Strecke
Salvatore La Torre  Jean-Marc Notin  Thomas Streicher
Martin Lange  Pawel Olszta  Wieslaw Szwast
Dominique Larchey  V. van Oostrom  Lidia Tendra
Alexander Leitsch  Fernando Orejas  Wolfgang Thomas
Pierre Lescanne  Geoffrey Ostrin  Alwen Tiu
Luigi Liquori  Martin Otto  Jacobo Toran
Christof Loeding  J. van de Pol  Christian Urban
Markus Lohrey  Chris Pollett  Alvaro del Val
Byron Long  Christophe Raffalli  Helmut Veith
Carsten Lutz  Alexander Rabinovich  Laurent Vigneron
P. Madhusudan  E. Ravve  Albert Visser
Savi Maharaj  Maarten de Rijke  Peter Vojtas
Yevginiy Makarov  Christophe Ringeissen  Benjamin Werner
Jean-Yves Marion  Philipp Rohde  Uwe Waldmann
Narciso Marti-Oliet  Marie-Christine Rousset  I. Walukiewicz
Maarten Marx  Paul Roziere  Heinrich Wansing
K. Meer  V. Rybakov  Andreas Weiermann
Daniel Mery  Luigi Santocanale  ToMasz Wierzbicki
Stephan Merz  Renate Schmidt  Lucian Wischik
George Metcalfe  Manfred Schmidt-Schauß  Alberto Zanardo
Dale Miller  Peter Selinger  Jeffrey Zucker
Oskar Mis  Kurt Sieber

Local Organizing Committee

Matthias Baaz, Chair
Arnold Beckmann
Agata Ciabattoni
Christian Fermüller
Rosalie Iemhoff
Norbert Preining
Sebastiaan Terwijn
Table of Contents

Deciding Monotonic Games ........................................ 1  
   Parosh Aziz Abdulla, Ahmed Bouajjani, and Julien d’Orso

The Commuting V-Diagram ........................................ 15  
   Bernhard K. Aichernig

Concurrent Construction of Proof-Nets ............................ 29  
   Jean-Marc Andreoli and Laurent Mazaré

Back to the Future: Explicit Logic for Computer Science  
(Invited Lecture) ................................................... 43  
   Sergei Artemov

Constraint Satisfaction with Countable Homogeneous Templates  .... 44  
   Manuel Bodirsky and Jaroslav Nešetřil

Quantified Constraints: Algorithms and Complexity ............... 58  
   Ferdinand Börner, Andrei Bulatov, Peter Jeavons,  
   and Andrei Krokhin

Verification of Infinite State Systems (Tutorial) ..................... 71  
   Ahmed Bouajjani

Parity of Imperfection or Fixing Independence ...................... 72  
   Julian C. Bradfield

Atomic Cut Elimination for Classical Logic ....................... 86  
   Kai Brünnler

Computational Mathematics, Computational Logic,  
and Symbolic Computation (Invited Lecture) ....................... 98  
   Bruno Buchberger

Simple Stochastic Parity Games .................................. 100  
   Krishnendu Chatterjee, Marcin Jurdziński,  
   and Thomas A. Henzinger

Machine Characterizations of the Classes of the W-Hierarchy .... 114  
   Yijia Chen and Jörg Flum

Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number  
of Sessions .......................................................... 128  
   Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch,  
   Mathieu Turuani, and Laurent Vigneron
On Relativisation and Complexity Gap for Resolution-Based Proof Systems ...................... 142
  Stefan Dantchev and Søren Riis

Strong Normalization of the Typed \( \lambda_{ws} \)-Calculus .................................. 155
  René David and Bruno Guillaume

A Fixed-Point Logic with Symmetric Choice .................................. 169
  Anuj Dawar and David Richerby

Positive Games and Persistent Strategies .................................. 183
  Jacques Duparc

Generating All Abductive Explanations for Queries on Propositional Horn Theories ................. 197
  Thomas Eiter and Kazuhisa Makino

Refined Complexity Analysis of Cut Elimination .................................. 212
  Philipp Gerhardy

Comparing the Succinctness of Monadic Query Languages over Finite Trees ...................... 226
  Martin Grohe and Nicole Schweikardt

The Arithmetical Complexity of Dimension and Randomness .................................. 241
  John M. Hitchcock, Jack H. Lutz, and Sebastiaan A. Terwijn

Towards a Proof System for Admissibility .................................. 255
  Rosalie Iemhoff

Program Complexity of Dynamic LTL Model Checking .................................. 271
  Detlef Kähler and Thomas Wilke

Coping Polynomially with Numerous but Identical Elements within Planning Problems .................. 285
  Max Kanovich and Jacqueline Vauzeilles

On Algebraic Specifications of Abstract Data Types .................................. 299
  Bakhadyr Khoussainov

On the Complexity of Existential Pebble Games .................................. 314
  Phokion G. Kolaitis and Jonathan Panttaja

Computational Aspects of \( \Sigma \)-Definability over the Real Numbers without the Equality Test .................................. 330
  Margarita Korovina

The Surprising Power of Restricted Programs and Gödel’s Functionals .................. 345
  Lars Kristiansen and Paul J. Voda
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pebble Games on Trees</td>
<td>359</td>
</tr>
<tr>
<td><em>Lukasz Krzeszczakowski</em></td>
<td></td>
</tr>
<tr>
<td>Bistability: An Extensional Characterization of Sequentiality</td>
<td>372</td>
</tr>
<tr>
<td><em>James Laird</em></td>
<td></td>
</tr>
<tr>
<td>Automata on Lempel-Ziv Compressed Strings</td>
<td>384</td>
</tr>
<tr>
<td><em>Hans Leiß and Michel de Rougemont</em></td>
<td></td>
</tr>
<tr>
<td>Complexity of Some Problems in Modal and Intuitionistic Calculi</td>
<td>397</td>
</tr>
<tr>
<td><em>Larisa Maksimova and Andrei Voronkov</em></td>
<td></td>
</tr>
<tr>
<td>Goal-Directed Calculi for Gödel-Dummett Logics</td>
<td>413</td>
</tr>
<tr>
<td><em>George Metcalfe, Nicola Olivetti, and Dov Gabbay</em></td>
<td></td>
</tr>
<tr>
<td>A Logic for Probability in Quantum Systems</td>
<td>427</td>
</tr>
<tr>
<td><em>Ron van der Meyden and Manas Patra</em></td>
<td></td>
</tr>
<tr>
<td>A Strongly Normalising Curry-Howard Correspondence for IZF Set Theory</td>
<td>441</td>
</tr>
<tr>
<td><em>Alexandre Miquel</em></td>
<td></td>
</tr>
<tr>
<td>The Epsilon Calculus (Tutorial)</td>
<td>455</td>
</tr>
<tr>
<td><em>Georg Moser and Richard Zach</em></td>
<td></td>
</tr>
<tr>
<td>Modular Semantics and Logics of Classes</td>
<td>456</td>
</tr>
<tr>
<td><em>Bernhard Reus</em></td>
<td></td>
</tr>
<tr>
<td>Validity of CTL Queries Revisited</td>
<td>470</td>
</tr>
<tr>
<td><em>Marko Samer and Helmut Veith</em></td>
<td></td>
</tr>
<tr>
<td>Calculi of Meta-variables</td>
<td>484</td>
</tr>
<tr>
<td><em>Masahiko Sato, Takafumi Sakurai, Yukiyoshi Kameyama, and Atsushi Igarashi</em></td>
<td></td>
</tr>
<tr>
<td>Henkin Models of the Partial λ-Calculus</td>
<td>498</td>
</tr>
<tr>
<td><em>Lutz Schröder</em></td>
<td></td>
</tr>
<tr>
<td>Nominal Unification</td>
<td>513</td>
</tr>
<tr>
<td><em>Christian Urban, Andrew Pitts, and Murdoch Gabbay</em></td>
<td></td>
</tr>
<tr>
<td>Friends or Foes? Communities in Software Verification (Invited Lecture)</td>
<td>528</td>
</tr>
<tr>
<td><em>Helmut Veith</em></td>
<td></td>
</tr>
<tr>
<td>More Computation Power for a Denotational Semantics for First Order Logic</td>
<td>530</td>
</tr>
<tr>
<td><em>Kess F.M. Vermeulen</em></td>
<td></td>
</tr>
</tbody>
</table>
Effective Model Completeness of the Theory
of Restricted Pfaffian Functions (Invited Lecture) .......... 544
    Nicolai Vorobjov

Effective Quantifier Elimination over Real Closed Fields (Tutorial) ...... 545
    Nicolai Vorobjov

Fast Infinite-State Model Checking in Integer-Based Systems
(Invited Lecture).................................................. 546
    Tatiana Rybina and Andrei Voronkov

Winning Strategies and Synthesis of Controllers (Tutorial) .............. 574
    Igor Walukiewicz

Logical Relations for Dynamic Name Creation .................... 575
    Yu Zhang and David Nowak

Author Index .......................................................... 589
Deciding Monotonic Games

Parosh Aziz Abdulla¹, Ahmed Bouajjani², and Julien d’Orso¹

¹ Uppsala University, Sweden
² University of Paris 7, France

Abstract. In an earlier work [ACJYK00] we presented a general framework for verification of infinite-state transition systems, where the transition relation is monotonic with respect to a well quasi-ordering on the set of states. In this paper, we investigate extending the framework from the context of transition systems to that of games. We show that monotonic games are in general undecidable. We identify a subclass of monotonic games, called downward closed games. We provide algorithms for analyzing downward closed games subject to winning conditions which are formulated as safety properties.

1 Introduction

One of the main challenges undertaken by the model checking community has been to develop algorithms which can deal with infinite state spaces. In a previous work [ACJYK00] we presented a general framework for verification of infinite-state transition systems. The framework is based on the assumption that the transition relation is monotonic with respect to a well quasi-ordering on the set of states (configurations). The framework has been used both to give uniform explanations of existing results for infinite-state systems such as Petri nets, Timed automata [AD90], lossy channel systems [AJ96], and relational automata [BBK77,Čer94]; and to derive novel algorithms for model checking of Broadcast protocols [EFM99,DEP99], timed Petri nets [AN01], and cache coherence protocols [Del00], etc.

A related approach to model checking is that of control [AHK97]. Behaviours of reactive systems can naturally be described as games [dAHM01,Tho02], where control problems can be reduced to the problem of providing winning strategies. Since the state spaces of reactive systems are usually infinite, it is relevant to try to design algorithms for solving games over infinite state spaces.

In this paper, we consider extending the framework of [ACJYK00] from the context of transition systems to that of games. This turns out to be non-trivial. In fact, for one of the simplest classes of monotonic transition systems, namely Petri nets, we show that the game problem is undecidable. The negative result holds for games with the simplest possible winning condition, namely that of safety. Such a game is played between two players A and B, where player A tries to avoid a given set of bad configurations, while player B tries to force the play into such a configuration. On the other hand, we show decidability of the safety game problem for a subclass of monotonic games, namely downward
closed games: if a player can make a move from a configuration $c_1$ to another configuration $c_2$, then all configurations which are larger than $c_1$ (with respect to the ordering on the state space) can also make a move to $c_2$. Typical examples of downward closed systems are those with lossy behaviours such as lossy channel systems \[AJ96b\] and lossy VASS \[BM99\].

We summarize our (un)decidability results as follows:

- Decidability of the safety problem for games where player $B$ has a downward closed behaviour (a $B$-downward closed game). Considering the case where only one player is downward closed is relevant, since it allows, for instance, modelling behaviours of systems where one player (representing the environment) may lose messages in a lossy channel system (a so called $B$-LCS game). In case player $A$ has a deterministic behaviour (has no choices), our algorithm for $B$-downward closed games degenerates to the symbolic backward algorithm presented in \[AJ96b,ACJYK00\] for checking safety properties. In fact, we give a characterization of the set of winning (and losing) configurations in such a game. Observe that this result implies decidability of the case when both players have downward closed behaviours.

- Decidability of the safety problem for $A$-downward closed games. In case player $B$ has a deterministic behaviour, our algorithm for $A$-downward closed games degenerates to the forward algorithms described in \[AJ96b,ACJYK00\] and \[Fin94,FS98\] for checking eventuality properties (of the form $\forall \Diamond p$). However, in contrast to $B$-downward closed games, we show it is not possible to give a characterization of the set of winning (or losing) configurations.

- Decidability results for downward closed games do not extend to monotonic games. In particular we show that deciding safety properties for games based on VASS (Vector Addition Systems with States) is undecidable. VASS is a variant of Petri nets. The undecidability result holds even if both players are assumed to have monotonic behaviours.

- Undecidability of parity games for both $A$- and $B$-downward closed games. In a parity game, each configuration is equipped with a rank chosen from a finite set of natural numbers. The winning condition is defined by the parity of the lowest rank of a configuration appearing in the play. In particular, we show undecidability of parity games for both A-LCS and B-LCS games. On the other hand, if both players can lose messages, then the problem is decidable.

Outline. In the next Section, we recall some basic definitions for games. In Section 3 we introduce monotonic and downward closed games. We present a symbolic algorithm for solving the safety problem for $B$-downward closed games in Section 4 and apply the algorithm to $B$-LCS in Section 5. In Section 6 we consider $A$-downward closed games. In Section 7 we show that the safety problem is undecidable for monotonic games. In Section 8 we study decidability of parity games for the above models.
2 Preliminaries

In this section, we recall some standard definitions for games.

A game $G$ is a tuple $(C, C_A, C_B, \rightarrow, C_F)$, where $C$ is a (possibly infinite) set of configurations, $C_A, C_B$ is a partitioning of $C$, $\rightarrow \subseteq (C_A \times C_B) \cup (C_B \times C_A)$ is a set of transitions, and $C_F \subseteq C_A$ is a finite set of final configurations. We write $c_1 \rightarrow c_2$ to denote that $(c_1, c_2) \in \rightarrow$. For a configuration $c$, we define $\text{Pre}(c) = \{c' | c' \rightarrow c\}$, and define $\text{Post}(c) = \{c' | c \rightarrow c'\}$. We extend $\text{Pre}$ to sets of configurations such that $\text{Pre}(D) = \cup_{c \in D} \text{Pre}(c)$. The function $\text{Post}$ can be extended in a similar manner. Without loss of generality, we assume that there is no deadlock, i.e., $\text{Post}(c) \neq \emptyset$ for each configuration $c$. For a set $D \subseteq C_A$ of configurations, we define $A^D$ to be the set $C_A \setminus D$. The operator $\sim$ is defined in a similar manner. For a set $D \subseteq C_A$, we use $\overline{\text{Pre}}(D)$ to denote $\overline{\text{Pre}}(A^D)$.

For $E \subseteq C_B$, we define $\overline{\text{Post}}(E)$ in a similar manner.

A play $P$ (of $G$) from a configuration $c$ is an infinite sequence $c_0, c_1, c_2, \ldots$ of configurations such that $c_0 = c$, and $c_i \rightarrow c_{i+1}$, for each $i \geq 0$. A play $c_0, c_1, c_2, \ldots$ is winning (for $A$) if there is no $j \geq 0$ with $c_j \in C_F$.

A strategy for player $A$ (or simply an $A$-strategy) is a partial function $\sigma_A : C_A \mapsto C_B$ such that $c \rightarrow \sigma_A(c)$. A $B$-strategy is a partial function $\sigma_B : C_B \mapsto C_A$ and is defined in a similar manner to $\sigma_A$. A configuration $c \in C_A$ together with strategies $\sigma_A$ and $\sigma_B$ (for players $A$ and $B$ respectively) define a play $P(c, \sigma_A, \sigma_B) = c_0, c_1, c_2, \ldots$ from $c$ where $c_{2i+1} = \sigma_A(c_{2i})$, and $c_{2i+2} = \sigma_B(c_{2i+1})$, for $i \geq 0$. A similar definition is used in case $c \in C_B$ (interchanging the order of applications of $\sigma_A$ and $\sigma_B$ to the configurations in the sequence).

An $A$-strategy $\sigma_A$ is said to be winning from a configuration $c$, if for all $B$-strategies $\sigma_B$, it is the case that $P(c, \sigma_A, \sigma_B)$ is winning. A configuration $c$ is said to be winning if there is a winning $A$-strategy from $c$.

We shall consider the safety problem for games:

The safety problem

Instance. A game $G$ and a configuration $c$.

Question. Is $c$ winning?

3 Ordered Games

In this section, we introduce monotonic and downward closed games.

Orderings. Let $A$ be a set and let $\preceq$ be a quasi-order (i.e. a reflexive and transitive binary relation) on $A$. We say that $\preceq$ is a well quasi-ordering (wqo) on $A$ if there is no infinite sequence $a_0, a_1, a_2, \ldots$ with $a_i \not\preceq a_j$ for $i < j$. For $B \subseteq A$, we say that $B$ is canonical if there are no $a, b \in B$ with $a \neq b$ and $a \preceq b$. We use $\text{min}$ to denote a function where, for $B \subseteq A$, the value of $\text{min}(B)$ is a canonical subset of $B$ such that for each $b \in B$ there is $a \in \text{min}(B)$ with $a \preceq b$. We say that $\preceq$ is decidable if, given $a, b \in A$ we can decide whether $a \preceq b$. A set $B \subseteq A$
is said to be upward closed if \( a \in B \) and \( a \preceq b \) imply \( b \in B \). A downward closed set is defined in a similar manner.

**Monotonic Games.** An ordered game \( G = (C, C_A, C_B, \rightarrow, C_F, \preceq) \) is a game and \( \preceq \subseteq (C_A \times C_A) \cup (C_B \times C_B) \) is a decidable wqo on the sets \( C_A \) and \( C_B \). The ordered game \( G \) is said to be monotonic with respect to player \( A \) (or simply \( A \)-monotonic) if, for each \( c_1, c_2 \in C_A \) and \( c_3 \in C_B \), whenever \( c_1 \preceq c_2 \) and \( c_1 \rightarrow c_3 \), there is a \( c_4 \) with \( c_3 \preceq c_4 \) and \( c_2 \rightarrow c_4 \). A \( B \)-monotonic game is defined in a similar manner. A monotonic game is both \( A \)-monotonic and \( B \)-monotonic.

**Downward Closed Games.** An ordered game \( G = (C, C_A, C_B, \rightarrow, C_F, \preceq) \) is said to be \( A \)-downward closed if, for each \( c_1, c_2 \in C_A \) and \( c_3 \in C_B \), whenever \( c_1 \rightarrow c_3 \) and \( c_1 \preceq c_2 \), then \( c_2 \rightarrow c_3 \). A \( B \)-downward closed game is defined in a similar manner. A game is downward closed if it is both \( A \)- and \( B \)-downward closed. Notice that each class of downward closed games is included in the corresponding class of monotonic games. For instance, each \( A \)-downward closed game is \( A \)-monotonic. From the definitions we get the following property.

**Lemma 1.** For an \( A \)-downward closed game \( G \) and any set \( E \subseteq C_B \), the set \( \text{Pre}(E) \) is upward closed. A similar result holds for \( B \)-downward closed games.

### 4 B-Downward Closed Games

We present a symbolic algorithm for solving the safety problem for \( B \)-downward closed games. In the rest of this Section, we assume an \( B \)-downward closed game \( G = (C, C_A, C_B, \rightarrow, C_F, \preceq) \).

**Scheme.** Given a configuration \( c \) in \( G \), we want to decide whether \( c \) is winning or not. To do that, we introduce a scheme by considering a sequence of sets of configurations of the form:

\[
s : \quad D_0, \; E_0, \; D_1, \; E_1, \; D_2, \; E_2, \; \ldots
\]

where \( D_i \subseteq C_A \) and \( E_i \subseteq C_B \). Intuitively, the sets \( D_i \) and \( E_i \) characterize the configurations (in \( C_A \) and \( C_B \) respectively) which are not winning. The elements of the sequence are defined by

\[
D_0 = C_F \quad \quad \quad E_0 = \text{Pre}(D_0)
\]

\[
D_{i+1} = D_i \cup \overline{\text{Pre}(E_i)} \quad \quad \quad E_{i+1} = E_i \cup \text{Pre}(D_{i+1}) \quad \quad \quad i = 0, 1, 2, \ldots
\]

We say that \( s \) converges (at \( \ell \)) if \( D_{\ell+1} \subseteq D_\ell \) or \( E_{\ell+1} \subseteq E_\ell \). In such a case, the set \( D_\ell \cup E_\ell \) characterizes exactly the set of configurations which are not winning. The question of whether a given configuration \( c \) is winning amounts therefore to whether \( c \not\in (D_\ell \cup E_\ell) \). To show that our characterization is correct, we show the following two Lemmas. The first Lemma shows that if \( c \) appears in one of the generated sets then it is not a winning configuration. The second Lemma states that if the sequence converges, then the generated sets contain all non-winning configurations.
Lemma 2. If $c \in D_i \cup E_i$, for some $i \geq 0$, then $c$ is not winning.

Lemma 3. If $s$ converges and $c \not\in D_i \cup E_i$ for each $i \geq 0$, then $c$ is winning.

Below, we present a symbolic algorithm based on the scheme above. We shall work with constraints which we use as symbolic representations of sets of configurations.

Constraints. An $A$-constraint denotes a (potentially infinite) set $[\phi] \subseteq C_A$ of configurations. A $B$-constraint is defined in a similar manner. For constraints $\phi_1$ and $\phi_2$, we use $\phi_1 \subseteq \phi_2$ to denote that $[\phi_2] \subseteq [\phi_1]$. For a set $\psi$ of constraints, we use $[\psi]$ to denote $\bigcup_{\phi \in \psi} [\phi]$. For sets of constraints $\psi_1$ and $\psi_2$, we use $\psi_1 \subseteq \psi_2$ to denote that for each $\phi_2 \in \psi_2$ there is a $\phi_1 \in \psi_1$ with $\phi_1 \subseteq \phi_2$. Notice that $\psi_1 \subseteq \psi_2$ implies $[\psi_2] \subseteq [\psi_1]$. Sometimes, we identify constraints with their interpretations, so we write $c \in \phi$, $\phi_1 \subseteq \phi_2$, $\phi_1 \cap \phi_2$, $\neg \phi$, etc. We consider a particular class of $B$-constraints which we call upward closed constraints. An upward closed constraint is of the form $c \uparrow$, where $c \in C_B$, and has an interpretation $[c \uparrow] = \{c', c \preceq c'\}$.

A set $\Psi$ of $A$-constraints is said to be effective with respect to the game $G$ if
- The set $C_F$ is characterized by a finite set $\psi_F \subseteq \Psi$ (i.e. $[\psi_F] = C_F$).
- For a configuration $c \in C_A$ and a constraint $\phi \in \Psi$, we can decide whether $c \in [\phi]$.
- For each $\phi \in \Psi$, we can compute a finite set $\psi'$ of upward closed constraints such that $[\psi'] = \text{Pre}([\phi])$. In such a case we use $\text{Pre}(\phi)$ to denote the set $\psi'$. Notice that $\text{Pre}([\phi])$ is upward closed by Lemma 1. Also, observe that computability of $\text{Pre}(\phi)$ implies that, for a finite set $\psi \subseteq \Psi$, we can compute a finite set $\psi'$ of upward closed constraints such that $[\psi'] = \text{Pre}([\psi])$.
- For each finite set $\psi$ of upward closed constraints, we can compute a finite set $\psi' \subseteq \Psi$ such that $[\psi'] = \overline{\text{Pre}}([\psi])$. In such a case we use $\overline{\text{Pre}}(\psi)$ to denote the set $\psi'$.

The game $G$ is said to be effective if there is a set $\Psi$ of constraints which is effective with respect to $G$.

Symbolic Algorithm. Given a constraint system $\Psi$ which is effective with respect to the game $G$, we can solve the safety game problem by deriving a symbolic algorithm from the scheme described above. Each $D_i$ will be characterized by a finite set of constraints $\psi_i \in \Psi$, and each $E_i$ will be represented by a finite set of upward closed constraints $\psi'_i$. More precisely:

$$
\psi_0 = \psi_F \quad \psi'_0 = \text{Pre}(\psi_0)
$$

$$
\psi_{i+1} = \psi_i \cup \overline{\text{Pre}}(\psi'_i) \quad \psi'_{i+1} = \psi'_i \cup \text{Pre}(\psi_{i+1}) \quad i = 0, 1, 2, \ldots
$$

The algorithm terminates in case $\psi'_i \subseteq \psi'_{i+1}$. In such a case, a configuration $c$ is not winning if and only if $c \in [\psi_j] \cup [\psi'_j]$. This gives an effective procedure for deciding the safety game problem according to the following
– Each step can be performed due to effectiveness of $\Psi$ with respect to $G$.
– For a configuration $c \in C_A$ and a constraint $\phi \in \psi_i$, we can check $c \in [\phi]$ due to effectiveness of $\Psi$ with respect to $G$. For a configuration $c \in C_B$ and a constraint $\phi \in \psi'_i$, we can check $c \in [\phi]$ due to decidability of $\preceq$.
– For a configuration $c$ and an upward closed constraint $\phi = c' \uparrow$, we can check $c \in [\phi]$, since $\preceq$ is decidable and since $c \in [\phi]$ if and only if $c' \preceq c$.
– The termination condition can be checked due to decidability of $\preceq$ (which implies decidability of $\sqsubseteq$).
– Termination is guaranteed due to well quasi-ordering of $\preceq$ (which implies well quasi-ordering of $\sqsubseteq$).

From this we get the following

**Theorem 1.** The safety problem is decidable for the class of effective $B$-downward closed games.

5 B-LCS

In this section, we apply the symbolic algorithm presented in Section 4 to solve the safety game problem for $B$-LCS games: games between two players operating on a finite set of channels (unbounded FIFO buffers), where player $B$ is allowed to lose any number of messages after each move.

A $B$-lossy channel system ($B$-LCS) is a tuple $(S, S_A, S_B, L, M, T, S_F)$, where

- $S$ is a finite set of (control) states, $S_A, S_B$ is a partitioning of $S$, $L$ is a finite set of channels, $M$ is a finite message alphabet, $T$ is a finite set of transitions, and $S_F \subseteq S_A$ is the set of final states. Each transition in $T$ is a triple $(s_1, op, s_2)$, where
  - $s_1 \in S_A$ and $s_2 \in S_B$, or $s_1 \in S_B$ and $s_2 \in S_A$.
  - $op$ is of one of the forms: $\ell!m$ (sending message $m$ to channel $\ell$), or $\ell?m$ (receiving message $m$ from channel $\ell$), or $nop$ (not affecting the contents of the channels).

A B-LCS $\mathcal{L} = (S, S_A, S_B, L, M, T, S_F)$ induces a $B$-downward closed game $G = (C, C_A, C_B, \rightarrow, C_F, \preceq)$ as follows:

- **Configurations:** Each configuration $c \in C$ is a pair $(s, w)$, where $s \in S$, and $w$, called a channel state, is a mapping from $L$ to $M^*$. In other words, a configuration is defined by the control state and the contents of the channels. We partition the set $C$ into $C_A = \{(s, w) | s \in S_A \}$ and $C_B = \{(s, w) | s \in S_B \}$.
- **Final Configurations:** The set $C_F$ is defined to be $\{(s, w) | s \in S_F \}$.
- **Ordering:** For $x_1, x_2 \in M^*$, we use $x_1 \preceq x_2$ to denote that $x_1$ is a (not necessarily contiguous) substring of $x_2$. For channel states $w_1, w_2$, we use $w_1 \preceq w_2$ to denote that $w_1(\ell) \preceq w_2(\ell)$ for each $\ell \in L$. We use $(s_1, w_1) \preceq (s_2, w_2)$ to denote that both $s_1 = s_2$ and $w_1 \preceq w_2$. The ordering $\preceq$ is decidable and wqo (by Higamn’s Lemma [Hig52]).
– **Non-loss transitions:** \((s_1, w_1) \rightarrow (s_2, w_2)\) if one of the following conditions is satisfied
  - There is a transition in \(T\) of the form \((s_1, \ell!m, s_2)\), and \(w_2\) is the result of appending \(m\) to the end of \(w_1(\ell)\).
  - There is a transition in \(T\) of the form \((s_1, \ell?m, s_2)\), and \(w_1\) is the result of appending \(m\) to the head of \(w_2(\ell)\).
  - There is a transition in \(T\) of the form \((s_1, nop, s_2)\), and \(w_2 = w_1\).

– **Loss transitions:** If \(s_1 \in S_B\) and \((s_1, w_1) \rightarrow (s_2, w_2)\) according to one of the previous two rules then \((s_1, w_1) \rightarrow (s'_2, w'_2)\) for each \((s'_2, w'_2) \preceq (s_2, w_2)\).

**Remark.** To satisfy the condition that there are no deadlock states in games induced by \(B\)-LCS, we can always add two “winning” states \(s^*_1 \in S_A\), \(s^*_2 \in S_B\), and two “losing” states \(s^*_3 \in S_A\), \(s^*_4 \in S_B\), where \(s^*_3 \notin S_F\), and \(s^*_4 \notin S_F\). We add four transitions \((s^*_1, nop, s^*_2)\), \((s^*_2, nop, s^*_4)\), \((s^*_3, nop, s^*_4)\), and \((s^*_4, nop, s^*_3)\). Furthermore, we add transitions \((s, nop, s^*_1)\) for each \(s \in S_A\), and \((s, nop, s^*_1)\) for each \(s \in S_B\). Intuitively, if player \(A\) enters a configuration, where he has no other options, then he is forced to move to \(s^*_4\) losing the game. A similar reasoning holds for player \(B\).

We show decidability of the safety problem for \(B\)-LCS using Theorem 1. To do that we first describe upward closed constraints for \(B\)-LCS, and then introduce constraints which are effective with respect to \(B\)-LCS. We introduce upward closed constraints in several steps. First, we define upward closed constraints on words, and then generalize them to channel states and configurations. An **upward-closed constraint** over \(M\) is of the form \(X \uparrow\) where \(X \subseteq M^*\), and has an interpretation \([X \uparrow] = \{x | \exists x' \in X. x' \preceq x\}\). An upward closed constraint \(\phi\) over channel states is a mapping from \(L\) to upward closed constraints over \(M\), with an interpretation \([\phi]\) = \{w | \forall \ell \in L. w(\ell) \in ([\phi(\ell)])\}. We use \(w \uparrow\) to denote the upward closed constraint \(\phi\) over channel states where \(\phi(\ell) = w(\ell) \uparrow\) for each \(\ell \in L\). An upward closed constraint \(\phi\) (over configurations) is of the form \((s, \phi')\), where \(s \in S\) and \(\phi'\) is an upward closed constraint over channel states, with an interpretation \([\phi'] = \{(s, w) | w \in ([\phi'])\}\}. We use \((s, w) \uparrow\) to denote the upward closed constraint \((s, w \uparrow)\).

We introduce **extended upward-closed constraints** which we show to be effective with respect to \(B\)-LCS games. An **extended upward-closed constraint** over \(M\) is of the form \(x \bullet \phi\), where \(x \in M^*\) and \(\phi\) is an upward closed constraint over \(M\), and has an interpretation \([x \bullet \phi] = \{x \bullet x' | x' \in [\phi]\}\}. Extended upward closed constraints are generalized to channel states and configurations in a similar manner to above.

In the rest of this section we prove the following lemma

**Lemma 4.** Extended upward closed constraints are effective for \(B\)-LCS games.

From Theorem 1 and Lemma 4 we get the following

**Theorem 2.** The safety problem is decidable for \(B\)-LCS games.

We devote the rest of this section to the proof of Lemma 4. This is achieved as follows:
– The set $C_F$ is characterized by the (finite) set of constraints of the form $(s, \phi)$ where $s \in S_F$ and $\phi$ is an extended upward closed constraint over channels, where $\phi(\ell) = \epsilon \bullet \epsilon \uparrow$ for each $\ell \in L$ (notice that $[[\epsilon \bullet \epsilon \uparrow]] = M^*$).
– For a configuration $c \in C_A$ and an extended constraint $\phi$ we can check whether $c \in [[\phi]]$. (Lemma 5).
– For each extended upward closed constraint $\phi$, we can compute a finite set $\psi$ of upward closed constraints, such that $\psi = Pre(\phi)$ (Lemma 7).
– For each finite set $\psi_1$ of upward closed constraints, we can compute a finite set $\psi_2$ of extended upward closed constraints, such that $\psi_2 = \overline{Pre}(\psi_1)$. (Lemma 6).

For words $x_1, x_2$, we use $x_1 \cap x_2$ to denote the (finite) set of minimal (with respect to $\leq$) words $x_3$ such that $x_1 \leq x_3$ and $x_2 \leq x_3$.

**Lemma 5.** For a configuration $c \in C_A$ and an extended constraint $\phi$ we can check whether $c \in [[\phi]]$.

**Lemma 6.** For each finite set $\psi_1$ of upward closed constraints, we can compute a finite set $\psi_2$ of extended upward closed constraints, such that $\psi_2 = \overline{Pre}(\psi_1)$.

**Lemma 7.** For each extended upward closed constraint $\phi$, we can compute a finite set $\psi$ of upward closed constraints, such that $\psi = Pre(\phi)$.

**Remark.** Theorem 1 holds also in the case where both players can lose messages. In fact, we can show that negation constraints are effective with respect to such games.

## 6 A-Downward Closed Games

We present an algorithm for solving the safety problem for $A$-downward closed games. We use the algorithm to prove decidability of the safety problem for a variant of lossy channel games, namely $A$-LCS.

An $A$-downward closed game is said to be effective if for each configuration $c$ we can compute the set $Post(c)$. Observe that this implies that the game is finitely branching.

Suppose that we want to check whether a configuration $c_{init} \in C_A$ is winning. The algorithm builds an AND-OR tree, where each node of the tree is labelled with a configuration. OR-nodes are labelled with configurations in $C_A$, while AND-nodes are labelled with configurations in $C_B$.

We build the tree successively, starting from the root, which is labelled with $c_{init}$ (the root is therefore an OR-node). At each step we pick a leaf with label $c$ and perform one of the following operations:

– If $c \in C_F$ then we declare the node unsuccessful and close the node (we will not expand the tree further from the node).
– If $c \in C_A$, $c \not\in C_F$, and there is a predecessor of the node in the tree with label $c'$ where $c' \preceq c$ then we declare the node successful and close the node.
– Otherwise, we add a set of successors, each labelled with an element in $Post(c)$. This step is possible by the assumption that the game is effective.
The procedure terminates by König’s Lemma and by well quasi-ordering of \( \preceq \). The resulting tree is evaluated interpreting AND-nodes as conjunction, OR-nodes as disjunction, successful leaves as the constant true and unsuccessful leaves as the constant false. The algorithm answers “yes” if and only if the resulting tree evaluates positively.

**Theorem 3.** The safety problem is decidable for effective \( A \)-downward closed games.

\( A \)-LCS An \( A \)-LCS has the same syntax as a \( B \)-LCS. The game induced by an \( A \)-LCS has a similar behaviour to that induced by a \( B \)-LCS. The difference is that in the definition of the loss transitions:

- If \( s_1 \in S_A \) and \( (s_1, w_1) \rightarrow (s_2, w_2) \) according to a non-loss transition then \( (s_1, w_1) \rightarrow (s'_2, w'_2) \) for each \( (s'_2, w'_2) \preceq (s_2, w_2) \).

It is straightforward to check that a game induced by an \( A \)-LCS is \( A \)-downward closed and effective. This gives the following.

**Theorem 4.** The safety problem is decidable for \( A \)-LCS games.

Although the safety problem is decidable for \( A \)-LCS games, it is not possible to give a characterization of the set of winning configurations as we did for \( B \)-LCS. By a similar reasoning to Lemma 1, the set \( \text{Pre}(\mathbb{E}_i) \) is upward closed and therefore can be characterized by a finite set of upward closed constraints for each \( i \geq 0 \). In turn, the set \( \bigcup_{i \geq 0} D_i \) can be characterized by a finite set of negation constraints. We show that we cannot compute a finite set of negation constraints \( \psi \) such that \( \{\psi\} = \bigcup_{i \geq 0} D_i \), as follows.

We reduce an uncomputability result reported in [BM99] for transition systems induced by lossy channel systems. The results in [BM99] imply that we cannot characterize the set of configurations \( c \) satisfying the property \( c \models \exists \infty \Box \neg \mathbb{S}_F \), i.e., we cannot characterize the set of configurations from which there is an infinite computation which never visits a given set \( \mathbb{S}_F \) of control states. Given a lossy channel system \( \mathcal{L} \) (inducing a transition system) and a set \( \mathbb{S}_F \) of states, we derive an \( A \)-LCS \( \mathcal{L}' \) (inducing an \( A \)-downward closed game). For each configuration \( c \) in \( \mathcal{L} \), it is the case that \( c \models \exists \infty \Box \neg \mathbb{S}_F \) if and only if the configuration corresponding to \( c \) is winning in the game induced by \( \mathcal{L}' \). Intuitively, player \( A \) simulates the transitions of \( \mathcal{L} \), while player \( B \) follows passively. More precisely, each state \( s \) in \( \mathcal{L} \) has a copy \( s \in C_A \) in \( \mathcal{L}' \). For each transition \( t = (s_1, op, s_2) \) in \( \mathcal{L} \), there is a corresponding “intermediate state” \( s_t \in C_B \) and two corresponding transitions \( (s_1, op, s_t) \) and \( (s_t, \text{nop}, s_2) \) in \( \mathcal{L}' \). Furthermore, we have two state \( s^*_1 \in C_A \) and \( s^*_2 \in C_B \) which are losing (defined in a similar manner to Section 5). Each configuration in \( C_A \) can perform a transition labelled with \( \text{nop} \) to \( s^*_2 \). It is straightforward to check that a configuration \( c \) is winning in \( \mathcal{L}' \) if and only if \( c \models \exists \infty \Box \neg \mathbb{F} \).

From this, we get the following:

**Theorem 5.** We cannot compute a finite set of negation constraints characterizing the set of non-winning configurations in an \( A \)-LCS (although such a set always exists).
7 Undecidability of Monotonic Games

We show that the decidability of the safety problem does not extend from downward closed games to monotonic games. We show undecidability of the problem for a particular class of monotonic games, namely VASS games. In the definition of VASS games below, both players are assumed to have monotonic behaviours. Obviously, this implies undecidability for $A$- and $B$-monotonic games.

In fact, it is sufficient to consider VASS with two dimensions (two variables). Let $N$ and $I$ denote the set of natural numbers and integers respectively.

**VASS Games** A (2-dimensional) VASS (Vector Addition System with States) game $V$ is a tuple $(S, S_A, S_B, T, S_F)$, where $S$ is a finite set of (control) states, $S_A, S_B$ is a partitioning of $S$, $T$ is a finite set of transitions, and $S_F \subseteq S$ is the set of final states. Each transition is a triple $(s_1, (a, b), s_2)$, where

- either $s_1 \in S_A$ and $s_2 \in S_B$, or $s_1 \in S_B$ and $s_2 \in S_A$.
- $a, b \in I$. The pair $(a, b)$ represents the change made to values of the variables during the transition.

A VASS $V = (S, S_A, S_B, T, S_F)$ induces a monotonic game $G = (C, C_A, C_B, \rightarrow, C_F, \preceq)$ as follows:

- Each configuration $c \in C$ is a triple $(s, x, y)$, where $s \in S$ and $x, y \in N$. In other words, a configuration is defined by the state and the values assigned to the variables.
- $C_A = \{(s, x, y) \mid s \in S_A\}$.
- $C_B = \{(s, x, y) \mid s \in S_B\}$.
- $(s_1, x_1, y_1) \rightarrow (s_2, x_2, y_2)$ iff $(s_1, (a, b), s_2) \in T$, and $x_2 = x_1 + a$, and $y_2 = y_1 + b$. Observe that since $x_2, y_2 \in N$, we implicitly require $x_2 \geq 0$ and $y_2 \geq 0$; otherwise the transition is blocked.
- $C_F = \{(s, x, y) \mid s \in S_F\}$.
- $(s_1, x_1, y_1) \preceq (s_2, x_2, y_2)$ iff $s_1 = s_2$, $x_1 \leq x_2$, and $y_1 \leq y_2$.

We can avoid deadlock in VASS games in a similar manner to Section 5.

**Theorem 6.** The safety problem is undecidable for VASS games.

Undecidability is shown through a reduction from an undecidable problem for 2-counter machines.

**2-Counter Machines.** A 2-counter machine $M$ is a tuple $(S_M, T_M)$, where $S_M$ is a finite set of states, and $T_M$ is a finite set of transitions. Each transition is a triple of the form $(s_1, (a, b), s_2)$, or $(s_1, x = 0?, s_2)$, or $(s_1, y = 0?, s_2)$, where $s_1, s_2 \in S_M$.

A configuration of $M$ is a triple $(s, x, y)$ where $s \in S_M$ and $x, y \in N$. We define a transition relation $\rightarrow$ on configurations such that $(s_1, x_1, y_1) \rightarrow (s_2, x_2, y_2)$ iff either

- $(s_1, (a, b), s_2) \in T_M$, and $x_2 = x_1 + a$, and $y_2 = y_1 + b$; or
(s_1, x = 0?, s_2) \in T_M \text{ and } x_2 = x_1 = 0, \text{ and } y_2 = y_1; \text{ or }
(s_1, y = 0?, s_2) \in T_M \text{ and } x_2 = x_1, \text{ and } y_2 = y_1 = 0.

The 2-counter reachability problem is defined as follows

2-counter reachability problem

Instance. A 2-counter machine M = (S_M, T_M) and two states s_{init}, s_f \in S_M.

Question. Is there a sequence (s_0, x_0, y_0) \rightarrow (s_1, x_1, y_1) \rightarrow (s_2, x_2, y_2) \rightarrow \cdots \rightarrow (s_n, x_n, y_n)

of transitions such that s_0 = s_{init}, x_0 = 0, y_0 = 0, and s_n = s_f?

It is well-known that the 2-counter reachability problem is undecidable. In
the following, we show how to reduce the 2-counter reachability problem to the
safety problem for VASS games. Given a 2-counter machine M = (S_M, T_M) and
two states s_{init}, s_f \in S_M, we construct a corresponding VASS game, such that
the reachability problem has a positive answer if and only if the game problem
has a negative answer. Intuitively, player B emulates the moves of the 2-counter
machine, while player A is passive. Tests for equality with 0 cannot be emulated
directly by a VASS system. This means that player B could try to make moves
not corresponding to an actual move of the 2-counter machine. However, if player
B tries to "cheat", i.e. make a forbidden move, then we allow player A to go
into a winning escape loop. This means that player B always chooses to make
legal moves. Furthermore, we add an escape loop accessible when the system has
reached the final state. This loop is winning for player B. Thus, player B wins
whenever the final state is reachable. More formally, we define the VASS game
\mathcal{V} = (S, S_A, S_B, T, S_F) as follows:

- S_A = \{s_t^A \mid t \in T_M\} \cup \{s^{A, reached}_t, s^{A, init}_t\}. In other words, for each transition t \in T_M there is a state s_t^A \in S_A. We also add three special states
  s^{A, reached}_t \text{ and } s^{A, init}_t \text{ to } S_A.
- S_B = \{s^B_s \mid s \in S_M\} \cup \{s^B, s^{B, init}\}. In other words, for each state in s \in S_M there
  is a corresponding state s^B \in S_B. We also add a special state s^B \text{ to } S_B.
- For each transition t of the form (s_1, (a, b), s_2) \in T_M, there are two transitions in T, namely (s^B_1, (a, b), s^A_t) \text{ and } (s^A_1, (0, 0), s^B_2). Player B chooses a
  move, and player A follows passively.
- For each transition t of the form (s_1, x = 0?, s_2) \in T_M, there are three transitions in T, namely (s^B_1, (0, 0), s^A_t), (s^A_1, (0, 0), s^B_2), \text{ and } (s^A_1, (-1, 0), s^B_t). Player B may cheat here. However, if this is the case, player A will be allowed
to move to s^B_t, which is winning.
- Transitions of the form (s_1, y = 0?, s_2) \in T_M are handled in a similar manner
to the previous case.
- There are five additional transitions in T, namely an initializing transition
  (s^{init, A}_1, (0, 0), s^{init, B}_1); an escape loop to detect that the final state has been
  reached \(s^B_f, (0, 0), s^{A, reached}_1\) \text{ and } \(s^{A, reached}_f, (0, 0), s^B_f\); a loop to detect il-
  legal moves \(s^B_f, (0, 0), s^A_1\) \text{ and } \(s^A_1, (0, 0), s^B_1\).
- S_F = \{s^{A, reached}\}. 

Let \( G = (C, C_A, C_B, \rightarrow, C_F, \preceq) \) be the monotonic game induced by \( V \). We show that there is a sequence 
\[
(s_0, x_0, y_0) \rightarrow (s_1, x_1, y_1) \rightarrow (s_2, x_2, y_2) \rightarrow \cdots \rightarrow (s_n, x_n, y_n)
\]
of transitions in \( M \) with \( s_0 = s_{\text{init}}, x_0 = 0, y_0 = 0, \) and \( s_n = s_f \) iff the configuration 
\( (s_{\text{init}}^A, 0, 0) \) is not winning in \( G \).

## 8 Parity Games

A parity game \( G \) of degree \( n \) is a tuple \( (C, C_A, C_B, \rightarrow, r) \) where \( C, C_A, C_B, \rightarrow \) are defined as in games (Section 2), and \( r \) is a mapping from \( C \) to the set \( \{0, \ldots, n\} \) of natural numbers. We use \( C^k \) to denote \( \{c \mid r(c) = k\} \). The sets \( C_A^k \) and \( C_B^k \) are defined in a similar manner. We call \( r(c) \) the rank of \( c \). Abusing notation, we define the rank \( r(P) \) of a play \( P = c_0, c_1, c_2, \ldots \) to be \( \min\{r(c_0), r(c_1), r(c_2), \ldots\} \). We say that \( P \) is parity winning if \( r(P) \) is even. We say that \( c \) is parity winning if there is an \( A \)-strategy \( \sigma_A \) such that, for each \( B \)-strategy \( \sigma_B \), it is the case that \( P(c, \sigma_A, \sigma_B) \) is parity winning.

### The parity problem

**Instance.** A parity game \( G \) and a configuration \( c \) in \( G \).

**Question.** Is \( c \) (parity) winning?

**Remark.** Notice that our definition of parity games considers parity of configurations which appear in the play, rather than the configurations which appear infinitely often (which is the standard definition). Our undecidability result can be extended for the latter case, too.

We show below that the parity problem is undecidable for \( A \)-downward closed games. In particular, we show undecidability of the problem for \( A \)-LCS games. The proof for \( B \)-downward closed games is similar.

**Theorem 7.** The parity problem is undecidable for \( A \)-LCS games.

In [AJ96a] we show undecidability of the recurrent state problem, for transition systems based on lossy channel systems.

### Recurrent State Problem

**Instance.** A lossy channel systems \( L \) and a control states \( s_{\text{init}} \).

**Question.** Is there a channel state \( w \) such that there is an infinite computation starting from \( (s_{\text{init}}, w) \)?

We reduce the recurrent state problem for LCS to the parity problem for \( A \)-LCS. We construct a new \( L' \) to simulate \( L \). Intuitively, we let player \( A \) choose the moves of the original system, while player \( B \) follows passively. An additional loop at the beginning of \( L' \) allows us to guess the initial contents \( w \) of the channels. If the system deadlocks, then player \( B \) wins. So the only way for player \( A \) to win is to make the system follow an infinite sequence of moves. More formally, \( L' = (S, S_A, S_B, L, M, T, S_F) \) is defined as follows. For each control state \( s \) in \( L \), we create a control state \( s_A^A \in S_A \). For each transition \( t \) in \( L \), we create a control state \( s_B^t \in S_B \). For each transition \( t = (s_1, op, s_2) \) in \( L \) there are two transitions
Deciding Monotonic Games

((s_i^A, op, s_j^B) and (s_k^B, nop, s_l^A)) in \(\mathcal{L}'\). Furthermore, there are five additional states \(s_1^*, s_2^*, s_3^*, s_4^*, s_5^* \in S_A, s_2^*, s_3^*, s_5^* \in S_B\), together with the following transitions:

- Two transitions \((s_1^*, \ell m, s_2^*)\) and \((s_2^*, \text{nop}, s_3^*)\) for each \(m \in M\) and \(\ell \in L\). These two allow to build up the initial channel contents.
- Two transitions \((s_3^*, \text{nop}, s_2^*)\) and \((s_3^*, \text{nop}, s_1^*)\). This is to get to the initial state of \(\mathcal{L}\) when the channel content is ready.
- A transition \((s^A, \text{nop}, s^*\text{A})\) for each control state \(s\) in \(\mathcal{L}\). This transition is only taken when \(\mathcal{L}\) is deadlocked.
- Two transitions \((s_4^*, \text{nop}, s_5^*)\) and \((s_5^*, \text{nop}, s_4^*)\). This loop indicates a deadlock in \(\mathcal{L}\).

The ranks of the configurations are defined as follows:

- \(r((s_1^*, w)) = r((s_2^*, w)) = r((s_3^*, w)) = 3\), for each \(w\).
- \(r((s_4^*, w)) = r((s_5^*, w)) = 2\), for each \(w\), each transition \(t\) in \(\mathcal{L}\), and each control state \(s\) in \(\mathcal{L}\).
- \(r((s_4^*, w)) = r((s_5^*, w)) = 1\), for each \(w\).

We show that \((s_1^*, \epsilon)\) is parity-winning if and only if there exists a \(w\) and an infinite sequence starting from \((s_{\text{init}}, w)\).

Remarks.

- In case both players can lose messages, we can show that the parity problem is decidable. The reason is that the best strategy for each player is to empty the channels after the next move. The problem can therefore be reduced into an equivalent problem over finite-state graphs.
- Using results in \cite{May00}, we can strengthen Theorem\[7\] showing undecidability for \(A\)-VASS (and \(B\)-VASS) games. Such games are special cases of the ones reported here where the message alphabet is of size one (each channel behaves as a lossy counter).

References


Concurrent Construction of Proof-Nets

Jean-Marc Andreoli\textsuperscript{1,2} and Laurent Mazaré\textsuperscript{1}

\textsuperscript{1} Xerox Research Centre Europe, Grenoble, France
\textsuperscript{2} Institut de Mathématiques de Luminy, France
Jean-Marc.Andreoli@xrce.xerox.com

Abstract. The functional paradigm of computation has been widely investigated and given a solid mathematical foundation, initiated with the Curry-Howard isomorphism, then elaborated and extended in multiple ways. However, this paradigm is inadequate to capture many useful programming intuitions, arising in particular in the development of applications integrating distributed, autonomous components. Indeed, in this context, non-determinism and true concurrency are the rule, whereas functional programming stresses determinism, and, although it allows some degree of concurrency, it is more as a “nice feature to have” rather than a primary assumption.

This paper is part of a program the ambition of which is to provide a logical foundation to a set of programming intuitions which, until now, have not been adequately accounted for. In particular, we are interested in the intuitions which lie behind the concept of transaction, a powerful and essential concept in distributed component-based application development. This concept is independent of the application domain and usually captured in an abstract form in middleware architectural layers.

We claim here that proof-construction, and more precisely proof-net construction in Linear Logic, offers the adequate basis for our purpose. We outline the relation, which is of the same nature as the Curry-Howard isomorphism, between transactional concepts and mechanisms on one hand, and proof-net construction on the other. Finally, we describe an algorithm which performs concurrent proof-net construction, where each expansion step is viewed as a transaction. Conflicts between such transactions are minimised using general topological properties of proof-nets, based on a variant of the notion of “domination tree”, introduced and proved here.

Keywords: Logical foundations of programming paradigms, Linear Logic, Proof-nets, Concurrency, Transactions.

1 Introduction

It has been recognised early on in the development of computer science that proof theory offers adequate concepts and tools to capture abstract computational mechanisms. Thus, the so-called Curry-Howard isomorphism establishes a direct correspondence between proof-theoretic notions such as Cut elimination and fundamental mechanisms of the functional programming paradigm, such as parameter passing and type-preserving evaluation. However, this paradigm is essentially characterised by its determinism: a well-typed computation may evolve in multiple ways, but always terminates and there
is a notion of “result” of the computation which is independent of the specific strategy chosen in the reductions (this is essentially the convergence property of typed lambda-calculus). This form of determinism, which is extremely intuitive when computation is viewed as pure (e.g. arithmetic) calculation, becomes a hindrance when computation is viewed as coordinated interactions between a set of autonomous components through their environment. And many computer programs are more naturally understood in this way rather than as pure calculation: operating systems, electronic brokers, workflow engines, monitoring tools, etc. These programs may involve deterministic calculations at some points in their execution, but, overall, they do not produce any final “result”, and constantly interact instead with their environment in a non fully deterministic way.

There is another computation paradigm, improperly called logic programming, which, especially in its concurrent flavour, stresses non-determinism and interaction rather than calculation. In proof-theoretic terms, it corresponds to a different intuition, namely proof-construction rather than proof-reduction. The mapping between computational and proof-theoretic concepts in the two paradigms can thus be summarised as follows:

<table>
<thead>
<tr>
<th>Computation</th>
<th>Proof theory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paradigm</td>
<td>“functional”</td>
</tr>
<tr>
<td>State</td>
<td>Proof</td>
</tr>
<tr>
<td></td>
<td>– possibly with Cuts</td>
</tr>
<tr>
<td></td>
<td>– without Proper axioms</td>
</tr>
<tr>
<td>Transformation</td>
<td>Proof reduction</td>
</tr>
<tr>
<td></td>
<td>ie. Cut elimination</td>
</tr>
<tr>
<td>Final state</td>
<td>Proof without Cut</td>
</tr>
<tr>
<td>Type</td>
<td>Formula</td>
</tr>
</tbody>
</table>

In this paper, we study the problem of the construction of a proof by a set of concurrent agents. The inference system considered here is the focusing bipolar sequent calculus of Linear Logic [1], which has been shown to be equivalent to that of full Linear Logic. However, to express concurrency in proof construction, a sequent system is not the adequate representation tool. Proof-nets, which offer a desquentialised representation of proofs, are more appropriate. The paradigm of proof-net construction by concurrent agents in the multiplicative, transitory fragment of bipolar Linear Logic has been presented in [2]. It is recalled in Section 2. The agents participating in the construction are bipoles from the “Universal program”, labelling the inferences of the sequent system.

Whereas it is quite easy to ensure that the structure collaboratively built by the bipole agents remains a proof-structure, it is not so easy to guarantee that it remains a proof-net. This requires checking a correctness criterion, which is a topological property of the structure that ensures that it can be sequentialised, and hence is a proof-net. A priori, checking the correctness criterion may lead each bipole agent to freeze an arbitrary large portion of the topology, resulting in conflicts between the agents, forcing them to take turn and, in the end, restoring the artificial sequentialisations of the sequent system that we had precisely tried to get rid of by turning to proof-nets. The challenge, addressed here, is therefore to identify precisely the portion of the topology that needs to be frozen by a candidate inference to ensure that it does not violate the correctness criterion, whatever the structure outside this portion. Only those inferences for which these portions overlap
(conflict) need to be sequentialised. In [2], a simplification procedure was presented, which, when applied to a restricted fragment of transitory multiplicative Linear Logic (basically modelling multiset rewriting), completely solves the concurrency problem: in fact, in this fragment, no correctness conflict ever occurs. Here, we address the problem in the whole fragment of transitory multiplicative Linear Logic, where conflicts may occur, but we show that they can be minimised, based on topological properties of proof-nets which are introduced and proved in Section 3. Note that we still restrict to transitory inferences (with at least one premiss), meaning that we ignore the problem of terminating agents (which, in fact, poses the same problem as the multiplicative unit in correctness checking). Section 4 defines an abstract implementation of a concurrent, collaborative proof-net construction mechanism using these properties.

The bipole agents act as typical infrastructure software (a.k.a. middleware). Indeed, they do not capture any application-specific behaviour (justifying their name of “Universal program”), but their role is to guarantee that some invariant is preserved, namely that the overall structure they build is a proof-net, i.e. could be sequentialised into a sequent proof. In that sense, they are closely analogous to transaction schedulers in distributed applications. Indeed, like bipole agents, transaction schedulers are not concerned with the application-level semantics of the actions they involve but only with their interdependencies, and, while bipole agents ensure that the inferences they perform could be sequentialised, transaction schedulers ensure that the transactions they enact could be serialised. Thus, proof-net construction by concurrent bipole agents provides a generic logical model of the behaviour of transaction schedulers, which are essential components of middleware infrastructures. The correspondence between transaction concepts, in particular the Atomicity and Isolation properties of “ACID” transactions, and proof-theoretic concepts is at the core of the present paper. It is claimed here that this correspondence is not just a coincidence, or a by-product of logic’s powerful expressivity, but reveals the deep computational nature of proof-construction just as the Curry-Howard correspondence does to proof-reduction. In particular, the ability to sequentialise, which characterises proof-nets, has an operational meaning in the paradigm of proof-net construction as a set of isolation constraints (the “I” of ACID transactions) among the agents that perform the construction. Not surprisingly, the correctness criterion for proof-nets is remarkably close to the traditional serialisability criterion of transaction schedulers (absence of cycle in the dependency graph of transactions). Hence, we have the following correspondence

<table>
<thead>
<tr>
<th>Transactions</th>
<th>Proof-Net construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialisability</td>
<td>Sequentialisability</td>
</tr>
<tr>
<td>Isolation</td>
<td>Correctness</td>
</tr>
</tbody>
</table>

2 Focussing Proof-Net Construction

We assume given an infinite set \( \mathcal{P} \) of places, and we define a link by a set of top places and a set of bottom places, together with a polarity (positive or negative). The sets of top and bottom places of a link must be disjoint; furthermore, a negative link must have exactly one bottom place, while a positive link must have at least one bottom place. A
Jean-Marc Andreoli and Laurent Mazaré

A (multiplicative) focussing proof-structure is a set $\pi$ of links satisfying the following conditions:

1. The sets of top (resp. bottom) places of any two links in $\pi$ are disjoint.
2. If two links in $\pi$ are adjacent, their polarities are opposite.

A place which is at the top of a link but not at the bottom of any link in a focussing proof-structure is called a top place of the structure. The bottom places of a structure are defined similarly, by permuting top/bottom. An example of focussing proof-structure is given in Figure 1. In the sequel, except when mentioned otherwise, we take proof-structure to mean focussing proof-structure.

Definition 2. A proof-structure $\pi$ is said to be bipolar if any place occurring at the top of some positive link in $\pi$ also occurs at the bottom of some negative link in $\pi$ and
vice-versa. Furthermore, \( \pi \) is said to be elementary if it is bipolar and contains exactly one positive link.

The example of Figure 1 is bipolar. Note that any bipolar proof-structure is the union of disjoint elementary proof-structures, and this decomposition is unique. Furthermore, each elementary proof-structure corresponds to a bipole.

It has been shown in [2] how each (possibly open) proof in the focussing bipolar sequent calculus can be mapped into a (focussing) proof-structure. This process is called desequentialisation, and is strictly analogous to the desequentialisation of sequent proofs in Linear Logic [3].

**Definition 3.** A proof-net is a proof-structure obtained by desequentialisation of a (possibly open) proof in the focussing bipolar sequent calculus.

A correctness criterion, characterising proof-nets among arbitrary proof-structures, is provided in [2] for transitory focussing proof-structures. It is an adaptation of the Danos-Regnier criterion for proof-modules [4], the focussing proof-structures manipulated here being, properly speaking, proof-modules.

**Definition 4.** A proof-structure is said to be transitory if all its positive links have at least one top place.

The criterion for a (transitory) proof-structure \( \pi \) is expressed as a property of the graph \( \mathcal{P}_\pi \) (ie. the restriction to \( \mathcal{P} \) of the relation “just-below” on links). We use the following notations. Let \( \mathcal{L} \) be the set of all links. For any \( x \in \mathcal{L} \), the expression \( x^+ \) (resp. \( x^- \)) means that \( x \) has a positive (resp. negative) polarity. For any binary graph \( \mathcal{R} \) over \( \mathcal{L} \), \( |\mathcal{R}| \) denotes the support set of \( \mathcal{R} \), ie. the set \( \bigcup_{x} x \mathcal{R} \{x, y\} \); also, \( \mathcal{R}^{op} \) denotes the reverse of \( \mathcal{R} \) and \( \mathcal{R}^{+} \) its transitive closure.

**Definition 5.** A trip \( \alpha \) is a non-empty binary relation on \( \mathcal{L} \) which is finite, connex and such that any element \( x \) in \( \mathcal{L} \) has at most one successor (written \( \alpha^+ (x) \) when it exists) and at most one predecessor (written \( \alpha^- (x) \) when it exists) by \( \alpha \). An element of \( \mathcal{L} \) is called a start-point, stop-point, middle-point of \( \alpha \) if it has, respectively, a successor but no predecessor, a predecessor but no successor, both a predecessor and a successor.

It is easy to show that for a given trip \( \alpha \), one and only one of the following conditions is true: (i) either \( \alpha \) has no start-point nor stop-point, in which case, \( \alpha \) is called a loop; (ii) or \( \alpha \) has a unique start-point and a unique stop-point, and they are distinct. Note that a loop can never be strictly contained in another loop.

**Definition 6.** Let \( \mathcal{R} \) be a binary graph over \( \mathcal{L} \).

- \( \mathcal{R} \) is said to be polarised if adjacent elements in \( \mathcal{R} \) are of opposite polarity:
  \( \forall x, y \in \mathcal{L} \ x \mathcal{R} y \Rightarrow (x^- \land y^+) \lor (x^+ \land y^-) \)
- \( \mathcal{R} \) is said to be bipolarised if it is finite, polarised, and negative elements have a unique predecessor:
  \( \forall x \in |\mathcal{R}| \ (x^- \Rightarrow \exists ! y \in \mathcal{L} \ y \mathcal{R} x) \)
- A trip \( \alpha \) is said to be over \( \mathcal{R} \) if \( \alpha \subset \mathcal{R} \cup \mathcal{R}^{op} \). A singularity for \( \mathcal{R} \) of a trip \( \alpha \) is a negative middle-point \( x \) of \( \alpha \) such that:
  \( (\alpha^- (x) \mathcal{R} x \mathcal{R} \alpha^+ (x)) \lor (\alpha^+ (x) \mathcal{R} x \mathcal{R} \alpha^- (x)) \)
R is said to be correct if any loop over R has at least one singularity for R.

The correctness criterion is defined for proof-structures as follows, and leads to the following result shown in [2].

**Definition 7.** A proof-structure \( \pi \) is said to be correct if the relation \( J_\pi \) (the restriction to \( \pi \) of the “just-below” relation on links) is correct.

**Theorem 1.** Any proof-net is a correct, bipolar proof-structure. Any correct transitory bipolar proof-structure is a proof-net.

## 3 Properties of the Focussing Proof-Nets

In this section, we analyse some properties of transitory focussing proof-nets which are used, in the next section, to ensure correctness preservation during the bottom-up construction of a shared proof-structure by multiple, concurrent agents performing only (transitory) bipole expansions. The demonstration of the main results (provided in the submitted version) are available from the authors.

### 3.1 Notations and Conventions

\( R \) denotes a bipolarised, correct, binary relation over \( L \). Being correct, \( R \) is obviously acyclic, and it is useful to visualise \( R \) as being oriented bottom-up. Unless specified otherwise, trips and singularities are all taken to be relative to \( R \). The elements of \( |R| \) are called points. A trip is said to be proper if it is not reduced to a loop with two points. A path is a trip that always go upward (while trips in general may go any direction). A point \( x \) is said to be below a point \( y \) if there is a path from \( x \) to \( y \). A point is called a root if there is no point below it. Note that, since \( R \) is bipolarised, roots are always positive. Let \( \alpha \) be a trip and \( x \) a point of \( |\alpha| \).

- If \( x \) is not a stop-point of \( \alpha \) then \( \alpha \) is said to exit \( x \) either upward when \( xR \check{\alpha} (x) \), or downward when \( \check{\alpha} (x)Rx \).
- If \( x \) is not a start-point of \( \alpha \), then \( \alpha \) is said to enter \( x \) either downward when \( xR \check{\alpha} (x), \) or upward when \( \check{\alpha} (x)Rx \).

### 3.2 The Domination Forest

We make use below of a notion of domination order which is different from the standard one [5]: whereas the standard notion is applicable to any kind of flow graph, our definition applies only to bipolarised and correct graphs. Our definition is formally similar to the standard one, but considers only singularity-free trips in the graphs, and the proofs of the main results are essentially different, as they exploit the specific properties of bipolarised correct graphs. Moreover, properly speaking, the domination order here is not a tree (as in the standard case) but a forest.

**Definition 8.** Let \( x, y \) be negative points. \( x \) is said to dominate \( y \) (notation \( x \leq y \)) if any singularity-free trip starting at a root and stopping upward at \( y \) visits \( x \) upward.
Note that for any negative point \( x \), there exists at least one singularity-free trip starting at a root and stopping upward at \( x \); any path from any root below \( x \) to \( x \) will do (a path is obviously a singularity-free trip); such paths always exist, since \( \mathcal{R} \) is acyclic and \( x \) cannot itself be a root since it is negative while the roots are positive.

**Proposition 1.** The relation \( \leq \) on negative points is a forest order.

We write \( x < y \) for \( x \leq y \land x \neq y \). Since \( \leq \) is a forest order, we have:

- Any set of negative points \( X \) which has a lower bound by \( \leq \) has a greatest lower bound, written \( \sqcap(X) \) and called the joint dominator of \( X \); as usual, \( \sqcap(\{x, y\}) \) is written \( x \sqcap y \).
- The set of predecessors by \( < \) of any negative point \( x \), if not empty, has a greatest element by \( \leq \), written \( \delta(x) \) and called the immediate dominator of \( x \).

Note that the knowledge of any one of \( \leq, <, \sqcap, \delta \) is sufficient to recover all the others.

**Theorem 2.** Let \( x, y \) be negative points, and let \( \alpha \) be a singularity-free trip starting downward at \( x \) and stopping upward at \( y \). Then any negative point visited by \( \alpha \) is strictly dominated by \( x \sqcap y \) (when defined), ie.

\[
\forall z \in |\alpha| \quad z^- \Rightarrow x \sqcap y < z
\]

### 3.3 Domination under Expansion

In a bipolar expansion step of the proof-net construction process, \( \mathcal{R} \) is turned into \( \mathcal{R}' = \mathcal{R} \cup (N \times \{p\}) \) where \( p \) is the (unique) positive link introduced by the bipole and \( N \) the set of already present negative links at the top places of which \( p \) is connected. In fact, \( \mathcal{R}' \) also contains the set of pairs \( (p, n) \) where \( n \) ranges over the negative links introduced by the bipole, but we will not consider them here as they do not affect the correctness of \( \mathcal{R}' \).

**Proposition 2.** Let \( N \subset |\mathcal{R}| \) be a set of negative elements, and \( p \in \mathcal{L} \setminus |\mathcal{R}| \) be a positive element. The graph \( \mathcal{R}' = \mathcal{R} \cup (N \times \{p\}) \) is bi-polarised. Furthermore, it is correct if and only if there is no singularity-free trip in \( \mathcal{R} \) starting downward at some \( x \in N \) and stopping upward at some \( y \in N \).

We now assume that the graph \( \mathcal{R}' = \mathcal{R} \cup (N \times \{p\}) \) is correct. The following result, expliciting the relation between \( \leq, \delta \) induced by \( \mathcal{R} \) and their counterparts \( \leq', \delta' \), induced by \( \mathcal{R}' \), underlies our concurrent proof-net construction algorithm.

**Theorem 3.** Let \( x \) be a negative point. Either \( \delta(x) \leq' x \) and \( \delta'(x) = \delta(x) \), or \( \neg \delta(x) \leq' x \) and \( \sqcap(N) \leq \delta'(x) < \delta(x) \).

### 4 A Proof-Net Construction Algorithm

#### 4.1 Information Containers

The proof-net construction mechanism considered here proceeds by expansion steps performed by concurrent agents, each applying a bipole to a set of places that match
its trigger. To ensure that the construction remains a proof-net, each agent must check, for each candidate expansion, that its application would not violate the criterion given by Theorem 1 (we assume here that the agents apply only transitory bipoles, so the condition of the Theorem is satisfied). Performing such a check amounts to some kind of traversal of the proof-net built so far: it is useful to visualise it in terms of a token moving from each link to adjacent links in the proof-net, gathering and checking information on the way. Hence, an expansion step is itself a transaction composed of a set of micro-steps, including the traversal of the net and, eventually, the installation of the successful candidate expansion. In each micro-step, the transaction retrieves and/or updates information about the proof-net. For example, it may retrieve information like: what are the links adjacent to link \( x \) in the proof-net? It may also update information, like: link \( x \) has a new adjacent link \( y \) in the proof-net (this happens when the expansion is actually performed, after a successful correctness check). Adjacency information is obviously crucial to correctness checking. We will see that other pieces of information are also relevant.

To be useful, the information retrieved in each micro-step of a transaction \( t \) needs to be protected against updates of that same information by concurrent transactions, until \( t \) is completed. For this purpose, we assume that the information is encapsulated into containers that keep track of which transactions access them. Whenever a transaction \( t \) accesses a container, it locks it, thus denying access to its content to any other transactions, until \( t \) either commits or aborts. We assume here a simple locking model which does not distinguish between access modes (read or write) of the operations. We assume that (i) whenever a transaction commits, it releases all the locks it has acquired; (ii) whenever a transaction aborts, all the updates made to the containers it has locked are cancelled before the locks are released; (iii) in order to avoid deadlocks, whenever a transaction is denied access to a container, the whole transaction is aborted (thus freeing all its locks) and retried from the start (this is a rather crude way of sequentialising conflicting transactions, used here for simplicity; any of the other traditional strategies, eg. based on arbitrary prioritisation of the transactions, is applicable here).

There is one container for each link in the proof-net, holding the adjacency information, ie. pointers to the containers of the links which are just-above and just-below in the proof-net. Note that, due to the way the construction proceeds, the information concerning the links just-below is never changed after creation of a link. The information concerning the links just-above is also never changed in the case of a positive link, whereas it may be changed by some further expansion of the proof-net in the case of a negative link. Hence, locking is only needed for negative links, and only when the information concerning the links just-above is accessed.

As an optimisation, two links need not be put in separate containers if it can be guaranteed that whenever a transaction locks one, it also eventually locks the other. Hence, containers may group more than one link. Grouping may be done by sessions as proposed in [2]. This optimisation could straightforwardly be incorporated in the sequel, but will be omitted here for simplification purpose.

Finally, the information attached to each place is also encapsulated into containers. Typically, a place container holds the type attached to that place (a negative atom, never changed) as well as pointers to the containers of the links of the proof-net at the top or
Table 1. A naive implementation of expansion checking

**Procedure** check$(T$: set of places$)$

Let $N = \emptyset$

For-each $a \in T$ Do:
   - Lock $a$; If $a$ is consumed Then: Abort
   - Add to $N$ the (negative) link at the top of which $a$ appears

For-each $x \in N$ Do: Call check-trip↓$(x,N,\emptyset)$

**Procedure** check-trip↓$(x$: negative link, $N,A$: set of negative links$)$

If $x \not\in A$ Then:
   - Let $y$ be the (positive) link just-below $x$
   - For-each (negative) link $x' \neq x$ just-above $y$ Do: Call check-trip↑$(x',N,A)$
   - For-each (negative) link $x'$ just-below $y$ Do: Call check-trip↓$(x',N,A)$

**Procedure** check-trip↑$(x$: negative link, $N,A$: set of negative links$)$

If $x \in N$ Then: Abort Else: Lock $x$

For-each (positive) link $y$ just-above $x$ Do:
   - For-each (negative) link $x'$ just-above $y$ Do: Call check-trip↑$(x',N,A \cup \{x\})$
   - For-each (negative) link $x' \neq x$ just-below $y$ Do: Call check-trip↓$(x',N,A \cup \{x\})$

bottom of which that place appears. The link at the top of which a place appears is never changed after creation of that place. On the other hand, the link at the bottom of which a place appears may be added dynamically, by an expansion step, but once it is set, it is never changed (the place is then said to be consumed). Place containers are used to ensure that the construction remains a proof-structure (a pre-requisite to correctness), simply by preventing that a place be consumed by several concurrent transactions.

To perform an expansion of the proof-net built so far from a given set of places, a bipole agent proceeds in two steps: (i) it first checks that the candidate expansion would not break the proof-structure nor violate the correctness criterion; (ii) it then installs the candidate expansion, creating and/or updating place and link containers with the appropriate information. The first phase reads information from the already existing containers, while the second phase updates the information in these containers, and creates new containers. These two phases are executed in a single transaction.

### 4.2 A Naive Procedure

A naive procedure for the first phase (preservation of proof-structure and correctness) is given in Table 1. It is a direct application Proposition 2 reformulated here as follows:

An expansion of a proof-net $\pi$ from a set of places $T$ (matching the trigger of a bipole) yields a proof-net if and only if (i) no place of $T$ is at the bottom of a link of $\pi$ and (ii) there exists no singularity-free trip over $\pi'$ starting downward at some link $x$ and stopping upward at some link $y$ such that both $x$, $y$ have a top place in $T$. 
Table 2. Computation of the joint dominator of a set of points

**Procedure** \(\text{J}\text{OINT-DOM}(N): \) set of negative links  
**Returns** negative link (or undefined)

1. **Sort** \(N\) by decreasing height
2. **While** \(N\) has more than 1 element:
   - **Remove** the first point \(x\) from \(N\)
   - **If** \(\delta(x)\) is undefined **Then** return undefined **Else** lock \(x\)
   - **If** \(\delta(x) \notin N\) **Then** insert \(\delta(x)\) in \(N\), so that it remains sorted
3. **Return** the single remaining element of \(N\)

Thus, procedure **check** of Table 1 first computes the set \(N\) of points having a top place in \(T\), checking at the same time condition (i) above. It then follows all the possible singularity-free trips starting downward at a link of \(N\). If one of them stops upward at another link of \(N\), then it means that the candidate expansion induces a violation of the correctness criterion and the transaction should be aborted. Otherwise, correctness is preserved and the transaction can proceed with the second phase (installation of the expansion) and be committed.

Following the singularity-free trips is achieved by the two procedures **check-trip**. Their first parameter \(x\) denotes the current negative link in the trip being built, and the direction of the arrow (↑ or ↓) denotes the direction of traversal of that link (upward or downward). The second parameter \(N\) is the set of links which must not be reached upward. The third parameter \(A\) denotes the set of negative links already visited upward by the trip, and is used to ensure that the trip does not visit them again downward (this is sufficient to ensure that no link is ever visited twice). Locking is only needed in the upward visit phases, since these are the only phases which use information which may be modified by concurrent transactions (namely the information concerning the links just-above).

However, the systematic exploration of all singularity-free trips starting at \(N\) amounts to locking a large portion of the proof-net, and decreases the potential concurrency of the transactions. We use below the results of Section 3 to reduce the amount of exploration required by each candidate expansion and thus improve the potential for concurrency in the construction. Basically, we enrich the containers with additional information about the proof-net (beyond bare adjacency), and use it to limit the scope of the exploration of singularity-free trips. The constraint here is two-fold: (i) the additional information should enable to compute a practical limit to the exploration (and locking); (ii) the update of the additional information in case of success should not itself require new explorations (and locking).

### 4.3 An Improved Procedure

Given a candidate expansion at a set \(T\) of places, the procedure of Table 1 computes all the singularity-free trips starting downward at a point \(x\) of \(N\), to check that none of them stops upward at another point \(y\) of \(N\) (where \(N\) is the set of links of \(\pi\) with a top place in \(T\)). Now, by Theorem 2 any negative point \(z\) of such a trip satisfies \(x \sqcap y < z\), hence \(\sqcap(N) < z\). Thus, in the procedure of Table 1 the exploration of the singularity-free trips
Table 3. An improved implementation of expansion checking

Procedure check\((T):\) set of places):
\[
\begin{array}{l}
\text{Let } N = \emptyset \\
\text{For-each } a \in T \text{ Do:} \\
\quad \text{Lock } a; \text{ If } a \text{ is consumed Then: Abort} \\
\quad \text{Add to } N \text{ the (negative) link at the top of which } a \text{ appears} \\
\quad \text{Let } \xi = \text{joint-dom}(N) \\
\quad \text{If } \xi \text{ is defined Then: Let } A = \{\xi\} \text{ Else: Let } A = \emptyset \\
\quad \text{Let } U = \emptyset \\
\quad \text{For-each } x \in N \text{ Do: Add check-trip}^\downarrow(x,N,A) \text{ to } U \\
\quad \text{For-each } x \in U \text{ Do: Set } \delta(x) = \text{joint-dom}\{\delta(x), \xi\} \\
\end{array}
\]

Procedure check-trip\(^\downarrow\)(\(x\): negative link, \(N, A\): set of negative links)
\[
\begin{array}{l}
\text{Returns set of negative links} \\
\text{Let } U = \emptyset \\
\text{If } x \not\in A \text{ Then:} \\
\quad \text{Let } y \text{ be the (positive) link just-below } x \\
\quad \text{For-each (negative) link } x' \neq x \text{ just-above } y \text{ Do: Add check-trip}^\uparrow(x',N,A) \text{ to } U \\
\quad \text{For-each (negative) link } x' \text{ just-below } y \text{ Do: Add check-trip}^\downarrow(x',N,A) \text{ to } U \\
\text{Return } U \\
\end{array}
\]

Procedure check-trip\(^\uparrow\)(\(x\): negative link, \(N, A\): set of negative links)
\[
\begin{array}{l}
\text{Returns set of negative links} \\
\text{If } x \in N \text{ Then: Abort Else: Lock } x \\
\text{Let } U = \{x\} \\
\text{For-each (positive) link } y \text{ just-above } x \text{ Do:} \\
\quad \text{For-each (negative) link } x' \text{ just-above } y \text{ Do: Add check-trip}^\downarrow(x',N,A \cup \{x\}) \text{ to } U \\
\quad \text{For-each (negative) link } x' \neq x \text{ just-below } y \text{ Do: Add check-trip}^\uparrow(x',N,A \cup \{x\}) \text{ to } U \\
\text{Return } U \\
\end{array}
\]

in the current proof-net can be limited by \(\sqcap(N)\) (when it exists). The computation of \(\sqcap(N)\) can itself be obtained using only mapping \(\delta\) (the immediate dominator mapping). Thus, the additional information we propose to store in each container for a (negative) link \(x\) is a pointer to the container of \(\delta(x)\) (when it exists, otherwise, some null pointer).

Note that each expansion step may update the immediate dominator information in several containers, and it needs to be locked when used in a transaction to compute \(\sqcap(N)\), as shown in the procedure of Table 2. This procedure also makes use of a height information, included in the containers. This information is a positive integer computed each time a container is created (and never modified afterwards) so as to respect the following condition: the height of a container is strictly greater than the height of any of the containers which are just-below it (the set of which is known at the time of the creation of the container and never changes).

Now, the joint dominator \(\sqcap(N)\) sets a good limit on the exploration of the singularity-free trips starting downward at \(N\). However, there is a price to pay: the immediate dominator information from which the joint dominator is computed has to be maintained. In particular, when a candidate expansion is successfully installed, the immediate dominator information stored in an a priori unknown number of containers may have to be
updated. Fortunately, Theorem 3 shows that the area which has to be considered for the update is itself limited by \( \cap(N) \). More precisely, there is no need to consider for update the containers other than those which have been visited, and locked. The procedure of Table 3 modifies the naive one to compute the set of containers which need to be considered for update (\( U \) in procedure \texttt{check}).

It is not obvious to compute exactly the updated value of \( \delta(x) \) at each container \( x \) in \( U \). Instead, we use Theorem 3 to assign a lower bound of this value: \( \delta(x) \cap \cap(N) \). This means that the actual algorithm does not maintain the exact value of the immediate dominator at each negative point, but a lower bound. Hence, the computation of the joint dominator may return a lower bound of the true value. This may result in wider explorations than strictly needed, but does not compromise the validity of the algorithm. The figure below illustrates the difference between the naive and improved procedure on a simple example (black/white nodes are positive/negative links):

![Diagram illustrating the difference between the naive and improved procedure on a simple example](image)

### 5 Related Work

The standard notion of domination tree has been used in the past for correctness checking of proof-nets [5]. It is not clear whether this work, which is not concerned with proof-net construction, has any relationship with the work presented here, since our notion of domination does not coincide with the standard one. The actual properties proved in each case are different and support entirely different uses of proof-nets.

Proof-search in the sequent system of Linear Logic has been widely investigated. Proof-net search by a sequential process has also been studied in [6]. Our approach addresses concurrency in proof-net construction as an essential feature.

Petri-nets [7], introduced to study concurrency of processes competing for bounded resources, are the closest formal tool to our proposal. It was remarked early in the development of Linear Logic that proof-nets have strong connections with Petri-nets [8]. There are major differences though.

- In the proof-net construction paradigm, proof-nets are used as a mean of representing a trace of past actions (the construction so-far) in order to constrain future actions (by imposing transactional isolation constraints). Petri-nets, on the other hand, are used as a specification tool for concurrent programs. In particular, as a trace, a proof-net (or even a proof-structure) never contains cycles, while, as a specification, a Petri-net naturally allows cycles. In a middleware paradigm, the purpose is not to model what
should take place in an application (specification) but rather to capture what actually takes place (trace). Proof-nets are therefore more appropriate for our purpose. Note that, of course, proof-nets as used here can naturally represent the trace of Petri-net executions, with a straightforward mapping between the bipoles used in the proof-nets and the actual execution of transitions in the Petri-net. However, usual Petri-net transitions (multiset rewriting) lead to simple proof-nets which do not need the kind of correctness checking procedure presented here: in fact they fall in the fragment considered in [2] for which no correctness problem can occur.

– The difference in purpose between specification and trace, which, by the way, corresponds to a deep cultural gap in the computing community between application programmers looking at how to implement things and system managers trying rather to understand what is going on in a system, has important consequences on the subsequent use of the tools. Petri-nets have been extended in many ways, in order to enhance their expressivity, according to needs arising from applications or intuition. And indeed, extensions of Petri-nets have been proposed to capture various form of transactional and contextual features [9,10,11] at the specification level. On the other hand, proof-nets need to be kept as bare as possible and avoid ad-hoc extensions which would not fit in the abstract proof machinery (the multiplication of concepts would make them useless as a tool to understand what is going on). The present paper tries to provide an understanding of concurrency and transaction concepts only in terms of the basic operations provided by Linear Logic.

The same discussion applies to other tools used to specify concurrent systems (or analyse them based on such specifications), such as the many process calculi [12,13,14] (the join calculus, in particular, is directly related to the work presented here), or the various coordination models based on shared tuple spaces (which also have many relations, including historical ones, with the work presented here).

6 Conclusion

Proof-nets are a powerful tool to represent and understand concurrency in computations. In particular, proof-net construction by a set of concurrent, decentralised agents provides an abstract model of infrastructure software in distributed applications (a.k.a. middleware). More precisely, proof-net construction captures in a natural way the behaviour of an essential component of middleware infrastructures, namely transaction schedulers. The concept of transaction has been widely investigated in the literature, and addresses basic intuitions such as atomicity and isolation in concurrent actions.

In this paper, we claim that the true computational content of proof-construction is given in terms of atomicity and isolation constraints between a set of agents performing expansion steps in a proof-net. We give a naive algorithm ensuring that the shared structure being built remains a proof-net. The proposed algorithm is at the same time incremental and truly concurrent. It does not rely on any kind of “behind the scene” interaction between the agents in order to synchronise: the proof-net being built is the only piece of data shared between the agents.

However, the naive algorithm may result in unbounded explorations of the net at each expansion step. A fine-grain analysis of proof-net properties allows us to improve
the algorithm, by setting a computationally tractable bound to the required exploration. One of the main tools used in this analysis is the concept of “domination” order attached to a graph. This concept is not new, and has been used in various contexts involving the study of flow-graphs in general. We use a different version of this concept here, to derive important properties of proof-nets used in the concurrent, incremental construction algorithm.

References

We will speak about three traditions in Logic:

- **Classical**, usually associated with Frege, Hilbert, Gödel, Tarski, and others;
- **Intuitionistic**, founded by Brouwer, Heyting, Kolmogorov, Gödel, Kleene, and others;
- **Explicit**, which we trace back to Skolem, Curry, Gödel, Church, and others.

The classical tradition in logic based on quantifiers \( \forall \) and \( \exists \) essentially reflected the 19th century mathematician’s way of representing dependencies between entities. A sentence \( \forall x \exists y A(x, y) \), though specifying a certain relation between \( x \) and \( y \), did not mean that the latter is a function of the former, let alone a computable one. The Intuitionistic approach provided a principal shift toward the effective functional reading of the mathematician’s quantifiers. A new, non-Tarskian semantics had been suggested by Kleene: realizability that revealed a computational content of logical derivations. In a decent intuitionistic system, a proof of \( \forall x \exists y A(x, y) \) yields a program \( f \) that computes \( y = f(x) \).

Explicit tradition makes the ultimate step by using representative systems of functions instead of quantifiers from the very beginning. Since the work of Skolem, 1920, it has been known that the classical logic can be adequately recast in this way. Church in 1936 showed that even the very basic system of function definition and function application is capable of emulating any computable procedure. However, despite this impressive start, the explicit tradition remained a Cinderella of the mathematical logic for decades. Now things have changed: due to its very explicitness, this third tradition became the one most closely connected with Computer Science.

In this talk we will show how switching from quantifiers to explicit functional language helps problem solving in both theoretical logic and its applications. A discovery of a natural system of self-referential proof terms, *proof polynomials*, was essential in the solution to an open problem of Gödel concerning formalization of provability. Proof polynomials considerably extend the Curry-Howard isomorphism and lead to a joint calculus of propositions and proofs which unifies several previously unrelated areas. It changes our conception of the appropriate syntax and semantics for reasoning about knowledge, functional programming languages, formalized deduction and verification.
Constraint Satisfaction with Countable Homogeneous Templates

Manuel Bodirsky1,* and Jaroslav Nešetřil2,**

1 Humboldt Universität zu Berlin, Germany
bodirsky@informatik.hu-berlin.de
2 Institute for Theoretical Computer Science (ITI) and Department of Applied Mathematics (KAM), Charles University, Prague, Czech Republic
nesetril@kam.mff.cuni.cz

Abstract. For a fixed countable homogeneous relational structure $\Gamma$ we study the computational problem whether a given finite structure of the same signature homomorphically maps to $\Gamma$. This problem is known as the constraint satisfaction problem CSP($\Gamma$) for $\Gamma$ and was intensively studied for finite $\Gamma$. We show that - as in the case of finite $\Gamma$ - the computational complexity of CSP($\Gamma$) for countable homogeneous $\Gamma$ is determined by the clone of polymorphisms of $\Gamma$. To this end we prove the following theorem which is of independent interest: The primitive positive definable relations over an $\omega$-categorical structure $\Gamma$ are precisely the relations that are invariant under the polymorphisms of $\Gamma$. Constraint satisfaction with countable homogeneous templates is a proper generalization of constraint satisfaction with finite templates. If the age of $\Gamma$ is finitely axiomatizable, then CSP($\Gamma$) is in NP. If $\Gamma$ is a digraph we can use the classification of homogeneous digraphs by Cherlin to determine the complexity of CSP($\Gamma$).

1 Introduction

For a fixed relational structure $\Gamma$ (called the template), the constraint satisfaction problem CSP($\Gamma$) is the following computational problem: Given a finite structure $S$ of the same signature as $\Gamma$, is there a homomorphism from $S$ to $\Gamma$?

Constraint satisfaction problems frequently occur in theoretical computer science, and have attracted much attention for finite templates $\Gamma$. It is conjectured that CSP($\Gamma$) has a dichotomy in the sense that every constraint satisfaction problem CSP($\Gamma$) for finite structure $\Gamma$ is either tractable or NP-complete. This is true for templates that are undirected graphs [20], for two element templates [35] or three element templates [6]. It is known that every constraint satisfaction problem is polynomial time equivalent to a digraph-homomorphism problem [17]. There are powerful classes of algorithms solving the known tractable

* Member of the European Graduate Program “Combinatorics, Geometry and Computation”, supported by DFG grant GRK 588/2 and supported by the COMBSTRU network of the European Union.

** Supported by grant LN00A056 of the Czech Ministry of Education and supported by the COMBSTRU network of the European Union.
constraint satisfaction problems [17], namely group theoretic algorithms and local-consistency based algorithms [14][21].

But many constraint satisfaction problems in the literature can not be formulated as a constraint satisfaction problem with a finite template. One example is Allen’s interval algebra [1] that has applications in temporal reasoning in artificial intelligence. The classification of the tractable and hard subalgebras of Allen’s algebra was completed only recently [30][24], and they also exhibit a complexity dichotomy. Other examples are tree description languages that were introduced in computational linguistics [11][4][8]. Even digraph-acyclicity can not be formulated as a constraint satisfaction problem with finite template $\Gamma$. However, arbitrary infinite templates $\Gamma$ might have undecidable constraint satisfaction problems.

We propose to study constraint satisfaction with countable homogeneous templates. This can be seen as a strict generalization of constraint satisfaction with finite templates, since every constraint satisfaction problem with a finite template is polynomial-time equivalent to a constraint satisfaction problem with a homogeneous template (see Section 3). Moreover, the constraint satisfaction problems mentioned above can be formulated naturally in this new framework. To prove tractability or hardness of constraint satisfaction problems with homogeneous templates reductions to different hard problems and new algorithms are used, which have not yet been considered for CSP($\Gamma$) with finite $\Gamma$. Countable homogeneous structures are intensively studied by model theorists, and they have many remarkable properties. For finite signatures they allow quantifier elimination and are $\omega$-categorical, i.e. their first-order theories have only one countable model up to isomorphism. Countable homogeneous structures have been classified for all digraphs [10].

Adding relations to a template $\Gamma$ that are primitive positive definable over $\Gamma$ does not change the computational complexity of CSP($\Gamma$). The central theorem here [3] is that a relation is primitive positive definable over a finite relational structure $\Gamma$ if and only if it is left invariant under the polymorphisms of $\Gamma$. This was first used in the context of constraint satisfaction by Jeavons et al. [23], and initiated the algebraic approach to constraint satisfaction, which has successfully been carried further e.g. in [13]. We generalize this result to $\omega$-categorical structures $\Gamma$: A relation is p.p.-definable in $\Gamma$ if and only if it is invariant under the polymorphisms of $\Gamma$.

We can determine the complexity of CSP($\Gamma$) and prove a dichotomy if $\Gamma$ is a homogeneous graph or a tournament. Since there are uncountably many countable homogeneous digraphs $\Gamma$ and uncountably many corresponding constraint satisfaction problems, the class of problems CSP($\Gamma$) contains undecidable problems. However, if we assume that the class of finite induced substructures of a countable homogeneous digraph $\Gamma$ is finitely axiomatized, one can determine the complexity of CSP($\Gamma$) with a classification result of Cherlin [10].

The paper is organized as follows. We first give some background on relational homogeneous structures. In the next section on combinatorial constraint satisfaction problems we explain the rôle of primitive positive definability in con-
We give a characterization of primitive positive definability on homogeneous structures in Section 5 after introducing the necessary tools from universal algebra in Section 4. We end with a catalog of homogeneous relational structures and a discussion of their constraint satisfaction problems.

2 Background

A relational signature \( \tau \) is a (in this paper always at most countable) set of relation symbols \( R_i \), each associated with an arity \( k_i \). A (relational) structure \( \Gamma \) over relational signature \( \tau \) is also called \( \tau \)-structure is a set \( D_\Gamma \) (the domain) together with a relation \( R_i \subseteq D_\Gamma^{k_i} \) for each relation symbol of arity \( k_i \). For simplicity we denote both a relation symbol and its corresponding relation with the same symbol. For a \( \tau \)-structure \( \Gamma \) and \( R \in \tau \) it will also be convenient to say that \( R(u_1, \ldots, u_k) \) holds in \( \Gamma \) if \( (u_1, \ldots, u_k) \in R \). We sometimes use the shortened notation \( \overline{x} \) for a vector \( x_1, \ldots, x_n \) of any length.

A first-order formula \( \varphi \) over the signature \( \tau \) is said to be primitive positive (we say \( \varphi \) is a p.p.-formula, for short) if it is of the form

\[
\exists \overline{x}(\varphi_1(\overline{x}) \wedge \cdots \wedge \varphi_k(\overline{x})).
\]

where \( \varphi_1, \ldots, \varphi_k \) are atomic formulas. For an introduction to first order logic and model theory see [22].

A relational structure \( \Gamma \) is called homogeneous (in the literature also ultra-homogeneous) if every partial isomorphism between two finite substructures can be extended to an automorphism of \( \Gamma \). Prominent examples of countable homogeneous structures are the Rado graph \( R \) and the dense linear order \( (\mathbb{Q}, <) \).

The Rado graph can be defined as the unique (up to isomorphism) model of the almost-sure theory of finite random graphs. Homogeneous structures have been classified for graphs [27], for tournaments, for posets [36], and finally digraphs [10] (there are continuum many homogeneous digraphs). For homogeneous structures with arbitrary relational signatures a classification is not yet known.

The age \( \text{Sub}(\Gamma) \) of a relational structure \( \Gamma \) over \( \tau \) is the set of all finite structures over \( \tau \) that (isomorphically) embed in \( \Gamma \). An important property of countable homogeneous structures is their characterization by amalgamation classes. A class of finite structures \( \mathcal{C} \) is an amalgamation class if \( \mathcal{C} \) is nonempty,
closed under isomorphism and taking induced substructures, and has the amalgamation property. The amalgamation property says that for all \( A, B_1, B_2 \in \mathcal{C} \) and embeddings \( e : A \to B_1 \) and \( f : A \to B_2 \) there exists \( C \in \mathcal{C} \) and embeddings \( g : B_1 \to C \) and \( h : B_2 \to C \) such that \( ge = hf \).

**Theorem 1 (Fraïssé [18]).** A countable class \( \mathcal{C} \) of finite relational structures with countable signature is the age of a unique (up to isomorphism) countable homogeneous structure if and only if \( \mathcal{C} \) is an amalgamation class.

If \( \mathcal{C} \) is an amalgamation class, we call the corresponding countable homogeneous structure the Fraïssé-limit of \( \mathcal{C} \). By definition amalgamation classes can be defined by a set of forbidden induced finite substructures. For a set of finite structures \( \mathcal{N} \) over \( \tau \) we denote by \( \text{Forb}(\mathcal{N}) \) the set of finite structures \( S \) over \( \tau \) such that no structure in \( \mathcal{N} \) is embeddable in \( S \). We say that a class of finite structures \( \mathcal{C} \) over \( \tau \) is finitely axiomatizable if there exists a first order formula \( \varphi \) over \( \tau \) such that for all \( \tau \)-structures \( A \) we have \( A \in \mathcal{C} \) if and only if \( A \) is a model of \( \varphi \). By compactness it follows that an amalgamation class \( \mathcal{C} \) is finitely axiomatizable if and only if \( \mathcal{C} = \text{Forb}(\mathcal{N}) \) for some finite set of forbidden induced substructures \( \mathcal{N} \).

A homogeneous structure \( \Gamma \) over a finite signature is \( \omega \)-categorical, i.e. every countable structure satisfying the same first order formulas as \( \Gamma \) is isomorphic to \( \Gamma \). A relational structure is called oligomorphic iff the automorphism group \( \text{Aut}(G) \) of the structure \( \Gamma \) has only a finite number of orbits on the set of \( n \)-tuples of elements of \( \Gamma \). The following theorem is essential (see [22]):

**Theorem 2 (Engeler, Ryll-Nardzewski, Svenonius).** A countable structure \( \Gamma \) is \( \omega \)-categorical if and only if \( \text{Aut}(\Gamma) \) is oligomorphic.

### 3 Combinatorial Constraint Satisfaction

Let \( \Gamma \) be an arbitrary structure with relational signature \( \tau \) - also called the template. Then the constraint satisfaction problem \( \text{CSP}(\Gamma) \) is the following computational problem:

**Given:** A finite \( \tau \)-structure \( S \).

**Question:** Is there some homomorphism from \( S \) to \( \Gamma \)?

Formally, we denote by \( \text{CSP}(\Gamma) \) the set of all finite \( \tau \)-structures that homomorphically map to \( \Gamma \). For finite \( \Gamma \) we can assume without loss of generality that \( \Gamma \) is a core, i.e. all endomorphisms of \( \Gamma \) are automorphisms. If \( \Gamma \) is a core, adding all the singleton relations to \( \Gamma \) does not change the complexity of \( \text{CSP}(\Gamma) \) (as stated in [7]). In this case \( \Gamma \) becomes a homogeneous relational structure. Therefore constraint satisfaction with homogeneous templates can be seen as a generalization of constraint satisfaction with finite templates.

All constraint satisfaction problems with finite \( \Gamma \) are clearly contained in NP. If the age of a relational homogeneous structure \( \Gamma \) of finite signature is finitely
axiomatizable then CSP(\(\Gamma\)) is also contained in NP. To see this, suppose we are given an instance \(S\) of CSP(\(\Gamma\)). An algorithm can then guess the image of \(S\) under a homomorphism, and verify that the image belongs to the age of \(\Gamma\) in polynomial time using the finite axiomatization. Thus we have

**Proposition 1.** Let \(\Gamma\) be a countable homogeneous relational structure of finite signature \(\tau\) with a finitely axiomatizable age. Then CSP(\(\Gamma\)) is in NP.

Note that we need the axiomatizability assumption in Proposition 1 as there exist homogeneous \(\Gamma\) such that CSP(\(\Gamma\)) is undecidable, see Section 6. In analogy with the dichotomy conjecture of Feder and Vardi [17], we can make the following conjecture.

**Conjecture 1 (Dichotomy).** Let \(\Gamma\) be a countable homogeneous relational structures with a finitely axiomatizable age. Then the class of constraint satisfaction problems CSP(\(\Gamma\)) has a dichotomy.

For both finite and infinite \(\Gamma\), the following simple lemma explains the relevance of p.p.-definable relations in constraint satisfaction. Suppose we extend a relational structure \(\Gamma\) by a p.p.-definable relation \(R\). This does not change the computational complexity of the corresponding constraint satisfaction problem, since we can replace every occurrence of \(R\) in an instance of CSP(\(\Gamma\)) by the \(\tau\)-structure that defines \(R\).

**Lemma 1.** Let \(\Gamma\) be a \(\tau\)-structure and let \(\Gamma'\) be the extension of this structure by a relation \(R\) that is p.p.-definable over \(\Gamma\). Then CSP(\(\Gamma\)) is polynomial-time equivalent to CSP(\(\Gamma'\)).

In the next section we introduce the algebraic notions that will be needed to characterize p.p.-definability.

## 4 The Clone of Polymorphisms

In this section, \(D\) will stand for a computable set and \(O\) for the set of finitary operations on \(D\), i.e., functions from \(D^k\) to \(D\) for finite \(k\). We say that \(f \in O\) preserves a \(k\)-ary relation \(R \subseteq D^k\) if \(R\) is a subalgebra of \((D, f)^k\). An operation that preserves all relations of a relational structure \(\Gamma\) is called a polymorphism of \(\Gamma\). The set of all \(k\)-ary polymorphisms of \(\Gamma\) is denoted by \(Pol^k(\Gamma)\), and we write \(Pol(\Gamma)\) for the set of all finitary polymorphisms \(Pol(\Gamma) = \bigcup_{i=1} Pol^i(\Gamma)\).

The notion of a product of relational structures allows an equivalent definition of polymorphisms, relating polymorphisms to homomorphisms. The (categorical-or cross-) product \(\Gamma_1 \times \Gamma_2\) of two relational \(\tau\)-structures \(\Gamma_1\) and \(\Gamma_2\) is a \(\tau\)-structure on the domain \(D_{\Gamma_1} \times D_{\Gamma_2}\). For all relations \(R \in \tau\) the relation \(R((x_1, y_2), \ldots, (x_k, y_k))\) holds in \(\Gamma_1 \times \Gamma_2\) iff \(R(x_1, \ldots, x_k)\) holds in \(\Gamma_1\) and \(R(y_1, \ldots, y_k)\) holds in \(\Gamma_2\). Comparing the corresponding definitions we see that a \(k\)-ary polymorphism \(f\) of a relational structure is a homomorphism from \(\Gamma^k = \Gamma \times \ldots \times \Gamma\) to \(\Gamma\), i.e., for an \(m\)-ary relation \(R\) in \(\tau\), if \(R(x_1, \ldots, x_m)\) holds in \(\Gamma^k\) then \(R(f(x_1), \ldots, f(x_m))\) holds in \(\Gamma\).
An operation $\pi$ is a projection (or a trivial polymorphism) if for all $n$-tuples, $\pi(x_1,\ldots,x_n) = x_i$ for some fixed $i \in \{1,\ldots,n\}$. The composition of a $k$-ary operation $f$ and $k$ operations $g_1,\ldots,g_k$ of arity $n$ is an $n$-ary operation defined by

$$f(g_1,\ldots,g_k)(x_1,\ldots,x_n) = f(g_1(x_1,\ldots,x_n),\ldots,g_k(x_1,\ldots,x_n)).$$

A clone $F$ is a set of operations from $O$ that is closed under composition and that contains all projections. We write $D_F$ for the domain $D$ of the clone $F$. For a set of operations $F$ from $O$ we write $\langle F \rangle$ for the smallest clone containing all operations in $F$ (the clone generated by $F$). Observe that $\text{Pol}(\Gamma)$ is a clone with the domain $D_\Gamma$.

Moreover, $\text{Pol}(\Gamma)$ is also closed under interpolation: We say that an operation $f \in O$ is an interpolation of a subset $F$ of $O$ if for every finite subset $B$ of $D$ there is some operation $g \in \langle F \rangle$ such that $f|_B = g|_B$ (f restricted to B equals g restricted to B, i.e., $f(a) = g(a)$ for every $a \in B^k$). The set of interpolations of $F$ is called the local closure of $F$. If the maximal arity of $\Gamma$ is bounded, $\text{Pol}(\Gamma)$ is also locally closed.

The converse was proved by Rosenberg and Schweigert [34]. Together with a theorem of Larose and Tardif [28] on infinite graphs this is one of the few known results on infinite structures and their polymorphisms. Many results on clones in general can be found in [38].

**Proposition 2 (Rosenberg and Schweigert [34]).** A set $F \subseteq O$ of operations is locally closed if and only if $X = \text{Pol}(\Gamma)$ for some relational structure $\Gamma$ of bounded maximal arity.

Important properties of operations in a clone: a $k$-ary operation $f$ is idempotent iff $f(x,\ldots,x) = x$ for all $x \in D$. An operation $f$ is called essentially unary iff there is a unary operation $f_0$ such that $f(x_1,\ldots,x_k) = f_0(x_i)$ for some $i \in \{1,\ldots,k\}$. A relational structure $\Gamma$ is called projective, iff all idempotent polymorphisms of $\Gamma$ are projections, and strongly projective, iff all polymorphisms of $\Gamma$ are projections [33].

Let $F$ be a (local) clone with domain $D$. Then $R \subseteq D^m$ is invariant under $F$, if every $f \in F$ preserves $R$. We denote by $\text{Inv}(F)$ the relational structure containing the set of all relations left invariant under $F$. A fundamental result of Bodnarcuk et al. [5] (other presentations can be found in [32][12]) says that for arbitrary finite relational structures $\Gamma$ the p.p.-definable relations can be characterized as the invariants of the polymorphisms of $\Gamma$.

**Theorem 3 (Bodnarcuk et al. [5]).** Let $\Gamma$ be a finite relational structure. Then

$$\langle \Gamma \rangle_{pp} = \text{Inv}(\text{Pol}(\Gamma)).$$

The proof of Theorem 3 also shows that it is decidable whether for a given finite relational structure $\Gamma$ a given relation $R$ is p.p.-definable or not. Generalizations of Theorem 3 and the related Galois correspondences were also studied.
for infinite domains. For arbitrary relational structures \( \Gamma \) the set of relations \( \text{Inv}(\text{Pol}(\Gamma)) \) was characterized with local closure operators on relational algebras in \[37\] (see also \[31\], page 32).

In the next section we will show that for countable homogeneous structures \( \Gamma \) any first-order definable relation is in \( \langle \Gamma \rangle_{pp} \) if and only if it is left invariant under all polymorphisms of finite arity. But first we note that the following is well-known for arbitrary cardinalities of the domain.

**Proposition 3** (see e.g. \[32\]). Let \( \Gamma \) be a relational structure. Then
\[
\langle \Gamma \rangle_{pp} \subset \text{Inv}(\text{Pol}(\Gamma)).
\]

**Proof.** Let \( R \) be a relation in \( \langle \Gamma \rangle_{pp} \). We prove that \( R \in \text{Inv}(\text{Pol}(\Gamma)) \) by induction on the length of a defining p.p.-formula \( \varphi \). The claim is true for \( \varphi = R(x_1, \ldots, x_n) \). For \( \varphi = \exists x. \varphi' \) we observe that every polymorphism that is left invariant by \( \varphi' \) also leaves \( \varphi \) invariant. The same holds for \( \varphi = \varphi_1 \land \varphi_2 \). \( \square \)

For a structure with a countable domain the inclusion of Proposition 3 might be strict. Consider for instance the following relational structure \( \Gamma = (\mathbb{N}; R_1, R_2, R_3) \) on the natural numbers communicated to the authors by Ferdinand Börner. We show that \( \text{Inv}(\text{Pol}(\Gamma)) \) contains relations that are not p.p.-definable.

\[
R_1 = \{(a, b, c, d) \mid a = b \text{ or } c = d, a, b, c, d \in \mathbb{N}\}
\]
\[
R_2 = \{(0)\}
\]
\[
R_3 = \{(a, a + 1) \mid a \in \mathbb{N}\}
\]

Every function preserving \( R_1 \) is essentially unary. If \( f \) is unary and preserves \( R_2 \) then \( f(0) = 0 \). Furthermore, if \( f \) preserves \( R_3 \) we have \( f(a + 1) = f(a) + 1 \) for all \( a \), and inductively follows \( f(a) = a \). Therefore \( \text{Pol}(\Gamma) \) is the set of all projections. Every projection preserves all relations, but even the unary first-order definable relation \( \{x \mid x = 1 \lor x = 3\} \) is not p.p.-definable.

For \( \omega \)-categorical structures \( \Gamma \) the situation looks better: It is known that the first-order definable relations are precisely the relations that are invariant under the automorphisms of \( \Gamma \), i.e. \( \langle \Gamma \rangle_{fo} = \text{Inv}(\text{Aut}(\Gamma)) \) (see e.g. \[22\]). The structure \( \text{Inv}(\text{Aut}(\Gamma)) \) is homogeneous and called the canonical structure of the permutation group \( \text{Aut}(\Gamma) \). We prove a corresponding theorem for primitive positive definability in the next section.

## 5 A Characterization of Primitive Positive Definability

We characterize the primitive positive first-order definable relations over an \( \omega \)-categorical structure \( \Gamma \) by the polymorphisms of \( \Gamma \) of finite arity.

**Theorem 4.** Let \( \Gamma \) be an \( \omega \)-categorical structure with relational signature \( \tau \). Then a relation \( R \) on \( \Gamma \) is invariant under the polymorphisms of \( \Gamma \) if and only if \( R \) is p.p.-definable, i.e.,
\[
\langle \Gamma \rangle_{pp} = \text{Inv}(\text{Pol}(\Gamma)).
\]
Proof. We already stated in Proposition 3 that the p.p.-definable relations over \( \Gamma \) are invariant under the polymorphisms of \( \Gamma \).

For the converse, let \( R \) be a \( k \)-ary relation from \( Inv(Pol(\Gamma)) \). Note that \( R \) is first-order definable in \( \Gamma \): By \( \omega \)-categoricity and Ryll-Nardzewski, and since \( \Gamma \) and \( Inv(Pol(\Gamma)) \) have the same automorphism group, the relation \( R \) is a union of finitely many orbits of the automorphism group of \( \Gamma \), and it can be defined by a disjunction \( \varphi \) of \( \tau \)-formulas that define these orbits. Let \( M_1, \ldots, M_w \) be the satisfiable monomials in this disjunction, and let \( x_1, \ldots, x_k \) be the variables of the monomials.

We have to construct a finite \( \tau \)-structure \( Q \) with designated vertices \( v_1, \ldots, v_k \) such that

\[
R = \{ (f(v_1), \ldots, f(v_k)) \mid f: Q \to \Gamma \text{ homomorphism} \}.
\]

The idea is to first consider an infinite \( \tau \)-structure, namely the categorical product \( \Gamma^w \), and then to apply König's Lemma to prove the existence of a suitable finite substructure.

For each monomial \( M_j \in M_1, \ldots, M_w \) of \( \varphi \) we find a substructure \( a^j_1, \ldots, a^j_k \) of \( \Gamma \), such that \( a^j_1, \ldots, a^j_k \) satisfies \( M_j \) in \( \Gamma \). Let \( b_1, b_2, \ldots \) be an enumeration of the \( w \)-tuples in \( D^w_1 \), starting with \( b_i = (a^j_1, \ldots, a^j_k) \) for \( 1 \leq i \leq k \). Let us call a partial mapping from \( \Gamma^w \) to \( \Gamma \) a bad mapping if it maps \( b_1, \ldots, b_k \) to a tuple not satisfying \( \varphi \). Since \( R \) is invariant under all polymorphisms, no homomorphism from \( \Gamma^w \) to \( \Gamma \) is bad.

We now claim that there is a finite substructure \( Q \) of \( \Gamma^w \) such that no homomorphism from \( Q \) to \( \Gamma \) is bad. Assume for contradiction that all finite substructures of \( \Gamma^w \) containing \( b_1, \ldots, b_k \) have a homomorphism to \( \Gamma \) mapping \( b_1, \ldots, b_k \) to a tuple not satisfying \( \varphi \). We now construct a bad homomorphism from \( \Gamma^w \) to \( \Gamma \), i.e. the images of \( b_1, \ldots, b_k \) do not satisfy \( \varphi \). This contradicts the fact that \( R \) is invariant under all polymorphisms.

To this end, consider the following infinite but finitely branching tree. The nodes on level \( n \) in the tree are the equivalence classes of the bad homomorphisms from \( \Gamma^w \) to \( \Gamma \), where two homomorphisms \( f_1 \) and \( f_2 \) are equivalent if \( f_1 = gf_2 \) for some \( g \in Aut(\Gamma) \). Adjacency between nodes on consecutive levels is defined by restriction. By our assumption, for each finite substructure of \( \Gamma^w \) there is a bad homomorphism, and thus the tree contains a node on each level. By the Ryll-Nardzewski, there are only finitely many nodes at each level. By König’s Lemma the tree contains an infinite path. We use this path to define a bad homomorphism from \( \Gamma^w \) to \( \Gamma \).

We proved by contradiction that there must be a finite substructure \( Q \) containing the vertices \( b_1, \ldots, b_k \) of \( \Gamma^w \) such that all homomorphisms from \( Q \) to \( \Gamma \) map \( b_1, \ldots, b_k \) to a tuple satisfying \( \varphi \). Conversely, every mapping \( f: Q \to \Gamma \) such that the tuple \((f(b_1), \ldots, f(b_k))\) satisfies \( \Gamma \) the monomial \( M_j \) can be extended to a homomorphism \( f: \Gamma^w \to \Gamma \). To see this note that both \( a^j_1, \ldots, a^j_k \) and \((f(b_1), \ldots, f(b_k))\) satisfy \( M_j \) and thus both lie in the same orbit of \( Aut(\Gamma) \). Thus we can choose \( f \) to be the \( j \)th projection combined with the automorphism sending \((a^j_1, \ldots, a^j_k)\) to \((f(b_1), \ldots, f(b_k))\). This completes the proof. \( \square \)
The clone of polymorphisms of an infinite structure is usually a very complicated object. However for homogeneous structures of finite signature we have the following:

**Proposition 4.** Let $\Gamma$ be a homogeneous structure with finite relational signature. Then the polymorphisms of $\Gamma$ are locally generated by countably many polymorphisms of $\Gamma$ and the automorphism group $\text{Aut}(\Gamma)$.

**Proof.** Let $u_1, u_2, \ldots, u_k$ be $m$-tuples from $\Gamma$, and let $v_i = g(u_i)$ for some $m$-ary polymorphism $g$ of $\Gamma$. Depending on $m$ and $k$ there are only finitely many isomorphism types of the substructure of $\Gamma$ induced by the elements of $u_1, u_2, \ldots, u_k$ and the elements $v_1, \ldots, v_k$. Let $F$ be a set of polymorphisms of arity $k$ containing a polymorphism $g$ for each of these isomorphism types in $\Gamma$.

Now let $f$ be an $m$-ary polymorphism of $\Gamma$. We show that $f$ is locally generated by $F \cup \text{Aut}(\Gamma)$. Let $B \subseteq D_\Gamma$ a set of finite cardinality. By the definition of $F$, the restriction $f|_B$ is isomorphic to the restriction of one of the operations $g \in F$. Let $\pi$ be the isomorphism. By homogeneity of $\Gamma$, $\pi$ can be extended to an automorphism $\pi'$ of $\Gamma$. The identity $f|_B = (\pi' g)|_B$ implies that $f$ is in the local closure of $\text{Aut}(\Gamma) \cup F$. \hfill $\square$

### 6 A Catalog of Homogeneous Templates

We consider various homogeneous and $\omega$-categorical structures, some of their polymorphisms and their corresponding constraint satisfaction problems. In particular we look at the binary structures from the classification project for countable homogeneous structures.

**The Countable Homogeneous Tournaments.** We start with the homogeneous tournaments, which have been classified by Lachlan [26]. There are a few types only: The oriented cycle $C_3$, the dense linear order $(\mathbb{Q}, \prec)$, the dense local order $S(2)$, and the generic tournament for the set of all finite tournaments.

The problem CSP($C_3$) is known to be tractable. The constraint satisfaction problem of the dense linear order $(\mathbb{Q}, \prec)$ is computationally equivalent to the problem whether a given digraph $D$ is acyclic. This tractable problem can not be formulated as a constraint satisfaction problem with a finite template. Note that the relational structure $(\mathbb{Q}, \prec)$ is not projective, e.g. $x, y \mapsto \max(x, y)$ is a polymorphism. The homogeneous tournament which is the Fraïssé-limit of all finite tournaments has a trivial constraint-satisfaction problem: Every finite tournament homomorphically maps to it. Thus the only interesting remaining case is the dense linear order $S(2)$ (see [10]). The problem CSP($S(2)$) is NP-hard, since it can simulate the hard problem Betweenness [19].

To define the density local order $S(2)$, consider two disjoint dense subsets $X$ and $Y$ of the rational numbers $\mathbb{Q}$ (i.e., for every rational number we will find sequences in $X$ and in $Y$ that converge against this number). Then the relation $\prec$ of $S(2)$ equals the dense linear order of $\mathbb{Q}$ on $X \cup Y$, but we reverse the edges between the sets $X$ and $Y$. It is easy to see that $S(2)$ is the up to
isomorphism unique countable tournament that satisfies the following property: for every vertex, both in-neighbourhood and out-neighbourhood are isomorphic to \((\mathbb{Q}, <)\).

The problem *Betweenness* can be stated as a constraint satisfaction problem \(\text{CSP}(\mathbb{Q}, R)\) where \(R = \{(x_1, x_2, x_3) \subseteq \mathbb{Q}^3 \mid x_2 < x_1 \text{ or } x_2 > x_3\}\). This relation can be simulated by the following p.p.-formula in \(S(2)\):

\[
\exists u, v : u < x_1 \land u < x_2 \land u < x_3 \land x_1 < x_3 \land x_2 < v \land v < x_1 \land v < x_3
\]

*The Countable Homogeneous Graphs.* Lachlan and Woodrow \[27\] showed that every infinite such graph is either the Rado-graph, the Fraïssé-limit of all \(K_n\)-free graphs, the complete \(n\)-partite graphs, or a complement of these.

The Rado graph \(R\) is the Fraïssé-limit of the class of all graphs, therefore the constraint satisfaction problem for the Rado graph is trivial: Every graph can be homomorphically mapped to it. The automorphism group of \(R\) has a rich structure (see e.g. \[9\] for an overview). Luczak and Nešetřil \[29\] showed that the Rado graph as well as the generic \(K_n\)-free graphs are projective. This is in interesting opposition to the finite case, where projectivity of a core of cardinality at least three implies NP-hardness for the corresponding constraint satisfaction problem, which can be seen using Theorem \[3\] by reduction of \(k\)-colorability. Again the constraint satisfaction problem is easy, since every graph which does not contain the \(K_n\) as a subgraph embeds into the generic \(K_n\)-free graph. The only interesting remaining cases are the complete \(n\)-partite graphs. Each such graph has a homomorphism to the finite graph \(K_n\), which has an NP-complete constraint satisfaction problem for \(n \geq 3\), and is tractable for \(n \leq 2\).

*Remark:* It is perhaps interesting to note that we can use the same results to give a new and short proof that almost all constraint satisfaction problems are NP-complete, if the template is a finite undirected graph. The fact that almost all graphs are strongly projective (Nešetřil and Luczak \[29\]) combined with Theorem \[3\] shows that almost all graphs can simulate the inequality-relation. This implies NP-hardness of the constraint satisfaction problem on a domain of size at least three. Note that we did not use the involved proof of the dichotomy for graphs in \[20\].

*Countable Homogeneous Digraphs.* The countable homogeneous digraphs have been classified by Cherlin \[10\], and there are uncountably many. But the classification shows that the age of all but a countable well-understood class of homogeneous digraphs has the strong free amalgamation property. Therefore it is easy to see that \(\text{CSP}(\Gamma)\) is the set of all (weak) subgraphs of \(\Gamma\), and that the set of constraint satisfaction problems is also uncountable. Thus there are homogeneous digraphs with an undecidable constraint satisfaction problem. However, if we consider the homogeneous structures \(\Gamma\) that have a finitely axiomatizable age and free amalgamation, it is easy to see that \(\text{CSP}(\Gamma)\) is tractable. The remaining homogeneous digraphs we described by Cherlin and its straightforward to determine the complexity of their constraint satisfaction problems (\[10\]; note that \(\text{CSP}(S(2)) = \text{CSP}(S(3)) = \text{CSP}(P(3))\)).
To summarize: Every homogeneous digraph problem is either undecidable, NP-complete or tractable. Note that this does not say anything about the mentioned constraint satisfaction problems for finite digraphs, since the corresponding homogeneous templates have a larger signature.

Tree Descriptions. The following constraint satisfaction problem was studied in [11]. Given: a finite structure $S$ over the signature $\tau = \{\rightarrow, \perp\}$ containing two binary relation symbols. Question: can we find a rooted forest $F$ on the vertices of $S$ such that every edge from $\rightarrow$ lies in the transitive closure of $F$, and every edge $\perp$ does not? Let us call such $\tau$-structures $S$ solvable.

Using the notion of dense trees (see [15]) we can formulate this problem (and related problems) as a constraint satisfaction problem CSP($\Gamma$) for appropriate $\Gamma$. This means, it is possible to find an $\omega$-categorical structure $\Gamma$ such that CSP($\Gamma$) contains precisely the solvable $\tau$-structures. The problem can be decided by a polynomial time algorithm [4]. The graph algorithm presented there can be generalized to various constraint satisfaction problems for tree-like structures. We already mentioned that every $\omega$-categorical structure $\Gamma$ can be made homogeneous by expanding the signature and $\Gamma$ by some first-order definable relations. In the case of countable dense trees this is possible by an additional ternary relation [16].

Allen’s Interval Algebra and Its Fragments. Consider as a base set $D$ the closed intervals on the rational numbers, and the following binary relations on these intervals: Let $x = (x^-, x^+)$ and $y = (y^-, y^+)$ be closed intervals. We define

- The interval $x$ precedes $y$, $x \mathbin{p} y$, iff $x^+ < y^-$.  
- The interval $x$ overlaps $y$, $x \mathbin{o} y$, iff $x^- < y^- < x^+$ and $x^+ < y^+$.  
- The interval $x$ is during $y$, $x \mathbin{d} y$, iff $y^- < x^- < x^+ < y^+$.  
- The interval $x$ starts $y$, $x \mathbin{s} y$, iff $x^- = y^-$ and $x^+ > y^-$.  
- The interval $x$ finishes $y$, $x \mathbin{f} y$, iff $x^+ = y^+$ and $x^- > y^-$.  
- The interval $x$ meets $y$, $x \mathbin{m} y$, iff $x^+ = y^-$.  
- The interval $x$ equals $y$, $x \equiv y$, iff $x^- = y^-$ and $x^+ = y^+$.

For any set of relations derived from $p, o, d, s, m, f$ and $\equiv$ by union and complementation the corresponding countable relational structure is $\omega$-categorical. The constraint satisfaction problems for these structures have a dichotomy [24,30]. Whereas for finite templates all known hard constraint satisfaction problems are hard because they can express the relation one-in-three-sat, here again the problem Betweenness [19] is used to prove hardness.

7 Related Work

We want to relate our work to previous unifying approaches to constraint satisfaction. The literature on combining constraint solving [25,2] has a broader view on constraint satisfaction, and also uses tools from universal algebra. However they are concerned mainly with decidability questions of more expressive constraint languages.
Various logical formalisms have been proposed to formulate constraint satisfaction problems as the model-checking problems of certain higher-order logics [17]. One of them is the class SNP, the class of existential second-order formulas $\Phi$ with a universal first-order part, which might use the relation symbols of the given signature $\tau$ and existentially quantified relation symbols. One of the results in [17] says that every problem in NP is equivalent to a problem in SNP, even if the relation symbols from $\tau$ occur only negatively in $\Phi$ (in which case the class is called monotone SNP). To answer the question whether the model checking problem of a given monotone SNP formula can be described as a constraint satisfaction problem with a countable homogeneous structure, the following problem posed by Cherlin [10] is of importance:

**Problem 1.** Let $\tau$ be a relational signature and $N$ a finite set of finite $\tau$-structures. Give a good criterion for $\text{Forb}(N)$ to be an amalgamation class.

## 8 Conclusion

Constraint satisfaction problems with countable homogeneous templates cover several classes of constraint satisfaction problems that were investigated in the literature. Examples are tree description languages and subalgebras of Allen’s interval algebra. Using the classification of homogeneous digraphs we can determine the complexity of the constraint satisfaction problems for all homogeneous digraphs. For larger signatures the classification of homogeneous structures is a difficult task. To study the complexity of a constraint satisfaction problem $\text{CSP}(\Gamma)$ it is useful to know whether a given first-order relation is p.p.-definable over $\Gamma$. In Section 5 we show that p.p.-definability of a first-order definable relation is characterized by a countable set of polymorphisms. We ask the following question:

**Problem 2.** Let $N$ be a finite set of relational structures over signature $\tau$ such that $\text{Forb}(N)$ is the age of a homogeneous structure $\Gamma$. Given a first-order formula $\varphi$ over $\tau$, is it decidable whether $\varphi$ is on $\Gamma$ equivalent to a primitive positive formula?

The techniques to describe p.p.-definability might be applied to simplify the technical and intricate proofs in the classification of the tractable fragments of Allen’s interval algebra.

## References

Quantified Constraints:
Algorithms and Complexity

Ferdinand Börner\textsuperscript{1}, Andrei Bulatov\textsuperscript{2,⋆},
Peter Jeavons\textsuperscript{2,⋆}, and Andrei Krokhin\textsuperscript{3,⋆⋆}

\textsuperscript{1} Institut für Informatik, University of Potsdam
Potsdam, D-14482, Germany
fboerner@cs.uni-potsdam.de
\textsuperscript{2} Oxford University Computing Laboratory
Oxford, OX1 3QD, UK
\{andrei.bulatov,peter.jeavons\}@comlab.ox.ac.uk
\textsuperscript{3} Department of Computer Science, University of Warwick
Coventry, CV4 7AL, UK
andrei.krokhin@dcs.warwick.ac.uk

Abstract. The standard constraint satisfaction problem over an arbitrary finite domain can be expressed as follows: given a first-order sentence consisting of a conjunction of predicates, where all of the variables are existentially quantified, determine whether the sentence is true. This problem can be parameterized by the set of allowed constraint predicates. With each predicate, one can associate certain predicate-preserving operations, called polymorphisms, and the complexity of the parameterized problem is known to be determined by the polymorphisms of the allowed predicates. In this paper we consider a more general framework for constraint satisfaction problems which allows arbitrary quantifiers over constrained variables, rather than just existential quantifiers. We show that the complexity of such extended problems is determined by the surjective polymorphisms of the constraint predicates. We give examples to illustrate how this result can be used to identify tractable and intractable cases for the quantified constraint satisfaction problem over arbitrary finite domains.

1 Introduction

The constraint satisfaction problem (CSP) provides a general framework in which a wide variety of combinatorial problems can be expressed in a natural way \cite{12,28,37}. A constraint satisfaction problem instance can be viewed as a collection of predicates on overlapping sets of variables. The aim is to determine whether there exist values for all of the variables such that all of the specified predicates hold simultaneously. The standard constraint satisfaction problem can

⋆ Partially supported by the UK EPSRC grant GR/R29598.
⋆⋆ Partially supported by the UK EPSRC grant GR/R29598 and by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
be parameterized by restricting the set of allowed predicates which can be used as constraints. The problem of determining (up to complete classification) the complexity of the CSP (and its many variants) for all possible parameter sets has attracted much attention, partly because constraint satisfaction problems play an important role in Artificial Intelligence [28,37], and partly because they “present a reasonably accurate bird’s-eye view on complexity theory” [12]. One important outcome of research in this direction has been the design of sophisticated new polynomial-time algorithms for solving a wide variety of problems (see, e.g., [5,6,15]). Another outcome has been progress with some important issues in complexity theory, such as the discovery of large subclasses of complexity classes that avoid intermediate complexity (e.g., dichotomy results in the case of NP, see [4,6,17,35]).

For the Boolean (i.e., two-valued) case, the complexity of the standard constraint satisfaction problem has been studied from the above perspective [35], as well as a number of related problems, including quantified and counting problems, maximum (and minimum) satisfiability, generating all solutions, optimizing the number of positive truth values in a solution (all of these are in [12]), minimal satisfiability [25], deciding equivalence and isomorphism of instances [2], maximizing Hamming distance between solutions [13], finding lexicographically minimal (or maximal) solutions [34], finding a second solution [22], inverse satisfiability [23], and solving random instances [11].

Analysing the complexity of non-Boolean CSPs is a significantly more difficult task: these problems usually withstand a direct combinatorial approach, and so require more involved techniques. A far-reaching approach via graph theory, logic and games has been developed in [16,17,27]. However, the most successful approach so far has been the algebraic approach developed in [8,20,21]. This approach has led to a number of new results (see, e.g., [5,7,9,15,16]), and has culminated (so far) in a complete classification of the complexity of parameterized CSPs for the three-valued case [4] and for the case when all unary predicates are available [6].

The standard CSP can be expressed as follows: given a first-order sentence consisting of a conjunction of predicates, where all of the variables are existentially quantified, determine whether the sentence is true. One of the most natural generalisations of this framework is to consider the quantified constraint satisfaction problem (QCSP), in which universal quantifiers are allowed in the sentence, as well as existential quantifiers [12,14]. This generalisation greatly increases the expressive power of the framework, but also increases the complexity of deciding whether an arbitrary instance is true — from NP-complete to PSPACE-complete.

Boolean QCSP (also known as QSAT or QBF) and some of its restrictions (such as Q3SAT) have always been standard examples of PSPACE-complete problems [19,29,35]. For some parameter sets, Boolean QCSP has been shown to be tractable: for all binary predicates in [1], and for Horn predicates in [26]. Finally, a complete classification for the Boolean case was obtained in [12,14]. For the non-Boolean case, some superpolynomial algorithms were given in [38].
However, to the best of our knowledge, only trivial results are known about the complexity of non-Boolean QCSPs.

In this paper we extend the algebraic approach for the first time to the more general framework of the quantified constraint satisfaction problem over an arbitrary finite set of values. We show that certain algebraic objects (surjective polymorphisms) determine the complexity of the QCSP for any given choice of parameter set. We then give examples to show how this result can be used to identify quantified constraint satisfaction problems lying in more restricted complexity classes such as NL and PTIME. Finally, we obtain the first complete classification result for a class of general (non-Boolean) quantified constraint satisfaction problems.

2 Preliminaries

Throughout the paper we use the standard correspondence between predicates and relations: a relation consists of all tuples of values for which the corresponding predicate holds. We will use the same symbol for a predicate and its corresponding relation, since the meaning will always be clear from the context. We will use $R_D^{(m)}$ to denote the set of all $m$-ary relations (or predicates) over a set $D$, and $R_D$ to denote the set $\bigcup_{m=1}^{\infty} R_D^{(m)}$.

**Definition 1.** Let $\Gamma \subseteq R_D$. An instance of CSP($\Gamma$) is a first-order sentence $\exists x_1 \ldots \exists x_l (\varrho_1 \land \ldots \land \varrho_q)$, where each $\varrho_i$ is an atomic formula involving a predicate from $\Gamma$, and $x_1, \ldots, x_l$ are the variables appearing in the $\varrho_i$. The question is whether the sentence is true.

The predicates $\varrho_i$ appearing in an instance will be referred to as constraints, since each of them restricts the possible models for the instance in some way.

In addition to predicates and relations we will also consider arbitrary operations on the set of values. We will use $O_D^{(n)}$ to denote the set of all $n$-ary operations on a set $D$ (that is, the set of mappings $f:D^n \to D$), and $O_D$ to denote the set $\bigcup_{n=1}^{\infty} O_D^{(n)}$.

Any operation on $D$ can be extended in a standard way to an operation on tuples over $D$, as follows. For any operation $f \in O_D^{(n)}$, and any collection of tuples $a_1, a_2, \ldots, a_n \in D^m$, where $a_i = (a_i(1), \ldots, a_i(m))$ ($i = 1 \ldots n$), define $f(a_1, \ldots, a_n)$ to be $(f(a_1(1), \ldots, a_n(1)), \ldots, f(a_1(m), \ldots, a_n(m)))$.

**Definition 2.** For any relation $\varrho \in R_D^{(m)}$, and any operation $f \in O_D^{(n)}$, if $f(a_1, \ldots, a_n) \in \varrho$ for all choices of $a_1, \ldots, a_n \in \varrho$, then $\varrho$ is said to be invariant under $f$, and $f$ is called a polymorphism of $\varrho$.

The set of all relations that are invariant under each operation from some set $C \subseteq O_D$ will be denoted $\text{Inv}(C)$. The set of all operations that are polymorphisms of every relation from some set $\Gamma \subseteq R_D$ will be denoted $\text{Pol}(\Gamma)$. We remark that the operators $\text{Inv}()$ and $\text{Pol}()$ form a Galois correspondence between $R_D$ and $O_D$ (see Proposition 1.1.14 of [32]). A basic introduction to this correspondence can be found in [30], and a comprehensive study in [32].
By considering certain properties of the set \( \text{Inv}(\text{Pol}(\Gamma)) \), the following result was obtained in [20].

**Theorem 1.** Let \( \Gamma_1 \) and \( \Gamma_2 \) be sets of predicates over a finite set, such that \( \Gamma_1 \) is finite. If \( \text{Pol}(\Gamma_2) \subseteq \text{Pol}(\Gamma_1) \) then \( \text{CSP}(\Gamma_1) \) is polynomial-time reducible to \( \text{CSP}(\Gamma_2) \).

This result shows that, when the set of values is finite, finite sets of predicates with the same polymorphisms give rise to constraint satisfaction problems which are mutually reducible. In other words, the complexity of \( \text{CSP}(\Gamma) \) is determined by the polymorphisms of \( \Gamma \).

A number of results on the complexity of constraint satisfaction problems have been obtained via this approach (e.g., [4,5,6,7,8,9,15,16,20,21]). For example, Schaefer’s Dichotomy Theorem [35], when appropriately re-stated, easily follows from Theorem 1 and well-known algebraic results [33] (see [20]).

**Theorem 2** ([35]). For any \( \Gamma \subseteq \mathbb{R}_{\{0,1\}} \), \( \text{CSP}(\Gamma) \) is in \( \text{PTIME} \) when \( \text{Pol}(\Gamma) \) contains at least one of the following:

- the constant 0 or constant 1 operations,
- the conjunction or disjunction operations,
- the affine operation \( x - y + z \mod 2 \),
- the majority operation \( (x \lor y) \land (x \lor z) \land (y \lor z) \).

In all other cases \( \text{CSP}(\Gamma) \) is \( \text{NP} \)-complete.

In this paper we consider the more general framework of the quantified constraint satisfaction problem, which is defined as follows.

**Definition 3.** Let \( \Gamma \subseteq R_D \). An instance of \( \text{QCSP}(\Gamma) \) is a first-order sentence

\[
\mathcal{Q} \mathcal{L} x_1 \ldots \mathcal{Q} \mathcal{L} x_l \ (\varphi_1 \land \ldots \land \varphi_q),
\]

where each \( \varphi_i \) is an atomic formula involving a predicate from \( \Gamma \), \( x_1,\ldots,x_l \) are the variables appearing in the \( \varphi_i \), and \( \mathcal{Q}_1,\ldots,\mathcal{Q}_l \) are arbitrary quantifiers. The question is whether the sentence is true.

Clearly, an instance of \( \text{CSP}(\Gamma) \) corresponds to an instance of \( \text{QCSP}(\Gamma) \) in which all the quantifiers happen to be existential.

For any finite set \( D \), and any set of relations \( \Gamma \subseteq R_D \), one can use an exhaustive search algorithm to show that \( \text{QCSP}(\Gamma) \) is in \( \text{PSPACE} \). The complexity of \( \text{QCSP}(\Gamma) \) has been completely characterized in the Boolean case [12,14].

**Theorem 3** ([12,14]). For any \( \Gamma \subseteq R_{\{0,1\}} \), \( \text{QCSP}(\Gamma) \) is in \( \text{PTIME} \) when \( \text{Pol}(\Gamma) \) contains at least one of the following:

- the conjunction or disjunction operations,
- the affine operation \( x - y + z \mod 2 \),
- the majority operation \( (x \lor y) \land (x \lor z) \land (y \lor z) \).

In all other cases \( \text{QCSP}(\Gamma) \) is \( \text{PSPACE} \)-complete.

**Corollary 1.** \( \text{QCSP}(\{N\}) \) is \( \text{PSPACE} \)-complete, where \( N \) is the ternary “not-all-equal” relation on \( \{0,1\} \), defined by \( N = \{0,1\}^3 \setminus \{(0,0,0),(1,1,1)\} \).
Even though Theorem 3 is stated here using polymorphisms, it was proved using a combinatorial rather than an algebraic approach, and that method of proof does not easily generalize to larger sets of values. In the remaining sections we introduce an algebraic approach which enables us to systematically analyse the complexity of quantified constraint satisfaction problems over an arbitrary finite set $D$.

3 Reduction in QCSP

The next result is the main result of this paper. It shows that, for quantified constraint satisfaction problems, surjective polymorphisms play a similar role to that played by arbitrary polymorphisms for ordinary CSPs (cf. Theorem 1). Let $s\text{-Pol}(\Gamma)$ denote the set of all surjective operations from $\text{Pol}(\Gamma)$.

Theorem 4. Let $\Gamma_1$ and $\Gamma_2$ be sets of predicates over a finite set, such that $\Gamma_1$ is finite. If $s\text{-Pol}(\Gamma_2) \subseteq s\text{-Pol}(\Gamma_1)$, then $\text{QCSP}(\Gamma_1)$ is polynomial-time reducible to $\text{QCSP}(\Gamma_2)$.

This theorem follows immediately from the next two propositions whose proofs can be found in [3].

Definition 4. For any set $\Gamma \subseteq R_D$, the set $[\Gamma]$ consists of all predicates that can be expressed using

1. predicates from $\Gamma$, together with the binary equality predicate $=D$ on $D$,
2. conjunction,
3. existential quantification,
4. universal quantification.

Proposition 1. Let $\Gamma_1$ and $\Gamma_2$ be sets of predicates over a finite set, such that $\Gamma_1$ is finite. If $[\Gamma_1] \subseteq [\Gamma_2]$, then $\text{QCSP}(\Gamma_1)$ is polynomial-time reducible to $\text{QCSP}(\Gamma_2)$.

Proposition 2. For any set of predicates $\Gamma$ over a finite set, $[\Gamma] = \text{lnv}(s\text{-Pol}(\Gamma))$.

Note that this proposition intuitively means that the expressive power of constraints in QCSP is determined by their surjective polymorphisms. Hence, in order to show that some relation $\varrho$ belongs to $[\Gamma]$, one does not have to give an explicit construction, but instead one can show that $\varrho$ is invariant under all surjective polymorphisms of $\Gamma$, which often turns out to be significantly easier.

We remark that the operators $\text{lnv}()$ and $s\text{-Pol}()$ used in Proposition 2 form a Galois connection between $R_D$ and the set of all surjective members of $O_D$ which has not previously been investigated (see, e.g., survey [31]).

4 Intractable Cases

In this section we will use Theorem 4 to show that certain small sets of predicates give rise to quantified constraint satisfaction problems that are intractable. In
particular, we will show that QCSP($\Gamma$) can be PSPACE-complete even in some cases where CSP($\Gamma$) is trivial.

We first establish that a particular QCSP problem is PSPACE-complete. This problem corresponds to a generalized form of the standard graph-$|D|$-colorability problem \cite{19,20} (which can be expressed as CSP($\{\neq = D\}$) where $\neq D$ is the binary disequality predicate on $D$).

**Proposition 3.** QCSP($\{\neq = D\}$) is PSPACE-complete when $|D| \geq 3$.

**Proof.** By reduction from QCSP($\{N\}$), where $N$ is the ternary not-all-equal predicate on a 2-element set, as defined in Corollary \cite{1}. For details see \cite{3}.

**Theorem 5.** For any finite set $D$ with $|D| \geq 3$, and any $\Gamma \subseteq R_D$, if every $f \in s$-$\text{Pol}(\Gamma)$ is of the form $f(x_1, \ldots, x_n) = g(x_i)$ for some $1 \leq i \leq n$ and some permutation $g$ on $D$, then QCSP($\Gamma$) is PSPACE-complete.

**Proof.** By Lemma 1.3.1 (b) of \cite{32}, $\text{Pol}(\{\neq = D\})$, for $|D| \geq 3$, consists of all operations of the form described in the Theorem. Hence $\text{Pol}(\{\neq D\}) = s$-$\text{Pol}(\{\neq D\})$, and we can apply Theorem \cite{4} and Proposition \cite{5}.

The next example uses this result to show that even predicates that give rise to trivial constraint satisfaction problems can give rise to intractable quantified constraint satisfaction problems. This can happen because non-surjective operations, which may guarantee the tractability of the CSP, do not affect the complexity of the QCSP.

**Example 1.** Let $\tau_s$ be the $s$-ary “not-all-distinct” predicate holding on a tuple $(a_1, \ldots, a_s)$ if and only if $|\{a_1, \ldots, a_s\}| < s$. Note that $\tau_s \supseteq \{(a, \ldots, a) | a \in D\}$, so every instance of CSP($\{\tau_s\}$) is trivially satisfiable by assigning the same value to all variables.

However, by Lemma 2.2.4 of \cite{32}, the set $\text{Pol}(\{\tau_{|D|}\})$ consists of all non-surjective operations on $D$, together with all operations of the form given in Theorem \cite{5}. Hence, $\{\tau_{|D|}\}$ satisfies the conditions of Theorem \cite{5} and QCSP($\{\tau_{|D|}\}$) is PSPACE-complete. Similar arguments can be used to show that QCSP($\{\tau_s\}$) is PSPACE-complete, for any $s$ in the range $3 \leq s \leq |D|$.

## 5 Tractable Cases

In spite of the results of the previous section, it is possible to identify sets of predicates which give rise to tractable QCSP problems. In this section we identify and describe two families of predicates of this kind.

### 5.1 Mal’tsev Predicates

An operation $m(x, y, z)$ on $D$ is said to be Mal’tsev if it satisfies the identities $m(x, y, y) = m(y, y, x) = x$ for all $x, y$. For example, for an Abelian group $G$, the operation $f(x, y, z) = x - y + z$, called the affine operation of $G$, is a Mal’tsev
operation. Relations invariant under the affine operation of a finite Abelian group play a significant role in the study of the complexity of the standard constraint satisfaction problem [5,17,20,21].

Throughout this subsection, let \( \Gamma = \lnv\{m\} \). By developing and using a deep algebraic structural theory, a sophisticated polynomial-time algorithm for deciding CSP(\( \Gamma \)) was given in [5]. Moreover, it was proved there that a satisfying assignment to any CSP(\( \Gamma \)) instance can also be found in polynomial time. We will show now that QCSP(\( \Gamma \)) can be solved using these algorithms.

**Theorem 6.** Let \( m \) be an arbitrary Mal’tsev operation on \( D \). The problem class QCSP(\( \lnv\{m\} \)) is in PTIME.

**Proof.** The proof is based on the following lemma.

**Lemma 1.** Let \( \mathcal{P} = Q_1x_1 \ldots Q_nx_n \Phi(x_1, \ldots, x_n) \) be an instance of QCSP(\( \Gamma \)), and \( j \) the maximal index such that \( Q_j \) is the universal quantifier.

1. If \( \Phi'(x_1, \ldots, x_{j-1}) = \forall x_j \exists x_{j+1} \ldots \exists x_n \Phi(x_1, \ldots, x_n) \) is satisfiable then, for any model \( (c_1, \ldots, c_n) \) of \( \Phi \), the tuple \( (c_1, \ldots, c_{j-1}) \) is a model of \( \Phi' \).

2. \( \mathcal{P} \) is true if and only if so is \( \mathcal{P'} = \mathcal{P}_1 \land \mathcal{P}_2 \) where

\[
\mathcal{P}_1 = Q_1x_1 \ldots Q_{j-1}x_{j-1} \exists x_j \exists x_{j+1} \ldots \exists x_n \Phi(x_1, \ldots, x_n), \\
\mathcal{P}_2 = \exists x_1 \ldots \exists x_{j-1} \forall x_j \exists x_{j+1} \ldots \exists x_n \Phi(x_1, \ldots, x_n).
\]

**Proof.** (1) Let \( (a_1, \ldots, a_{j-1}) \) be a model for \( \Phi' \), and \( (a^b_1, \ldots, a^b_n) \), \( b \in D \), its extensions such that \( a^b = (a_1, \ldots, a_{j-1}, a^b_j, \ldots, a^b_n) \) is a model of \( \Phi \) and \( a^b_j = b \).

Take an arbitrary model \( c = (c_1, \ldots, c_{j-1}, c_j, \ldots, c_n) \) of \( \Phi \). We need to show that \( (c_1, \ldots, c_{j-1}) \) is a model of \( \Phi' \). Fix an arbitrary \( b \in D \) and let \( d = (d_1, \ldots, d_n) \) be equal to \( m(a^b, a^{c_j}, c) \). Proposition [2] implies that the predicate defined by \( \Phi \) is invariant under \( m \), so \( d \) is a model of \( \Phi \), too. Moreover, we have \( d_i = m(a_i, a_i, c_i) = c_i \) for \( i \in \{1, \ldots, j - 1\} \) and \( d_j = m(a^b_j, a^c_j, c_j) = m(b, c_j, c_j) = b \). Thus, \( (c_1, \ldots, c_{j-1}) \) is a model of \( \Phi' \).

(2) Obviously, if \( \mathcal{P} \) is true then \( \mathcal{P'} \) is also true. The inverse implication easily follows from part (1). Indeed, since \( \mathcal{P}_2 \) is true, we can apply (1); then, (1) implies that every tuple \( (c_1, \ldots, c_{j-1}) \) that can be extended to a model of \( \Phi \) can be extended so with \( c_j \) being any given element. Thus, since \( \mathcal{P}_1 \) is true, so is \( \mathcal{P} \).

Repeatedly applying Lemma [1](2), one can show that every instance of QCSP(\( \Gamma \)) can be decomposed into a conjunction of instances which have the same quantifier-free part and each contain at most one universal quantifier. Moreover, if we can find a model of \( \Phi \) then part (1) of the lemma implies that initial segments of this model can be used in deciding whether each of the instances is true. It remains to notice that, as is easy to check, fixing a value for any variable in a predicate from \( \Gamma \) gives another predicate invariant under \( m \) which implies that \( \exists x_{j+1} \ldots \exists x_n \Phi(c_1, \ldots, c_{t-1}, b, x_{t+1}, \ldots, x_n) \) is also an instance of CSP(\( \lnv\{m\} \)). Now it follows that the algorithm shown in Fig. [1] is correct.
Input $P = Q_1x_1 \ldots Q_nx_n \Phi(x_1, \ldots, x_n)$ where $\Phi = \varphi_1 \land \ldots \land \varphi_q$, and $\varphi_1, \ldots, \varphi_q \in \Gamma$.

Output ‘YES’ if $P$ is true, ‘NO’ otherwise.

Step 1 Solve the instance $\exists x_1 \ldots \exists x_n \Phi$

Step 2 If $\Phi$ has a model then find one, $(c_1, \ldots, c_n)$, else OUTPUT(‘NO’) and STOP.

Step 3 For $l = n, \ldots, 1$ do
  If $Q_l$ is the universal quantifier then
    For each $b \in D$ do
      Solve the instance $\exists x_{l+1} \ldots \exists x_n \Phi(c_1, \ldots, c_{l-1}, b, x_{l+1}, \ldots, x_n)$.
      If the instance has no solution then OUTPUT(‘NO’) and STOP.
    enddo
  enddo
enddo

Step 4 OUTPUT(‘YES’).

Fig. 1. Algorithm for deciding QCSP($\Gamma$) for Mal’tsev $\Gamma$

This algorithm uses $k|D| + 1$ applications of an algorithm solving CSP($\Gamma$), where $k$ is the number of universal quantifiers in an instance, and one application of an algorithm finding a model. Now we can use the general polynomial-time algorithms developed in [5]. This finishes the proof of Theorem [6].

Note that if the operation $m$ has a special form then the method described above may lead to better algorithms. For example, let $G$ be a finite Abelian group, with affine operation $f$, and unit element 0, and let $\Gamma$ be a set of relations over $G$ which are invariant under $f$. Note that, by straightforward algebraic manipulation, it can be shown that any $(n$-ary) relation invariant under $f$ is a coset of a subgroup of the group $G^n$.

In the simplest case, when the order of $G$ is prime, $G$ can be considered as a prime field, and hence $G^n$ as a vector space over $G$. In this case, each coset of a subgroup of $G^n$ is a linear manifold, and it is well-known that such manifolds can be defined by systems of linear equations, whose coefficients are elements of the field $G$. Therefore, in this case, QCSP($\Gamma$) can be considered as the problem of solving quantified linear systems over $G$, which can be done by applying standard techniques from linear algebra, or by using them in the above algorithm.

5.2 Implicational Predicates

Our second example of predicates which give rise to tractable quantified constraint satisfaction problems concerns predicates that are invariant under an operation known as the dual discriminator. These problems can be viewed as generalized Q2SAT, and our algorithm generalizes and extends the algorithm for Q2SAT given in [1].

Definition 5. For any finite set $D$, the dual discriminator operation $d$ on $D$, is given by

$$d(x, y, z) = \begin{cases} 
  y & \text{if } y = z \\
  x & \text{otherwise}
\end{cases}$$
Note that, in the special case when \( D = \{0, 1\} \), the dual discriminator is exactly the majority operation mentioned in Theorems 2 and 3.

**Theorem 7.** Let \( D \) be a finite set, and let \( d \) be the dual discriminator operation on \( D \). The problem class \( \text{QCSP}(\text{Inv}(\{d\})) \) is in \( \text{NL} \).

To establish this result, we consider a graph structure associated with any instance of a constraint satisfaction problem, which is sometimes known as the “microstructure graph” \[18\]. For any instance \( \mathcal{P} \) of QCSP, with variables \( V \) and set of values \( D \), the microstructure graph of \( \mathcal{P} \) is the graph \((W, E)\), where the set of nodes \( W \), is the subset of pairs \( V \times D \) representing all possible assignments of values to individual variables which are compatible with the constraints on those variables, and the set of edges, \( E \), is the set of all assignments to pairs of variables which are compatible with the constraints on those pairs. In other words, \( E \) is the set of all (ordered) pairs of nodes \(((v_i, a), (v_j, b)) \in W \times W\) such that \( v_i \neq v_j \) and the partial assignment \( v_i = a, v_j = b \) is compatible with each individual predicate in \( \mathcal{P} \).

We will call an arc \(((v_i, a), (v_j, b))\) in the microstructure graph of \( \mathcal{P} \) implicative if it is the unique arc in \( E \) with first component \((v_i, a)\) and second component \((v_j, b')\), for some value \( b' \). If we remove all arcs from the microstructure graph of \( \mathcal{P} \) which are not implicative, then we obtain a directed graph which will be called the implication graph of \( \mathcal{P} \), and denoted \( I(\mathcal{P}) \).

Note that the arcs in the implication graph \( I(\mathcal{P}) \) represent logical implications which can be made about the possible models for the conjunction of predicates in \( \mathcal{P} \). If any model gives value \( a \) to variable \( v_i \), and \( I(\mathcal{P}) \) contains the arc \(((v_i, a), (v_j, b))\), then that model must also give the value \( b \) to the variable \( v_j \). By the transitivity of implication, we obtain the same conclusion if there is any path from \((v_i, a)\) to \((v_j, b)\) in \( I(\mathcal{P}) \).

For some sets of predicates the implication graph has certain additional properties, which can be used to obtain further restrictions on the possible models. We define one such property as follows.

**Definition 6.** For any QCSP instance \( \mathcal{P} \), the implication graph \( I(\mathcal{P}) \) will be called invertible if whenever there is a path from a node \((v_i, a)\) to a node \((v_j, b)\), then for each node \((v_j, b')\) with \( b' \neq b \) there is a path from \((v_j, b')\) to \((v_i, a')\) for some \( a' \neq a \) (which may depend on \( b' \)).

The second property we define holds when the implication graph captures all the restrictions on the possible models. In other words, it says that a given assignment is a model for the conjunction of predicates in \( \mathcal{P} \) whenever it satisfies all of the implications encoded in the implication graph of \( \mathcal{P} \).

**Definition 7.** Let \( \mathcal{P} \) be a QCSP instance of the form \( Q_1 x_1 \ldots Q_l x_l \Phi \). The implication graph \( I(\mathcal{P}) \) will be called sufficient if a given assignment \( \sigma \) is a model of \( \Phi \) whenever \( I(\mathcal{P}) \) contains the node \((v, \sigma(v))\) for each variable \( v \), and does not contain an arc \(((v_i, \sigma(v_i)), (v_j, b))\), with \( b \neq \sigma(v_j) \).

We will now show that, when the implication graph for an instance is both invertible and sufficient, then there is a non-deterministic logarithmic-space algorithm for that instance.
Proposition 4. There is a non-deterministic logarithmic-space algorithm that decides any QCSP instance $\mathcal{P}$ whose implication graph $I(\mathcal{P})$ is both invertible and sufficient.

Proof. See [3].

Finally, we show that both of these properties hold for all predicates that are invariant under the dual discriminator.

Proposition 5. Let $D$ be a finite set, and let $d$ be the dual discriminator operation on $D$. For any instance $\mathcal{P}$ in QCSP($\text{Inv}(\{d\})$), the implication graph $I(\mathcal{P})$ is invertible and sufficient.

Proof. It was shown in [21] that the predicates invariant under $d$ are precisely the predicates which can be expressed as conjunctions of binary predicates, each of which is of the form of a “0/1/all” binary relation, as described in [10]. Such predicates are also described in [24], where they are referred to as “implicational”.

The defining characteristic of a “0/1/all” relation on a pair of variables is that for both variables each value is either disallowed, or else allowed with either precisely one value, or all possible values, for the other variable [10]. Hence, it is straightforward to show that they give rise to implication graphs which are invertible. (In fact, an equivalent statement appears as Proposition 2.3 in [24].)

Furthermore, it is straightforward to verify that, in any problem instance whose constraints are specified by such relations, the restrictions imposed by these constraints on possible assignments are precisely those captured by the implication graph of the instance. Hence this implication graph is sufficient.

6 A Trichotomy Result

In this section we apply results from the previous sections to obtain a complete classification of complexity of QCSP($\Gamma$) in those cases where $\Gamma$ contains the set $\Delta$ of all graphs of permutations. Recall that the graph of a permutation $\pi$ is the binary relation $\{(x, y) \mid y = \pi(x)\}$ (or the binary predicate $\pi(x) = y$). The complexity of CSP($\Gamma$) for such sets $\Gamma$ is completely classified in [15].

We will need two new surjective operations:

– The $k$-ary near projection operation,

$$l_k(x_1, \ldots, x_k) = \begin{cases} x_1 & \text{if } x_1, \ldots, x_k \text{ are all different}, \\ x_k & \text{otherwise}. \end{cases}$$

– The ternary switching operation,

$$s(x, y, z) = \begin{cases} x & \text{if } y = z, \\ y & \text{if } x = z, \\ z & \text{otherwise}. \end{cases}$$
Proposition 6. If $\Gamma \subseteq R_D$, $|D| \geq 3$, and $l_{|D|} \in s\text{-Pol}(\Gamma)$ then QCSP($\Gamma$) is polynomial-time reducible to CSP($\Gamma'$) where $\Gamma' = \text{Inv}(\text{Pol}(\Gamma))$. In particular, QCSP($\Gamma$) is in $\text{NP}$.

Proof. See [3].

Theorem 8. Let $\Delta \subseteq \Gamma \subseteq R_D$, and $|D| \geq 3$.
- If $s\text{-Pol}(\Gamma)$ contains the dual discriminator $d$, or the switching operation $s$, or an affine operation, then QCSP($\Gamma$) is in $\text{PTIME}$;
- else, if $s\text{-Pol}(\Gamma)$ contains $l_{|D|}$, then QCSP($\Gamma$) is $\text{NP}$-complete;
- else QCSP($\Gamma$) is $\text{PSPACE}$-complete.

Proof. Chapter 5 of [30] shows that, either $s\text{-Pol}(\Gamma)$ consists of all projections (that is, all functions of the form $f(x_1, \ldots, x_n) = x_i$ for some $1 \leq i \leq n$), or else $s\text{-Pol}(\Gamma)$ contains the dual discriminator operation, $d$, or the near-projection operation, $l_{|D|}$, or (when $|D| \in \{3, 4\}$) an affine operation. If $s\text{-Pol}(\Gamma)$ consists of all projections then, by Theorem 5, QCSP($\Gamma$) is $\text{PSPACE}$-complete. If $s\text{-Pol}(\Gamma)$ contains $d$ or an affine operation then, by Theorem 6 or Theorem 7 QCSP($\Gamma$) is tractable.

Suppose that $s\text{-Pol}(\Gamma)$ contains $l_{|D|}$. Then, by Proposition 6 this problem is polynomial-time reducible to CSP($\Gamma'$), where $\Gamma' = \text{Inv}(\text{Pol}(\Gamma))$. If $s\text{-Pol}(\Gamma)$ contains $s$ or $d$ then clearly so does $\text{Pol}(\Gamma')$. It follows from the results of [15] that in this case CSP($\Gamma'$) is tractable, and hence so is QCSP($\Gamma$). If $s\text{-Pol}(\Gamma)$ contains neither $s$ nor $d$ then, by [15], CSP($\Gamma$) is $\text{NP}$-complete. Since, obviously, CSP($\Gamma$) is polynomial-time reducible to QCSP($\Gamma$), and QCSP($\Gamma$) is in $\text{NP}$, the result follows.

Note that, for any fixed finite set $D$, the conditions in Theorem 8 can be efficiently checked.

7 Conclusion
We have shown that the algebraic theory relating complexity and polymorphisms, which was originally developed for the standard constraint satisfaction problem allowing only existential quantifiers, can be extended to deal with the more general framework of the quantified constraint satisfaction problem.

In this extension of the theory it turns out that it is the surjective polymorphisms of the predicates used in problem instances which determine the complexity of the corresponding problems. Using this information we have been able to identify subproblems of the quantified constraint satisfaction problem lying in (or complete for) some standard complexity classes, and to obtain a complete classification of complexity for certain special cases. We expect that by developing the results and ideas from this paper one will be able to successfully build a far-reaching theory linking algebraic properties of relations with the computational complexity of certain associated problems, just as it was in the case of ordinary constraint satisfaction problems.

Acknowledgement The authors thank Reinhard Pöschel who helped launch this project.
References

The aim of this tutorial is to give an overview on the state-of-the-art in infinite-state model checking and its applications.

We present a unified modeling framework based on word/term rewrite systems and show its relevance in reasoning about several important classes of systems (communication protocols, parametrized distributed algorithms, multi-threaded programs, etc).

Then, we address the verification problem of various classes of such models. We consider especially the basic problem of reachability analysis which consists in computing a (finite) representation of the (potentially infinite) set of reachable configurations.

We show the main existing approaches to tackle this problem:

- Specialized constructions for several significant classes of models for which this problem is shown to be decidable,
- General principles to prove the termination and the completeness of the iterative computation of the reachability sets for classes of models,
- Generic constructions and fixpoint acceleration techniques, leading to powerful semi-algorithms applicable to general classes of models in order to compute exact/approximate reachability sets.
Parity of Imperfection or Fixing Independence

Julian C. Bradfield

Laboratory for Foundations of Computer Science, School of Informatics, King’s Bldgs, University of Edinburgh, Edinburgh EH9 3JZ, UK
jcb@inf.ed.ac.uk

Abstract. We introduce a fixpoint extension of Hintikka and Sandu’s IF (independence-friendly) logic. We obtain some results on its complexity and expressive power. We relate it to parity games of imperfect information, and show its application to defining independence-friendly modal mu-calculi.

1 Introduction

Independence-friendly logic [7] is a logic introduced by Hintikka and Sandu which gives an alternative account of branching quantifiers (Henkin quantifiers) in terms of games of imperfect information. It allows the expression of quantifiers where the choice must be independent of specified earlier choices; it has existential second-order power. In the last few years, it has attracted study from both philosophical logicians and mathematical logicians. Also, in earlier work, we have argued that its modal analogues have a role to play in concurrency theory. (See [2] and [3] for discussions of this role and its relation to other work in concurrency theory.)

Given a first-order logic, or a logic like IF that is supposed to look first-order (even though it isn’t), it is natural to want to add fixpoint operators. One motivation is just the mathematical interest of studying inductive definability in many contexts; a more computer-science-based motivation is the desire to be able to produce an IF analogue of the modal mu-calculus, a popular and interesting temporal logic.

In [2], we asserted that using the semantics given to IF by Hodges [8], it was possible to define an IF fixpoint logic. In this article, we give a detailed definition of IF least fixpoint logic (which, typically of IF logics, is a little more subtle than one first thinks), and then study it.

In section 2 we deal with the preliminaries, the existing syntax and semantics of IF logic. Sections 3, 4 and 5 are the main part of the paper; in section 3 we give the detailed definitions of IF fixpoint logic and its semantics; in section 4 we give a couple of interesting examples; and in section 5 we establish some partial results on complexity and expressive power. Then in section 6 we return to the game-theoretic roots of IF by giving a suitable notion of parity game of imperfect information, which gives an alternative semantics for IF fixpoint logic. Finally, in section 7 we briefly sketch the application to IF modal mu-calculus that was one of the original motivations for looking at IF with fixpoints.
2 IF-FOL Syntax and Semantics

First of all, we state one important notational convention: we take the scope of all quantifiers and fixpoint operators to extend as far to the right as possible.

For the purposes of this article, we will use only a sublanguage of IF-FOL (sometimes just IF for short). The full languages advocated by Hintikka and analysed by Hodges and others include the possibility of conjunctions and disjunctions that are independent of previous quantifiers. These operators do not introduce inherently new problems, but they do introduce some additional complexity (and space) in defining the semantics. We will therefore ignore them, and consider only the independent quantifiers; the interested reader can use [8] to put back the independent junctions.

One of the more tedious features of IF-FOL is the need to be more pedantic than usual in keeping track of free variables etc., as not all the things one takes for granted in usual logic are true in IF-FOL. When introducing fixpoint operators, even more care is needed, and we shall therefore give the semantics even more pedantically than Hodges did.

**Definition 1.** Assume the usual FOL set of proposition (P, Q etc.), relation (R, S etc.), function (f, g etc.) and constant (a, b etc.) symbols, with given arities. Assume also the usual variables v, x etc. We write \( \vec{x}, \vec{v} \) etc. for tuples of variables, and similarly for tuples of other objects; we use concatenation of symbols to denote concatenation of tuples with tuples or objects.

For formulae \( \phi \) and terms \( t \), the (meta-level) notations \( \phi[\vec{x}] \) and \( t[\vec{x}] \) mean that the free variables of \( \phi \) or \( t \) are included in the variables \( \vec{x} \), without repetition\(^1\).

The terms of IF-FOL are as usual constructed from variables, constants and function symbols. The free variables of a term are as usual; the free variables of a tuple of terms are the union of the free variables of the terms.

We assume equality = is in the language, and atomic formulae are defined as usual. The free variables of the formula \( R(\vec{t}) \) are those of \( \vec{t} \).

The compound formulae are given as follows:

**Conjunction and disjunction.** If \( \phi[\vec{x}] \) and \( \psi[\vec{y}] \) are formulae, then \( (\phi \lor \psi)[\vec{z}] \) and \( (\phi \land \psi)[\vec{z}] \) are formulae, where \( \vec{z} \) is the union of \( \vec{x} \) and \( \vec{y} \).

**Quantifiers.** If \( \phi[\vec{y}, x] \) is a formula, \( x \) a variable, and \( W \) a finite set of variables, then \( (\forall x/W. \phi)[\vec{y}] \) and \( (\exists x/W. \phi)[\vec{y}] \) are formulae. If \( W \) is empty, we write just \( \forall x. \phi \) and \( \exists x. \phi \).

**Game negation.** If \( \phi[\vec{x}] \) is a formula, so is \( (\sim \phi)[\vec{x}] \).

**Flattening.** If \( \phi[\vec{x}] \) is a formula, so is \( (\downarrow \phi)[\vec{x}] \).

(Negation. \( \neg \phi \) is an abbreviation for \( \sim \downarrow \phi \).)

**Definition 2.** IF-FOL\(^+\) is the logic in which \( \sim \), \( \downarrow \) and \( \neg \) are applied only to atomic formulae.

\(^1\)[8] writes \( \phi(\vec{x}) \), but we wish to distinguish the meta-notation for free variables from the object-level syntax for atomic formulae and the meta-notation for assigning values to variables.
In the independent quantifiers the intention is that $W$ is the set of independent variables, whose values the player is not allowed to know at this choice point: the Henkin quantifier $\forall x \exists y \phi_{x,y}$ can be written as $\forall x/\emptyset. \exists y/\emptyset. \forall u/\{x, y\}. \exists v/\{x, y\}$. If one then plays the usual model-checking game with this additional condition, which can be formalized by requiring strategies to be uniform in the ‘unknown’ variables, one gets a game semantics which characterizes the Skolem function semantics in the sense that Eloise has a winning strategy iff the formula is true. However, these games are not determined, so it is not true that Abelard has a winning strategy iff the formula is untrue. For example, $\forall x. y/x. x = y$ (or $\forall x. \exists y/\{x\}. x = y$) is untrue in any structure with more than one element, but Abelard has no winning strategy.

The trump semantics of Hodges [9], with variants by others, gives a Tarski-style semantics for this logic, equivalent to the imperfect information game semantics given by Hintikka and Sandu. The semantics is as follows:

**Definition 3.** Let a structure $A$ be given, with constants, propositions and relations interpreted in the usual way. A deal $\bar{a}$ for $\phi[\bar{x}]$ or $\bar{t}[\bar{x}]$ is an assignment of an element of $A$ to each variable in $\bar{x}$. Given a deal $\bar{a}$ for a tuple of terms $\bar{t}[\bar{x}]$, let $\bar{t}(\bar{a})$ denote the tuple of elements obtained by evaluating the terms under the deal $\bar{a}$.

If $\phi[\bar{x}]$ is a formula and $W$ is a subset of the variables in $\bar{x}$, two deals $\bar{a}$ and $\bar{b}$ for $\phi$ are $\simeq_{W}$-equivalent ($\bar{a} \simeq_{W} \bar{b}$) iff they agree on the variables not in $W$. A $\simeq_{W}$-set is a non-empty set of pairwise $\simeq_{W}$-equivalent deals.

The interpretation $[\phi]$ of a formula is a pair $(T, C)$ where $T$ is the set of trumps, and $C$ is the set of cotrumps.

- If $(R(\bar{t}))[\bar{x}]$ is atomic, then a non-empty set $D$ of deals is a trump iff $\bar{t}(\bar{a}) \in R$ for every $\bar{a} \in D$; $D$ is a cotrump iff it is non-empty and $\bar{t}(\bar{a}) \notin R$ for every $\bar{a} \in D$.
- $D$ is a trump for $(\phi \land \psi)[\bar{x}]$ iff $D$ is a trump for $\phi[\bar{x}]$ and $D$ is a trump for $\psi[\bar{x}]$; $D$ is a cotrump iff there are cotrumps $E, F$ for $\phi, \psi$ such that every deal in $D$ is an element of either $E$ or $F$.
- $D$ is a trump for $(\phi \lor \psi)[\bar{x}]$ iff it is non-empty and there are trumps $E$ of $\phi$ and $F$ of $\psi$ such that every deal in $D$ belongs either to $E$ or $F$; $D$ is a cotrump iff it is a cotrump for both $\phi$ and $\psi$.
- $D$ is a trump for $(\forall y/W. \psi)[\bar{x}]$ iff the set $\{ \bar{a}b \mid \bar{a} \in D, b \in A \}$ is a trump for $\psi[\bar{x}, y]$. $D$ is a cotrump iff it is non-empty and there is a cotrump $E$ for $\psi[\bar{x}, y]$ such that for every $\simeq_{W}$-set $F \subseteq D$ there is a $b$ such that $\{ \bar{a}b \mid \bar{a} \in F \} \subseteq E$.
- $D$ is a trump for $(\exists y/W. \psi)[\bar{x}]$ trump iff there is a trump $E$ for $\psi[\bar{x}, y]$ such that for every $\simeq_{W}$-set $F \subseteq D$ there is a $b$ such that $\{ \bar{a}b \mid \bar{a} \in F \} \subseteq E$; $D$ is a cotrump iff the set $\{ \bar{a}b \mid \bar{a} \in D, b \in A \}$ is a cotrump for $\psi[\bar{x}, y]$.
- $D$ is a trump for $\sim \phi$ iff $D$ is a cotrump for $\phi$; $D$ is a cotrump for $\sim \phi$ iff it is a trump for $\phi$.
- $D$ is a trump (cotrump) for $\downarrow \phi$ iff $D$ is a non-empty set of members (non-members) of trumps of $\phi$. 
A sentence is true in the usual sense if \(\{\langle\rangle\} \in T\) (the empty deal is a trump set), and false in the usual sense if \(\{\langle\rangle\} \in C\); this corresponds to Eloise or Abelard having a uniform winning strategy. Otherwise, it is undetermined.

Note that the game negation \(\sim\) provides the usual de Morgan dualities.

A trump for \(\phi\) is essentially a set of winning positions for the model-checking game for \(\phi\), for a given uniform strategy, that is, a strategy where choices are uniform in the ‘hidden’ variables. The most intricate part of the above definition is the clause for \(\exists y/W.\psi\): it says that a trump for \(\exists y/W.\psi\) is got by adding a witness for \(y\), uniform in the \(W\)-variables, to trumps for \(\psi\).

It is easy to see that any subset of a trump is a trump. In the case of an ordinary first-order \(\phi(\vec{x})\), the set of trumps of \(\phi\) is just the power set of the set of tuples satisfying \(\phi\). To see how a more complex set of trumps emerges, consider the following formula, which has \(x\) free: \(\exists y/\{x\}.x = y\). Any singleton set of deals is a trump, but no other set of deals is a trump. Thus we obtain that \(\forall x.\exists y/\{x\}.x = y\) has no trumps (unless the domain has only one element).

The following definition is for later convenience: a set \(T\) of sets of deals is well-dealt if for every \(D \in T\), \(D\) is non-empty and \(D' \in T\) for every non-empty \(D' \subseteq D\). A formula has well-dealt semantics \((T, C)\) if \(T\) and \(C\) are well-dealt; the above semantics ensures that all IF-FOL formulae have well-dealt semantics.

\[\text{[8]}\] shows that every well-dealt set is the semantics of some IF formula (given suitable atomic relations), giving us

**Proposition 4.** On a structure \(A\) with \(n\) elements, IF formulae of length \(m\) require space exponential in \(n^m\) to represent their semantics.

**Proof.** The set of tuples for \(m\) free variables has \(n^m\) elements; Given a \(k\) element set, there are \(2^k\) subsets, but not all sets of subsets are well-dealt; however, there are about \(2^k/\sqrt{k}\) sets of size \(k/2\), and hence at least \(2^{2^k/\sqrt{k}}\) well-dealt sets of subsets. (Cameron and Hodges [4] look in more detail at the combinatorics of trumps.) \(\square\)

We can record the easy loose upper bounds on the time complexity of IF-FOL operations:

**Proposition 5.** In a structure \(A\) of size \(n\), the trump components of the IF operators can be calculated in the following times on formulae with \(m\) free variables, where \(k = n^m\): \(\lor\) and \(\land\) in \(2^{k+1} \cdot k^2\); \(\forall x\) in \(2^k \cdot k^3 n\); \(\exists x/W\) in \(2^{k+k \log n}\).

**Proof.** A crude analysis of the cost of computing the trump semantics more or less directly from the definitions. Note that the computation for \(\exists\) has further exponential factors above the \(2^k\) from the number of possible trumps, effectively due to the computation of choice functions. \(\square\)

In the case of IF, these exponential upper bounds are much worse than is really required for determining whether a deal satisfies (i.e. is a singleton trump for) an IF formula, since IF expressible properties are in NP (because we can guess values for choice functions).
3 Adding Fixpoint Operators

The prime motivation for considering fixpoint extensions is in the modal setting, where it is a standard way to produce temporal logics from modal logics. However, fixpoint extensions to IF logics raise a number of issues, and it is useful to recall briefly the first-order case.

In the classical settings, fixpoint operators are added to allow sets or relations to be inductively defined by formulae: $\mu(x, X).\phi(x, X)$, where $X$ is a set variable, is the least set $A$ such that $A = \{ x \mid \phi(x, A) \}$, and the syntax of formulae is extended to allow terms of the form $t \in X$ or $t \in \mu(x, X).\phi(x, X)$ (among set theorists) or $X(t)$ and $(\mu(x, X).\phi(x, X))(t)$ (among finite model theorists).

In applying this directly to IF-FOL, there is the obvious problem that we no longer have a simple notion of an element satisfying a formula, so the usual definition no longer type-checks. There are two possible approaches, depending on how one views the use of fixpoint terms. If one takes the view that their purpose is to define sets, and the logic is a means to this end, then it is natural to retain the use of set variables, and work out how to make $\phi(x, X)$ reduce to a boolean. On the other hand, if one views fixpoint operators as a means of introducing recursion into the logical formulae, it is more natural to decide that fixpoint terms should have the same semantics as other formulae, namely sets of trumps, and that therefore the variables $X$ range over trump sets rather than sets. We then have to decide the meaning of $X(t)$. This is the approach we suggested in [2], and will now pursue.

**Definition 6.** IF-LFP extends the syntax of IF-FOL as follows:

- There is a set $\text{Var} = \{ X, Y, \ldots \}$ of fixpoint variables. Each variable $X$ has an arity $(\text{ar}_1(X), \text{ar}_2(X))$; $\text{ar}_1(X)$ is the arity of the fixpoint, and $\text{ar}_2(X)$ is the number of free parameters of the fixpoint.
- If $X$ is a fixpoint variable, and $\vec{t}$ an $\text{ar}_1(X)$-vector of terms then $X(\vec{t})$ is a formula.
- The notation $\phi(X)$ indicates that $X$ is among the free fixpoint variables of $\phi$. If $\phi(X)[\vec{x}, \vec{z}]$ is a formula with $\text{ar}_1(X)$ free individual variables $\vec{x}$ and $\text{ar}_2(X)$ free individual variables $\vec{z}$, and $\vec{t}$ is a sequence of $\text{ar}_1(X)$ terms with free variables $\vec{y}$, then $(\mu(X, \vec{x}).\phi(\vec{t}))[\vec{z}, \vec{y}]$ is a formula; provided that $\phi$ is IF-FOL$^+$.
- Similarly for $\nu(X, \vec{x}).\phi$.

The process of extending the trump semantics to fixpoint formulae is not entirely straightforward. First we define valuations for free fixpoint variables.

**Definition 7.** A fixpoint valuation $\mathcal{V}$ maps each fixpoint variable $X$ to a pair $(\mathcal{V}_T(X), \mathcal{V}_C(X)) \in (\wp((\wp(\text{Var})))^{\text{ar}_1(X)}+\text{ar}_2(X)))^2$.

Let $D$ be a non-empty set of deals for $X(\vec{t})[\vec{x}, \vec{z}, \vec{y}]$, where $\vec{y}$ are the free variables of $\vec{t}$ not already among $\vec{x}, \vec{z}$. A deal $d = \vec{a}\vec{c}\vec{b} \in D$, where $\vec{a}, \vec{c}, \vec{b}$ are the deals for $\vec{x}, \vec{z}, \vec{y}$ respectively, determines a deal $d' = \vec{t}(d)\vec{c}$ for $X[\vec{x}, \vec{z}]$. Let $D' = \{ d' \mid d \in D \}$. $D$ is a trump for $X(\vec{t})$ iff $D' \in \mathcal{V}_T(X)$; it is a cotrump iff $D' \in \mathcal{V}_C(X)$. 
Then we define a suitable complete partial order on denotations:

**Definition 8.** If \((T_1, C_1)\) and \((T_2, C_2)\) are elements of \(\wp(\wp(A^n))^2\), define \((T_1, C_1) \preceq (T_2, C_2)\) iff \(T_1 \subseteq T_2\) and \(C_1 \supseteq C_2\).

**Lemma 9.** If \(\phi(X)[\vec{z}]\) is an IF-FOL\(^+\) formula and \(\mathcal{V}\) is a fixpoint valuation, the map on \(\wp(\wp(A^{\text{ar}_1}(X) + \text{ar}_2(X)))^2\) given by

\[
(T, C) \mapsto \llbracket \phi \rrbracket_{\mathcal{V}[X := (T, C)]}
\]

is monotone with respect to \(\preceq\); hence it has least and greatest fixpoints, with ordinal approximants defined in the usual way.

**Definition 10.** \(\llbracket \mu(X, x).\phi(X)[\vec{x}, \vec{z}] \rrbracket\) is the least fixpoint of the map just defined; \(\llbracket \nu(X, x).\phi(x)[\vec{x}, \vec{z}] \rrbracket\) is the greatest fixpoint. \(\mu(X, x).\phi\) means the \(\zeta\)th approximant of \(\mu(X, x)\).

The following lemma records the usual basic properties (which have to be checked again in this setting), and one new basic property, particular to the IF case.

**Lemma 11.** 1. The trump and cotrump components of \(\llbracket \mu(X, x).\phi \rrbracket\) are well-dealt.
2. If \(Y\) is free in \(\phi\), then \(\llbracket \mu(X, x).\phi \rrbracket\) is monotone in \(Y\); hence the definition extends to further fixpoints in the usual way, as does this lemma.
3. \(\mu\) and \(\nu\) are dual: \(T\) is a trump for \(\mu(X, x).\phi(X)\) iff it is a cotrump for \(\nu(X, x).\sim\phi(\sim X)\) (with the outer negation pushed in by duality).

**Proof.** (1) by induction on approximants; (2) as usual; (3) from definitions.

A distinctive feature of the definition, compared to the normal LFP definition, is the way that free variables are explicitly mentioned. Normally, one can fix values for the free variables, and then compute the fixpoint, but because of independent quantification this is not possible in the IF setting. For example, consider the formula fragment

\[
\forall z. \ldots \mu(X, x) \ldots \lor \exists y/\{z\}. X(y)
\]

The independent choice of \(y\) means that the trumps for the fixpoint depend on the possible deals for \(z\), not just a single deal.

Another point is that the trump set of a least fixpoint is the union of the trump sets of its approximants; but the interpretation of logical disjunction is not union of trump sets, but union of trumps (applied pointwise to the trump sets). Thus the usual view of a least fixpoint as a transfinite disjunction is not valid in general. The following explains why, despite this, the IF-LFP semantics is consistent with classical LFP semantics.

**Proposition 12.** Call a set \(T\) of trumps or cotrumps full iff it is the set of non-empty subsets of \(\bigcup T\). Call a formula \(\phi\) of IF-LFP classical iff it is in IF-FOL\(^+\) and it contains no independent quantification (i.e. all quantifiers are \(\exists x/\emptyset\) and \(\forall x/\emptyset\)). Then
1. if \( \phi \) is fixpoint free, then \( \llbracket \phi \rrbracket_{\text{IF}} = (T, C) \) is full, \( \bigcup T = \llbracket \neg \phi \rrbracket_{\text{FO}} \);  
2. if \( T_\zeta \) is a (transfinite) sequence of full well-dealt deal sets, then \( \bigcup \zeta T_\zeta \) and \( \bigcap \zeta T_\zeta \) are full well-dealt sets;  
3. hence (1) is true for any classical IF-LFP formula.

4 Examples of IF-LFP

IF logic is not entirely easy to understand and mu-calculi are also traditionally hard to understand, so we now consider some examples that demonstrate interesting features of the combination. For convenience, we introduce the abbreviation \( \phi \Rightarrow \psi \) for \( \psi \lor \neg \phi \) provided that \( \phi \) is atomic.

Let \( G = (V, E) \) be a directed graph. The usual LFP formula \( R(y, z) \overset{\text{def}}{=} ( \mu(X, x).z = x \lor \exists w. E(x, w) \land X(w))(y) \) asserts that the vertex \( z \) is reachable from \( y \). Hence the formula \( \forall y. \forall z. R(y, z) \) asserts that \( G \) is strongly connected. Now consider the IF-LFP formula

\[
\forall y. \forall z. (\mu(X, x).z = x \lor \exists w/\{y, z\}. E(x, w) \land X(w))(y).
\]

At first sight, one might think this asserts not only that every \( z \) is reachable from every \( y \), but that the path taken is independent of the choice of \( y \) and \( z \). This is true exactly if \( G \) has a directed Hamiltonian cycle, a much harder property than being strongly connected.

Of course, the formula does not mean this, because the variable \( w \) is fresh each time the fixpoint is unfolded. In the trump semantics, the denotation of the fixpoint will include all the possible choice functions at each step, and hence all possible combinations of choice functions. Thus the formula reduces to strong connectivity.

It may be useful to look at the approximants of this formula in a little more detail, to get some intuitions about the trump semantics. Considering just

\[
H \overset{\text{def}}{=} (\mu(X, x).z = x \lor \exists w/\{y, z\}. E(x, w) \land X(w))[x, y, z],
\]

we see that in computing each approximant, the calculation of \( \llbracket \exists w/\{y, z\} \ldots \rrbracket \) involves generating a trump for every possible value of a choice function \( f: x \mapsto w \). This is a feature of the original trump semantics, and can be understood by viewing it as a second-order semantics: just as the compositional Tarskian semantics of \( \exists x. \phi(x) \) involves computing all the witnesses for \( \phi(x) \), so computing the trumps of \( \exists x/\{y\}. \phi \) involves computing all the Skolem functions; and unlike the first-order case, it is necessary to work with functions (as IF can express existential second-order logic). Consequently, the \( n \)th approximant includes all states such that \( x \rightarrow f_1(x) \rightarrow f_2f_1(x) \rightarrow \ldots \rightarrow f_n \ldots f_1(x) = z \) for any sequence of successor-choosing functions \( f_i \). Thus we see that the cumulative effect is the same as for a normal \( \exists w \), and the independent choice has indeed not bought us anything.
It is, however, possible\textsuperscript{2} to produce a slightly more involved formula expressing the Hamiltonian cycle property in this inductively defined way, by using the standard trick for expressing functions in Henkin quantifier logics. We replace the formula $H$ by

$$\forall s. \exists t/\{y, z\}. E(s, t) \land \mu(X, x). x = z \lor \forall u. \exists v/\{x, z, s, t\}. (s = u \Rightarrow t = v) \land (x = u \Rightarrow X(v)).$$

This works because the actual function $f$ selecting a successor for every node is made outside the fixpoint by $\forall s. \exists t/\{y, z\}. E(s, t) \land \ldots$; then inside the fixpoint, a new choice function $g$ is made so that $X(g(x))$, and $g$ is constrained to be the same as $f$ by the clause $(s = u \Rightarrow t = v)$. (The reader who is not familiar with the IF/Henkin to existential second-order translation might wish to ponder why $\forall s. \exists t/\{y, z\}. E(s, t) \land \mu(X, x). x = z \lor (x = s \Rightarrow X(t))$ does not work.)

5 Complexity and Expressive Power of IF-LFP

The above examples have shown IF-LFP being used to express relatively simple NP properties. Since, as remarked, it is well known that Henkin quantifiers and IF logic express just the NP properties, and since it is also known\textsuperscript{6} that LFP plus Henkin quantifiers express $P_{\text{NP}}$, one might imagine that IF-LFP (which is not closed under classical negation) also expresses only NP properties, or at worst some subset of $P_{\text{NP}}$. This is not the case; adding fixpoints to the IF formulation gives a more significant increase in expressive power.

Firstly, we note that the approximant semantics of fixpoints gives the usual behaviour in simple upper bounds:

**Proposition 13.** If $\phi(X)[x, z_1, \ldots, z_m]$ is an IF-FOL\textsuperscript{+} formula, then in a structure of size $n$, the approximants of $\mu(X, x). \phi$ close after at most $2^{n^m}$ steps. Hence in an IF-LFP formula with $d$ alternating fixpoints and $m$ variables, $2^{d n^m}$ evaluations of IF formulae are required. If the formula size is $l$, this gives a total cost of $2^{d n^m} \cdot l \cdot 2^{n^m(1+\lg n)} = l \cdot 2^{n^m(1+d+\lg n)}$.

Observe, however, that the contribution from fixpoint alternation is small compared to the cost of computing independent existential quantifiers.

Despite the relative weakness of adding fixpoints, they do in some sense release the power of independent quantification. This is shown by the following theorem.

**Theorem 14.** There is an IF-LFP sentence (with one least fixpoint) which is EXPTIME-hard to evaluate.

\textsuperscript{2} Since IF logic is equi-expressive with Henkin quantified logic, it is also equi-expressive with existential second-order logic, and so can express ‘Hamiltonian cycle’ without using fixpoints. Thus we are not, in this example, adding technical expressive power. However, the pure IF definition is quite complex, as it involves defining a binary relation coded via functions; so we are adding expressive convenience.
Proof. We give a reduction from the EXPTIME-complete problem of determining whether Player 1 has a winning strategy for the game of generalized chess.

A structure for a generalized chess game between 1 and 2 of order \( n \) comprises a board \( R \) with \( n^2 \) (or any other fixed polynomial) squares \( r \) and a set \( P \) of \( n \) (or any other fixed polynomial) pieces \( p \). A position of the game is a function \( \pi: P \rightarrow R \). There may be some relations on \( P \) and \( R \) in the signature. The game is defined by three first-order formulae with parameter \( \pi \): a formula \( \phi_1(\pi) \) true only of the initial position, a formula \( \phi_W(\pi) \) which is true if player 1 has won at \( \pi \), and a formula \( \phi_M(\pi, i, p, r) \) which is true if moving piece \( p \) to square \( r \) is a legal move for player \( i \) from position \( \pi \). (Without loss of generality, we assume that a move consists of moving exactly one piece.)

Given a position \( \pi \) and a move \( p, r \), the ‘next position’ formula \( N(\pi') \) is defined to be \( \forall p'. (p' = p \land \pi'(p) = r) \lor \pi'(p') = \pi(p') \).

The set \( X \) of winning positions (i.e. from which 1 can force a win) for 1 can then be inductively defined by the type 3 functional

\[
F(X, \pi) \iff \Phi \stackrel{\text{def}}{=} \phi_W(\pi) \lor ((\forall p, r. \phi_M(\pi, 2, p, r) \Rightarrow \exists \pi'. N(\pi') \land X(\pi'))) \\
\land (\exists p, r. \phi_M(\pi, 1, p, r) \land \exists \pi'. N(\pi') \land X(\pi')))\]

We now show how to express this inductive definition in IF-LFP. Part of the coding is the well-known expression of existential second-order logic in IF or Henkin logic, which we have already seen in the Hamiltonian cycle example. The general technique is thus: assume given an ESO formula \( \exists f. \psi \). Let \( Q_1(f(\tau_1)), \ldots, Q_n(f(\tau_n)) \) be the instances in \( \psi \) of applications of \( f \) occurring in atoms \( Q_i \). Then the translation is

\[
\forall x. \exists y. \forall x_1. \exists y_1/\{x\}. \ldots \forall x_n. \exists y_n/\{x, x_1, \ldots, x_{n-1}\}. \bigwedge_i (x_i = x \Rightarrow y_i = y) \land \hat{\psi},
\]

where \( \hat{\psi} \) is obtained from \( \psi \) by replacing \( Q_i(f_i(\tau_i)) \) with \( x_i = \tau_i \Rightarrow Q_i(y_i) \).

The second part is passing a function through a fixpoint. This is fairly simple to do: one just passes the domain and codomain as normal parameters, and relies on the quantification outside forcing them to represent a function. In this case, the classical type 2 relation \( X(\pi) \) is replaced by a binary IF type 1 relation \( Y(s, t) \), so that the classical \( \exists \pi. (\mu X. \Phi)(\pi) \) becomes \( \forall x. \exists y. (\mu Y. \Phi)(x, y) \), where \( \Phi \) is obtained from \( \Phi \) by applying the ESO–IF translation using \( s, t \) for \( \pi \) and replacing \( X(\pi') \) with \( Y(p'', r'') \), where \( p'', r'' \) are the variables bound by the translation \( \forall p''. \exists r''/\ldots. \) of \( \exists \pi' \).

One then shows by an inductive argument on \( \zeta \) that \( \exists \pi. (\mu^\zeta X. \Phi)(\pi) \) holds iff \( \forall x. \exists y. (\mu^\zeta Y. \Phi)(x, y) \) holds. Finally, if we wish to determine whether the initial position is winning for 1, we evaluate \( \exists \pi. \phi_1(\pi) \land (\mu X. \Phi)(\pi) \).

(We should note that we have extended the IF abbreviation \( \phi \Rightarrow \psi \) to the case where \( \phi \) is classical, not just atomic. This is acceptable because game negation coincides with classical negation for classical formula.)

The above argument was applied to the case of finite structures, but there is nothing in it that depends on finiteness. We can therefore obtain the following theorem, which refutes our conjecture in [2] that a fixpoint extension of IF would be within \( \Delta^1_2 \).
Theorem 15. Let $F(X,\alpha)$ be a positive $\Sigma^1_3$ type 3 functional in the language of arithmetic. Then a set of integers definable from the set of reals inductively defined by $F$ can be expressed in IF-LFP. It follows that that IF-LFP (even with just one fixpoint) over the natural numbers can express $\Sigma^1_2$ properties.

Proof. $F$ is defined by a $\Sigma^1_1$ formula $\phi(X,\alpha)$. Use the technique of the previous proof to express $\exists \alpha. (\mu X.\phi(\alpha)) \land \psi(\alpha, n)$, where $\psi$ is first-order. Cenzer [5] showed that any $\Sigma^1_2$ set of reals is the closure of a $\Sigma^1_1$ positive inductive definition over the reals. Since if $\alpha$ is a $\Sigma^1_2$ real, the set $\{\alpha\}$ is also $\Sigma^1_2$, we also have the stated consequence. □

Cenzer’s results also allow us to obtain an improvement (for those who don’t believe CH) on the closure ordinal for a single IF fixpoint over $\omega$. The usual cardinality argument for fixpoints tells us merely that an IF fixpoint over $\omega$ must close by $2^{\aleph_0}$. The improvement is

Theorem 16. If $\phi(X)$ is an IF-FOL$^+$ formula (i.e. with $\sim$ and $\downarrow$ applied only to atoms), then $\mu X.\phi$ has closure ordinal $\leq \aleph_1$.

Proof. Seen as operations on $\wp(2^\omega)$, the semantics of the IF boolean operators and quantifiers are $\Sigma^1_1$. (This is not immediately apparent from the definitions as presented above, but a small amount of rearrangement reveals it.) Cenzer showed that the closure ordinal of a $\Sigma^1_1$ monotone inductive definition over the reals is $\leq \aleph_1$. □

It remains to investigate lower bounds on the complexity of multiple IF fixpoints. We remark only that the absence of classical negation makes this less easy than it otherwise would be.

6 IF Parity Games

We briefly recall the game semantics of first-order logic and of IF logic.

Given a FO formula $\psi$ (in positive form) and a structure $A$, a position is a subformula $\phi(\vec{x})$ of $\psi$ together with a deal for $\phi$, that is, an assignment of values $\vec{v}$ to its free variables $\vec{x}$. At a position $(\forall x. \phi_1, \vec{v})$, Abelard chooses a value $v$ for $x$, and play moves to the position $(\phi_1, \vec{v} \cdot v)$; similarly Eloise moves at $\exists x. \phi$. At $(\phi_1 \land \phi_2, \vec{v})$, Abelard chooses a conjunct $\phi_i$, and play moves to $(\phi_i(\vec{x}', \vec{v}'))$, where $\vec{x}', \vec{v}'$ are $\vec{x}, \vec{v}$ restricted to the free variables of $\phi_i$; and at $(\phi_1 \lor \phi_2, \vec{v})$, Eloise similarly chooses a disjunct. A play of the game terminates at (negated) atoms $(P(\vec{x}), \vec{v})$ (resp. $(\neg P(\vec{x}), \vec{v})$), and is won by Eloise (resp. Abelard) iff $P(\vec{v})$ is true. Then it is standard that $M \models \phi$ exactly if Eloise has a winning strategy in this game, where a strategy is a function from sequences of legal positions to moves.

These games have perfect information; both players know everything that has happened, and in particular when one player makes a choice, they know the other player’s previous choices. Game semantics for IF logic 7 use games of imperfect information: at the position $\exists x/W. \phi$, when Eloise chooses a value $v$ for $x$, she does not know what Abelard chose for the values of the independent
variables $W$. A uniform Eloise strategy for the game is one in which her choice of $v$ is indeed uniform in the values of $W$, and we say a formula is true if Eloise has a uniform winning strategy.

Now recall that in a parity game the positions are assigned ranks $0, \ldots, r$, and if a run of the game is infinite, Eloise wins if the highest rank appearing infinitely often is even. The model-checking game for FOL extends to a model-checking game for LFP by assigning even ranks to maximal fixpoints and odd to minimal, such that the rank of an inner fixpoint is less than the rank of its enclosing fixpoints. Then the formula is true iff Eloise has a winning strategy for the defined parity game.

Combining these two concepts, a general parity game of imperfect information is given by a usual parity game together with imperfect information requirements at each position, requiring a player to move uniformly in some part of the game history. The winning runs are those given by the usual parity winning conditions; a player wins the game if she has winning strategy for the parity game that is uniform as required by the imperfect information requirements.

In general, infinite imperfect information games are undecidable even on finite structures, since they require players to keep arbitrary knowledge (and lack of knowledge) of the history of the game. To obtain a class of decidable imperfect parity games, we will first give a parity game semantics for IF-LFP, and then define a class of imperfect parity games characterized by IF-LFP.

**Definition 17.** The model-checking game for an IF-LFP formula is defined by adding the following clauses to the Hintikka–Sandu game for IF. The moves are extended by the usual fixpoint unfolding rule: at a position $((\mu(X,x).\phi)(t), \vec{u})$, play moves to $(\phi, \vec{uv})$, where $v$ is the value of $t$; at a position $(X(t), \vec{uvw})$, where $\vec{u}$ is the deal for the free variables of $X$, $v$ for $x$, and $\vec{w}$ for the variables bound inside $\phi$, play moves to $(\phi, \vec{uv})$ where $v'$ is the value of $t$. Parities are assigned to positions in the usual way, and the usual infinite parity winning condition is added.

The independence requirements are that at a quantifier $\exists x/W$. (and dually), Eloise must choose $x$ without knowing the values of the $W$ variables and without knowing the values of any variables bound in some currently enclosing fixpoint but chosen before the most recent unfolding of that fixpoint. (In other words, she does not remember choices that have gone out of scope and have no value in the current deal.)

Correspondingly, a uniform strategy in the parity game is a strategy where the choice function is uniform in the independent variables and the out-of-scope variables.

**Theorem 18.** If $\phi$ is an IF-LFP sentence, then Eloise has a uniform winning strategy for the model-checking game if and only if $\{()\}$ is a trump for $\phi$. Moreover, the strategy can be history-free.

**Proof.** (Sketch) The argument relating parity conditions to alternating fixpoints applies to any set of monotone operators with fixpoints added, not just to FOL
or modal logic. Thus by combining the usual proof of equivalence of parity games and fixpoints with the proof of equivalence of IF games and trump semantics, we get the result.

History-freeness follows from the inductive construction of the set of winning positions, but is intuitively obvious, as all the information a player is supposed to be able to remember is included in the game position.

This game account of the IF-LFP semantics brings out the key factor, which may have been less obvious in the trump semantics, that keeps model-checking decidable. This is that passing through a fixpoint variable throws away all information about choices made within the body of the fixpoint, unless they are explicitly passed as parameters. Of course, this is also true in usual LFP, but in the IF case knowledge of previous choices is explicitly part of the semantics.

This suggests the following definition:

**Definition 19.** An imperfect information parity game on a structure $A$ is finite-memory if each player is equipped with a finite memory in which they can remember previous moves. A player’s choice at a move is required to depend only on the current position and memory, with additional imperfect information requirements imposed by the game on the memory (i.e. a player may have to temporarily forget things).

A player wins the game if they have a uniform history-free winning strategy.

The expected theorem is

**Theorem 20.** Given a finite-memory imperfect parity game on $A$, the statement ‘Eloise wins the game’ is expressible by an IF-LFP formula whose fixpoint alternation depth is the parity rank of the game.

**Proof.** (Sketch) The finite memory is modelled by parameters of fixpoints. We will use fixpoints $X$ which carry one parameter $p$ for the position in the game, and parameters $m_i$ for the memory ‘cells’. The inner loop of an inductive definition of winning positions is the usual expression of ‘it is Eloise’s move and there exists a move such that the next position is in $X$, or it is Abelard’s move and all next moves are in $X’’, as in the formula we used early for generalized chess. The quantifiers are made explicitly independent of the memory items required to be unknown (which may require a case analysis of the moves of the game).

To deal with the parities, we use the first-order version of the usual ‘parity game formula’ from parity automata and modal mu-calculus (see [1] for a detailed explanation of the parity game formula): for each rank $j = 0, \ldots, r$, there is a fixpoint variable $X_j$. Then the inner loop is enclosed by $\nu X_0, \mu X_1, \ldots, \mu/\nu X_r$, and the formula $X(p, \vec{m})$, where $p$ and $\vec{m}$ are the position and memory after the next move, is conjoined with

$$\bigwedge_{0 \leq j \leq r} (R_j \Rightarrow X_j(p, \vec{m}))$$

where $R_j$ is the formula expressing that the next position has rank $j$.

The usual proof now applies to give the result. \qed
Corollary 21. Finite-memory imperfect parity games on finite structures are decidable.

7 Application to IF Modal Mu-Calculus

Our original motivation for looking at fixpoint extensions of IF logic was the desire to combine two threads of work. Firstly, modal mu-calculus is a well studied and widely used temporal logic. Secondly, we have argued in [2] and [3] that modal versions of Henkin quantifiers and independence logics provide a natural expression of some properties of concurrent systems. Given a concurrent modal logic, it is natural to extend it to a concurrent temporal logic by adding fixpoint operators. In [2] we looked at modal analogues of Henkin quantifiers acting on systems composed of several concurrent components; since a single Henkin quantifier gives an operator on the powerset of states, there was no difficulty in adding such modalities to mu-calculus. In [3], we designed a modal analogue of IF logic, defined on certain structures appropriate for true concurrency. The full definition of the structures and the logic is, for a number of technical reasons, rather long and complex. We refer the reader to [3] for details, and here just give the idea.

IFML extends the syntax of usual modal logic as follows. Instead of the simple ‘next step’ modality $\langle a \rangle \Phi$, each modality carries a tag $\alpha$, and may be declared to be independent of previous tags $\beta$ by the Hintikka slash, giving a syntax $\langle a \rangle_{\alpha/\beta} \Phi$. The intended interpretation is that the choice of a action must be independent of the action chosen in the modality tagged by $\beta$; for this to make sense, the action at $\beta$ should be concurrent (in the technical sense of event structures etc.) with the action at $\alpha$. The semantics is given in terms of runs (sequences of states) of the system, directly via an imperfect information model-checking game.

As we have not the space to import the full definition of IFML here, we shall not give detailed propositions in this section, but remarks that can be refined into theorems with the material in [3].

Remark 22. The game semantics of IFML given in [3] can be equivalently expressed by translating to IF as a meta-language (modulo the introduction of some fairly messy defined functions and relations on runs of the system) such that the main variable holding the state ranges over runs (as in the game), and auxiliary variables range over actions. Consequently, IFML has a trump semantics. The evaluation of a formula on a finite system is decidable, since the maximum length of runs that must be considered is bounded by the modal depth of the formula.

Remark 23. We can define an IF modal mu-calculus by adding fixpoint formulae of the form $\mu(X, \chi).\Phi$ and $X(\alpha)$, where the fixpoint variable $X$ has not only an implicit parameter for the current ‘state’, but also explicit parameters $\chi$ for tags to be passed through the fixpoint.

3 This result is probably long known in some form, but I do not know the reference.
This can be given a semantics via IF-LFP. However, since a ‘state’ in the semantics is a run, not a system state, it is not obvious that decidability of model-checking is maintained for IF mu-calculus. (We conjecture that it is, but some results from concurrency theory, such as the undecidability of hereditary history-preserving bisimulation, give some cause for doubt.)

The IF modal mu-calculus has a model-checking game that is an IF version of the usual parity games for modal logic, as done above for IF-LFP.

8 Conclusion

We have defined a suitable fixpoint extension of independence-friendly logic, and established some results. We have related it to parity games of imperfect information, and we have shown how it may be applied to the construction of independence-friendly modal mu-calculi.

For IF-LFP itself, there are still many questions remaining. Chief among these are better upper and lower bounds on the complexity of model-checking (in the finite case) and descriptive complexity (in the infinite case). We have shown that IF-LFP is more complex than we surmised in earlier work, and it is not unlikely that it will turn out to be much more complex.

References

Atomic Cut Elimination for Classical Logic

Kai Brünnler

Technische Universität Dresden, Fakultät Informatik
D-01062 Dresden, Germany
kai.bruennler@inf.tu-dresden.de

Abstract. System SKS is a set of rules for classical propositional logic presented in the calculus of structures. Like sequent systems and unlike natural deduction systems, it has an explicit cut rule, which is admissible. In contrast to sequent systems, the cut rule can easily be reduced to atomic form. This allows for a very simple cut elimination procedure based on plugging in parts of a proof, like normalisation in natural deduction and unlike cut elimination in the sequent calculus. It should thus be a good common starting point for investigations into both proof search as computation and proof normalisation as computation.

Keywords: sequent calculus, natural deduction, cut elimination, classical logic, atomic cut.

1 Introduction

The two well-known connections between proof theory and language design, proof search as computation and proof normalisation as computation, have mainly used different proof-theoretic formalisms. While designers of functional programming languages prefer natural deduction, because of the close correspondence between proof normalisation and reduction in related term calculi [4,8], designers of logic programming languages prefer the sequent calculus [7], because infinite choice and much of the unwanted non-determinism is limited to the cut rule, which can be eliminated.

System SKS [2] is a set of inference rules for classical propositional logic presented in a new formalism, the calculus of structures [5]. This system admits the good properties usually found in sequent systems: in particular, all rules that induce infinite choice in proof search are admissible. Thus, in principle, it is as suitable for proof search as systems in the sequent calculus. In this paper I will present a cut elimination procedure for SKS that is very similar to normalisation in natural deduction. It thus allows us to develop, at least for the case of classical logic, both the proof search and the proof normalisation paradigm of computation in the same formalism and starting from the same system of rules.

Cut elimination in the sequent calculus and normalisation in natural deduction, widely perceived as ‘morally the same’, differ quite a bit, technically. Compared to cut elimination, (weak) normalisation is simpler, involving neither
permutation of a multicut rule, nor induction on the cut-rank. The equivalent of a cut in natural deduction, for example,

\[
\Delta_1 \\
\vdash I \quad \Gamma, A \vdash B \\
\vdash E \quad \Gamma \vdash A \supset B \quad \Gamma \vdash B \\
\Delta_2
\]

is eliminated as follows: first, assumption \( A \) and all its copies are removed from \( \Delta_1 \). Second, the derivation \( \Delta_2 \), with the context strengthened accordingly, is plugged into all the leaves of \( \Delta_1 \) where assumption \( A \) was used.

This method relies on the fact that no rule inside \( \Delta_1 \) can change the premise \( A \), which is why it does not work for the sequent calculus. To eliminate a cut in the sequent calculus, one has to cope with the fact that logical rules may be applied to both eigenformulas of the cut. This is usually done by permuting up the cut rule step-by-step. However, given a cut with an atomic cut formula \( a \) inside a sequent calculus proof, we can trace the occurrence of \( a \) and its copies produced by contraction, identify all the leaves where they are used in identity axioms, and plug in subproofs in very much the same way as in natural deduction. The problem for the sequent calculus is that cuts are not atomic, in general.

The calculus of structures generalises the one-sided sequent calculus. It has led not only to inference systems with interesting new properties for classical and linear logic [2,9,10], but also to inference systems for new logics that are problematic for the sequent calculus [5,6,3].

Derivations in the calculus of structures enjoy a top-down symmetry that is not available in the sequent calculus: they are chains of one-premise inference rules. ‘Meta-level conjunction’ (the branching of the proof tree) and ‘object-level conjunction’ (the connective in a formula) are identified. The two notions of formula and sequent are also identified, they merge into the notion of structure, which is a formula subject to equivalences that are usually imposed on sequents. This simplification makes explicit the duality between the identity axiom and the cut rule [5]:

\[
\begin{align*}
\text{identity} & \quad S\{\text{true}\} \\
& \quad S\{\text{R} \lor \bar{\text{R}}\} \\
\text{cut} & \quad S\{\text{R} \land \bar{\text{R}}\} \\
& \quad S\{\text{false}\}
\end{align*}
\]

The identity rule is read bottom-up as: if inside a structure there occurs a disjunction of a structure \( R \) and its negation, then it can be replaced by the constant \( \text{true} \). The notion of duality between cut and identity is precisely the one that is known as contrapositive.

Just like in the sequent calculus, the identity axiom can easily be reduced to atomic form. The symmetry of the calculus of structures allows to reduce the cut to atomic form in the same way as the identity axiom, i.e. without having to go through cut elimination. Atomicity of the cut then admits a very simple cut elimination procedure that is similar to normalisation in natural deduction.
After introducing basic notions of the calculus of structures, I show system SKS with atomic contraction, weakening, identity and, most significantly, atomic cut. Then, after establishing some lemmas, I present the cut elimination procedure.

2 The Calculus of Structures

Definition 1. Propositional variables \( p \) and their negations \( \bar{p} \) are atoms, with the negation of \( \bar{p} \) defined to be \( p \). Atoms are denoted by \( a, b, \ldots \). The structures of the language \( KS \) are generated by

\[
S ::= t \mid f \mid a \mid [S_1, \ldots, S_h] \mid (S_1, \ldots, S_h) \mid \bar{S} ,
\]

where \( t \) and \( f \) are the units true and false, \([S_1, \ldots, S_h]\) is a disjunction and \((S_1, \ldots, S_h)\) is a conjunction. \( \bar{S} \) is the negation of the structure \( S \). The units are not atoms. Structures are denoted by \( S, R, T, U \) and \( V \). Structure contexts, denoted by \( S\{\} \), are structures with one occurrence of \( \{\} \), the empty context or hole, that does not appear in the scope of a negation. \( S\{R\} \) denotes the structure obtained by filling the hole in \( S\{\} \) with \( R \). We drop the curly braces when they are redundant: for example, \( S[R, T] \) stands for \( S\{[R, T]\} \). Structures are equivalent modulo the smallest congruence relation induced by the equations shown in Fig. 1, where \( R \) and \( T \) are finite, non-empty sequences of structures.

In general we do not distinguish between equivalent structures.

Definition 2. An inference rule is a scheme of the kind \( \rho \frac{S\{T\}}{S\{R\}} \), where \( \rho \) is the name of the rule, \( S\{T\} \) is its premise and \( S\{R\} \) is its conclusion. In an instance
of \( \rho \), the structure taking the place of \( R \) is called \textit{redex} and the structure taking the place of \( T \) is called \textit{contractum}. A (formal) system \( \mathcal{S} \) is a set of inference rules. To clarify the use of the equational theory where it is not obvious, I will use the rule \( \frac{T}{R} \) where \( R \) and \( T \) are equivalent structures.

**Definition 3.** The \textit{dual} of a rule is obtained by exchanging premise and conclusion and replacing each connective by its De Morgan dual.

**Definition 4.** A \textit{derivation} \( \Delta \) in a certain formal system is a finite chain of instances of inference rules in the system:

\[
\frac{T}{V} \\
\frac{\pi'}{\pi} \\
\vdots \\
\frac{\rho'}{U} \\
\frac{\rho}{R}
\]

A derivation can consist of just one structure. The topmost structure in a derivation is called the \textit{premise} of the derivation, and the structure at the bottom is called its \textit{conclusion}. A derivation \( \Delta \) whose premise is \( T \), whose conclusion is \( R \), and whose inference rules are in \( \mathcal{S} \) will be indicated with \( \Delta \parallel \mathcal{S} \). A \textit{proof} \( \Pi \) in the calculus of structures is a derivation whose premise is the unit true. It will be denoted by \( \frac{\Pi}{R} \). A rule \( \rho \) is \textit{derivable} for a system \( \mathcal{S} \) if for every instance of \( \frac{\rho}{R} \) there is a derivation \( \frac{T}{R} \). A rule \( \rho \) is \textit{admissible} for a system \( \mathcal{S} \) if for every proof \( \frac{\Pi}{S} \) there is a proof \( \frac{\Pi}{\mathcal{S} \cup \{\rho\}} \).

## 3 System SKS

System SKS, shown in Fig. 2 has been introduced and shown to be sound and complete for classical propositional logic in [2]. The first \( S \) stands for “symmetric” or “self-dual”, meaning that for each rule, its dual (or contrapositive) is also in the system. The \( K \) stands for “klassisch” as in Gentzen’s \( \text{LK} \) and the last \( S \) says that it is a system on structures.

The rules \( \text{ai} \downarrow, s, m, \text{aw} \uparrow, \text{ac} \downarrow \) are called respectively \textit{atomic identity}, \textit{switch}, \textit{medial}, \textit{atomic weakening} and \textit{atomic contraction}. Their dual rules carry the same name prefixed with a “co-”, so e.g. \( \text{aw} \uparrow \) is called \textit{atomic co-weakening}. The rules \( s \) and \( m \) are their own duals. The rule \( \text{ai} \uparrow \) is special, it is called \textit{atomic
Fig. 2. System SKS

Fig. 3. System KS

cut. Rules $\textit{ai}\downarrow$, $\textit{aw}\downarrow$, $\textit{ac}\downarrow$ are called down-rules and their duals are called up-rules. In [2], by a semantic argument, all up-rules were shown to be admissible. By removing them we obtain system KS, shown in Fig. 3 which is complete.

Cut-free sequent systems fulfill the subformula property. Our case is different, because the notions of formula and sequent are merged. System KS does not fulfill a “substructure property” just as sequent systems do not fulfill a “subsequent property”. However, when seen bottom-up, no rule in system KS introduces new atoms. It thus satisfies the main aspect of the subformula property: when given a conclusion of a rule there is only a finite number of premises to choose from. In proof search, for example, the branching of the search tree is finite.

Identity, cut, weakening and contraction are restricted to atoms in system SKS. The general versions of those rules are shown in Fig. 4.

Theorem 5. The rules $\textit{i}\downarrow$, $\textit{w}\downarrow$ and $\textit{c}\downarrow$ are derivable in $\{\textit{ai}\downarrow, s\}$, $\{\textit{aw}\downarrow, s\}$ and $\{\textit{ac}\downarrow, m\}$, respectively. Dually, the rules $\textit{i}\uparrow$, $\textit{w}\uparrow$ and $\textit{c}\uparrow$ are derivable in $\{\textit{ai}\uparrow, s\}$, $\{\textit{aw}\uparrow, s\}$ and $\{\textit{ac}\uparrow, m\}$, respectively.
Fig. 4. General identity, weakening, contraction and their duals

Proof. By an easy structural induction on the structure that is cut, weakened or contracted. Details are in [2]. The case for the cut is shown here. A cut introducing the structure \((R,T)\) together with its dual structure \([\bar{R},\bar{T}]\) is replaced by two cuts on smaller structures:

\[
\begin{array}{c}
i\dn S(t) \\
\Downarrow \\
S[R,R] \\
\end{array}
\quad
\begin{array}{c}
i\dn S(R,\bar{R}) \\
\Downarrow \\
S\{f\} \\
\end{array}
\]

\[
\begin{array}{c}
w\dn S(R) \\
\Downarrow \\
S\{R\} \\
\end{array}
\quad
\begin{array}{c}
w\dn S\{\bar{f}\} \\
\Downarrow \\
S[R] \\
\end{array}
\]

\[
\begin{array}{c}
c\dn S[R,R] \\
\Downarrow \\
S\{R\} \\
\end{array}
\quad
\begin{array}{c}
c\dn S\{R\} \\
\Downarrow \\
S[R,R] \\
\end{array}
\]

\[
\begin{array}{c}
i\dn S(R,T,[\bar{R},\bar{T}]) \\
\Downarrow \\
S\{f\} \\
\end{array}
\quad
\begin{array}{c}
i\dn S(R,R) \\
\Downarrow \\
S\{f\} \\
\end{array}
\]

\[
\begin{array}{c}
s\dn S[R,R,[\bar{R},\bar{T}]] \\
\Downarrow \\
S(R,T,(T,T)) \\
\end{array}
\quad
\begin{array}{c}
s\dn S[R,R] \\
\Downarrow \\
S[R,R] \\
\end{array}
\]

\[
\begin{array}{c}
s\dn S[R,R] \\
\Downarrow \\
S[R,R] \\
\end{array}
\quad
\begin{array}{c}
s\dn S[R,R] \\
\Downarrow \\
S[R,R] \\
\end{array}
\]

\[
\begin{array}{c}
i\dn S[R,R] \\
\Downarrow \\
S[R,R] \\
\end{array}
\quad
\begin{array}{c}
i\dn S[R,R] \\
\Downarrow \\
S[R,R] \\
\end{array}
\]

So, while general identity, weakening, contraction and their duals do not belong to SKS, they will be freely used in derivations in SKS to denote multiple instances of the corresponding rules in SKS according to Theorem 5.

Remark 6. Sequent calculus derivations easily correspond to derivations in system SKS. For instance, the cut of sequent systems in Gentzen-Schütte form [11]:

\[
\begin{array}{c}
\vdash \varphi, A \\
\Downarrow \\
\vdash \psi, \bar{A} \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi, \psi \\
\Downarrow \\
\vdash \psi, A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi \\
\Downarrow \\
\vdash \psi \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi, \psi \\
\Downarrow \\
\vdash \psi, A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi, \psi \\
\Downarrow \\
\vdash \psi, A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi, \psi \\
\Downarrow \\
\vdash \psi, A \\
\end{array}
\]

\[
\begin{array}{c}
\vdash \varphi, \psi \\
\Downarrow \\
\vdash \psi, A \\
\end{array}
\]

4 Cut Elimination

In the calculus of structures, there is more freedom in applying inference rules than in the sequent calculus. While this allows for a richer combinatorial analysis of proofs, it is a significant challenge for cut elimination. During cut elimination,
the sequent calculus allows to get into the crucial situation where on one branch a logical rule applies to the main connective of the eigenformula and on the other branch the corresponding rule applies to the dual connective of the dual eigenformula. In the calculus of structures, rules apply deep inside a context, they are not restricted to main connectives. The methodology of the sequent calculus thus does not apply. For example, one cannot permute the cut over the switch rule. One can generalise the cut in order to permute it over switch, but this requires a case analysis that is far more complicated than in the sequent calculus. Contraction is an even bigger problem. Despite many efforts, no cut elimination procedure along these lines has been found for system SKS.

Two new techniques were developed to eliminate cuts in the calculus of structures. The first is called decomposition, and has been used in [6,9] for some systems related to linear logic. Proving termination of decomposition is rather involved [9]. It makes essential use of the exponentials of linear logic which restrict the use of contraction. So far, this technique could not be used for classical logic with its unrestricted contraction. The second technique is called splitting [5], and essentially makes available a situation corresponding to the one described above for the sequent calculus. Splitting covers the broadest range of systems in the calculus of structures, it not only applies to the systems mentioned above, but has recently also been applied to system SKS (but the proof is not published.
yet). Compared to splitting, the procedure given here is much simpler. In fact, I
do not know of any other system with such a simple cut elimination procedure.

In the sequent calculus as well as in sequent-style natural deduction, a deriva-
tion is a tree. Seen bottom-up, a cut splits the tree into two branches. To apply
a cut, one is forced to split the context among the two branches (in the case
of multiplicative context treatment) or to duplicate the context (in the case of
additive context treatment). In the calculus of structures, the cut rule does not
split the proof.

The crucial idea, illustrated in Fig. 5, is that we can do that during cut
elimination. This allows us to plug-in proofs just like in natural deduction: we
duplicate the proof above a cut and remove atom $a$ from the copy shown on the
left and the atom $\bar{a}$ from the copy shown on the right. We choose one copy, the
one on the left in this case, and replace $a$ by $R$ throughout the proof, breaking
some instances of identity. They are fixed by substituting the proof on the right.
A contraction is applied to obtain a cut-free proof of $R$.

In contrast to the sequent calculus, the cut is not the only problematic rule
in system SKS. The rule $aw^\uparrow$ also induces infinite choice in proof-search. Fortu-
nately, we can not only eliminate the cut rule, but also the other up-rules. Each
up-rule individually can be shown to be admissible for system KS. However,
since we are going to eliminate the cut anyway, to eliminate rules $aw^\uparrow$ and $ac^\uparrow$
the following lemma is sufficient.

**Lemma 7.** Each rule in SKS is derivable for identity, cut, switch and its dual
rule.

Proof. An instance of $\rho^\uparrow \frac{S\{T\}}{S\{R\}}$ can be replaced by $\frac{S(T, [R, R])}{S(R)}$. The
same holds for down-rules.

When plugging in a derivation in natural deduction, its context has to be
strengthened, to fit into the leaf into which it is plugged. Adding to a context
in natural deduction is easy, since it is a flat object, a set or a multiset. In the
calculus of structures, contexts are more general, nested objects. The following
definition is used to strengthen contexts.

**Definition 8.** Given a derivation $\Delta$, the derivation $S\{\Delta\}$ is obtained as follows:
Definition 9. An instance of atomic cut is called shallow if it is of the following form:

\[ \bar{a} \uparrow \frac{[S, (a, \bar{a})]}{S} \].

Lemma 10. The atomic cut is derivable for shallow atomic cut and switch.

Proof. An easy induction locally replaces an instance of atomic cut by a shallow atomic cut followed by instances of switch. Details are in [1]. \( \square \)

Lemma 11. Each proof \( \frac{T{\{a\}}}{\varnothing} \) can be transformed into a proof \( \frac{T{\{t\}}}{\varnothing} \).

Proof. Starting with the conclusion, going up in the proof, in each structure we replace the occurrence of \( a \) and its copies, that are produced by contractions, by the unit \( t \). Replacements inside the context of any rule instance do not affect the validity of this rule instance. Instances of the rules \( m \) and \( s \) remain valid, also in the case that atom occurrences are replaced inside redex and contractum. Instances of the other rules are replaced by the following derivations:

\[
\begin{align*}
\text{ac} & \downarrow \frac{S[a, a]}{S\{a\}} \leadsto \frac{S[t, t]}{S\{t\}} \\
\text{aw} & \downarrow \frac{S\{f\}}{S\{a\}} \leadsto \frac{S[t, (t, f)]}{S\{t\}} \\
\text{ai} & \downarrow \frac{S\{t\}}{S[a, \bar{a}]} \leadsto \text{aw} \downarrow \frac{S[t, f]}{S[t, \bar{a}]}.
\end{align*}
\]

Properly equipped, we now turn to cut elimination.

Theorem 12. Each proof \( \frac{T{\{a\}}}{\varnothing} \) can be transformed into a proof \( \frac{T}{\varnothing} \).

Proof. By Lemma 11 the only rule left to eliminate is the cut. By Lemma 10 we replace all cuts by shallow cuts. The topmost instance of cut, together with the proof above it, is singled out:

\[
\frac{T{\{a\}}}{\varnothing} = \text{ai} \uparrow \frac{[R, (a, \bar{a})]}{R} \uparrow \frac{T{\{a\}}}{\varnothing}.
\]
Lemma 11 is applied twice on \( \Pi \) to obtain

\[
\begin{align*}
\Pi_1 & \vdash_{KS} R, a \\
\Pi_2 & \vdash_{KS} R, \bar{a}
\end{align*}
\]

Starting with the conclusion, going up in proof \( \Pi_1 \), in each structure we replace the occurrence of \( a \) and its copies, that are produced by contractions, by the structure \( R \).

Replacements inside the context of any rule instance do not affect the validity of this rule instance. Instances of the rules \( m \) and \( s \) remain valid, also in the case that atom occurrences are replaced inside redex and contractum. Instances of \( \text{ac} \downarrow \) and \( \text{aw} \downarrow \) are replaced by their general versions:

\[
\begin{align*}
\text{ac} \downarrow & \quad S[a, a] \quad \sim \quad S[R, R] \\
\text{aw} \downarrow & \quad S[f] \quad \sim \quad S[R]
\end{align*}
\]

Instances of \( \text{ai} \downarrow \) are replaced by \( S\{\Pi_2\} \):

\[
\begin{align*}
\text{ai} \downarrow & \quad S[t] \quad \sim \quad S\{\Pi_2\} \vdash_{KS} R, \bar{a}
\end{align*}
\]

The result of this process of substituting \( \Pi_2 \) into \( \Pi_1 \) is a proof \( \Pi_3 \), from which we build

\[
\begin{align*}
\Pi_3 & \vdash_{KS} \Pi_2 \vdash_{KS} \Pi_1 \\
\Pi_3 & \vdash_{KS} R, R \\
\Pi_3 & \vdash_{KS} T \cup \{\text{ai} \downarrow\}
\end{align*}
\]

Proceed inductively downward with the remaining instances of cut. \( \square \)

5 Conclusion

System \( \text{SKS} \) seems a good starting point for developing both the proof search as well as the proof normalisation paradigm in one system. Since all up-rules are admissible, it is suitable for proof search as computation. The cut elimination procedure given is simpler than those for sequent calculi. The way in
which proofs are substituted resembles normalisation in natural deduction. This hopefully allows for a computational interpretation in the proof normalisation as computation paradigm.

Of course, a lot of work remains to be done. In the proof search as computation realm, given the admissibility of cut, a suitable notion of uniform proof as in [7] should be obtainable. For proof normalisation as computation, natural questions to be considered are strong normalisation and confluence of the cut elimination procedure when imposing as little strategy as possible. Similarly to [8], a term calculus should be developed and its computational meaning be made precise. Intuitionistic logic is a more familiar setting for this, so the possibility of treating intuitionistic logic should be explored.

A natural question is whether this procedure scales to more expressive cases, for example to predicate logic. System SKSq extends system SKS by first-order quantifiers [1]. There, cut elimination is proved via a translation to the sequent calculus. The procedure presented here does not appear to easily scale to system SKSq. The problem, which does not occur in shallow inference systems like sequent calculus or natural deduction, are existential quantifiers in the context of a cut which bind variables both in $a$ and $\bar{a}$. The procedure easily extends to closed atomic cuts, that is, cuts where the eigenformula is an atom prefixed by quantifiers that bind all its variables. The question then is how to reduce general cuts to closed atomic cuts. If this problem were solved, then the procedure would scale to predicate logic. Hopefully this will lead to a cut elimination procedure for predicate logic, which is simpler than other cut elimination procedures, as happened for propositional logic.

Acknowledgements

This work has been supported by the DFG Graduiertenkolleg 334. I would like to thank Alessio Guglielmi, for inspiration, and Alwen Fernanto Tiu and Michel Parigot, for comments which helped simplifying an earlier version of the cut elimination procedure. Alessio Guglielmi, Charles Stewart and Lutz Straburger carefully read previous drafts of this work and prompted improvements.

Web Site
Information about the calculus of structures is available from:

http://www.wv.inf.tu-dresden.de/~guglielm/Research/.

References


The Commuting V-Diagram
On the Relation of Refinement and Testing

Bernhard K. Aichernig

United Nations University
International Institute for Software Technology (UNU/IIST)
P.O. Box 3058, Macau
bka@iist.unu.edu

Abstract. This article discusses the relations between the step-wise development through refinement and the design of test-cases. It turns out that a commuting diagram inspired by the V-process model is able to clarify the issues involved. This V-diagram defines the dependencies of specifications, implementations and test-cases in the category of contracts. The objects in this category are contracts defined in the formalism of the refinement calculus. The maps are the refinement steps between these objects. Our framework is able to define the correctness notion of test-cases, testing strategies as refinement rules, and which test-cases should be added under refinement.

Keywords: formal methods, specification-based testing, refinement, refinement calculus, contracts.

1 Introduction

The synergy of formal methods and testing has become a popular area of research. In the last years, test-generation tools have been invented for almost every popular specification language. One reason is the industry’s demand to cut the efforts of software testing. Another is the academics’ insight that testing is complementary to proving the correctness of a program (see e.g. Hoare’s comments on testing in [5]).

However, only little research has been put into the question how the related development techniques such as refinement contribute to testing. Our current research addresses this open issue. In our previous work we have demonstrated that test design can be viewed as a reverse program synthesis problem of finding adequate abstractions [1]. The consequence of this insight is that we are able to define test-case synthesis rules in order to calculate correct test-cases from specifications. The mathematical framework in our work is the refinement calculus of Back and von Wright [6] including a simple but powerful contract language.

Our general approach is able to cover rather different test-selection techniques like domain partitioning [3], interactive scenario selection [2], and mutation testing [4]. These previous work demonstrated that a test-case selection strategy can be represented by means of formal synthesis rules that define how specifications
should be changed into test-cases. This approach is inspired by the refinement calculus where refinement rules define correctness preserving development steps. The difference is that our rules represent the derivation of test-cases by means of abstraction steps.

In this paper we focus on the question: “Which new test-cases are needed if we refine a specification or implementation?” In order to give a scientific answer, the role of testing and step-wise development has to be clarified. This can be done using a simple diagram (Figure 1) to which we give a precise mathematical semantics.

![Image of V-Diagram]

**Fig. 1.** The V-Diagram.

The V-diagram is inspired by the V process model, a derivative of the waterfall model where the development phases are explicitly linked to testing. Both visualizations stress the importance of testing. The left-hand side of the V in Figure 1 represents the step-wise development of a specification $C_1$ into an implementation $C_n$. In the following $C_1$ to $C_n$ are called contracts representing commitments on different levels of abstractions. All the arrows in the diagram denote refinement. On the right-hand side of the V, the corresponding test-cases are shown. The test-cases are refined in accordance with the contracts. In our view of test-cases, contract $C_i$ must be a refinement of its test-cases $T_i$. Consequently, test-cases can be viewed as a special form of specification. During discussions we have found that not too many colleagues are aware of this fact.

The formal refinement relation between test-cases $T_i$ and a contract $C_i$ can be interpreted in two directions:

- **test-cases as specifications:** if test-cases are given, an implementation or formal specification must be a correct refinement of the intended test-cases.
- **test-synthesis as an abstraction problem:** As a refinement calculus is a technique to derive correct implementations from a specification by following correctness preserving refinement rules, dual abstraction rules can be used to calculate test-cases. The names of the refinement arrows $t_i^{-1}$ in Figure 1 should indicate this reverse process of abstraction.
In the following sections we will discuss this diagram and its consequences in more detail. Section 2 presents the mathematical interpretation of the V-diagram as a commuting diagram in the category of contracts. Section 3 briefly introduces the notions of contracts and refinements in the refinement calculus. This section covers the left-hand side of the V. Next, Section 4 shows that different kinds of test-cases are in fact abstractions of formal specifications. Section 5 presents a main contribution of this paper: properties for new test-cases under refinement. In Section 6 an example serves to illustrate these findings. Finally, we draw our conclusions in Section 7.

2 A Formal Interpretation of the V-Diagram

The explanations of the V-Diagram have been informal so far. We neither gave a definition of contracts, nor a definition of refinement. These definitions will be provided in Section 3. However, taking a category theoretic view on Figure 1 reveals interesting properties without a detailed knowledge of the refinement calculus.

A category $\mathcal{C}$ is an algebraic structure consisting of a class of objects and a class of maps. Each map $f$, has one object $A$ as domain and one object $B$ as codomain, denoted $A \xrightarrow{f} B$. For each object $A$ an identity map $A \xrightarrow{1_A} A$ exists. Furthermore, composition $A \xrightarrow{g} B$ is defined on maps. In a category the identity laws $1_B \circ f = f$ and $f \circ 1_A = f$ hold and composition is associative: $(h \circ g) \circ f = h \circ (g \circ f)$.

It can be easily seen that contracts form such a category, here called $\mathbf{Con}$. The objects in $\mathbf{Con}$ are contracts formulated in the contract language of the refinement calculus (see Section 3). A map $A \xrightarrow{r} B$ in $\mathbf{Con}$ is a refinement step of $A$ into $B$. We write $A \sqsubseteq B$ for $B$ being a refinement of $A$. The identity refinement $A = B$ exists as well as composition of refinements which is associative.

In category theory equalities can be easily described by means of commuting diagrams. For example, the identity law $1_B \circ f = f$ can be expressed via the following commuting diagram:

\[
\begin{array}{ccc}
A & \xrightarrow{f} & B \\
\downarrow & \ & \downarrow \\
A & \xrightarrow{1_B} & B
\end{array}
\]

Consequently, by interpreting Figure 1 as a diagram in the category $\mathbf{Con}$, several properties can be derived.

The commutations in the V-diagram represent $n - 1$ equations for $1 \leq i < n$:

\[
r_i^c \circ t_i^{-1} = t_{i+1}^{-1} \circ r_i^t
\]

This equation implies that given two sets of test-cases, $T_i$ for a contract $C_i$ and $T_{i+1}$ for a refined contract $C_{i+1}$ such that $C_i \sqsubseteq C_{i+1}$, then test-cases $T_{i+1}$ must

\footnote{We use the terminology of [9].}
be a refinement of $T_i$ or $T_i \subseteq T_{i+1}$. This means that under refinement the test-cases have to be refined as well. What this refinement looks like will be discussed in Section 5.

Since composition exists we know that step-wise refinement preserves refinement. For example, if we extend $\subseteq$ point-wise to maps, then $C_1 \subseteq C_n$ can be expressed as

$$1_{C_1} \subseteq r_{n-1}^c \circ \cdots \circ r_2^c \circ r_1^c$$

The same holds for refining test-cases, e.g.:

$$1_{T_1} \subseteq r_{n-1}^t \circ \cdots \circ r_2^t \circ r_1^t$$

As a consequence even more similar equations relating the several refinements on different levels of abstraction can be derived. All these properties are captured in the V-diagram.

3 Refining Contracts: $C \xrightarrow{r_c} C'$

3.1 Contracts

The prerequisite for testing is some form of contract between the user and the provider of a system that specifies what it is supposed to do. In case of system-level testing usually user and software requirement documents define the contract. Formal methods propose mathematics to define such a contract unambiguously and soundly. In the following the formal contract language of Back and von Wright [6] is used. It is a generalization of the conventional pre- and post-condition style of formal specifications known from VDM, B and Z. The foundation of this refinement calculus is based on lattice-theory and classical higher-order logic (HOL).

A system is modeled by a global state space $\Sigma$. A single state $x$ in this state space is denoted by $x : \Sigma$. Functionality is either expressed by functional state transformers $f$ or relational updates $R$. A state transformer is a function $f : \Sigma \to \Gamma$ mapping a state space $\Sigma$ to the same or another state space $\Gamma$.

A relational update $R : \Sigma \to (\Gamma \to \text{Bool})$ specifies a state change by relating a state before ($\sigma : \Sigma$) with a state after execution ($\gamma : \Gamma$). In HOL, relations are modeled by functions mapping the states to Boolean valued predicates. For convenience, a relational assignment ($x := x' | b$) is available and generalizes assignment statements. It sets a state variable $x$ to a new state $x'$ such that $b$, relating $x$ and $x'$, holds.

The language further distinguishes between the responsibilities of communicating agents in a contract. Here, the contract models the viewpoint of one agent called the angel who interacts with the rest of the system called the demon. In our work following [65], the user is considered the angel and the system under test the demon. Relational contract statements denoted by $\{R\}$ express relational updates under control of the angel (user). Relational updates of the demon are denoted by $[R]$ and express updates that are non-deterministic from the angel’s point of view. Usually, we take the viewpoint of the angel.
The contract statement \( \langle f \rangle \) denotes a functional update of the state determined by a state transformer \( f \). There is no choice involved here, neither for the angel nor the demon agent, since there is only one possible next state for a given state.

Two contracts can be combined by sequential composition \( C_1; C_2 \) or choice operators. The angelic choice \( C_1 \sqcup C_2 \) and the demonic choice \( C_1 \sqcap C_2 \) define non-deterministic choice of the angel or demon between two contracts \( C_1 \) and \( C_2 \). Furthermore, predicate assertions \( \{ p \} \) and assumptions \( [ p ] \) define conditions the angel, respectively the demon, must satisfy. In this language of contract statements \( \{ p \}; \langle f \rangle \) denotes partial functions and \( \{ p \}; [ R ] \) pre-postcondition specifications. Furthermore, recursive contracts defined by means of least (\( \mu \)) and greatest fix-point operators (\( \nu \)) may express several patterns of iteration.

The core contract language used in this work can be summarized by the following BNF grammar, where \( p \) is a condition and \( R \) a relation over states.

\[
C := \{ p \} | [ p ] | \{ R \} | [ R ] | C; C | C \sqcup C | C \sqcap C | \mu X \cdot C
\]

As needed, we will extend this core language by our own contract statements. However, all new statements will be defined by means of the above core language. Thus, our language extensions are conservative. This means that no inconsistencies into the theory of the refinement calculus are introduced by our new definitions.

### 3.2 Example Contracts

A few simple examples should illustrate the contract language. The following contract is a pre-postcondition specification of a square root algorithm:

\[
\{ x \geq 0 \land e > 0 \}; [ x := x' | -e \leq x - x'^2 \leq e ]
\]

The precondition is an assertion about an input variable \( x \) and a precision \( e \). A relational assignment expresses the demonic update of the variable \( x \) to its new value \( x' \). Thus, the contract is breached unless \( x \geq 0 \land e > 0 \) holds in the state initially. If this condition is true, then \( x \) is assigned some value \( x' \) for which \( -e \leq x - x'^2 \leq e \) holds.

Consider the following version of the square root contract that uses both kinds of non-determinism:

\[
\{ x, e := x', e'| x' \geq 0 \land e' > 0 \}; [ x := x' | -e \leq x - x'^2 \leq e ]
\]

In this contract the interaction of two agents is specified explicitly. This contract requires that our agent, called the angel, first chooses new values for \( x \) and \( e \). Then the other agent, the demon, is given the task of computing the square-root in the variable \( x \).

The following example should demonstrate that programming constructs can be defined by means of the basic contract statements. A conditional statement can be defined by an angelic choice as follows:

\[
\text{if } P \text{ then } S_1 \text{ else } S_2 \text{ fi } \triangleq \{ P \}; S_1 \sqcup \{ \neg P \}; S_2
\]
Thus, the angel agent can choose between two alternatives. The agent will, however, always choose only one of these, the one for which the assertion is true, because choosing the alternative where the guard is false would breach the contract. Hence, the agent does not have a real choice if he wants to satisfy the contract.\footnote{An alternative definition can be given by demonic choice.}

Iteration can be specified by recursive contracts ($\mu X \cdot C$). Here $X$ is a variable that ranges over contract statements, while ($\mu X \cdot C$) is the contract statement $C$, where each occurrence of $X$ in $C$ is interpreted as a recursive invocation of the contract $C$. For example, the standard while loop is defined as follows:

\[
\text{while } g \text{ do } S \text{ od } \triangleq (\mu X \cdot \text{if } g \text{ then } S; X \text{ else skip fi})
\]

We write $\text{skip } \triangleq \langle \text{id} \rangle$ for the action that applies the identity function to the present state.

### 3.3 Semantics

The semantics of the contract statements is defined by weakest precondition predicate transformers. A predicate transformer $C : (\Gamma \rightarrow \text{Bool}) \rightarrow (\Sigma \rightarrow \text{Bool})$ is a function mapping postcondition predicates to precondition predicates. The set of all predicate transformers from $\Sigma$ to $\Gamma$ is denoted by $\Sigma \rightarrow \rightarrow \Gamma \triangleq (\Gamma \rightarrow \text{Bool}) \rightarrow (\Sigma \rightarrow \text{Bool})$.

The different roles of the angel and the demon are reflected in the following weakest-precondition semantics. Here $q$ denotes a postcondition predicate and $\sigma$ a particular state, $p$ is an arbitrary predicate, and $R$ a relation. The weakest-precondition predicate transformer is denoted by $\text{wp}$. The notation $f.x$ is used for function application instead of the more common form $f(x)$.

\[
\begin{align*}
\text{wp.}\{p\}.q & \triangleq p \cap q \quad \text{(assertion)} \\
\text{wp.}\{p\}.q & \triangleq \neg p \cup q \quad \text{(assumption)} \\
\text{wp.}\{R\}.q.\sigma & \triangleq (\exists \gamma \in \Gamma \cdot R.\sigma.\gamma \land q.\gamma) \quad \text{(angelic update)} \\
\text{wp.}\{R\}.q.\sigma & \triangleq (\forall \gamma \in \Gamma \cdot R.\sigma.\gamma \Rightarrow q.\gamma) \quad \text{(demonic update)} \\
\text{wp.}(C_1; C_2).q & \triangleq C_1.(C_2.q) \quad \text{(sequential composition)} \\
\text{wp.}(C_1 \sqcup C_2).q & \triangleq C_1.q \sqcup C_2.q \quad \text{(angelic choice)} \\
\text{wp.}(C_1 \sqcap C_2).q & \triangleq C_1.q \sqcap C_2.q \quad \text{(demonic choice)}
\end{align*}
\]

In this semantics, the breaching of a contract by our angel agent, means that the weakest-precondition is false. If a demon agent breaches a contract, the weakest-precondition is trivially true. The semantics of the specification constructs above can be interpreted as follows:

- The weakest precondition semantics of an assertion contract reflects the fact that, if the final state of the contract should satisfy the post-condition $q$, then in addition the assertion predicate $p$ must hold. It can be seen that the global state is not changed by an assertion statement. Consequently, the angel breaches this contract if $p \cap q$ evaluates to false.
The semantics of an assumption shows that the demon is responsible for satisfying an assumption predicate $p$. If the assumption does not hold, the demon breaches the contract and the angel is released from the contract. In this case, the weakest-precondition trivially evaluates to $\text{true}$.

The angelic update definition says that a final state $\gamma$ must exist in the relation $R$, such that the postcondition $q$ holds. The existential quantifier in the weakest-precondition shows that the angel has control of this update. The angel can satisfy the contract, as long as one update exists that satisfies the postcondition. In the set notation this update is defined as $\wp\{R\}.q.\sigma \triangleq R.\sigma \cap q \neq \emptyset$.

This is in contrast to the definition of the demonic update. Here, all possible final states $\gamma$ have to satisfy the postcondition. The reason is that the demonic update is out of our control. It is not known, to which of the possible states, described by the relation $R$, the state variables will be set. In the set notation this update is defined as $\wp\{R\}.q.\sigma \triangleq R.\sigma \subseteq q$.

The weakest-precondition of two sequentially combined contracts is defined by the composition of the two weakest-preconditions.

The angelic choice definition shows that the weakest-precondition is the union of the weakest-precondition of the two contracts. Thus, a further choice of the angel further weakens the weakest-preconditions.

The demonic choice is defined as the intersection of the weakest-preconditions of the two contracts. Thus, demonic choice represents a strengthening of the weakest-preconditions.

For further details of the predicate transformer semantics, we refer the reader to [6].

### 3.4 Refinement and Abstraction

The notion of contracts includes specification statements as well as programming statements. More complicated specification statements as well as programming statements can be defined by the basic contract statements presented above. The refinement calculus provides a synthesis method for refining specification statements into programming statements that can be executed by the target system. The refinement rules of the calculus ensure by construction that a program is correct with respect to its specification.

Formally, refinement of a contract $C$ by $C'$, written $C \sqsubseteq C'$, is defined by the point-wise extension of the subset ordering on predicates: For $\Gamma$ being the after state space of the contracts, we have

$$C \sqsubseteq C' \triangleq \forall q \in (\Gamma \to \text{Bool}) \cdot C.q \subseteq C'.q$$

This ordering relation defines a lattice of predicate transformers (contracts) with the lattice operators meet $\sqcap$ and join $\sqcup$. The top element $\top$ is $\text{magic}.q \triangleq \text{true}$, a statement that is not implementable since it can magically establish every postcondition. The bottom element $\bot$ of the lattice is $\text{abort}.q \triangleq \text{false}$ defining
the notion of abortion. The choice operators and negation of contracts are defined by point-wise extension of the corresponding operations on predicates. A large collection of refinement rules can be found in [6,10].

Abstraction is dual to refinement. If $C \subseteq C'$, we can interchangeably say $C$ is an abstraction of $C'$. In order to emphasize rather the search for abstractions than for refinements, we write $C \sqsupseteq C'$ to express $C'$ is an abstraction of $C$. Trivially, abstraction can be defined as:

$$C \sqsupseteq C' \equiv C' \subseteq C$$

Hence, abstraction is defined as the reverse of refinement. The reader should keep this technical definition of abstraction in mind when we identify test-cases being abstractions in the following section.

4 Test-Cases through Abstraction: $T \xrightarrow{t^{-1}} C$

In the following we will demonstrate that test-cases are in fact contracts — highly abstract contracts. To keep our discussion simple, we do not consider parameterized procedures, but only global state manipulations. In [6] it is shown how procedures can be defined in the contract language. Consequently, our approach scales up to procedure calls.

4.1 Input-Output Tests

The simplest form of test-cases are pairs of input $i$ and output $o$ data. We can define such an input-output test-case $TC$ as a contract between the user and the unit under test:

$$TC_{i \ o} \triangleq \{ x = i \}; [y := y' | y' = o]$$

Intuitively, the contract states that if the user provides input $i$, the state will be updated such that it equals $o$. Here, $x$ is the input variable and $y$ the output variable.

In fact, such a $TC$ is a formal pre-postcondition specification solely defined for a single input $i$. This demonstrates that a collection of $n$ input-output test-cases $TC$s are indeed point-wise defined formal specifications:

$$TCs \triangleq TC_{i_1 \ o_1} \sqcup \ldots \sqcup TC_{i_n \ o_n}$$

Moreover, such test-cases are abstractions of general specifications, if the specification is deterministic for the input-value of the test-case, as the following theorem shows.

**Theorem 1.** Let $p : \Sigma \rightarrow \text{Bool}$ be a predicate, $Q : \Sigma \rightarrow \Gamma \rightarrow \text{Bool}$ a relation on states, and $TC_{i \ o}$ a test-case, where all state variables (the whole state) are
observable and not just a designated input variable $x$ and an output variable $y$. Thus, input $i : \Sigma$ and output $o : \Gamma$. Then

$$\{p\}; [Q] \sqsupseteq TC\, i\, o \equiv p.i \land (Q.i = o)$$

The intuition behind Theorem 1 is that input-output test-cases can be viewed as pre-postcondition specifications with a special pre-condition restricting the input to a single test-value. If the specification is deterministic for the given input, a derived test-case is an abstraction of this specification. Note that in the case of non-deterministic specifications a form of non-deterministic test-cases are needed, as has been shown in [2].

Furthermore, the selection of certain test-cases out of a collection of test-cases can be considered an abstraction:

**Corollary 1.**

$$TC\, i_1\, o_1 \sqcup \ldots \sqcup TC\, i_n\, o_n \sqsupseteq TC\, i_k\, o_k$$

for all $k$, $1 \leq k \leq n$.

**Proof.** The theorem is valid by definition of the join operator $a \sqcup b \sqsubseteq a$ or $a \sqcup b \sqsubseteq b$, respectively.

The fact that test-cases are indeed formal specifications and, as Theorem 1 shows, abstractions of more general contracts shows an aspect of why test-cases are so popular: First, they are abstract in the technical sense, and thus easy to understand. Second, they are formal and thus unambiguous.

Here, only the commonly used pre-postcondition contracts have been considered. They are a normal form for all contracts not involving angelic actions. This means that arbitrary contracts excluding $\sqcup$ and $\{R\}$ can be formulated in a pre-postcondition style (see Theorem 26.4 in [6]). However, our result that test-cases are abstractions holds for general contract statements involving user interaction. We refer to [2] where we have demonstrated that even sequences of interactive test-cases, so called scenarios, are abstractions of interactive system specifications. In the same work we have shown how test-synthesis rules can be formulated for deriving test-partitions or scenarios.

5 Refining Test-Cases: $T \xrightarrow{r^t} T'$

5.1 Defining the Problem

In the previous sections it has been demonstrated that test-cases are abstractions of specification or implementation contracts. If contracts are refined towards an implementation, the test-cases can be refined accordingly in order to test the correctness of the chosen refinement steps.

---

3 The slightly more complex case of partly observable states including proofs can be found in [2].
However, the refinement relation $\sqsubseteq$ alone is a too weak criterion in order to design new test-cases. For example, the relation $\sqsubseteq$ includes the equality of contracts $=$. This means that the same test-cases could be used for the refined contract. Note that in this case the V-diagram still commutes, but that $T \xrightarrow{r} T' = T \xrightarrow{1r} T$. More useful criteria for designing new test-cases are needed.

The problem of finding new test-cases can be formulated as follows:

**Theorem 2.** Given a contract $C$ and its refinement $C'$. Furthermore, it is assumed that test-cases $T$ for $C$ have been correctly designed, thus $C \sqsupseteq T$. Then the refined test-cases $T'$ for testing $C'$ have the general form

$$T' = (T \sqcup T'_{\text{new}})$$

for arbitrary new test-cases $T'_{\text{new}}$ such that $C' \sqsupseteq T'_{\text{new}}$.

**Proof.** It must be shown that $(T \sqcup T'_{\text{new}})$ is a refinement of $T$ and that they are test-cases of $C'$, thus $T \sqsubseteq (T \sqcup T'_{\text{new}})$ and $C' \sqsupseteq (T \sqcup T'_{\text{new}})$ must hold.

The first property $T \sqsubseteq (T \sqcup T'_{\text{new}})$ holds by definition of the join operator $\sqcup$ in the lattice of contracts.

Similarly, by the definition of $\sqcup$, the second property $C' \sqsupseteq (T \sqcup T'_{\text{new}})$ holds iff $C' \sqsubseteq T$ and $C' \sqsupseteq T'_{\text{new}}$. Trivially, $C' \sqsupseteq T'_{\text{new}}$ is a premise. $C' \sqsubseteq T$ follows from the facts that $C'$ is a refinement of $C$, thus $C' \sqsupseteq C$, and that $T$ are correct test-cases designed for $C$, thus $C \sqsupseteq T$ holds.

This proves that the diagram commutes for refined test-cases $(T \sqcup T'_{\text{new}})$. $\square$

**Theorem 3.** A mutant $C_i$ of a contract $C$ is a useful mutant iff $C \not\sqsubseteq C_i$. $\square$

### 5.2 Abstract Contracts as Mutations

A main contribution of this paper is the insight that the problem of finding useful test-cases $T'_{\text{new}}$ can be mapped to the author’s approach to mutation testing [4].

In analogy to program mutation in the traditional approaches of mutation testing, in contract-based mutation testing introduced in [4] we produce a mutant contract by introducing small changes to the formal contract definition. Then we select test-cases that are able to detect the introduced mutations. What kind of changes to be introduced is heuristically defined by a set of mutation operators that take a contract and produce its mutated version. However, not all mutants are useful:

**Definition 1.** (useful mutant) A useful mutant of a contract $C$ is a mutated contract $C_i$ such that there exists a test-case $TC_i \sqsubseteq C$ that is able to distinguish an implementation of $C_i$ from an implementation of $C$. $\square$

**Theorem 3.** A mutant $C_i$ of a contract $C$ is a useful mutant iff $C \not\sqsubseteq C_i$. $\square$
In [4] we pointed out that abstract mutants $C_i \sqsubseteq C$ are useful. Furthermore we defined adequate test-cases for mutation testing:

**Definition 2. (adequate test-case)** A test-case for a contract $C$ is called adequate with respect to a set of mutation operators $M$ if it is able to distinguish at least one useful mutant $M_i$, with $m.C = M_i$ and $m \in M$.

Here, in our quest for useful new test-cases $T'_{new}$ for a contract $C'$ and $C \sqsubseteq C'$, we consider $C$ as a useful mutant. Then $T'_{new}$ should be adequate test-cases that are able to distinguish the abstract mutant from its refinement.

Thus the criterion for useful new test-cases is an adaptation of the abstraction rule for deriving mutation test-cases (Theorem 12 in [4]):

**Theorem 4.** Given a contract $C$ and its refinement $C'$. Furthermore, test-cases $T$ for $C$ have been previously designed following some selection strategy. Then the refined test-cases $T'$ for testing $C'$ have the general form

$$T' = (T \sqcup T'_{new})$$

and

$$C' \sqsupseteq T'_{new} \land C \not\sqsubset T'_{new}$$

Alternatively an abstraction rule for producing useful new test-cases can be formulated as follows:

$$\begin{array}{ll}
C \sqsubseteq C', C \sqsupseteq T \\
C' \sqsupseteq T'_{new}, C \not\sqsubset T'_{new} \\
\hline
C' \sqsupseteq T \sqcup T'_{new}
\end{array}$$

It can be seen that $C \not\sqsubset T'_{new}$ is the central property for adding new test-cases that are able to distinguish between contract $C$ and its refinement $C'$ or between their implementations, respectively. An example serves to illustrate this.

### 6 Example

Assume that we want to find an index $i$ pointing to the smallest element in an array $A[1..n]$, where $n = \text{len} . A$ is the length of the array and $1 \leq n$ (so the array is nonempty).

We define the predicate $\text{minat} . i . A$ to hold when the minimum value in $A$ is found at $A[i]$.

$$\text{minat} . i . A \triangleq 1 \leq i \leq \text{len} . A \land (\forall j \mid 1 \leq j \leq \text{len} . A \cdot A[i] \leq A[j])$$

Then the contract

$$\text{MIN} \triangleq \{ 1 \leq \text{len} . A \}; [i := i' \mid \text{minat} . i' . A]$$

constitutes the problem of finding an index $i$ with the minimum $A[i]$ in $A$. 

In this example only input-output test-cases are considered. We define
$$\text{TC } i \ o \triangleq \{A = i\}; [i := i'|i' = o]$$
to be the corresponding deterministic test-case contracts with the input array $A$ and the output index $i$ containing the minimum of the array.

Assume the following test-cases for $MIN$ have been designed:

$$T_1 \triangleq \text{TC } [2, 1, 3] \ 2$$
$$T_2 \triangleq \text{TC } [2, 3, 1] \ 3$$
$$T_3 \triangleq \text{TC } [1, 2, 3] \ 1$$

In [4] it is shown how these test-cases are derived by following a mutation testing strategy. Note that only deterministic test-cases are used, although the specification is non-deterministic. We have to add a non-deterministic test-case with two possible outputs

$$T_4 \triangleq \text{TC } [1, 1, 3] \ (i = 1 \lor i = 2)$$

in order to obtain sufficient coverage with respect to mutation testing.

Next, consider the following implementation (refinement) of $MIN$ using a guarded iteration statement:

$$MIN \sqsubseteq MIN' \triangleq \begin{array}{l}
\text{begin var } k := 2; \ i := 1;
\text{do } k \leq n \land A[k] < A[i] \rightarrow i, k := k, k + 1 \\
\quad \quad \quad \quad [\ k \leq n \land A[k] \geq A[i] \rightarrow k := k + 1 \\
\text{od}
\end{array} \text{ end}$$

What test-cases should be added in order to test the refinement? The rule in Theorem [4] points us to the answer, since a correct test-case of $MIN'$ should be found that is not an abstraction of $MIN$. It is straightforward to reconsider the non-deterministic test-case, since the implementation $MIN'$ is deterministic. Hence, new test-cases should be able to distinguish the deterministic from the non-deterministic version (mutant):

$$T_{\text{new}} \triangleq \text{TC } [1, 1, 3] \ 1$$

is such a useful test-case. Our framework actually allows us to prove this fact by showing that $MIN' \equiv T_{\text{new}}$ and $MIN \not\equiv T_{\text{new}}$.

7 Conclusion

Summary. In this article we have presented a unified view on testing and refinement. First, category theory has been used to analyze the relations of testing and refinement on a high-level of abstraction. A commuting diagram was able to capture the essential properties. Then, the refinement calculus was used to take a closer look at this diagram. Refinement and abstraction have been defined
formally. It has been shown that test-cases are in fact abstract contracts of specifications or implementations. Finally, the novel framework was able to answer the question “Which new test-cases are needed if we refine a specification or implementation?” A new formal inference rule for assessing new test-cases under refinement has been provided.

**Related Work.** To our current knowledge no other work has been published that uses a refinement calculus for deriving test-cases. Stepney was the first who made the abstraction relation between test-cases and object-oriented Z specifications explicitly [12]. Her group developed a tool for interactively calculating partition abstractions and to structure them for reuse. Our work can be seen as an extension of this view on testing. We are happy to see that recently other colleagues seem to pick up our ideas and transfer the results to other frameworks and application domains like modal logics and security [11].

Derrick and Boiten looked at testing refinements in state-based Z specifications [7]. We assume that our framework can express the properties they have collected for refining this kind of specifications. This is part of future work.

**Discussion.** Our approach to testing is a novel unification of the area of testing and formal methods in several senses: First, test-cases are considered as a special form of specifications or contracts. Next, program synthesis techniques, like the refinement calculus can be applied to test-synthesis — it is just abstraction. Finally, the notion of refinement is sufficient to clarify the relations of testing and step-wise development — it forms a category of contracts. The work presented demonstrates that such a unification leads to simpler theories and to a better understanding of the different test-approaches available. In this paper, for example, we have found a scientifically defensible formula for which test-case to be added under refinement.

Here, we did not address the important issue of test coverage explicitly, but have given a rule for calculating test-cases for refinements. In our previous papers we have given abstraction rules for different selection strategies [2,3,4]. The new rule for refinement test-cases can be easily combined with these rules such that the new test-cases follow a certain test-strategy. One has just to assure that the new test-cases \( T'_{\text{new}} \) in Theorem 4 are calculated according to a strategy rule and that \( T'_{\text{new}} \) is not an abstraction of \( C \).

**Future Work.** We hope that our newly gained insight that testing refinements can be mapped to mutation testing will stimulate further research. Our approach can be easily applied to other frameworks that form a similar category. In category theory we would say that a functor must exists that maps our category of contracts into other frameworks.

Finally, automation is of major concern. Future work must involve the research that analyzes to which extent our approach can be automated and which mathematical framework is best suited to support this automation.
References


The Arithmetical Complexity of Dimension and Randomness

John M. Hitchcock\(^1,\star\), Jack H. Lutz\(^1,\star\), and Sebastiaan A. Terwijn\(^2,\star\star\)

\(^1\) Department of Computer Science, Iowa State University, Ames, IA 50011, USA 
\{jhitchco,lutz\}@cs.iastate.edu

\(^2\) Technische Universität Wien, Wiedner Hauptstrasse 8-10, A-1040 Vienna, Austria.
terwijn@logic.at.

Abstract. Constructive dimension and constructive strong dimension are effectivizations of the Hausdorff and packing dimensions, respectively. Each infinite binary sequence \(A\) is assigned a dimension \(\text{dim}(A) \in [0, 1]\) and a strong dimension \(\text{Dim}(A) \in [0, 1]\).

Let \(\text{DIM}^0\) and \(\text{DIM}^{\alpha}_{\text{str}}\) be the classes of all sequences of dimension \(\alpha\) and of strong dimension \(\alpha\), respectively. We show that \(\text{DIM}^0\) is properly \(\Pi^0_2\), and that for all \(\Delta^0_2\)-computable \(\alpha \in (0, 1]\), \(\text{DIM}^\alpha\) is properly \(\Pi^0_3\).

To classify the strong dimension classes, we use a more powerful effective Borel hierarchy where a co-enumerable predicate is used rather than an enumerable predicate in the definition of the \(\Sigma^0_1\) level. For all \(\Delta^0_2\)-computable \(\alpha \in [0, 1)\), we show that \(\text{DIM}^\alpha_{\text{str}}\) is properly in the \(\Pi^0_3\) level of this hierarchy. We show that \(\text{DIM}^1_{\text{str}}\) is properly in the \(\Pi^0_2\) level of this hierarchy.

We also prove that the class of Schnorr random sequences and the class of computably random sequences are properly \(\Pi^0_3\).

1 Introduction

Hausdorff dimension – the most extensively studied fractal dimension – has recently been effectivized at several levels of complexity, yielding applications to a variety of topics in theoretical computer science, including data compression, polynomial-time degrees, approximate optimization, feasible prediction, circuit-size complexity, Kolmogorov complexity, and randomness\([14,15,3,1,8,5,7,17]\).

The most fundamental of these effectivizations is constructive dimension, which is closely related to Kolmogorov complexity and algorithmic randomness. For every subset \(\mathcal{X}\) of \(C\), the Cantor space of all infinite binary sequences, a constructive dimension \(\text{cdim}(\mathcal{X}) \in [0, 1]\) is assigned. Informally, this dimension is determined by the maximum rate of growth that a lower semicomputable martingale can achieve on all sequences in \(\mathcal{X}\).

\(^\star\) This research was supported in part by National Science Foundation Grant 9988483.

\(^\star\star\) Supported by the Austrian Research Fund (Lise Meitner grant M699-N05). Part of this author’s research was done while visiting the second author at Caltech in January 2002.
Just as Martin-Löf [16] used constructive measure to define the randomness of individual sequences, Lutz [15] used constructive dimension to define the dimensions of individual sequences. Each sequence \( A \in C \) is assigned a dimension \( \dim(A) \in \langle 0, 1 \rangle \) by \( \dim(A) = \text{cdim}({A}) \). Every Martin-Löf random sequence has dimension 1, but there are nonrandom sequences with dimension 1. For every real number \( \alpha \in [0, 1] \), there is a sequence with dimension \( \alpha \).

It is useful to understand the arithmetical complexity of a class of sequences. For example, knowing that \( \text{RAND} \), the class of Martin-Löf random sequences, is a \( \Sigma_0^2 \) class allows the application of Kreisel’s Basis Lemma [12,18] to give a short proof [25] that

\[
\text{RAND} \cap \Delta_2^0 \neq \emptyset. \quad (1.1)
\]

For any \( \alpha \in [0, 1] \), let

\[
\text{DIM}^\alpha = \{ A \in C \mid \text{dim}(A) = \alpha \}.
\]

Lutz [15] showed that

\[
\text{DIM}^\alpha \cap \Delta_2^0 \neq \emptyset \quad (1.2)
\]

for any \( \Delta_2^0 \)-computable \( \alpha \in [0, 1] \). As these dimension classes do not appear to be \( \Sigma_2^0 \), Lutz was unable to apply the Basis Lemma to them, so he used different techniques to prove (1.2).

We investigate the complexities of these dimension classes in terms of the arithmetical hierarchy of subsets of \( C \). We show that \( \text{DIM}^0 \) is properly \( \Pi_2^0 \), and for all \( \Delta_2^0 \)-computable \( \alpha \in (0, 1] \) we show that \( \text{DIM}^\alpha \) is properly \( \Pi_3^0 \). Therefore, the proof for (1.1) using Kreisel’s Basis Lemma cannot directly be used to establish (1.2). (See however the comments made after Corollary 4.10.)

More recently, packing dimension, another important fractal dimension, has also been effectivized by Athreya, Hitchcock, Lutz, and Mayordomo [2]. At the constructive level, this is used in an analogous way to define the strong dimension \( \text{Dim}(A) \in [0, 1] \) for every sequence \( A \). For any \( \alpha \in [0, 1] \), let

\[
\text{DIM}^\alpha_{\text{str}} = \{ A \in C \mid \text{Dim}(A) = \alpha \}.
\]

To classify these strong dimension classes, we use introduce a more powerful effective Borel hierarchy where a co-enumerable predicate is used rather than an enumerable predicate in the definition of the \( \Sigma_1^0 \) level. We show that \( \text{DIM}^1_{\text{str}} \) is properly in the \( \Pi_2^3 \) level of this stronger hierarchy. For all \( \Delta_2^0 \)-computable \( \alpha \in [0, 1] \), we show that \( \text{DIM}^\alpha_{\text{str}} \) is properly in the \( \Pi_3^3 \) level of this hierarchy.

Our techniques for classifying the dimension and strong dimension classes include Baire category, Wadge reductions, and Kolmogorov complexity. We also classify some effective randomness classes.

Section 2 gives an overview of the randomness and dimension notions used in this paper. In Section 3 we introduce the stronger effective Borel hierarchy that we use for the strong dimension classes. Section 4 presents the classification of \( \text{DIM}^\alpha \) and \( \text{DIM}^\alpha_{\text{str}} \).
2 Background on Randomness and Dimension

This section provides an overview of the notions of randomness and dimension used in this paper. We write $\{0,1\}^*$ for the set of all finite binary strings and $C$ for the Cantor space of all infinite binary sequences. In the standard way, a sequence $A \in C$ can be identified with the subset of $\{0,1\}^*$ or $\mathbb{N}$ for which it is the characteristic sequence, or with a real number in the unit interval. The length of a string $w \in \{0,1\}^*$ is $|w|$. The string consisting of the first $n$ bits of $x \in \{0,1\}^* \cup C$ is denoted by $x \upharpoonright n$. We write $w \succeq x$ if $w$ is a prefix of $A$.

2.1 Martin-Löf Randomness

Martin-Löf [16] introduced the notion of a constructive null set. A set is constructively null if it can be covered by a uniform sequence of c.e. open sets that are shrinking in size. That is, $A \subseteq C$ is constructive null if $A \subseteq \bigcap_i U_i$, where $\{U_i\}_{i \in \mathbb{N}}$ is uniformly c.e. such that $\mu(U_i) \leq 2^{-i}$. The sequence $\{U_i\}_{i \in \mathbb{N}}$ is called a Martin-Löf test. An individual sequence $A \in C$ is Martin-Löf random if $\{A\}$ is not constructively null. The Martin-Löf random sequences play an important role in algorithmic information theory, see e.g. Li and Vitányi [13].

Schnorr [20], following Ville [26], characterized constructive null sets in terms of martingales. A function $d : \{0,1\}^* \rightarrow [0,\infty)$ is a martingale if for every $w \in \{0,1\}^*$, $d$ satisfies the averaging condition

$$2d(w) = d(w0) + d(w1),$$

and $d$ is a supermartingale if it satisfies

$$2d(w) \geq d(w0) + d(w1).$$

The success set of $d$ is

$$S^\infty[d] = \left\{ A \in C \left| \limsup_{n \rightarrow \infty} d(A \upharpoonright n) = \infty \right. \right\},$$

i.e., it is the set of all sequences on which $d$ has unbounded value. We say that $d$ succeeds on a class $A \subseteq C$ if $A \subseteq S^\infty[d]$.

Ville [26] proved that a set $A \subseteq C$ has Lebesgue measure 0 if and only if there is a martingale $d$ that succeeds on $A$. Schnorr [20] showed that $A$ is constructively null if and only if $d$ can be chosen to be lower semicomputable, that is, if $d$ can be computably approximated from below. We call such a $d$ constructive.

Martin-Löf [16] proved that there is a universal constructive null set. That is, he proved that there is a Martin-Löf test $\{U_i\}$ such that for every other test $\{V_i\}$ it holds that $\bigcap_i V_i \subseteq \bigcap_i U_i$. By Schnorr’s analysis this implies that there is also a universal constructive supermartingale $d$. That is, for any constructive supermartingale $d'$ there is a $c > 0$ such that $d(w) \geq cd'(w)$ for all $w \in \{0,1\}^*$. We will use this universal supermartingale in section 4. We denote the complement of $S^\infty[d]$ by RAND, so that RAND consists of all the Martin-Löf random sequences.
2.2 Schnorr Randomness

Schnorr [20] criticized the notion of constructive null for an actual lack of constructiveness, and introduced the more constructive notion of a Schnorr null set, which is defined by requiring that the measure of the levels $\mathcal{U}_i$ in a Martin-Löf test be computably approximable to within any given precision. It is easy to see that this is equivalent to the following: $A$ is Schnorr null if $A \subseteq \bigcap_i \mathcal{U}_i$, where $\{\mathcal{U}_i\}_{i \in \mathbb{N}}$ is uniformly c.e. such that $\mu(\mathcal{U}_i) = 2^{-i}$. The sequence $\{\mathcal{U}_i\}_{i \in \mathbb{N}}$ is called a Schnorr test.

Following Schnorr [20], we call an unbounded nondecreasing function $h : \{0, 1\}^* \rightarrow \varnothing \{0, 1\}^*$ an order. (N.B. An “Ordnungsfunktion” in Schnorr’s terminology is always computable, whereas we prefer to leave the complexity of orders unspecified in general.) For any order $h$ and martingale $d$, we define the order $h$ success set of $d$ as

$$S^h[d] = \left\{ A \in C \left| \limsup_{n \rightarrow \infty} \frac{d(A \upharpoonright n)}{h(n)} \geq 1 \right. \right\}.$$

Schnorr pointed out that the rate of success of a constructive martingale $d$ can be so slow that it cannot be computably detected. Thus rather than working with constructive null sets of the form $S^\infty[d]$ with $d$ constructive, he worked with null sets of the form $S^h[d]$, where both $d$ and $h$ are computable. He proved that a set $A$ is Schnorr null if and only if it is included in a null set of the form $S^h[d]$, with $d$ and $h$ computable.

A sequence $A \in C$ is Schnorr random if $\{A\}$ is not Schnorr null. This is related the notion of computable randomness. A sequence $A$ is computably random if for any computable martingale $d$, $A \not\in S^\infty[d]$.

We write $\text{RAND}_{\text{Schnorr}}$ for the class of all Schnorr random sequences and $\text{RAND}_{\text{comp}}$ for the class of all computably random sequences. By definition we have that

$$\text{RAND} \subseteq \text{RAND}_{\text{comp}} \subseteq \text{RAND}_{\text{Schnorr}}.$$

The first inclusion was proved strict by Schnorr [20], and the second inclusion was proved strict by Wang [27].

2.3 Constructive Dimension

Hausdorff [6] introduced the concept of null covers that “succeed exponentially fast” to define what is now commonly called Hausdorff dimension, the most widely used dimension in fractal geometry. Basically, this notion allows one to discern structure in classes of measure zero, and to calibrate them. As for constructive measure, already Schnorr drew special attention to null sets of “exponential order”, although he did not make an explicit connection to Hausdorff dimension.

Lutz [14,15] gave a characterization of Hausdorff dimension in terms of gales, which are a generalization of martingales. Let $s \in [0, \infty)$. An $s$-gale is a function $d : \{0, 1\}^* \rightarrow [0, \infty)$ that satisfies the averaging condition

$$2^s d(w) = d(w0) + d(w1) \quad (2.1)$$
for every $w \in \{0, 1\}^*$. Similarly, $d$ is an $s$-supergale if (2.1) holds with $\geq$ instead of equality. The success set $S^\infty[d]$ is defined exactly as was done for martingales above. Lutz showed that for any class $\mathcal{A} \subseteq C$, the Hausdorff dimension of $\mathcal{A}$ is

$$\dim_H(\mathcal{A}) = \inf \left\{ s \mid \text{there exists an } s\text{-gale } d \text{ for which } \mathcal{A} \subseteq S^\infty[d] \right\}.$$  \hfill (2.2)

Lutz \cite{15} effectivized this characterization to define the constructive dimensions of sets and sequences. An $s$-(super)gale is called constructive if it is lower semi-computable. The constructive dimension of a class $\mathcal{A} \subseteq C$ is

$$\cdim(\mathcal{A}) = \inf \left\{ s \mid \text{there exists a constructive } s\text{-gale } d \text{ for which } \mathcal{A} \subseteq S^\infty[d] \right\}$$ \hfill (2.3)

and the constructive dimension of an individual sequence $A \in C$ is

$$\dim(A) = \cdim(\{A\}).$$

(Supergales can be equivalently used in place of gales in both (2.2) and (2.3) \cite{14, 9, 4}.)

Constructive dimension has some remarkable properties. For example, Lutz \cite{15} showed that for any class $\mathcal{A}$,

$$\cdim(\mathcal{A}) = \sup_{A \in \mathcal{A}} \dim(A).$$ \hfill (2.4)

Also, Mayordomo \cite{17} established a strong connection with Kolmogorov complexity: for any $A \in C$,

$$\dim(A) = \liminf_{n \to \infty} \frac{K(A \upharpoonright n)}{n},$$ \hfill (2.5)

where $K(A \upharpoonright n)$ is the size of the smallest program that causes a fixed universal self-delimiting Turing machine to output the first $n$ bits of $A$. (For comments on the relation of this result to earlier results, see the report \cite{23} by Staiger and section 6 of \cite{15}. For more details on Kolmogorov complexity, we refer to \cite{13}.)

One can also characterize constructive dimension using the Schnorr null sets (see Section 2.2) of exponential order. The following proposition was observed by several authors, including those of \cite{11, 24}.

\textbf{Proposition 2.1.} \textit{Let }$d$\textit{ be the universal constructive supermartingale. For any }$\mathcal{A} \subseteq C$\textit{,}

$$\cdim(\mathcal{A}) = \inf \{ s \in \mathbb{Q} : \mathcal{A} \subseteq S_{2^{(1-s)n}}[d] \}.$$  

\subsection{2.4 Constructive Strong Dimension}

More recently, Athreya, Hitchcock, Lutz, and Mayordomo \cite{2} also characterized packing dimension, another important fractal dimension, in terms of gales. For
this, the notion of strong success of an s-gale $d$ was introduced. The strong success set of $d$ is

$$S_{\text{str}}^\infty[d] = \left\{ A \in C \left| \liminf_{n \to \infty} d(A \upharpoonright n) = \infty \right. \right\}. $$

Analogously to what was done for Hausdorff dimension, packing dimension can be characterized using strong success sets of gales. Effectivizing this in the same way leads to the definition of the constructive strong dimension of a class $A \subseteq C$ as

$$c\text{Dim}(A) = \inf \left\{ s \left| \text{there exists a constructive s-gale } d \text{ for which } A \subseteq S_{\text{str}}^\infty[d] \right. \right\}. $$

The constructive strong dimension of a sequence $A \in C$ is

$$\text{Dim}(A) = c\text{Dim}\{A\}). $$

A pointwise stability property analogous to (2.4) also holds for strong dimension, as well as a Kolmogorov complexity characterization [2]:

$$\text{Dim}(A) = \limsup_{n \to \infty} \frac{K(A \upharpoonright n)}{n} \quad (2.6)$$

for any $A \in C$.

3 Borel Hierarchies

We use $\Sigma^0_n$ and $\Pi^0_n$ to denote the levels of the Borel hierarchy for subsets of Cantor space. The levels of the arithmetical hierarchy (the corresponding effective hierarchy) are denoted by $\Sigma^0_n$ and $\Pi^0_n$.

We will also make use of the following more general hierarchy definition.

**Definition.** Let $\mathcal{P}$ be a class of predicates, let $n \geq 1$, and let $\mathcal{X} \subseteq C$.

- $\mathcal{X} \in \Sigma^0_n[\mathcal{P}]$ if for some predicate $P \in \mathcal{P}$,
  
  $$A \in \mathcal{X} \iff (\exists k_n)(\forall k_{n-1}) \cdots (Qk_1)P(k_n, \ldots, k_2, A \upharpoonright k_1),$$
  
  where $Q = \exists$ if $n$ is odd and $Q = \forall$ if $n$ is even.

- $\mathcal{X} \in \Pi^0_n[\mathcal{P}]$ if for some predicate $P \in \mathcal{P}$,
  
  $$A \in \mathcal{X} \iff (\forall k_n)(\exists k_{n-1}) \cdots (Qk_1)P(k_n, \ldots, k_2, A \upharpoonright k_1),$$
  
  where $Q = \forall$ if $n$ is odd and $Q = \exists$ if $n$ is even.

If we take $\mathcal{P}$ to be $\Delta^0_1$ (decidable predicates), then the above definition is equivalent to the standard arithmetical hierarchy; that is,

$$\Sigma^0_n = \Sigma^0_n[\Delta^0_1]$$

and

$$\Pi^0_n = \Pi^0_n[\Delta^0_1]$$
hold for all $n$. Also, if ALL is the class of all predicates, then we obtain the classical Borel hierarchy:

$$\Sigma^0_n = \Sigma^0_n[\text{ALL}]$$

and

$$\Pi^0_n = \Pi^0_n[\text{ALL}].$$

In this paper, we will also be interested in the cases where $P$ is $\Sigma^0_1$ (enumerable predicates) or $\Pi^0_1$ (co-enumerable predicates). In some cases, the classes in the generalized hierarchy using these sets of predicates are no different that the standard arithmetical hierarchy classes. If $n$ is odd, then $\Sigma^0_n = \Sigma^0_n[\Sigma^0_1]$ as the existential quantifier in the $\Sigma^0_1$ predicate can be absorbed into the last quantifier in the definition of $\Sigma^0_n[\Delta^0_1] = \Sigma^0_n$. Analogously, $\Pi^0_n = \Pi^0_n[\Pi^0_1]$ for odd $n$, and for even $n$ we have $\Sigma^0_n = \Sigma^0_n[\Pi^0_1]$ and $\Pi^0_n = \Pi^0_n[\Sigma^0_1]$. On the other hand, using the complementary set of predicates defines an effective hierarchy that is distinct from and interleaved with the arithmetical hierarchy.

**Proposition 3.1.** 1. If $n$ is odd, then

$$\Sigma^0_n \subsetneq \Sigma^0_n[\Pi^0_1] \subsetneq \Sigma^0_{n+1}$$

and

$$\Pi^0_n \subsetneq \Pi^0_n[\Sigma^0_1] \subsetneq \Pi^0_{n+1}.$$  

2. If $n$ is even, then

$$\Sigma^0_n \subsetneq \Sigma^0_n[\Sigma^0_1] \subsetneq \Sigma^0_{n+1}$$

and

$$\Pi^0_n \subsetneq \Pi^0_n[\Pi^0_1] \subsetneq \Pi^0_{n+1}.$$  

Intuitively, the classes $\Sigma^0_1[\Pi^0_1]$, $\Pi^0_1[\Sigma^0_1]$, $\Sigma^0_2[\Sigma^0_1]$, $\Pi^0_2[\Pi^0_1]$, ... are slightly more powerful than their respective counterparts in the arithmetical hierarchy because they use one additional quantifier that is limited to the predicate. We now give a simple example of a class that is best classified in this hierarchy: the class of all 1-generic sequences is $\Pi^0_2[\Pi^0_1]$ but not $\Pi^0_2$.

**Example 3.2.** Recall that a sequence $X \in C$ is 1-generic (see e.g. Jockusch [10]) if

$$(\forall e)(\exists \sigma \sqsubseteq X)[\{e\}^\sigma(e) \downarrow \lor (\forall \tau \sqsubseteq \sigma)[\{e\}^\tau(e) \uparrow]]$$

From this definition it is immediate that the class $G = \{X \mid X \text{ is 1-generic}\}$ is in $\Pi^0_2[\Pi^0_1]$. To show that $G$ is not $\Pi^0_2$, suppose that it is. Then there is a computable predicate $R$ such that

$$X \in G \iff (\forall n)(\exists m)[R(n, X \upharpoonright m)].$$

As $G$ is dense, we can now easily construct a computable element of it by a computable finite extension argument, which gives a contradiction. (Given $\sigma$ at stage $n$, search for extension $\sigma' \sqsubseteq \sigma$ such that $R(n, \sigma')$. Such extension will be found by density. Take this extension and proceed to stage $n+1$.)
Staiger has pointed out to us that the class $\Pi^0_1[\Sigma^0_1]$ already occurred under a different guise in his paper [22], where it was called $\mathcal{P}$, and several presentations were proven to be equivalent to it. The following definitions are from [21]. For any set $W$ of initial segments define

$$\lim W = \{ A \in C \mid (\forall \sigma \subseteq A) \sigma \in W \}$$

$$W^\sigma = \{ A \in C \mid (\forall \infty \sigma \subseteq A) \sigma \in W \}.$$  

Staiger proved that the classes in $\Pi^0_1[\Sigma^0_1]$ are those of the form $\lim W$, for $W \in \Sigma^0_1$, and the classes in $\Sigma^0_2[\Sigma^0_1]$ are those of the form $W^\sigma$, for $W \in \Sigma^0_1$.

4 Classification of DIM$^\alpha$ and DIM$^\alpha_{\text{str}}$

In this section we investigate the arithmetical complexity of the following dimension and strong dimension classes.

$$\text{DIM}^\alpha = \{ A \in C \mid \dim(A) = \alpha \}$$

$$\text{DIM}^{\leq \alpha} = \{ A \in C \mid \dim(A) \leq \alpha \}$$

$$\text{DIM}^{\geq \alpha} = \{ A \in C \mid \dim(A) \geq \alpha \}$$

$$\text{DIM}^\alpha_{\text{str}} = \{ A \in C \mid \text{Dim}(A) = \alpha \}$$

$$\text{DIM}^{\leq \alpha}_{\text{str}} = \{ A \in C \mid \text{Dim}(A) \leq \alpha \}$$

$$\text{DIM}^{\geq \alpha}_{\text{str}} = \{ A \in C \mid \text{Dim}(A) \geq \alpha \}$$

Let $\alpha \in [0, 1]$ be $\Delta^0_2$-computable. For any such $\alpha$, it is well known that there is a computable function $\hat{\alpha} : \mathbb{N} \to \mathbb{Q}$ such that $\lim_{n \to \infty} \hat{\alpha}(n) = \alpha$. Using (2.5), we have

$$\dim(X) \leq \alpha \iff \liminf_{n \to \infty} \frac{K(X \upharpoonright n)}{n} \leq \alpha$$

$$\iff (\forall k)(\forall N)(\exists n \geq N)K(X \upharpoonright n) < (\hat{\alpha}(n) + 1/k)n,$$

so DIM$^{\leq \alpha}$ is a $\Pi^0_2$-class. Also,

$$\dim(X) \geq \alpha \iff \liminf_{n \to \infty} \frac{K(X \upharpoonright n)}{n} \geq \alpha$$

$$\iff (\forall k)(\exists N)(\forall n \geq N)K(X \upharpoonright n) > (\hat{\alpha}(N) - 1/k)n,$$

so DIM$^{\geq \alpha}$ is a $\Pi^0_3$-class. Therefore we have the following:

**Proposition 4.1.** 1. The class DIM$^0$ is $\Pi^0_2$.

2. For all $\Delta^0_2$-computable $\alpha \in (0, 1]$, DIM$^\alpha$ is a $\Pi^0_3$-class.

3. For arbitrary $\alpha \in (0, 1]$, DIM$^\alpha$ is a $\Pi^0_3$-class.
The situation is slightly more complicated for strong dimension. By (2.6), we have

$$\text{Dim}(X) \leq \alpha \iff \limsup_{n \to \infty} \frac{K(X \upharpoonright n)}{n} \leq \alpha$$

$$\iff (\forall k)(\exists N)(\forall n \geq N) K(X \upharpoonright n) < (\hat{\alpha}(N) + 1/k)n$$

$$\iff (\forall k)(\exists N)(\forall n \geq N) (\exists \langle \pi, t \rangle) |\pi| < (\hat{\alpha}(N) + 1/k)n$$

and $U(\pi) = X \upharpoonright n$ in $\leq t$ computation steps,

where $U$ is the fixed universal self-delimiting Turing machine used to define $K$. From this it is clear that $\text{DIM}_\alpha \in \Pi_4^0$. However, the “$(\exists \langle \pi, t \rangle)$” quantifier is local to the defining predicate, so we have $\text{DIM}_{\text{str}} \leq \alpha \in \Pi_3^0$, and in fact, it is a $\Pi_3^0[\Sigma_1^0]$-class. Also,

$$\text{Dim}(X) \geq \alpha \iff \limsup_{n \to \infty} \frac{K(X \upharpoonright n)}{n} \geq \alpha$$

$$\iff (\forall k)(\forall N)(\exists n \geq N) K(X \upharpoonright n) > (\hat{\alpha}(n) - 1/k)n,$$

so $\text{DIM}_{\text{str}}^\geq \alpha$ is a $\Pi_2^0[\Pi_1^0]$-class. This establishes the following analogue of Proposition 4.1.

**Proposition 4.2.** 1. The class $\text{DIM}_{\text{str}}^1$ is a $\Sigma_0^2[\Pi_1^0]$.
2. For all $\Delta_2^0$-computable $\alpha \in [0, 1)$, $\text{DIM}_{\text{str}}^\alpha$ is a $\Pi_3^0[\Sigma_1^0]$-class.
3. For arbitrary $\alpha \in [0, 1)$, $\text{DIM}_{\text{str}}^\alpha$ is a $\Pi_3^0$-class.

In the remainder of this section we prove that the classifications in Propositions 4.1 and 4.2 cannot be improved in their respective hierarchies.

### 4.1 Category Methods

Recall that a class $\mathcal{X}$ is *meager* if it is included in a countable union of nowhere dense subsets of $C$, and *comeager* if its complement $\overline{\mathcal{X}}$ is meager. The following lemma (implicit in Rogers [19, p341]) will be useful.

**Lemma 4.3.** If $\mathcal{X} \in \Sigma_2^0$ and $\overline{\mathcal{X}}$ is dense then $\mathcal{X}$ is meager.

The class RAND of Martin-Löf random sets can easily be classified with category methods.

**Theorem 4.4.** (folk) $\text{RAND}$ is a $\Sigma_1^0$-class, but not a $\Pi_2^0$-class.

As $\text{DIM}^0$ and $\text{DIM}_{\text{str}}^1$ are dense $\Pi_2^0$-classes that have dense complements, an argument similar to the one used for Theorem 4.4 shows that they are not $\Sigma_2^0$-classes.

**Theorem 4.5.** The classes $\text{DIM}^0$ and $\text{DIM}_{\text{str}}^1$ are not $\Sigma_2^0$-classes.

We now develop category methods for the other $\text{DIM}^\alpha$ classes. For every rational $s$, define the computable order $h_s(n) = 2^{(1-s)n}$. Let $d$ be the optimal constructive supermartingale.
Lemma 4.6. For every rational \( s \in (0, 1) \), \( S^{h_s}[d] \) is a comeager \( \Pi^0_2 \)-class.

Proof. Notice that \( S^{h_s}[d] \in \Sigma^0_2 \) and \( S^{h_s}[d] \) is dense. Now apply Lemma 4.3 \( \square \)

Lemma 4.7. For all \( \alpha \in (0, 1] \), \( \text{DIM}^\alpha \) is meager.

Proof. Let \( s < \alpha \) be rational. Lutz [15] showed that \( d^{(s)}(w) = 2^{(s-1)|w|}d(w) \) is an optimal constructive \( s \)-supergale. It follows that for any \( A \subseteq C \), \( A \in S^{h_s}[d] \Rightarrow \dim(S) < \alpha \). Therefore \( \text{DIM}^\alpha \subseteq S^{h_s} \), so \( \text{DIM}^\alpha \) is meager by Lemma 4.6 \( \square \)

Proposition 4.8. For all \( \alpha \in (0, 1] \), \( \text{DIM}^\alpha \) is not a \( \Pi^0_2 \)-class.

Proof. If \( \text{DIM}^\alpha \in \Pi^0_2 \), then Lemma 4.3 implies that \( \text{DIM}^\alpha \) is comeager, contradicting Lemma 4.7 \( \square \)

To strengthen Proposition 4.8 to show that \( \text{DIM}^\alpha \) is not \( \Sigma^0_3 \)-class, we now turn to Wadge reductions.

4.2 Wadge Reductions

Let \( A, B \subseteq C \). A Wadge reduction of \( A \) to \( B \) is a function \( f : C \to C \) that is continuous and satisfies \( A = f^{-1}(B) \), i.e., \( X \in A \iff f(X) \in B \). We say that \( B \) is Wadge complete for a class \( \Gamma \) of subsets of \( C \) if \( B \in \Gamma \) and every \( A \in \Gamma \) Wadge reduces to \( B \). As the classes of the Borel hierarchy are closed under Wadge reductions, Wadge completeness can be used to properly identify the location of a subset of \( C \) in the hierarchy.

We now prove that \( \text{DIM}^1 \) is Wadge complete for \( \Pi^0_3 \). We will then give Wadge reductions from it to \( \text{DIM}^\alpha \) for the other values of \( \alpha \).

Theorem 4.9. \( \text{DIM}^1 \) is Wadge complete for \( \Pi^0_3 \). Therefore \( \text{DIM}^1 \) is not a \( \Sigma^0_3 \)-class, and in particular is not a \( \Sigma^0_3 \)-class.

Proof. One could prove this by reducing a known \( \Pi^0_3 \)-complete class to \( \text{DIM}^1 \), e.g. the class of sets that have a limiting frequency of 1's that is 0 (this class was proved to be \( \Pi^0_3 \)-complete by Ki and Linton [11]), but it is just as easy to build a direct reduction from an arbitrary \( \Pi^0_3 \)-class.

Let \( d \) be the universal constructive supermartingale. Note that we have (cf. Proposition 2.1)

\[
S^{2^n}[d] \subseteq \ldots \subseteq S^{2^{\frac{1}{2^n}}}[d] \subseteq S^{2^{\frac{1}{2^{n+1}}}}[d] \subseteq \ldots \subseteq \text{DIM}^1.
\]

Let \( \bigcup_k \bigcap_s \mathcal{O}_{k,s} \) be a \( \Sigma^0_3 \)-class. Without loss of generality \( \mathcal{O}_{k,s} \supseteq \mathcal{O}_{k,s+1} \) for all \( k,s \). We define a continuous function \( f : C \to C \) such that

\[
\forall k \left( X \in \bigcap_s \mathcal{O}_{k,s} \iff f(X) \in S^{2^{\frac{1}{2^n}}}[d] \right) \quad (4.1)
\]
so that we have
\[ X \not\in \bigcup_k \bigcap_s O_{k,s} \iff \forall k \left( f(X) \not\in S^{2^{t_n}}[d] \right) \iff f(X) \in \text{DIM}^1. \]

The image \( Y = f(X) \) is defined in stages, \( Y = \bigcup_s Y_s \), such that every initial segment of \( X \) defines an initial segment of \( Y \).

At stage 0 we define \( Y_0 \) to be the empty sequence.

At stage \( s > 0 \) we consider \( X \rceil s \), and for each \( k \) we define \( t_{k,s} \) to be the largest stage \( t \leq s \) such that \( X \rceil t \in O_{k,t} \). (Let \( t_{k,s} = 0 \) if such a \( t \) does not exist.) Define \( k \) to be expansionary at stage \( s \) if \( t_{k,s-1} < t_{k,s} \).

Now we let \( k(s) = \min\{k : k \text{ is expansionary at } s\} \). There are two substages.

Substage (a). First consider all strings \( \sigma \) extending \( Y_{s-1} \) of minimal length with \( d(\sigma) \geq 2^{\frac{1}{n} |\sigma|} \), and take the leftmost one of these \( \sigma \)'s. Such \( \sigma \)'s exist because \( S^{2^{\frac{1}{n} |\cdot|}}[d] \) is dense. If \( k(s) \) does not exist, let \( \sigma = Y_{s-1} \).

Substage (b). Next consider all extensions \( \tau \supseteq \sigma \) of minimal length such that \( d(\tau \rceil i) \leq d(\tau \rceil (i-1)) \) for every \( |\sigma| < i < |\tau| \), and \( d(\tau) \leq |\tau| \). Clearly such \( \tau \) exist, by direct diagonalization against \( d \). Define \( Y_s \) to be the leftmost of these \( \tau \). This concludes the construction.

So \( Y_s \) is defined by first building a piece of evidence \( \sigma \) that \( d \) achieves growth rate \( 2^{\frac{1}{n} |\cdot|} \) on \( Y \) and then slowing down the growth rate of \( d \) to the order \( n \). Note that \( f \) is continuous. If \( X \in \bigcup_k \bigcap_s O_{k,s} \), then for the minimal \( k \) such that \( X \in \bigcap_s O_{k,s} \), infinitely many pieces of evidence \( \sigma \) witness that \( d \) achieves growth rate \( 2^{\frac{1}{n} |\cdot|} \) on \( Y \), so \( Y \notin \text{DIM}^1 \). On the other hand, if \( X \notin \bigcup_k \bigcap_s O_{k,s} \), then for every \( k \) only finitely often \( d(Y_s) \geq 2^{\frac{1}{n} |Y_s|} \) because in substage (a) the extension \( \sigma \) is chosen to be of minimal length, so \( Y \notin S_{h_k}[d] \). Hence \( Y \in \text{DIM}^1 \). \( \square \)

As \( \text{RAND} \) is a \( \Sigma^0_2 \)-class, we have the following corollary (which can also be proved by a direct construction).

**Corollary 4.10.** (Lutz [15]) \( \text{RAND} \) is a proper subset of \( \text{DIM}^1 \).

In order to establish the existence of \( \Delta^0_2 \)-computable sequences of any \( \Delta^0_2 \)-computable dimension \( \alpha \in [0,1) \), Lutz [15] defined a dilution function \( g_\alpha : C \to C \) that is computable and satisfies \( \dim(g_\alpha(X)) = \alpha \cdot \dim(X) \) for all \( X \in C \). Applying this to any \( \Delta^0_2 \)-computable Martin-Löf random sequence (which must have dimension 1) establishes the existence theorem. (We note that \( g_\alpha(X) \) has the same Turing degree as \( X \). Since by the Low Basis Theorem of Jockusch and Soare [18, Theorem V.5.32] there are Martin-Löf random sets of low degree, we immediately obtain that there are low sets of any \( \Delta^0_2 \)-computable dimension \( \alpha \).) As \( g_\alpha \) is continuous, it is a Wadge reduction from \( \text{DIM}^1 \) to \( \text{DIM}^\alpha \) if \( \alpha > 0 \). Combining this with the previous theorem, we have that \( \text{DIM}^\alpha \) is Wadge complete for \( \Pi^0_3 \) for all \( \Delta^0_2 \)-computable \( \alpha \in (0,1) \). We now give a similar dilution construction that will allow us to prove this for arbitrary \( \alpha \in (0,1) \).
Let $X \in C$ and let $\alpha \in (0, 1)$. Write $X = x_1x_2x_3 \ldots$ where $|x_n| = 2n - 1$ for all $n$, noting that $|x_1 \cdots x_n| = n^2$. For each $n$, let $k_n = \lceil n \frac{1-\alpha}{\alpha} \rceil$ and $y_n = 0^{k_n}$. We then define $f_\alpha(X) = x_1y_1x_2y_2 \cdots x_ny_n \cdots$. Observe that $f_\alpha$ is a continuous function mapping $C$ to $C$. We now show that it modifies the dimension of $X$ in a controlled manner.

**Lemma 4.11.** For any $X \in C$ and $\alpha \in (0, 1)$, $\dim(f_\alpha(X)) = \alpha \cdot \dim(X)$ and $\Dim(f_\alpha(X)) = \alpha \cdot \Dim(X)$.

The function $f_\alpha$ establishes the completeness of $\text{DIM}^\alpha$.

**Theorem 4.12.** For all $\alpha \in (0, 1)$, $\text{DIM}^\alpha$ is Wadge complete for $\Pi^0_3$. Therefore it is not a $\Sigma^0_3$-class, and in particular not a $\Sigma^0_3$-class.

**Proof.** By Lemma 4.11, $f_\alpha$ is a Wadge reduction from $\text{DIM}^1$ to $\text{DIM}^\alpha$. Therefore $\text{DIM}^\alpha$ is Wadge complete for $\Pi^0_3$ by composing $f_\alpha$ with the reduction from Theorem 4.9. □

For lack of space, we state the following theorems without proof.

**Theorem 4.13.** For all $\alpha \in [0, 1)$, $\text{DIM}^\alpha_{\text{str}}$ is Wadge complete for $\Pi^0_3$. Therefore $\text{DIM}^\alpha_{\text{str}}$ is not a $\Sigma^0_3$-class, and in particular is not a $\Sigma^0_3[[\Pi^0_1]]$-class.

**Theorem 4.14.** $\text{RAND}_{\text{Schnorr}}$ is a $\Pi^0_3$-class, but not a $\Sigma^0_3$-class.

**Theorem 4.15.** $\text{RAND}_{\text{comp}}$ is a $\Pi^0_3$-class, but not a $\Sigma^0_3$-class.

### 4.3 Ad Hoc Methods

When the level of the class in the effective hierarchy is not the same as the level in the classical hierarchy one often needs to resort to ad hoc arguments. One might think that the notion of effective Wadge reduction, or recursive functional, would be the proper notion to use in classifying classes of reals in the effective hierarchy. However, this notion is rarely useful for the following reason. Let $\mathcal{X}$ be a class without computable elements, such as the class of Martin-Löf random sets or the class of 1-generic sets. Then $\mathcal{X}$ cannot be proven to be complete for any level of the effective hierarchy by a recursive Wadge reduction $f$. For if $\mathcal{X}$ is recursive, then so is $f(\mathcal{X})$, so we can never have $X \in C \iff f(X) \in \mathcal{X}$. So we see that “easy” classes like $C$ that contain recursive elements cannot be reduced in such a way to many “difficult” classes, which renders the notion rather useless.

We have left open the question whether $\text{DIM}^1_{\text{str}}$ is not in $\Pi^0_2$, and whether $\text{DIM}^\alpha_{\text{str}}$ is not in $\Pi^0_3$ for any $\Delta^0_2$-computable $\alpha \in [0, 1)$. We have no answer to the second question, but we provide an answer to the first in the next theorem. We make use of the following lemma:

**Lemma 4.16.** If $\mathcal{X} \in \Pi^0_2$ is dense then there is a computable $X \in \mathcal{X}$. 

Proof. This is an easy finite extension argument. Suppose that $\mathcal{X} = \{ X : (\forall m)(\exists k)R_X^X(m, k) \downarrow = 1 \} \in \Pi^0_2$ is dense. (Here $R$ is a computable predicate. Note that $R$ does not have to be defined with oracles $X$ that are not in $\mathcal{X}$.) Given any initial segment $\tau$ such that

$$(\forall n < m)(\exists k)R^\tau(m, k) \downarrow = 1,$$

we show how to compute an extension $\sigma \supseteq \tau$ such that

$$(\exists k)R^\sigma(m, k) \downarrow = 1. \quad (4.2)$$

Because $\mathcal{X}$ is dense, there are $X \supseteq \tau$ and $k$ such that $R_X^X(m, k) \downarrow = 1$. Let $u$ be the use of this computation, i.e. the part of the oracle $X$ used in it. Now define $\sigma = \max\{ X \upharpoonright u, \tau \}$. Then $\sigma \supseteq \tau$ satisfies (4.2).

Now it is clear that for every $m$ we can compute appropriate extensions $\sigma_m$ such that $X = \bigcup_m \sigma_m$ is computable and $(\forall m)(\exists k)R^{\sigma_m}(m, k) \downarrow = 1$, so that $X \in \mathcal{X}$. \hfill □

Theorem 4.17. $\text{DIM}^1_{\text{str}}$ is not a $\Pi^0_2$-class. Hence it is properly $\Pi^0_2[\Pi^0_1]$.

Proof. Suppose that $\text{DIM}^1_{\text{str}}$ is $\Pi^0_2$. Then, since clearly $\text{DIM}^1_{\text{str}}$ is dense, by Lemma 4.16 it contains a computable real, contradicting that every computable real has strong dimension 0. \hfill □

We conclude this section by summarizing its main results in the following table.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$\text{DIM}^\alpha$</th>
<th>$\text{DIM}^\alpha_{\text{str}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha = 0$</td>
<td>$\Pi^0_2 - \Sigma^0_3$</td>
<td>$\Pi^0_3 \setminus \Sigma^0_1 \setminus \Sigma^0_3$</td>
</tr>
<tr>
<td>$\alpha \in (0, 1) \cap \Delta^0_2$</td>
<td>$\Pi^0_3 - \Sigma^0_3$</td>
<td>$\Pi^0_3 \setminus \Sigma^0_1 \setminus \Sigma^0_3$</td>
</tr>
<tr>
<td>$\alpha = 1$</td>
<td>$\Pi^0_3 - \Sigma^0_3$</td>
<td>$\Pi^0_3 \setminus \Pi^0_1 \setminus (\Sigma^0_2 \cup \Pi^0_2)$</td>
</tr>
<tr>
<td>arbitrary $\alpha \in (0, 1)$</td>
<td>$\Pi^0_3 - \Sigma^0_3$</td>
<td>$\Pi^0_3 - \Sigma^0_3$</td>
</tr>
</tbody>
</table>

Question 4.18. Is it the case that $\text{DIM}^\alpha_{\text{str}}$ is not in $\Pi^0_3$ for any $\Delta^0_2$-computable $\alpha \in [0, 1)$?

References

Towards a Proof System for Admissibility

Rosalie Iemhoff

Technical University Vienna, Wiedner Hauptstrasse 8–10, A-1040 Vienna, Austria
iemhoff@logic.at
http://www.logic.at/people/iemhoff

Abstract. In [4] a basis for the admissible rules of intuitionistic propositional logic IPC was given. Here we strengthen this result by presenting a system ADM that has the following two properties. $A \vdash_{ADM} B$ implies that $A$ admissibly derives $B$. ADM is complete in the sense that for every formula $A$ there exists a formula $A \vdash_{ADM} A_A$ such that the admissibly derivable consequences of $A$ are the (normal) consequences of $A_A$. This work is related to and partly relies upon research by Ghilardi on projective formulas [2] [3]... .

1 Introduction

An interesting meta-mathematical property of intuitionistic propositional logic IPC is the existence of non-derivable admissible rules for this logic. That is, there exist rules that, when added to IPC, do not lead to new theorems, but that are not derivable in IPC either. We write $A \vdash B$ if $A/B$ is an admissible rule. Some results during the last decade have shed some light on the structure of the admissible derivability relation $\vdash$. Rybakov [6] showed that $\vdash$ is decidable and Ghilardi [3] presented a transparent algorithm. In [4] a simple syntactical characterization for $\vdash$ was given. This result implied that Visser’s rules $V = \{V_n \mid n = 1, 2, \ldots\}$, where

$$V_n (\bigwedge_{i=1}^n (A_i \to B_i) \to A_{n+1} \lor A_{n+2}) \vdash \bigvee_{j=1}^{n+2} \bigwedge_{i=1}^n (A_i \to B_i) \to A_j,$$

form a basis for the admissible rules of IPC. Intuitively, this means that all admissible rules of IPC can be obtained from Visser’s rules via derivability in IPC. Here we strengthen this result in the following way. Our aim is to present a decent proof system for $\vdash$, i.e. a system that given $A \vdash B$, tells what are the syntactical manipulations one has to apply to $A$ to obtain $B$. With decent we mean something like “as cut-free as possible”. The mentioned characterization does not fulfill this aim since it contains a full cut rule: if $A \vdash B$ and $B \vdash C$, then $A \vdash C$. The form of $V$ does not seem to suggest the existence of a neat cut-free Gentzen calculus for $\vdash$, as the rules $V_n$ violate the subformula property.

* Research supported by a Marie Curie fellowship of the European Union under grant HPMC-CT-2001-01383.
Although cut elimination might not be possible in full extent, it might however still be possible to reach the above mentioned goal in a satisfying way. Here we present a result in this direction. We present a proof system $\text{ADM}$ such that $A \vdash_{\text{ADM}} B$ implies $A \Vdash B$. The converse does not hold. However, the system is complete in the sense that for every formula $A$ there exists a unique formula $A \vdash_{\text{ADM}} \Lambda_A$ such that

$$\forall B : A \Vdash B \text{ iff } \Lambda_A \vdash B.$$ 

Thus, given $A$, once one has obtained $\Lambda_A$ via $\text{ADM}$, $A \Vdash B$ is reduced to $\Lambda_A \vdash B$.

The system $\text{ADM}$ is “cut-free” in the sense that it consists solely of “rewriting” rules and applications of $V$. With “rewriting” rules we mean in this context simple rules that e.g. infer $(A \rightarrow B) \land (A \rightarrow C)$ from $(A \rightarrow B \land C)$. In contrast to $V$, all these rules of $\text{ADM}$ are derivable admissible rules. Note that this result implies that $V$ is a basis for the admissible rules of $\text{IPC}$. However, it is stronger in the sense that it provides more information about $\Vdash$.

These results are intimately linked with and inspired by results from Ghilardi’s papers [2] and [3]. In fact, our main theorem heavily relies on results in [3]. His results stem from research on unification in intermediate and modal logics. To this end he defines in [2] the notion of a projective formula. A formula $A$ is called projective if there exists a substitution $\sigma$ such that $\vdash \sigma A$ and $\forall B (A \vdash \sigma B \leftrightarrow B)$. Ghilardi shows in [2] that every formula $A$ has a projective approximation $\Pi(A)$. $\Pi(A)$ is a finite set of projective formulas that derive $A$ and such that for every projective $C$ that derives $A$ there exists $D \in \Pi(A)$ such that $C \vdash D$. It follows from [2] that, like for $\Lambda_A$, it holds that $A \Vdash B$ iff $\bigvee \Pi(A) \vdash B$. However, for our aim of giving a proof system that derives $\Lambda_A$ from $A$, it seems that the notion of admissible projective sets instead of projective approximations is more useful. An admissible projective set for a formula $A$ is a set of formulas $X$ such that all formulas in $X$ are either inconsistent or projective and

$$A \Vdash \bigvee X \vdash A.$$ 

A formula $A$ is inconsistent if $A \vdash \bot$. The special behavior of projective formulas w.r.t. admissibility clearly holds for inconsistent formulas too:

For every projective or inconsistent formula $A$, for all formulas $B$: $A \Vdash B$ iff $A \vdash B$. Now the main theorems of the paper, Theorem 1 and 2, state the following: Theorem 1: $A \vdash_{\text{ADM}} B$ then $A \Vdash B$.

Theorem 2: For every formula $A$ there exists a unique formula $\Lambda_A = \bigvee dj(\Lambda_A)$, such that $dj(\Lambda_A)$ is an admissible projective set for $A$ and $A \vdash_{\text{ADM}} \Lambda_A$.

These results have various consequences for other intermediate logics. The most interesting one being that $V$ is a basis for the admissible rules for any
Towards a Proof System for Admissibility

intermediate logic for which it is admissible. Due to lack of space we are not able to include these results here. They are discussed in [5]. The system ADM is not yet as elegant and neat, and whence as useful, as one would hope and expect. In particular, instead of replacing previously derived expressions, it just adds expressions to the initial formula. We hope to be able to present a better system in the near future.

2 Preliminaries

In this paper we will only be concerned with intuitionistic propositional logic IPC. We write $\vdash$ for derivability in IPC. The letters $A, B, C, D, E, F, H$ range of formulas, the letters $p, q, r, s, t$, range over propositional variables. We assume $\top$ and $\bot$ to be present in the language. $\neg A$ is defined as $(A \to \bot)$. We omit parentheses when possible; $\land$ binds stronger than $\lor$, which in turn binds stronger than $\to$. A substitution $\sigma$ will in this paper always be a map from propositional formulas to propositional formulas that commutes with the connectives. A (propositional) admissible rule of IPC is a rule $A/B$ such that adding the rule to the logic does not change the theorems of IPC, i.e. $\forall \sigma : \vdash \sigma A \Rightarrow \vdash \sigma B$. We write $A \vdash B$ if $A/B$ is admissible. The rule is called derivable if $A \vdash B$ and proper if $A \not\vdash B$. We say that a collection $\mathcal{R}$ of rules, e.g. $V$, is admissible if all rules in $\mathcal{R}$ are admissible. We write $A \vdash^\mathcal{R} B$ if $B$ is derivable from $A$ in the logic consisting of IPC extended with the rules $\mathcal{R}$, i.e. there are $A = A_1, \ldots, A_n = B$ such that for all $i < n$, $A_i \vdash A_{i+1}$ or there exists a $\sigma$ such that $\sigma B_i/\sigma B_{i+1} = A_i/A_{i+1}$ and $B_i/B_{i+1} \in \mathcal{R}$. A set $\mathcal{R}$ of admissible rules is a basis for the admissible rules if for every admissible rule $A \vdash B$ we have $A \vdash^\mathcal{R} B$.

3 The System ADM

The aim of the system ADM is to derive a formula $\Lambda_A$, which is the disjunction of an admissible projective set for $A$, by applying some syntactical rules to $A$. Intuitively it works as follows. First $A$ is rewritten into a disjunction of conjunctions. Under all other rewriting rules that follow it will remain in this form, i.e. in each step the algorithm will only change one of the conjuncts of some disjunct of the formula. We write these conjuncts as sequents, which are interpreted in the usual way as implications. For the outer conjunctions and disjunctions we use different symbols, to distinguish them from the symbols in the sequent. The symbol $\cdot$ is interpreted as conjunction, the symbol $|$ as disjunction, where $|$ binds stronger than $\cdot$. Thus e.g. $S_1 \cdot S_2 \cdot S_3$ stands for $S_1 \lor (S_2 \land S_3)$. The interpretation of $|$ as disjunction is in accordance with its interpretation in hypersequent calculi, see e.g. [1]. Our sequents will be of the form $\Gamma, \Pi \Rightarrow \Delta$, where $\Gamma$, $\Pi$ and $\Delta$ are sets of formulas. We denote these sets without the {}‘s and the comma’s. Thus e.g. $A (B \land C), D \Rightarrow E F$ is a sequent in the sense defined above, and means $\{A, B \land C\}, \{D\} \Rightarrow \{E, F\}$. Also, if we write $\Gamma \{A_i \mid i = 1, 2\}, \Pi \Rightarrow \Delta$ we mean $\Gamma A_1 A_2, \Pi \Rightarrow \Delta$. We let $S$ range over sequents, $\Theta$ over expressions of the form $S_1 \cdot S_2 \cdot \ldots \cdot S_n$, and $\Lambda$ over expressions
of the form $\Theta_1 \mid \Theta_2 \mid \ldots \mid \Theta_n$. The $\Theta$ are called conjunction expressions and the $\Lambda$ are called disjunction expressions. Both $\Theta$ and $\Lambda$ are considered as sets which components are respectively sequents and conjunction expressions. Thus e.g. $S_1 \cdot S_2 \cdot S_1 = S_2 \cdot S_1$. If $\Theta = S_1 \cdot S_2 \cdot \ldots \cdot S_n$, the $S_i$ are called the conjuncts of $\Theta$, and $\Theta$ is interpreted as the formula $\bigwedge S_i' \land \ldots \land S_n'$, where $S_i'$ is the formula corresponding to the sequent $S_i$. We write $S \in \Theta$ if $\Theta$ is of the form $\Theta' \cdot S$, i.e. if $S$ is a conjunct of $\Theta$. For $A = \Theta_1 \mid \Theta_2 \mid \ldots \mid \Theta_n$, the $\Theta_i$ are called the disjuncts of $\Lambda$, and $\Lambda$ is interpreted as the formula $(\bigvee \Theta_i') \lor \ldots \lor (\bigvee \Theta_n')$, where $\Theta_i'$ is the formula corresponding to $\Theta_i$. Sometimes we use sequents and conjunction and disjunction expressions also for their corresponding propositional formula. If we write a sequent but mean a formula, we often leave out the comma between disjunction expressions since it plays no role in the context of formulas: $\Gamma \Pi \Rightarrow \Delta$ has the same interpretation as $\Gamma, \Pi \Rightarrow \Delta$. We write $\Gamma \Pi$ for $\Gamma \cup \Pi$. The empty conjunction is taken to be $\top$ and the empty disjunction is $\bot$. For a formula $A$ we define its corresponding disjunction expression (the $s$ for succedent):

$$A_{sA} \equiv_{df} ( \, \Rightarrow \, A).$$

Note that $\vdash A \leftrightarrow A_{sA}$. In fact, following our convention, $A_{sA}$ is interpreted as the formula $(\top \rightarrow A)$ in $\vdash A \leftrightarrow A_{sA}$.

**Definition 1.** We associate the following sets with a given set $\Gamma$:

$$\Gamma^a = \{ A \mid \exists B \ (A \rightarrow B \in \Gamma) \} \quad \Gamma^p = \{ A \in \Gamma \mid \exists B \ (A = p \rightarrow B) \}$$

$$C^{A \rightarrow B} = \begin{cases} B \rightarrow D & \text{if } C = (A \rightarrow B) \rightarrow D \\ C & \text{otherwise} \end{cases}$$

$$C^{A \wedge} = \begin{cases} A \rightarrow (A' \rightarrow B) & \text{if } C = (A \wedge A' \rightarrow B) \\ C & \text{otherwise} \end{cases}$$

$$C^{\Lambda \wedge A} = \begin{cases} A \rightarrow (A' \rightarrow B) & \text{if } C = (A' \wedge A \rightarrow B) \\ C & \text{otherwise} \end{cases}$$

$$C^{A \vee A'} = \begin{cases} (A \rightarrow B) \land (A' \rightarrow B) & \text{if } C = (A \vee A' \rightarrow B) \\ C & \text{otherwise} \end{cases}$$

For a set $\Gamma$ we define $\Gamma^{A \rightarrow B} = \{ C^{A \rightarrow B} \mid C \in \Gamma \}$. Similarly for the other superscripts.
Towards a Proof System for Admissibility 259

Let us check that the following holds. If derivations have some special properties, as we will see in Proposition 1. We consider all sequents \( \Gamma, \Pi \Rightarrow \Delta \in \Theta \) that are explicitly indicated in the rules above. The expressions \( \Delta = \Lambda/\Lambda \) are the principal conjunction expressions of the inference. The sequents that do not occur in these derivations are called principal sequents of the inference. Note that rules have in general more than one principal formula, sequent and conjunction expression. If \( \Lambda/\Lambda' \) is an instance of a rule \( R \), then we write \( \Lambda \vdash R^i_{\text{ADM}} \Lambda' \) or \( \Lambda \vdash 1_{\text{ADM}} \Lambda' \). If there are \( A_1, \ldots, A_n \) such that \( A = A_1 \vdash 1_{\text{ADM}} \ldots \vdash 1_{\text{ADM}} A_n = \Lambda' \), we write \( \Lambda \vdash \Lambda_{\text{ADM}} \Lambda' \). The depth of a disjunction expression \( \Lambda \) in a derivation \( D = A_1, \ldots, A_n \) is the minimal \( i \) such that \( A = A_i \). The depth of a conjunction expression \( \Theta \) in \( D \) is the minimal \( i \) such that \( \Theta \) is a conjunct of \( A_i \).

We will be only interested in derivations \( \Lambda_{sA} \vdash \Lambda_{\text{ADM}} \Lambda \), where the first expression is of the special form \( \Lambda_{sA} \) for some formula \( A \). The \( \Lambda \) that occur in these derivations have some special properties, as we will see in Proposition 1. We leave it to the reader to check that the following holds. If \( \Lambda_{sA} \vdash \Lambda_{\text{ADM}} \Lambda \), then for all sequents \( \Gamma, \Pi \Rightarrow \Delta \) that occur in \( \Lambda \), the set \( \Pi \) consists only of implications.

**Definition 2.** A set \( \Gamma \) is called simple if all its elements are implications or propositional variables. A sequent \( \Gamma, \Pi \Rightarrow \Delta \) is called simple if \( \Gamma \Pi \Delta \) is simple. A derivation in \( \Lambda_{\text{ADM}} \) is in normal form if the principal sequents in any \( R \) or \( Vg \) inference are simple. If \( D \) is the smallest normal form derivation \( \Lambda_{sA} \vdash \Lambda_{\text{ADM}} \Lambda \) such that for all \( A \vdash 1_{\text{ADM}} \Lambda' \) we have \( \Lambda' = \Lambda \), then we denote \( \Lambda \) by \( \Lambda_{A} \). Observe that \( \Lambda_{A} \) is unique. We denote the set of all disjuncts of a disjunction expression \( \Lambda \) by \( dj(\Lambda) \). Thus \( \Lambda \leftrightarrow \bigvee dj(\Lambda) \). We define

\[
p(\Lambda) = \{ \Theta \mid \Theta \in dj(\Lambda), \Theta \text{ has no conjunct } (\Gamma, \Pi \Rightarrow \bot) \text{ for which } \Gamma^a = \{ \bot \} \}.
\]
3.1 Properties of ADM

All rules of ADM except \( \wedge \), \( \lor \) and \( \rightarrow \) have a subformula property: the formulas that occur in the sequents of the lower expression are subformulas of the formulas in the sequents in the upper expression. The reason that e.g. \( \lor \) is not formulated in the naive way \( \Gamma, \Pi \Rightarrow A \lor B \Delta / \Gamma, \Pi \Rightarrow A \lor B \Delta \) (we have left out the side expressions), in which case it would have the subformula property, is best explained by an example. Consider a formula \( A = (B \rightarrow p \lor q) \) where \( B = (p \lor q \rightarrow r) \). Formulating \( \lor \) in the naive way would give the following derivation.

\[
\begin{align*}
R & \Rightarrow A \\
R & \lor B, \Rightarrow p \lor q \\
Vg & \Rightarrow B, \Rightarrow p \lor q \\
& \Rightarrow B, \Rightarrow p \lor q
\end{align*}
\]

It is not difficult to see that this implies that \( \Lambda_A \) will contain a disjunct that is equivalent to \( A \land (r \rightarrow p \lor q) \). The semantical characterization of projective formulas given by Ghilardi [2], and recalled in the appendix, shows that neither is this formula projective, nor is it inconsistent. Therefore, the naive formulation of \( \lor \) does not suffice for our purposes. In our case the above derivation will become:

\[
\begin{align*}
R & \Rightarrow A \\
R & \lor B, \Rightarrow p \lor q \\
Vg & \Rightarrow (p \rightarrow r)(q \rightarrow r), \Rightarrow p q \\
& \Rightarrow B, \Rightarrow p \lor q
\end{align*}
\]

This shows that the non-projective disjunct will not appear in this way. Similar reasons explain the form of \( \wedge \) and \( \rightarrow \).

Although there is no subformula property, the situation is not that bad. All formulas in the sequents that appear in a derivation from \( \Lambda_{sA} \) belong to the smallest set \( A \) that contains the subformulas of \( A \) and that is closed under the operations \( (\overset{3}{B \rightarrow C}), (\overset{3}{B \lor C}), (\overset{3}{B \land}), (\overset{3}{\land B}) \). Clearly, this set is finite. Note that the finiteness of \( A \) implies that \( \Lambda_A \) exists.

The rule \( Vg \) is different from the other rules in that it adds a sequent \( S \) to a conjunct \( \Theta \) that is in general not derivable from \( \Theta \). However, as we will see in Theorem 11, the upper expression of the \( Vg \) rule admissibly derives the lower expression, at least in derivations that start with a sequent of the form \( \Lambda_{sA} \). If in the application of the \( Vg \) rule to a sequent \( \Gamma, \Pi \Rightarrow \Delta \), the set \( \Pi \) is empty, this follows already from the admissibility of the rule \( V \) mentioned in the introduction. If \( \Pi \) is non-empty, \( Vg \) is a generalization of the rule \( V \).

In the remainder of this section we show that the admissibility of \( V \) implies the admissibility of \( Vg \). This is the main ingredient of Theorem 11 in which we show that \( A \vdash \Lambda_A \vdash A \). We need this generalization of \( V \) to make the proof of Theorem 2 in which we show that \( dj(A_A) \) is an admissible projective set, work.

Remark 1. We will need the following consequences of the admissibility of \( V \), the verification of which we leave to the reader. If \( \Gamma \) and \( \Sigma \) are two sets of
implications, it holds, if at least one of the conditions stated below is satisfied, that

\[ \Gamma \Sigma \Rightarrow \Delta \sim \bigvee_{A \in \Delta \cup \Sigma^a} (\Gamma \Sigma \Rightarrow A). \]

The conditions are as follows. (1) If \( \Gamma^a \subseteq \Delta \). (2) If for all \( A \in \Gamma^a \) there exists a \( B \in \Delta \) such that \( A \vdash B \). (3) If \( \Gamma^a = \bigvee \Delta \). For in this case \( \bigwedge \Gamma \) is equivalent (over IPC, and whence over L) to \( \bigwedge \{ A \rightarrow B \mid \bigvee \Delta \rightarrow B \in \Gamma, A \in \Delta \} \).

**Definition 3.** For a set of formulas \( I \) we denote \{ \( A \rightarrow B \in \Pi \mid A \in I \} \) by \( \Pi^I \).

For a sequent \( (\Gamma, \Pi \Rightarrow) \) we define

\[ (\Gamma, \Pi \Rightarrow) = \Gamma \{ A \in \Pi^a \mid A \in I \}, \Pi \setminus \Pi^I \Rightarrow \Delta. \]

For a sequent \( S \) we say that another sequent \( (\Gamma, \Pi \Rightarrow) \) is \( S \)-correct if \( \forall I \subseteq \Pi^a : S \vdash (\Gamma, \Pi \Rightarrow) \). For a set of implications \( \Pi \) we define

\[ \Pi_\Delta = \{ \bigvee \Delta \rightarrow D \mid \exists C(C \rightarrow D \in \Pi) \}, \]

i.e. \( \Pi_\Delta \) is the result of replacing all antecedents of the implications in \( \Pi \) by \( \bigvee \Delta \). \( \Pi_\Delta^I = (\Pi^I)_\Delta \).

**Remark 2.** If \( \Gamma, \Pi \Rightarrow \Delta \) is \( S \)-correct then \( S \vdash \Gamma, \Pi \Rightarrow \Delta \). If \( \Pi \) is empty, then \( \Gamma, \Pi \Rightarrow \Delta \) is \( S \)-correct iff \( S \) derives \( \Gamma, \Pi \Rightarrow \Delta \).

The reader is advised to first read the proof of Proposition [before embarking on the following two lemma’s, that may otherwise seem somewhat mysterious.

**Lemma 1.** If the sequent \( \Gamma, \Pi \Rightarrow \Delta \) is \( S \)-correct then for all \( I \subseteq \Pi^a \) the sequent \( \Gamma \Pi_\Delta^I, \Pi \setminus \Pi^I \Rightarrow \Delta \) is \( S \)-correct too.

**Proof.** We use induction on \(| I | \). The case \(| I | = 0 \) follows from the \( S \)-correctness of \( \Gamma, \Pi \Rightarrow \Delta \). Suppose \(| I | > 0 \). Let \( \Sigma = \Pi \setminus \Pi^I \). We have to show that for all \( J \subseteq \Sigma^a \)

\[ S \vdash \Gamma \Pi_\Delta^J \{ A \in \Sigma^a \mid A \in J \}, \Sigma \setminus \Sigma^J \Rightarrow \Delta. \quad (1) \]

Pick a \( C \in I \) and let \( I' = I \setminus \{ C \} \) and \( \Sigma' = \Pi \setminus \Pi^{I'} \). Note that \( C \notin \Sigma^a \), whence \( C \notin J \) and \( \Sigma^a \cup \{ C \} = \{ \Sigma' \}^a \). The induction hypothesis says that \( \Gamma \Pi_\Delta^{I'}, \Pi \setminus \Pi^{I'} \Rightarrow \Delta \) is \( S \)-correct. This implies that we have for all \( H \subseteq (\Sigma')^a \)

\[ S \vdash \Gamma \Pi_\Delta^{I'} \{ A \in (\Sigma')^a \mid A \in H \}, \Sigma' \setminus (\Sigma')^H \Rightarrow \Delta. \quad (2) \]

Thus we have \( S \vdash \Gamma \Pi_\Delta^J \{ A \in (\Sigma')^a \mid A \in J \}, \Sigma' \setminus (\Sigma')^J \Rightarrow \Delta \) by taking \( H = J \). We show that

\[ S \vdash \Gamma \Pi_\Delta^J \{ A \in \Sigma^a \mid A \in J \}, \Sigma' \setminus (\Sigma')^J \Rightarrow \bigwedge \Gamma \Pi_\Delta^{I'} \{ A \in (\Sigma')^a \mid A \in J \}, (\Sigma')^J \setminus (\Sigma')^J \}. \]

Then \((1)\) will follow from \((2)\). Clearly, \( S \vdash \Gamma \Pi_\Delta^I \Rightarrow \bigwedge \Pi_\Delta^{I'} \). Since \( \Sigma^a \cap J = (\Sigma')^a \cap J \) also \( S \vdash \{ A \in \Sigma^a \mid A \in J \} \Rightarrow \bigwedge \{ A \in (\Sigma')^a \mid A \in J \} \). It remains
to show that $S \vdash \Gamma \Pi^I_\Delta \{ A \in \Sigma^a \mid A \in J \}, \Sigma \setminus \Sigma^J \Rightarrow \bigwedge \Sigma^J \setminus (\Sigma^a)^J$. Observe that $\Sigma^J \setminus (\Sigma^a)^J = \Sigma \setminus \Sigma^J \cup \{ C \rightarrow D \mid C \rightarrow D \in \Pi \}$ since $C \not\in J$. Therefore, it suffices to show that for all $(C \rightarrow D) \in \Pi$ we have

$$\Gamma \Pi^I_\Delta \{ A \in \Sigma^a \mid A \in J \}, \Sigma \setminus \Sigma^J \Rightarrow C \rightarrow D.$$  

The following derivation shows that this holds.

$$
\begin{align*}
S \vdash \Gamma \Pi^I_\Delta \big\{ A \in (\Sigma^a)^a \mid A \in J \big\}, \Sigma \setminus (\Sigma^a)^{J \cup \{ C \}} \Rightarrow \Delta \\
\Gamma \Pi^I_\Delta \big\{ A \in (\Sigma^a)^a \mid A \in J \big\}, \Sigma \setminus (\Sigma^a)^{J \cup \{ C \}} \Rightarrow C \rightarrow \Delta \\
\Gamma \Pi^I_\Delta \big\{ (\forall \Delta \rightarrow D) \big\} \{ A \in (\Sigma^a)^a \mid A \in J \big\}, \Sigma \setminus (\Sigma^a)^{J \cup \{ C \}} \Rightarrow C \rightarrow D \\
\Gamma \Pi^I_\Delta \{ A \in \Sigma^a \mid A \in J \}, \Sigma \setminus \Sigma^J \Rightarrow C \rightarrow D \\
\end{align*}
$$

The first step is [2] with $H = J \cup \{ C \}$. The step from the 3th to 4th line holds, as $\Sigma^a \cap J = (\Sigma^a)^a \cap J$. The step from the 4th to 5th line holds, as $\Sigma \setminus \Sigma^J = (\Sigma^a)^{J \cup \{ C \}}$. This proves the lemma.

**Lemma 2.** If $\Pi$ consist of implications only and the sequent $\Gamma, \Pi \Rightarrow \Delta$ is $S$-correct, then for all $I \subseteq \Pi^a$:

$$S \land \bigwedge \{ \Gamma \Pi \Rightarrow A \mid A \in I \} \lor \bigvee \{ \Gamma \Pi \Rightarrow A \mid A \in \Delta \lor \Gamma^a \cup (\Pi^a \setminus I) \}.$$  

**Proof.** Consider $I \subseteq \Pi^a$. Let $S'$ denote $S \land \bigwedge \{ \Gamma \Pi \Rightarrow A \mid A \in I \}$. Let $\Sigma$ denote $\Gamma \Pi^I_\Delta, \Pi^I_\Pi$ and consider the sequent $T = (\Sigma \Rightarrow \Delta)$. By Lemma [1] $T$ is $S$-correct, whence it is $S'$-correct. In particular, $S' \vdash T$. By Remark [1] $S' \vdash \bigwedge \{ \Sigma \Rightarrow A \mid A \in \Delta \lor \Gamma^a \cup (\Pi^a \setminus I) \}$. Whence we are done once we can show that $S' \vdash \Gamma \Pi \Rightarrow A$ for all $A \in \Sigma$. Clearly, $S' \vdash \Gamma \Pi \Rightarrow A$ for all $A \in I$. Therefore, for all $(A \rightarrow B) \in \Pi$ for which $A \in I, S' \vdash \Gamma \Pi \Rightarrow B$, and whence $S' \vdash \Gamma \Pi \Rightarrow \bigwedge \Delta \rightarrow B$. Therefore, $S' \vdash \Gamma \Pi \Rightarrow A$ for all $A \in \Sigma$. This proves the lemma.

**Proposition 1.** If $\Gamma$ and $\Pi$ are sets of implications such that the sequent $\Gamma, \Pi \Rightarrow \Delta$ is $S$-correct and $\Pi^a \subseteq \Delta$, then $S \vdash \bigvee \{ \Gamma \Pi \Rightarrow D \mid D \in \Delta \}$.

**Proof.** The lemma follows from the following derivation.

$$
\begin{align*}
S \vdash \bigvee_{D \in \Delta} \big( \Gamma \Pi \Rightarrow D \big) \lor \bigvee_{A \in \Pi^a} \big( \Gamma \Pi \Rightarrow A \big) & \quad \text{(Remark [1] and [2])} \\
\bigvee_{D \in \Delta} \big( \Gamma \Pi \Rightarrow D \big) \lor \bigvee_{A \in \Pi^a} \big( \Gamma \Pi \Rightarrow A \big) \land \\
\land \big( \bigvee_{D \in \Delta} \big( \Gamma \Pi \Rightarrow D \big) \lor \bigvee_{B \in \Pi^a, B \neq A} \big( \Gamma \Pi \Rightarrow B \big) \big) & \quad \text{(Lemma [2])} \\
\bigvee_{D \in \Delta} \big( \Gamma \Pi \Rightarrow D \big) \lor \bigvee_{A, B \in \Pi^a, A \neq B} \big( \Gamma \Pi \Rightarrow A \big) \land \big( \Gamma \Pi \Rightarrow B \big) \\
\vdots \\
\bigvee_{D \in \Delta} \big( \Gamma \Pi \Rightarrow D \big) \lor \bigwedge_{A \in \Pi^a} \big( \Gamma \Pi \Rightarrow A \big) & \quad \text{(Lemma [2])} \\
\end{align*}
$$
Lemma 3. If \( \Pi \) consist of implications only, \( S' = (\Gamma, \Pi \Rightarrow \bot) \) is \( S \)-correct, and \( \Gamma^a = \{ \bot \} \), then \( S \vdash \bot \).

\( \text{Proof.} \) Let \( n = |\Pi^a| \). With induction to \( n - |I| \) we show that for all \( I \subseteq \Pi^a \)
\[
S \vdash \bigwedge \{ A \in \Pi^a \mid A \in I \} \rightarrow \bot. \tag{3}
\]
Let us first see that this implies the lemma. For all \( B \in \Pi^a \), (3) gives \( S \vdash \neg B \), by taking \( I = \{ B \} \). This implies \( S \vdash \bigwedge \Pi \). Also, \( S \vdash \Gamma \) because \( \Gamma^a = \{ \bot \} \). The \( S \)-correctness of \( S' \) gives \( S \vdash (\bigwedge \Gamma \Pi \Rightarrow \bot) \), by taking \( I = \emptyset \). Whence \( S \vdash \bot \).
It remains to prove (3). The case \( |I| = n \) follows from the \( S \)-correctness of \( (\Gamma, \Pi \Rightarrow \bot) \) and the fact that \( S \vdash \bigwedge \Gamma \). The case that \( |I| < n \). By the induction hypothesis we have for all \( B \in \Pi^a \setminus I \) that \( S \vdash B \land \bigwedge \{ A \in \Pi^a \mid A \in I \} \rightarrow \bot \). Whence
\[
\forall B \in \Pi^a \setminus I : S \vdash \bigwedge \{ A \in \Pi^a \mid A \in I \} \rightarrow \neg B. \tag{4}
\]
This implies that \( S \vdash \bigwedge \{ A \in \Pi^a \mid A \in I \} \rightarrow \bigwedge (\Pi \setminus I) \). By \( S \)-correctness we have
\[
S \vdash \bigwedge \Gamma \land \bigwedge \{ A \in \Pi^a \mid A \in I \}, \bigwedge (\Pi \setminus I) \rightarrow \bot.
\]
Therefore, \( S \vdash \bigwedge \{ A \in \Pi^a \mid A \in I \} \rightarrow \bot \).

Lemma 4. If \( A_{sA} \vdash_{\text{ADM}} \Lambda \) and \( \Lambda = \Theta_1 \mid \ldots \mid \Theta_n \), then for all \( i \leq n \), all \( S \in \Theta_i \) are \( \Theta_i \)-correct.

\( \text{Proof.} \) We use induction to the depth \( d \) of \( \Lambda \) in the derivation \( D \). The case \( d = 0 \) follows from Remark 2. Assume \( d > 0 \). We distinguish by cases according to the last rule of \( D \). Observe that if \( \Gamma, \Pi \Rightarrow \Delta \) is \( S \)-correct and if \( S \vdash \Gamma' \Rightarrow \Gamma \) and \( S \vdash \Gamma \Gamma' \Delta \Rightarrow \Delta' \), then \( \Gamma', \Pi \Rightarrow \Delta' \) is \( S \)-correct. This observation suffices for all cases except \( L \Pi \rightarrow Vg \) and \( \text{Res} \). We leave \( L \Pi \rightarrow \) to the reader. For the case \( Vg \), apply Remark 2. We treat the case \( \text{Res} \). In this case the one but last disjunction expression in \( D \) is \( \Lambda' = \Lambda'' \mid \Theta'' \cdot \Gamma, \Pi \Rightarrow p \Delta \cdot p \Gamma', \Pi' \Rightarrow \Delta' \) and
\[
\Lambda = \Lambda'' \mid \Theta'' \cdot \Gamma, \Pi \Rightarrow p \Delta \cdot p \Gamma', \Pi' \Rightarrow \Delta' \\
\text{Let } \Theta' = \Theta'' \cdot \Gamma, \Pi \Rightarrow p \Delta \cdot p \Gamma', \Pi' \Rightarrow \Delta' \text{ and } \\
\Theta = \Theta' \cdot (\Gamma \setminus \Gamma^p) \Gamma', \Gamma^p \Pi \Pi' \Rightarrow \Delta \Delta'.
\]
For the sequents in \( \Lambda'' \), \( \Theta'' \) and the sequents \( S_1 = (\Gamma, \Pi \Rightarrow p \Delta) \) and \( S_2 = (p \Gamma', \Pi' \Rightarrow \Delta') \) the induction hypothesis applies. Whence they are \( \Theta' \)-correct, and thus also \( \Theta \)-correct. We consider the remaining sequent of \( \Lambda \). Let \( \Sigma = \Gamma^p \Pi \Pi' \). We have to show that for all \( I \subseteq \Sigma^a \),
\[
\Theta \vdash (\Gamma \setminus \Gamma^p) \Gamma' \{ A \in \Sigma^a \mid A \in I \}, \Sigma \setminus \Sigma^I \Rightarrow \Delta \Delta'. \tag{5}
\]
Let \( I_1 = I \cap \Pi^a \) and \( I_2 = I \cap (\Pi')^a \). Observe that \( \Pi \setminus \Pi I_1 \) and \( (\Pi')^I \Pi' \) are subsets of \( \Sigma \setminus \Sigma^I \). The fact that \( S_2 \) is \( \Theta \)-correct implies that we have \( \Theta \vdash p \Gamma' \{ A \in (\Pi')^a \mid A \in I_2 \}, \Pi \setminus (\Pi')^I \Pi' \Rightarrow \Delta' \). Whence
\[
\Theta \vdash p (\Gamma \setminus \Gamma^p) \Gamma' \{ A \in \Sigma^a \mid A \in I \}, \Sigma \setminus \Sigma^I \Rightarrow \Delta \Delta'. \tag{6}
\]
If \( p \in I \), then \( p \in \{ A \in \Sigma^a \mid A \in I \} \). Hence (5) follows from (6). If \( p \notin I \) we use the fact that \( S_1 \) is \( \Theta \)-correct, i.e. \( \Theta \vdash (\Gamma \vDash I \mid A \in I_1), \Pi \vdash \Pi_1 \Rightarrow p \). Since \( p \notin I \), \( \Gamma \vdash \Sigma \setminus \Sigma \). Therefore it follows that
\[
\Theta \vdash (\Gamma \setminus \Gamma^p) \Gamma'_\{ A \in \Sigma^a \mid A \in I \}, \Sigma \setminus \Sigma \Rightarrow p \Delta. \tag*{(7)}
\]
Combining (7) with (6) gives (5).

Note that the previous lemma does not hold without the assumption that \( \Lambda_s A \vdash \text{ADM} \Lambda \). The special form of the first expression plays a role in the induction.

**Theorem 1.** If \( \Lambda_s A \vdash \text{ADM} \Lambda \vdash \text{ADM} \Lambda' \) then \( \Lambda \vdash \Lambda' \vdash \Lambda \). In particular, \( \Lambda \vdash \Lambda_A \vdash \Lambda \). In fact, for all rules \( R \) except \( Vg \), even \( \Lambda \vdash R \vdash \text{ADM} \Lambda \) implies \( \Lambda \vdash \Lambda' \vdash \Lambda \).

**Proof.** We use induction to the depth \( d \) of the derivation. The proof for \( \Lambda' \vdash \Lambda \) is trivial. For \( \Lambda \vdash \Lambda' \), in the case \( d = 0 \) we have \( \Lambda = \Lambda' \). For \( d > 0 \) we distinguish by cases according to the last rule that is applied. We only treat the case \( Vg \). Suppose \( \Lambda \) derived from the application of the rule \( Vg \) to \( \Lambda \). Suppose \( \Lambda = \Lambda' \mid \Theta \cdot S \), where \( S = (\Gamma, \Pi \Rightarrow \Delta) \) with \( \Delta = A_0 \ldots A_n \) and \( \Gamma \) a set of implications such that \( \Gamma^a \subseteq \Delta \). As observed before, the fact that \( \Lambda_s A \vdash \text{ADM} \Lambda \) implies that \( \Pi \) consists of implications only. For \( 1 \leq i \leq n \), let \( S_i = (\Gamma \Pi_i, \Rightarrow A_i) \). Whence
\[
\Lambda' = \Lambda'' \mid \Theta \cdot S_0 \mid \ldots \mid \Theta \cdot S_n.
\]

Therefore, to prove the lemma it suffices to show that \( \Theta \land S \vdash S_0 \lor \ldots \lor S_n \). This follows from Lemma [4] in combination with Proposition 1.

Note that in the previous theorem, the fact that \( \Lambda \) and \( \Lambda' \) appear in derivations from \( \Lambda_s A \) is crucial: in general, \( \Lambda \vdash \text{ADM} \Lambda' \) does not imply \( \Lambda \vdash \Lambda' \vdash \Lambda \). The previous theorem together with the second main theorem, Theorem 2 proved in the next section, show that \( \text{ADM} \) has the properties we want it to have, as discussed in the introduction.

### 3.2 Properties of \( \Lambda_A \)

In this section we discuss only \( \text{IPC} \). Whence \( \vdash \) and \( \vdash \) stand again for \( \vdash \text{IPC} \) and \( \vdash \text{IPC} \). Recall from Theorem 1 that \( \Lambda \vdash \Lambda_A \vdash \Lambda \). The last fact implies that for all disjuncts \( \Theta \) of \( \Lambda_A \) we have \( \Theta \vdash \Lambda \).

**Lemma 5.** For every formula \( A \), for every \( \Theta \in \text{dj}(\Lambda_A) \setminus p(\Lambda_A) \), it holds that \( \vdash \neg \Theta \).

**Proof.** Let \( \Theta \) be a disjunct of \( \Lambda_A \) that is not in \( p(\Lambda_A) \). Since \( \Theta \notin p(\Lambda_A) \) it follows that \( \Theta \) contains a conjunct \( (\Gamma, \Pi \Rightarrow \bot) \) for which \( \Gamma^a = \{ \bot \} \). By Lemma 4 \( (\Gamma, \Pi \Rightarrow \bot) \) is \( \Theta \)-correct. Thus by Lemma 3 we have \( \Theta \vdash \bot \).

**Lemma 6.** For any formula \( A \), every formula in \( p(\Lambda_A) \) is projective.
Proof. Let $\Theta$ be one of the disjuncts of $A_A$. We show that if $\Theta \in p(A_A)$, then $\Theta$ is projective. We will use Ghilardi’s algorithm, described in the appendix, to show this. As Ghilardi remarks in [3], the order in which the steps of the algorithm are applied is irrelevant. We will apply them in the following order.

$S1$ Apply $T\top, T\bot, Tp, T\land, T\lor, T\rightarrow, T\neg$ and the two Simplification Rules.

$S2$ Apply $F\top, F\bot, Fp, F\land, F\lor, F\rightarrow$ and $F\neg$.

$S3$ Apply the Resolution Rule.

We apply these steps $S1 – S3$ as long as possible, and we will not apply $S3$ ($S2$) as long as we can apply $S2$ ($S1$). When computing the algorithm on a formula $A$, we will initialize with $\{FA\}$. The input formulas of the algorithm are conjunctions of implications. Thus in this case it would e.g. be $(\top \rightarrow A)$. According to the Initialization we should then start with $\{T\top, FA\}$. After one application of $T\top$ we have $\{FA\}$ instead. Thus we see that starting with $\{FA\}$ gives the same result as starting with the Initialization according to the algorithm. We do this only to obtain a smooth induction. Namely, we will show in Claim 1 that when applying the algorithm in this order to $\Theta$, for every set $O$ that is created there exists a conjunct $S_O = (\Gamma, \Pi \Rightarrow \Delta)$ of $\Theta$ with certain properties. Then we show, Claim 2 and Claim 3, that this shows that either all output sets of $\Theta$ contain atomic modalities or $\Theta \notin p(A_A)$. By Theorem 4.2 of [3] (cited here as Theorem 4) all formulas for which all output sets contain atomic modalities are projective. This then proves the theorem. We denote $\{A \in \Gamma \mid TA \notin O\}$ by $\Gamma \setminus TO$. For $X, Y, Z \in \{T, T^+, T_c\}$ or $X, Y, Z \in \{F, F^+, F_c\}$, $XYO$ denotes $XO \cup YO$ and $XYZO$ denotes $XO \cup YO \cup ZO$.

Claim. For every $O$ that is created and not removed there exists a conjunct $S_O = (\Gamma, \Pi \Rightarrow \Delta)$ of $\Theta$ that satisfies the following properties:

(a) $TT^+O \subseteq \Gamma \Pi$

(b) $\Delta = FF^−FcO$

(c) $\forall B \in (\Pi \setminus TO) \ (B$ is an implication and $\Theta \top \land T_cO \rightarrow B)$

(d) $\forall (B \rightarrow C) \in (\Gamma \setminus TO)(B \in \Delta \cup \{\bot\})$.

Proof. With induction to the number of sets that were created before $O$.

The Initialization is clear. We consider the induction step. We distinguish by cases according to the step $Si$ in which $O$ is created. Assume $O$ is the result of an application of a rule to $O'$ or, in the case of $Res$ to $(O', O'')$. The induction hypothesis for $O'$ implies that there is a conjunct $S_{O'}$ of $\Theta$ that satisfies (a) – (d) w.r.t. $O'$. Clearly, the Simplification Rules do not have to be considered. Observe that we can also omit the rules $T\rightarrow$ and $F\neg$. As they can be seen as consecutive applications of $T \rightarrow$ and $T \bot$ or $F \rightarrow$ and $F \bot$ respectively.

$S1$. $O$ is the result of an application of one of the rules $T\top, T\bot, Tp, T\land, T\lor$ and $T\rightarrow$ to some formula in $O'$. In the cases $T\top$ and $Tp$ we take $S_O = S_{O'}$. In the case $T\bot$, $O'$ is removed, so $O$ does not exist. We treat the cases $T\land$ and $T\rightarrow$ and leave $T\lor$ to the reader. In the first case $T\land$ is applied to some formula $T(B \land C) \in O'$. By (a), $S_{O'} = (\Gamma B \land C, \Pi \Rightarrow \Delta)$ for some $\Gamma, \Pi, \Delta$. We let $S_O = (\Gamma BC, \Pi \Rightarrow \Delta)$. This sequent is a conjunct of $\Theta$ by the rule $L\land$. We
leave it to the reader to check that $S_O$ satisfies $(a) – (d)$. For the case $T \rightarrow$, $O$ is the result of an application of $T \rightarrow$ to some formula $T(B \rightarrow C)$ in $O'$. By $(a)$, $S_{O'} = (\Gamma B \rightarrow C, \Pi \Rightarrow \Delta)$ or $S_{O'} = (\Gamma, B \rightarrow C \Pi \Rightarrow \Delta)$. Assume we are in the first situation. The second one is analogous. We have

$$O = O' \setminus \{T(B \rightarrow C)\} \cup \{T_c(B \rightarrow C), FB\} \text{ or } O = O' \setminus \{T(B \rightarrow C)\} \cup \{TC\}.$$  

In the first case, take $S_O = (\Gamma B \rightarrow C, \Pi \Rightarrow B \Delta)$. In the second case take $S_O = (\Gamma C, \Pi \Rightarrow \Delta)$. The rule $L\Gamma \rightarrow$ in ADM guarantees that $S_O$ is a conjunct of $\Theta$. We leave it to the reader to check that in both cases $S_O$ satisfies $(a) – (d)$.

$S2$. $O$ is the result of an application of $F \top, F \bot, F p, F \wedge, F \vee$ or $F \rightarrow$ to some formula in $O'$. We treat the cases $F \wedge, F \rightarrow$. The case $F \wedge$ is similar to the case $F \land$. First, the case $F \land$. Then $F \land$ is applied to some $F(B \land C) \in O'$. By $(b)$ $S_{O'} = (\Gamma, \Pi \Rightarrow B \land C \Delta)$. Thus

$$O = O' \setminus \{FB \land C\} \cup \{FB\} \text{ or } O = O' \setminus \{FB \land C\} \cup \{FC\}.$$  

For reasons of symmetry we only have to consider one case, suppose the first one. The rule $R \land$ in ADM implies that $\Theta$ has a conjunct $F^{B \land}, \Pi \Rightarrow B \Delta$. Let $S_O$ be this conjunct. To see that $(a)$ holds, note that since $S1$ cannot be applied, there are no formulas $TB$ in $O'$ and whence in $O$. It is easy to see that $(b)$ holds. For $(c)$, observe that formulas in $\Gamma$, if replaced in going from $S_{O'}$ to $S_O$, are only replaced by equivalent formulas. Whence the induction hypothesis applies. For $(d)$, note that formulas $(B \land C \rightarrow D) \in \Gamma$ are replaced by $(B \rightarrow (C \rightarrow D)) \in \Gamma^{B \land}$.

Second, the case $F \rightarrow$. There is some $F(B \rightarrow C) \in O$ such that

$$O = O' \setminus \{FB \rightarrow C\} \cup \{TB, FC\} \text{ or } O = O' \setminus \{FB \rightarrow C\} \cup \{F_c(B \rightarrow C)\}.$$  

By $(b)$, $S_{O'} = (\Gamma, \Pi \Rightarrow B \rightarrow C \Delta)$. In the first case take $S_O = (B \Gamma^{B \rightarrow C}, \Pi \Rightarrow C \Delta)$. This is a conjunct of $\Theta$ by $R \rightarrow$. To see that $(a)$ holds: since $S1$ cannot be applied to $O'$, there are no formulas of the form $TD$ in $O$ except $TB$. $(b)$ is easy. The only problematic case for $(c)$ might be formulas $(B \rightarrow C) \rightarrow D \in \Gamma$. However, note that the fact that $\Theta \vdash T_c O' \rightarrow ((B \rightarrow C) \rightarrow D)$ implies $\Theta \vdash T_c O \rightarrow (C \rightarrow D)$. For $(d)$, observe that formulas $(B \rightarrow C) \rightarrow D \in \Gamma$ are replaced by $(C \rightarrow D)$ in $\Gamma^{B \rightarrow C}$. In the second case take $S_O = S_{O'}$. Here we leave the verification of $(a) – (d)$ to the reader.

$S3$. $O$ is the result of applying the Resolution Rule to $(O', O'')$. By $(a)$ and $(b)$ we may assume that $S_{O'}$ and $S_{O''}$ have the following form.

$$S_{O'} = (\Gamma, \Pi \Rightarrow p \Delta) \text{ and } S_{O''} = (p \Gamma', \Pi' \Rightarrow \Delta').$$  

Note that since $S1$ and $S2$ cannot be applied, both sequents are simple. We take $S_O = (\Gamma \setminus \Gamma^p) \Gamma', \Gamma^p \Pi \Pi' \Rightarrow \Delta \Delta'$. Note that this sequent is a conjunct of $\Theta$ by the rule $Res$ and the fact that $S_{O'}$ and $S_{O''}$ are simple (recall that we assume the proof to be in normal form). We leave the verification of $(a)$, $(b)$ and $(c)$ to the reader. For $(d)$, observe that the only possibly problematic case would be formulas $(p \rightarrow A) \in \Gamma$. However, for these formulas the property $(d)$ does not have to hold anymore, because of their place in $S_O$. This finishes the proof of the claim.
Claim. If an algorithm set $O$ contains only context modalities, and $F_c O$ is non-empty, then $O$ is removed by the Simplification Rule.

Proof. Let $S_O = (\Gamma, \Pi \Rightarrow \Delta)$. By $(c)$ this implies that $\Gamma \Pi$ consists of implications only. By $(b)$ and the fact that $O$ contains only context modalities, $\Delta$ consists of implications and propositional variables only. In particular, $S_O$ is simple. Note that $(b)$ implies that $\Delta$ is non-empty. By $(d)$, $\Gamma^a \subseteq \Delta$. Whence the rule $Vg$ implies that there exists an $A \in \Delta$ such that $(\Gamma \Pi, \Rightarrow A)$ is a conjunct of $\Theta$. By $(e)$, $\Theta \vdash T_c O \rightarrow \bigwedge \Gamma \Pi$. Whence $\Theta \vdash T_c O \rightarrow A$. Since $A \in F_c O$, this shows that $O$ will be removed. This proves the claim.

Claim. If an algorithm set $O$ contains only context modalities, and $F_c O$ is empty, then $\Theta$ contains a conjunct $(\Gamma, \Pi \Rightarrow)$ with $\Gamma^a = \{ \bot \}$.

Proof. Let $S_O = (\Gamma, \Pi \Rightarrow \Delta)$. By $(d)$, $\Gamma^a = \{ \bot \}$. By $(b)$, $\Delta$ is empty. This proves the claim.

Observe that if $\Theta$ contains a conjunct $(\Gamma, \Pi \Rightarrow)$, then it also contains a conjunct $(\Gamma, \Pi \Rightarrow \bot)$. Since $\Theta \in p(A_A)$, it does not contain such conjuncts. Therefore, Claim 3 implies that if an algorithm set of $\Theta$ contains only context modalities, then $F_c O$ is non-empty. Claim 2 implies that this set will be removed. This proves that all output sets of $\Theta$ contain atomic modalities.

Theorem 2. For every formula $A$, the set $d_j(A_A)$ is an admissible projective set for $A$.

Proof. By the previous lemma and theorem, all formulas in $d_j(A_A)$ are either projective or inconsistent. By Theorem 1 we have $A \vdash \bigvee d_j(A_A) \vdash A$. 

References


A Appendix

In [2] introduced the notion of a projective formula. The paper contains the following useful semantical characterization of projective formulas. For Kripke
models $K_1, \ldots, K_n$ we denote by $(\sum_i K_i)'$ the Kripke model which is the result of attaching one new node below all nodes in $K_1, \ldots, K_n$, at which no propositional variable is valid. We say that two rooted Kripke models are variants of each other when they have the same domain and partial order, and their forcing relations only possibly differ at the roots. A class of rooted Kripke models has the extension property when for every finite set of Kripke models $K_1, \ldots, K_n$ in this class, there is a variant of $(\sum_i K_i)'$ which is in this class as well.

**Theorem 3.** (Ghilardi [2]) A formula is projective if and only if its class of models has the extension property.

In [3] Ghilardi presents an algorithm called CHECK-PROJECTIVITY that checks if a formula is projective. We will use this algorithm in the next section to show that the formulas in $p(A_A)$ are projective. In this section we will state the algorithm and cite the results in [3] that we will need in the next section. Thus this section is a recapituring of results in [3]. Only at one point, which we will indicate, our notation differs slightly from the one used by Ghilardi. In this section $\vdash$ stands again for $\vdash_{\text{IPC}}$.

An input formula $A$ of the CHECK-PROJECTIVITY algorithm is assumed to have the form $\bigwedge (A_i \rightarrow A'_i)$ (w.l.o.g. one may assume that all formulas have this form). The algorithm manipulates sets $O, O'$ of so-called signed subformulas of $A$. A signed formula is an expression of the following six kinds: $TB, FB, T_cB, F_cB, p^+, p^-$, where $B$ is a subformula of $A$ and $p$ is an atomic formula. (Ghilardi uses $x$ for atomic formulas, and $\Delta$ and $\Gamma$ for the sets we denote by $O$). Signed formulas of the form $TB, FB$ are called truth modalities, signed formulas of the form $T_cB, F_cB$ are called context modalities and signed formulas of the form $p^+$ or $p^-$ are called atomic modalities. Signed formulas express conditions on models $K$: $TB$ ($FB$) means that $B$ is true (false) at the root of $K$, $T_cB$ ($F_cB$) means that $B$ is true in the context of $K$ (the context of $K$ are all nodes different from the root), $p^+$ is a synonymous of $Tp$ and $p^-$ means that $p$ is false at the root and true in the context of $K$. If $O$ is a set of signed subformulas then $K \models O$ means that $K$ matches all the requirements expressed by $O$. A set $O$ is inconsistent iff it contains either $F_c \top$ or $TB$ and $FB$, or $TB$ and $F_cB$, or $p^+$ and $p^-$, or $Tp$ and $p^-$ or $p^+$ and $Fp$, for some $B$ or some atomic formula $p$. For sets $O, TO$ is the set of truth modalities of $O$, $T^+O$ is the set of atomic modalities of $O$ and $T_cO$ is the set of context modalities of $O$. For $X, Y, Z \in \{T, T^+, T_c\}$, $XYO$ denotes $XO \cup YO$ and $XYZO$ denotes $XO \cup YO \cup ZO$. The sets $FO$, $F^-O$ and $F_cO$ are defined in a similar way.

The algorithm analyzes reasons for which we may have that $A$ is false in the root and true in the context of a given model. From the rules of the algorithm given below it is easy to deduce that the following Soundness lemma holds. We will not need this lemma, but thought it instructive to state since it captures the idea behind the algorithm.

**Lemma 7.** Let $A$ be the input formula and $O_1, \ldots, O_n$ a stage of the algorithm. If $K$ is a model such that $k \models A$ for all $k \in K$ different from the root, then $K \not\models A$ iff $(K \models O_1$ or ... or $K \models O_n)$. 


The **CHECK-PROJECTIVITY** Algorithm. (The names that are given to the rules do not come from Ghilardi, but for our purposes it is convenient to have them available.)

**Initialization**

\[
\{TA_1, FA'_1\}, \ldots, \{TA_n, FA'_n\}
\]

**Tableaux Rules**

\[
\begin{align*}
T \land & \quad \frac{O_1, \ldots, O_j \cup \{TB_1 \land B_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{TB_1, TB_2\}, \ldots, O_n} \\
T \lor & \quad \frac{O_1, \ldots, O_j \cup \{T\lor\}, \ldots, O_n}{O_1, \ldots, O_j, \ldots, O_n} \\
T \top & \quad \frac{O_1, \ldots, O_j \cup \{TB_1 \lor TB_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{TB_1\}, O_j \cup \{TB_2\}, \ldots, O_n} \\
T \bot & \quad \frac{O_1, \ldots, O_j \cup \{T\bot\}, \ldots, O_n}{O_1, \ldots, O_{j-1}, O_{j+1}, \ldots, O_n} \\
F \land & \quad \frac{O_1, \ldots, O_j \cup \{FB_1 \land FB_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{FB_1\}, O_j \cup \{FB_2\}, \ldots, O_n} \\
F \lor & \quad \frac{O_1, \ldots, O_j \cup \{FB_1 \lor B_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{FB_1, FB_2\}, \ldots, O_n} \\
F \top & \quad \frac{O_1, \ldots, O_j \cup \{F\top\}, \ldots, O_n}{O_1, \ldots, O_j, \ldots, O_n} \\
F \bot & \quad \frac{O_1, \ldots, O_j \cup \{F\bot\}, \ldots, O_n}{O_1, \ldots, O_j, \ldots, O_n} \\
T \rightarrow & \quad \frac{O_1, \ldots, O_j \cup \{TB_1 \rightarrow B_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{FB_1, TcB_1 \rightarrow B_2\}, O_j \cup \{TB_2\}, \ldots, O_n} \\
F \rightarrow & \quad \frac{O_1, \ldots, O_j \cup \{FB_1 \rightarrow B_2\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{FB_1, TB_2\}, O_j \cup \{TB_1\}, \ldots, O_n} \\
T \neg & \quad \frac{O_1, \ldots, O_j \cup \{T\neg B\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{FB, Tc\neg B\}, \ldots, O_n} \\
F \neg & \quad \frac{O_1, \ldots, O_j \cup \{F\neg B\}, \ldots, O_n}{O_1, \ldots, O_j \cup \{Fc\neg B\}, O_j \cup \{TB\}, \ldots, O_n}
\end{align*}
\]
Resolution Rule

\[
\begin{array}{c}
T_p \quad O_1, \ldots, O_j \cup \{T_p\}, \ldots, O_n \\
O_1, \ldots, O_j \cup \{p^+\}, \ldots, O_n \\
\end{array}
\]

\[
\begin{array}{c}
F_p \quad O_1, \ldots, O_j \cup \{F_p\}, \ldots, O_n \\
O_1, \ldots, O_j \cup \{p^-\}, O_j \{F_{cp}\}, \ldots, O_n \\
\end{array}
\]

Simplification Rule

\[
\begin{array}{c}
O_1, \ldots, O_j \cup \{p^+\}, \ldots, O_i \cup \{p^-\}, \ldots, O_n \\
O_1, \ldots, O_i \cup \{p^+\}, \ldots, O_i \cup \{p^-\}, \ldots, O_n, O_j \cup O_i \cup \{T_{cp}\} \\
\end{array}
\]

Auxiliary Simplification Rule

\[
\begin{array}{c}
O_1, \ldots, O_j, \ldots, O_n \\
O_1, \ldots, O_{j-1}, O_{j+1}, \ldots, O_n \\
\end{array}
\]

provided \( A \land \bigwedge T_c O_j \vdash C \), for some \( C \in F_c(O_j) \).

Ghilardi shows that the algorithm terminates after a finite number of steps. The following theorem is the one we will need. It is Theorem 4.2 of \[3\].

**Theorem 4.** (Ghilardi \[3\]) \( A \) is projective iff each of its output sets contains at least one atomic modality.
Program Complexity
of Dynamic LTL Model Checking*

Detlef Kähler and Thomas Wilke
Christian-Albrechts University, 24098 Kiel, Germany
{kaehler,wilke}@ti.informatik.uni-kiel.de

Abstract. Using a recent result by Hesse we show that for any fixed
linear-time temporal formula the dynamic model checking problem is in
Dyn-TC^0, a complexity class introduced by Hesse, Immerman, Patnaik,
containing all dynamic problems where the update after an operation has
been performed can be computed by a DLOGTIME-uniform constant-
depth threshold circuit. The operations permitted to modify the tran-
sition system to be verified include insertion and deletion of transitions
and relabeling of states.

1 Introduction

Usually, a model checking problem is defined as follows. Given a transition sys-
tem and a system specification, also called property, check whether the system
has the property. This is motivated by the fact that in order to ensure that a
piece of hardware or software is correct one checks that it has certain desirable
properties. Looking more closely one observes that as the system evolves during
the design process the desirable properties are checked over and over again, be-
cause properties once true can be made false by a modification and properties
once false hopefully become true. This gives rise to a dynamic view of model
checking and motivates a dynamic version of the model checking problem: re-
compute the truth value of the property in question after a modification of the
system. Clearly, the recomputation may be facilitated by keeping around auxil-
ary data, but then this must also be adapted as the system is modified. From a
theoretical point of view it is now interesting to determine how difficult the re-
computation is (both the mere recomputation of the truth value of the property
and the update of the auxiliary data), that is, how efficiently the recomputation
can be carried out. In the present paper, this is studied in a very specific setting,
namely when the properties are specified in linear-time temporal logic.

We work in the logical framework of dynamic complexity provided by Hesse,
Immerman, and Patnaik [11,8]. In this framework, hereafter called the HIP ap-
proach, the problem instances as well as the auxiliary data are represented by
relational structures, and the changes to the auxiliary data are described in an
appropriate logic (of low complexity). For instance, Hesse and Immerman show

* Work supported by German-Israeli Foundation for Scientific Research and Develop-
ment, Project No. I-638-95.6/1999

© Springer-Verlag Berlin Heidelberg 2003
in [8] that a dynamic version of the circuit value problem is complete for Dyn-FO, the class of dynamic problems where the update can be described by a first-order formula. The most sophisticated result, which is also the basis for our classification of the model checking problem, is Hesse’s Theorem [7], which says that dynamic graph reachability is in Dyn-TC\textsuperscript{0} or, equivalently, Dyn-FO(Count) [9]. This means that the update can be carried out by threshold circuits of constant depth or first-order logic with counting, respectively.

Graph reachability is almost like LTL model checking; recall that in the automata-theoretic approach to LTL model checking [17] a given formula (its negation) is converted into a Büchi automaton, then a product of this automaton and the transition system is built, and finally an emptiness test for the resulting Büchi automaton, which amounts to a number of reachability tests, is carried out. This explains briefly how we obtain our main theorem that says that the formula complexity of dynamic LTL model checking is in Dyn-TC\textsuperscript{0}.

The situation with model checking is somewhat more complicated than with graph reachability, because we also consider the possibility of relabeling states (which is similar to adding and deleting nodes) whereas Hesse is mainly interested in adding and removing edges.

To the best of our knowledge, there are as yet no non-trivial results on the dynamic complexity of model checking. (This is somewhat different for the related area of dynamic graph problems and the dynamic evaluation of database queries with transitive closure, see, e.g., [3, 10, 12].) Concrete dynamic model checking algorithms (as opposed to complexity results) have been dealt with in various papers, see, e.g., [14, 15, 16]. The most recent results on dynamic graph reachability are by Roditty and Zwick [13].

This paper is organized as follows. In Sect. 2, we review the HIP approach, in Sect. 3 we review LTL model checking, Sect. 4 describes how dynamic model checking can be reduced to dynamic selective transitive closure, a problem specifically tailored for our purposes, and, finally, Sect. 5 explains why dynamic selective transitive closure is in Dyn-TC\textsuperscript{0}, which also proves that dynamic LTL model checking is in Dyn-TC\textsuperscript{0}.

For background on circuit complexity, descriptive complexity, and model checking, see [18], [9], and [2], respectively.

## 2 Dynamic Complexity Framework

In this section, we provide the necessary background on dynamic complexity by looking at examples; for detailed information, see [8]. As stated in the introduction, we use the HIP approach and modify it slightly in order to be able to treat computation problems—in [8], only decision problems are considered.

### 2.1 Background on Descriptive Complexity

Our entire approach is based on descriptive complexity, and we start off with some background on this.
We are only interested in relational structures $\mathcal{A}$ where the universe, denoted $|\mathcal{A}|$, is an initial segment of the natural numbers, that is, $|\mathcal{A}| = \{0, 1, \ldots, n-1\}$. In addition, we assume that every structure is provided with the built-in predicate $\leq$ (with the natural interpretation) and the built-in predicate $\text{BIT}^{(2)}$, which may be used to query the binary representation of the numbers building the universe: $\text{BIT}^{\mathcal{A}}(i, j)$ holds if the $j$th bit of $i$ is 1. Moreover, we assume the structures have at least 2 elements, and we identify 0 with false and 1 with true. Structures of this kind will be referred to as arithmetic structures. Given a relational vocabulary $\tau$, we write $\text{Struc}[\tau]$ for the set of all arithmetic structures with vocabulary $\tau$ (more precisely, with vocabulary $\tau \cup \{\leq, \text{BIT}\}$) and $\text{Struc}_n[\tau]$ for all such structures with $n$ elements.

For specifying properties of arithmetic structures, we only consider first-order logic, denoted FO, and extensions thereof. Our main theorem involves FO[$\text{Count}$], the logic obtained from FO by adding so-called counting quantifiers $\exists_i x \phi$ with the meaning “there exists $i$ many $x$ such that $\phi$ holds” and where the occurrence of $i$ is free.

Observe that already in FO, with the built-in predicates $\leq$ and BIT, it is easy to express addition and multiplication on the universe, see [9], as well as the boolean operations on the subuniverse $\{0, 1\}$. This is the reason why in the following we will use $+$ and $\cdot$ freely in our first-order formulas as well as boolean operations on terms meant to denote boolean values.

An important fact, which ties the logical approach to the computational side, is the following.

**Theorem 1 ([1,4])** In FO[Count], one can express exactly the properties that can be computed by DLOGTIME-uniform polynomial-size constant-depth threshold circuits.

### 2.2 Dynamic Problems

The basic idea is that a dynamic problem is specified by (1) a set of operations that can be used to build instances of the problem and (2) for every sequence of operations a solution to the instance represented by this sequence. A particular aspect of the HIP approach is that the problems are parametrized according to the size of the input; we will use $n$ to denote this size.

As an example, we consider the problem dynamic transitive closure, Dyn-TransClos, where we are interested in computing (maintaining) the transitive closure of a binary relation, which we view as the edge set of a directed graph. The graph can (dynamically) be constructed by deleting and inserting edges. Therefore, we use the vocabulary $\Sigma_{tc} = \{\text{Insert}^{(2)}, \text{Delete}^{(2)}\}$ of two binary operators. For the parameter $n$, the set of operations for constructing the instances of DynTransClos is then given by

$$\Sigma_{tc}^n = \{\text{Insert}(i, j) \mid i, j < n\} \cup \{\text{Delete}(i, j) \mid i, j < n\}. \quad (1)$$

The transitive closure of a graph is simply a binary relation on the vertex set of the given graph, which is why for representing the solutions to the instances we
will use structures over the vocabulary $\tau^{tc} = \{C^{(2)}\}$. For any $n$, the solutions are described by the mapping

$$s_n^{tc}: u \mapsto A_{u}^{tc}, \quad u \in (\Sigma_n^{tc})^*, \quad A_{u}^{tc} \in \text{Struc}_n[\tau^{tc}]$$

(2)

where $A_{u}^{tc}$ represents the transitive closure of the graph obtained from the empty graph with $n$ vertices by inserting and deleting edges as described by $u$.

Formally, our dynamic problem is now given by

$$\text{DynTransClos} = (\Sigma^{tc}, \tau^{tc}, \{s_n^{tc}\})$$

(3)

In this paper, we will be interested in two other dynamic problems as well, namely selective transitive closure, DynSelTransClos, which we introduce below, and a family of model checking problems, for each LTL formula $\phi$ a problem DynLTLModCheck$\phi$, which we introduce in the subsequent section.

DynSelTransClos is a modification of DynTransClos where we try to model that vertices can be added and removed. To this end, we imagine that the vertices of the graph in question can be in two different states, selected or unselected. In determining the transitive closure only edges incident with selected vertices can be used. In other words, we only consider paths with selected nodes on them, also called active paths. Formally, we set

$$\Sigma^{stc} = \Sigma^{tc} \cup \{\text{Select}^{(1)}, \text{Deselect}^{(1)}\}$$

(4)

and have

$$\tau^{stc} = \tau^{tc}$$

(5)

and have

$$s_n^{stc}: u \mapsto A_{u}^{stc}, \quad u \in (\Sigma_n^{stc})^*, \quad A_{u}^{stc} \in \text{Struc}_n[\tau^{stc}]$$

(6)

where $A_{u}^{stc}$ represents the transitive closure of the graph obtained from the empty graph with $n$ vertices by performing the operations listed in $u$.

In fact, we are interested in a variant of DynSelTransClos, denoted DynSelTransClos$\psi$, in which the set of selected vertices of the initial graph is described by a fixed first-order formula $\psi$. That is, the operation sequence $u \in (\Sigma_n^{stc})^*$ is applied to the graph on $n$ vertices with no edges and with vertex $i$ selected iff $\psi(i)$ holds.

2.3 Dynamic Programs

An algorithm for solving a dynamic problem will usually maintain some auxiliary data structure where crucial information is stored. For instance, we will see that for computing the transitive closure Hesse maintains counts of paths of certain lengths. In the HIP approach, the auxiliary data structure is modeled by additional relations. The computation itself—how the data structure is updated when an operation is performed—is then described by appropriate formulas.

Consider a very simple example, DynDivBy6, where an element of a structure is either selected or not and suppose we want to determine if the number of selected elements is divisible by 6. In analogy to before, we choose
\( \Sigma^{\text{div}6} = \{ \text{Select}^{(1)}, \text{Deselect}^{(1)} \} \),

and we have a description of the solutions,

\[
s_n^{\text{div}6} : u \mapsto A_u^{\text{div}6}, \quad u \in (\Sigma_n^{\text{div}6})^*, \quad A_u^{\text{div}6} \in \text{Struc}_n[\tau^{\text{div}6}] .
\]

In order to be able to determine (efficiently) whether or not the number of selected vertices is divisible by six we (have to) remember which vertices are selected and which aren’t and their count modulo 6. For this purpose, our auxiliary data structure will have a unary relation, denoted \( S \), and six boolean constants \( m_0, \ldots, m_5 \). The updates of the auxiliary data and the updates of the solution relation \( d_6 \) are described by simple formulas. For \( \text{Select}(i) \), we have

\[
S'(x) \equiv i = x \lor S(x) ,
\]

\[
m_0' \equiv (S(i) \land m_0) \lor (\neg S(i) \land m_5) ,
\]

\[
m_1' \equiv (S(i) \land m_1) \lor (\neg S(i) \land m_0) ,
\]

\[
m_2' \equiv (S(i) \land m_2) \lor (\neg S(i) \land m_1) ,
\]

\[
m_3' \equiv (S(i) \land m_3) \lor (\neg S(i) \land m_2) ,
\]

\[
m_4' \equiv (S(i) \land m_4) \lor (\neg S(i) \land m_3) ,
\]

\[
m_5' \equiv (S(i) \land m_5) \lor (\neg S(i) \land m_4) ,
\]

\[
d_6' \equiv (S(i) \land m_0) \lor (\neg S(i) \land m_5) .
\]

Note that \( \equiv \) is to be read as an assignment where the unprimed variables refer to the relations and constants before the operation \( \text{Select}(i) \) has been carried out, while the primed variables refer to the relations and constants after the operation has been carried out. Note also that (17) simply says that \( d_6 \) and \( m_0 \) are the same. The entire block of assignments, (10)–(17), is called the dynamic procedure for \( \text{Select} \). For \( \text{Deselect} \), we could set up a very similar dynamic procedure. In addition to this, we have to specify the initial interpretation of the auxiliary data structure and the result we expect before any operation has been carried out:

\[
S(x) \equiv \text{FALSE} ,
\]

\[
m_0 \equiv \text{TRUE} , \quad m_1 \equiv \text{FALSE} , \quad m_2 \equiv \text{FALSE} ,
\]

\[
m_3 \equiv \text{FALSE} , \quad m_4 \equiv \text{FALSE} , \quad m_5 \equiv \text{FALSE} ,
\]

\[
d_6 \equiv \text{TRUE} .
\]

This block of assignments, (18)–(21), is called the initialization procedure. The assignments all-together constitute our dynamic program for \( \text{DynDivBy6} \).

### 2.4 Dynamic Complexity

In the HIP approach, dynamic complexity classes are made up from logics. Given a logic \( \mathcal{L} \), the complexity class \( \text{Dyn-} \mathcal{L} \) consists of all dynamic problems that can
be solved via a dynamic program (as above) where the initial settings and the updates are described by formulas from \(L\).

Hesse’s result, which we rely on here heavily, gives an upper bound for the complexity of dynamic transitive closure:

**Theorem 2** ([7]) \(\text{DynTransClos}\), is in Dyn-FO[Count] (= Dyn-TC\(^0\)).

The equality Dyn-FO[Count] = Dyn-TC\(^0\) is justified by Theorem 1.

### 2.5 Redductions

Just as in ordinary complexity theory the notion of a reduction is quite useful, both for defining completeness and for ease in proving membership in a certain complexity class.

As a simple example, consider the problem DynDivBy3, the obvious modification of DynDivBy6. Clearly, divisibility by three can be deduced from divisibility by six if one can double: 3 divides \(x\) iff 6 divides \(2x\). Therefore, in our reduction from DynDivBy3 to DynDivBy6, we will double the universe.— One might be tempted to use the simple formula \(d_0 \lor m_3\) as a reduction; this does not work because \(m_3\) is not part of the specification of DynDivBy6.

In general, a reduction \(\text{from a dynamic problem } \mathcal{P} = (\Sigma, \tau, \{s_n\}_n) \text{ to a dynamic problem } \mathcal{Q} = (\Gamma, \sigma, \{t_n\}_n)\) is composed of
- a universe expansion function, \(e: \mathbb{N} \to \mathbb{N}\), a polynomial with \(e(n) \geq n\) for all \(n\),
- a family \(\{h_n\}_n\) of reduction homomorphisms \(h_n: \Sigma^*_n \to \Gamma^*_e(n)\), and
- for each symbol \(R^{(k)} \in \tau\) a first-order result definition \(R(x_0, \ldots, x_{k-1}) \equiv \phi_R(x_0, \ldots, x_{k-1})\) where \(\phi\) is in the vocabulary \(\sigma\).

It is required that the family \(\{h_n\}_n\) is bounded, that is, there must exist some \(k\) such that for all \(n\) and for all \(a \in \Sigma_n\), \(|h_n(a)| \leq k\). Moreover, all images \(h_n(a)\) must be specified in a uniform way, say, by a first-order formula.

In ordinary complexity theory, for a reduction \(f\) from \(L\) to \(L'\) we need that \(x \in L\) iff \(f(x) \in L\) for all strings \(x\). This translates to our dynamic setting as follows. For every \(n\), every sequence \(u \in \Sigma_n\), every \(k\)-ary relation symbol \(R \in \tau\), and \(a_0, \ldots, a_{k-1} < n\), we have \(s_n(u) \models R(a_0, \ldots, a_{k-1})\) iff \(t_{e(n)}(h_n(u)) \models \phi_R(a_0, \ldots, a_{k-1})\).

For the concrete reduction from DynDivBy3 to DynDivBy6 we choose the above components as follows. The expansion function is given by \(e(n) = 2n\), the reduction homomorphisms are determined by \(h_n(\text{Select}(i)) = \text{Select}(i)\text{Select}(n+i)\) and \(h_n(\text{Deselect}(i)) = \text{Deselect}(i)\text{Deselect}(n+i)\), and the result definition is simply \(d_3 := d_6\).

The important lemma we will need is the following.

**Lemma 1** If \(P\) and \(P'\) are dynamic problems such that \(P' \in \text{Dyn-FO[Count]}\) and \(P\) can be reduced to \(P'\), then \(P \in \text{Dyn-FO[Count]}\).

The proof is similar to the corresponding lemma from [8]; the fact that we deal with computation problems is only a technical point.
3 LTL Model Checking and Main Result

We first recall ordinary LTL model checking, then define dynamic LTL model checking, and conclude with the description of the automata-theoretic method for solving the ordinary LTL model checking problem.

3.1 Ordinary LTL Model Checking

In ordinary LTL model checking we are given (1) a transition system
\[ T = (S, P, R, L, s_I) \]  \hspace{1cm} (22)
with state set \( S \), set of propositional variables \( P \), edge relation \( R \subseteq S \times S \), labeling function \( L: S \rightarrow 2^P \), and initial state \( s_I \), and (2) a temporal formula \( \phi \), built from the propositional variables in \( P \), boolean connectives and temporal operators such as “next”, “eventually”, and “until”, and we ask whether \( T \models \phi \).

The relation \( T \models \phi \) holds if each infinite computation \( s_0s_1s_2 \ldots \) through \( T \) (meaning \( s_0 = s_I \), \( (s_i, s_{i+1}) \in R \) for each \( i \)) satisfies \( \phi \), where, in turn, this means that \( L(s_0)L(s_1)L(s_2) \cdots \models \phi \) holds in the usual sense.

3.2 Dynamic LTL Model Checking and Main Result

In our dynamic version of the problem defined above, we allow the transition system to be changed, but leave the LTL formula fixed. This means we are interested in the program complexity of LTL model checking [17].

The operations of the dynamic version of the problem defined above with fixed formula \( \phi \), \( \text{DynLTLModCheck}_\phi \), are insertion and deletion of edges and relabeling of the states, more precisely,
\[ \Sigma_{mc} = \{ \text{Insert}^{(2)}, \text{Delete}^{(2)} \} \cup \{ \text{SetVar}_U(i) \mid U \subseteq P \} \]  \hspace{1cm} (23)
where the intended meaning of \( \text{SetVar}_U(i) \) is that the \( i \)th state of the transition system gets label \( U \). The solution to the model checking problem is represented by a boolean constant, \( v \) (for verified), that is
\[ \tau_{mc} = \{ v^0 \} \]  \hspace{1cm} (24)
and finally,
\[ s_n^{mc}: u \mapsto A_u^{mc}, \quad u \in (\Sigma_n^{mc})^*, \quad A_u^{mc} \in \text{Struct}_n[\tau_{mc}] \]  \hspace{1cm} (25)
where \( s_n^{mc}(u) \models v \) iff \( \phi \) holds in the transition system with state set \( \{0, \ldots, n - 1\} \), initial state 0 and edge relation and labeling according to \( u \).

The main result of this paper is:

**Theorem 3** For every LTL formula \( \phi \), dynamic model checking of the property \( \phi \), \( \text{DynLTLModCheck}_\phi \), is in Dyn-FO[Count] (= Dyn-TC^0).

The remainder of the paper explains our proof of this theorem. We start with a brief review of the automata-theoretic approach to model checking.
3.3 The Automata-Theoretic Method

In the automata-theoretic approach to LTL model checking [17], given an LTL formula $\phi$, one first constructs a Büchi automaton

$$A_\phi = (Q, 2^P, q_I, \delta, F)$$ (26)

with finite state set $Q$, alphabet $2^P$, initial state $q_I$, nondeterministic transition function $\delta: Q \times 2^P \to 2^Q$, and accepting state set $F$, which recognizes exactly all infinite strings over $2^P$ which do not satisfy $\phi$.

Then, in the second step, one constructs the product of the given transition system $T$ (as above):

$$A_{T \times \phi} = (S \times Q, 2^P, s_I \times q_I, \delta', S \times F),$$ (27)

where for $(s, q) \in S \times Q$ and $U \subseteq P$,

$$(s', q') \in \delta'((s, q), U) \quad \text{iff} \quad L(s) = U \land (s, s') \in R \land q' \in \delta(q, U).$$ (28)

This construction guarantees that if the language of $A_{T \times \phi}$ is empty, one can conclude no computation of $T$ violates $\phi$, which means $T \models \phi$, and if the language is not empty, one can conclude there exists a computation of $T$ violating $\phi$, which means $T \not\models \phi$. So, when we denote the language recognized by a Büchi automaton $A$ by $L(A)$, we can state:

**Lemma 2** $T \models \phi$ iff $L(A_{T \times \phi}) = \emptyset$.

As a consequence, in the third and last step of the automata-theoretic approach, one checks $L(A_{T \times \phi})$ for emptiness.

Observe that in general, the language of a Büchi automaton $(Q, \Sigma, q_I, \delta, F)$ is non-empty iff there exists $q \in F$ such that $q$ is reachable from $q_I$ and $q$ is reachable from itself (via a non-trivial path). This can easily be expressed using the transitive closure of the underlying transition graph.

3.4 Outline of Proof

Our proof of Theorem 3 now goes as follows. The first step is to show that $\text{DynLTLModCheck}_\phi$ can be reduced to $\text{DynSelTransClos}_\psi$, for a certain formula $\psi$, using the automata-theoretic approach just described. We then show that $\text{DynSelTransClos}_\psi$ is in Dyn-FO[Count], which, together with Lemma 1, will complete the proof of Theorem 3.

4 Reducing Dynamic LTL Model Checking to Selective Transitive Closure

We first show how to model the product Büchi automaton $A_{T \times \phi}$ as a selective graph $G$ in such a way that we will be able to reduce $\text{DynLTLModCheck}_\phi$ to
DynSelTransClos$_\psi$. On the one hand we want to decide whether $L(A_{T \times \phi}) = \emptyset$ by considering the transitive closure of $G$, on the other hand the selective graph $G$ should be easily updatable according to the possible changes the transition system $T$ may undergo. The main problem here is to handle the relabeling of states in $T$. To cope with this we had to introduce the notion of selective graphs.

4.1 The Product Automaton as a Selective Graph

For determining if a transition system $T$ is a model of an LTL-formula $\phi$ it suffices to know the transitive closure of the transition graph of the product Büchi automaton $A_{T \times \phi}$, see Section 3. The edge relation of this transition graph is described by (28). In order to reduce DynLTLModCheck$_\phi$ to DynSelTransClos we have to simulate each operation of $\Sigma_{mc}$ by a bounded sequence of operations from $\Sigma_{stc}$. An insertion or deletion of a transition in $T$ can only cause at most $|Q|^2$ changes in the transition graph of $A_{T \times \phi}$ and $|Q|^2$ is independent of the size $n$ of the transition system $T$. But if we change the label of state $i$ of $T$ we may have to change up to $n^2$ edges in the transition graph of $A_{T \times \phi}$. As a consequence, the operations SetVar$_U$ cannot be simulated by a uniformly bounded sequence of insertions and deletions in the transition graph associated with $A_{T \times \phi}$.

With selective graphs, we can handle relabeling as well. We split up (28) into two parts: (i) $L(i) = U$ and (ii) $(i, i') \in R \land q' \in \delta(q, U)$, and model the second part by transitions and the first part by selecting vertices.

The selective graph $G$ representing the product Büchi automaton

$$A_{T \times \phi} = (S \times Q, 2^P, s_I \times q_I, \delta', S \times F),$$

has vertex set $V = S \times 2^P \times Q$. There is an edge in $G$ from vertex $(i, U, q)$ to vertex $(i', U', q')$ if (ii) holds. So the edges of $G$ represent all possible labelings of $T$ at once. Which labeling is actually present is described by the selected vertices in $G$. At any point during the computation, a vertex $(i, U, q) \in V$ will be selected iff the label of $i$ in $T$ (at that point) is $U$, reflecting condition (i).

Then the following lemma holds.

**Lemma 3** $L(A_{T \times \phi}) \neq \emptyset$ iff there exists $q \in F$ and $s \in S$ such that there is an active path from $(0, L(0), q_I)$ to $(s, L(s), q)$ and an active path from $(s, L(s), q)$ to $(s, L(s), q)$.

4.2 Reducing DynLTLModCheck to DynSelTransClos

We may assume that $Q = \{q_0, \ldots, q_{2^p-1}\}$, that is, the size of $S \times 2^P \times Q$ is $n \cdot 2^{m+p}$. We want to encode the vertices of $G$ as numbers in order to encode it as an instance of DynSelTransClos$_\psi$. For this purpose, we identify each element $U$ of $2^P$ with a unique number from $\{0, \ldots, 2^m-1\}$, where, by convention, 0 represents the empty labeling. A vertex $(i, U, q_j)$ is encoded as the number $i \cdot 2^{m+p} + U \cdot 2^p + j$. We represent the vertex set $V$ of $G$ by the set $\{0, \ldots, n \cdot 2^{m+p} - 1\}$. Initially,
the vertices of the selective graph $G$ correspond to the initial labeling of transition system $T$, i.e., all vertices $i$ with $L(i) = \emptyset$ are selected. For the reduction from $\text{DynLTLModCheck}_\phi$ to $\text{DynSelTransClos}_\psi$ we have to provide (i) the formula $\psi$, (ii) the reduction homomorphisms that map operation sequences of $\text{DynLTLModCheck}_\phi$ to operation sequences of $\text{DynSelTransClos}_\psi$, and (iii) the formula that expresses the solution of $\text{DynLTLModCheck}_\phi$ in terms of the solution of the corresponding problem in $\text{DynSelTransClos}_\psi$. This is what we will do in the remainder of this section.

(i) According to what was said above, we set

$$\psi(i) \equiv \forall j(p \leq j < p + m \rightarrow \neg \text{BIT}(i, j)).$$

(ii) If we insert the edge $(i, j)$ into $T$ we have to insert the uniform set

$$\{(i, U, q), (j, U', q')\} \mid q' \in \delta(q, U)$$

into the corresponding selective graph $G$. If we delete edge $(i, j)$ the above set of edges has to be removed. So we can simulate insertion and deletion of an edge in the transition system $T$ by a bounded sequence of insertions and deletions in the corresponding selective graph:

$$h(\text{Insert}(i, j)) = \text{Insert}(i \cdot 2^{p+m} + u_1, j \cdot 2^{p+m} + v_1) \ldots$$

$$\ldots \text{Insert}(i \cdot 2^{p+m} + u_t, j \cdot 2^{p+m} + v_t),$$

$$h(\text{Delete}(i, j)) = \text{Delete}(i \cdot 2^{p+m} + u_1, j \cdot 2^{p+m} + v_1) \ldots$$

$$\ldots \text{Delete}(i \cdot 2^{p+m} + u_t, j \cdot 2^{p+m} + v_t),$$

where $\{(u_1, v_1), \ldots, (u_t, v_t)\} = \{(2^p \cdot U + i, 2^p \cdot U' + j) \mid q_j \in \delta(q_i, U)\}$. Note that the operation sequences are indeed bounded because $t$ is a fixed number not depending on $n$.

If we change the label of state $i$ of $T$ to $U$, we first deselect all vertices of the form $(i, \cdot, \cdot)$ and then select all vertices of the form $(i, U, \cdot)$. This leads to

$$h(\text{SetVar}_U(i)) = \alpha \beta$$

where

$$\alpha = \text{Deselect}(i \cdot 2^{p+m}) \ldots \text{Deselect}(i \cdot 2^{p+m+1} - 1),$$

$$\beta = \text{Select}(i \cdot 2^{p+m} + 2^p \cdot U) \ldots \text{Select}(i \cdot 2^{p+m} + 2^p \cdot U + 2^m - 1).$$

(iii) By Lemma 3 we can express $v \in \tau^{mc}$ in terms of $C \in \tau^{ste}$:

$$v \equiv \neg \exists x \exists y (\rho_{\text{initial}}(x) \land \rho_{\text{final}}(y) \land (C(x, y) \land C(y, y)),$n

where

$$\rho_{\text{initial}}(x) \equiv \forall i(i < p \rightarrow (\neg \text{BIT}(x, i)) \land \forall i(i > m + p \rightarrow (\neg \text{BIT}(x, i)),$n

$$\rho_{\text{final}}(x) \equiv \bigvee_{q_j \in F} \forall i(i < p \rightarrow (\text{BIT}(j, i) \leftrightarrow \text{BIT}(x, i)).$$

Taking everything into account, we have shown the following lemma.

**Lemma 4** For any fixed LTL-formula $\phi$ the problem $\text{DynLTLModCheck}_\phi$ is reducible to $\text{DynSelTransClos}_\psi$. 

5 DynSelTransClos Is in Dyn-TC\(^0\)

The last step of our proof is to show that DynSelTransClos\(_\psi\) is in Dyn-TC\(^0\). We do this by extending Hesse’s proof which shows that DynTransClos is in Dyn-TC\(^0\). Without loss of generality, we only consider the non-parametrized version DynSelTransClos in the following.

The main idea of Hesse’s proof is to count the number of paths up to a certain length between any two vertices of the considered graph. These “path numbers” are stored as polynomials whose coefficients stand for the number of paths. To use these “path counting polynomials” in the dynamic setting one needs update formulas according to the possible graph operations. Hesse, in his paper, provides update formulas for inserting and deleting edges and deleting vertices. In order to use the idea of the path counting polynomials for DynSelTransClos we give an update formula for selecting (and deselecting) vertices. To place DynTransClos in Dyn-TC\(^0\) Hesse represents the counting polynomials as large integers and uses the results from [6] to show that one can perform the updates on this number representation of the path counting polynomials by a Dyn-TC\(^0\) (or Dyn-FO[Count]) procedure.

In the following, we explain Hesse’s ideas in more detail and present our lemma for modeling the selection of vertices in selective graphs.

5.1 Path Counting Polynomials

For vertices \(s, t\) of a directed graph \(G\) we represent the number of paths between \(s\) and \(t\) by the path counting polynomial

\[
f_{s,t}(x) = \sum_{k=0}^{\infty} p_{s,t}(k)x^k
\]  

where \(p_{s,t}(k)\) is the number of directed paths from \(s\) to \(t\) of length \(k\).

If we insert or delete edges in \(G\) the coefficients \(p_{s,t}(k)\) of the path counting polynomials have to be recomputed. Hesse gives update formulas for the path counting polynomials for three operations: insertion and deletion of edges and for deleting all edges incident with a single vertex at once. As an example we give the update formula for inserting a new edge \((i, j)\) from vertex \(i\) to \(j\). By \(f'_{s,t}(x)\) we denote the path counting polynomial after insertion of edge \((i, j)\). With this notation, we can state one of Hesse’s results:

\[
f'_{s,t}(x) = f_{s,t}(x) + f_{s,i}(x) \left( \sum_{k=0}^{\infty} (f_{j,i}(x)x^k) \right) x f_{j,t}(x) .
\]  

In DynSelTransClos we consider selective graphs, which means we count only active paths between vertices: in our setting, the coefficient \(p_{s,t}(k)\) is the number of active paths between \(s\) and \(t\) of length \(k\). In DynSelTransClos there are four operations to be considered: insertion and deletion of edges and selection and deselection of vertices. For the insertion and deletion of edges and deselection
of vertices we can apply the update formulas given by Hesse mentioned above. For solving DYNSELTRANS CLOS we prove the following lemma, where, for a vertex \( i \) we denote by \( \langle i \rangle \) and \([i]\) the set of all predecessors and successors of \( i \), respectively.

**Lemma 5** Let \( i \) be a non-active vertex of graph \( G \) and \( G' \) the graph \( G \) with activated vertex \( i \) and edge \((i,i)\) not contained in \( G \). For all vertices \( s,t \) of \( G \), let \( f_{s,t}(x) \) and \( f'_{s,t}(x) \) be the path counting power series of \( G \) and \( G' \), respectively:

\[
 f_{s,t}(x) = \sum_{k=0}^{\infty} p_{s,t}(k)x^k, \quad f'_{s,t}(x) = \sum_{k=0}^{\infty} p'_{s,t}(k)x^k. \tag{41}
\]

Then, for \( i \notin \{s,t\} \),

\[
 f'_{s,t}(x) = f_{s,t}(x) + f_{s,\langle i \rangle}(x)x^2 \left( \sum_{k=0}^{\infty} (f_{\langle i \rangle,\langle i \rangle}(x)x^2)^k \right) f_{\langle i \rangle,\langle i \rangle}(x). \tag{42}
\]

where

\[
f_{s,\langle i \rangle}(x) = \sum_{u \in \langle i \rangle} f_{s,u}(x), \tag{43}
\]

\[
f_{\langle i \rangle,\langle i \rangle}(x) = \sum_{v \in \langle i \rangle} f_{v,t}(x), \tag{44}
\]

\[
f_{\langle i \rangle,\langle i \rangle}(x) = \sum_{(v,u) \in \langle i \rangle \times \langle i \rangle} f_{v,u}(x). \tag{45}
\]

**Proof (following the proof of [7, Lemma 2])**. The main idea is to write \( p'_{s,t}(l) \) as a sum where the \( j \)-th addend is the number of active paths of length \( l \) from \( s \) to \( t \) containing \( i \) exactly \( j \) times.

Each active path from \( s \) to \( t \) in \( G' \) of length \( l \) containing \( i \) exactly once \((j = 1)\) can be decomposed into three parts: an active path from \( s \) to an active predecessor \( u \) of \( i \) in \( G \) of length \( m \), edges \((u,i)\) and \((i,v)\) for an active successor of \( i \) in \( G \), and an active path from \( v \) to \( t \) of length \( l - 2 - m \). Thus, the number of active paths from \( s \) to \( t \) in \( G' \) containing vertex \( i \) exactly once is:

\[
\sum_{m=0}^{l-2} p_{s,\langle i \rangle}(m)p_{\langle i \rangle,t}(l - 2 - m). \tag{46}
\]

This is the coefficient of \( x^l \) in \( f_{s,\langle i \rangle}(x)x^2f_{\langle i \rangle,\langle i \rangle}(x) \).

An active path from \( s \) to \( t \) using \( i \) exactly twice \((j = 2)\) can be decomposed into an active path from \( s \) to an active predecessor \( u \) of \( i \), edges \((u,i)\) and \((i,v)\) for an active successor \( v \) of \( i \), an active path from \( v \) to an active predecessor \( u' \) of \( i \), edges \((u',i)\) and \((i,v')\) for an active successor \( v' \) of \( i \), and an active path from \( v' \) to \( t \) in \( G \). The number of such paths of length \( l \) is:

\[
\sum_{m=0}^{l-4} \sum_{o=0}^{l-4-m} p_{s,\langle i \rangle}(m)p_{\langle i \rangle,\langle i \rangle}(o)p_{\langle i \rangle,\langle i \rangle}(l - 4 - m - o). \tag{47}
\]
This is the coefficient of \(x^l\) in \(f_{s,\langle i \rangle}(x)x^2 f_{\langle i \rangle, \langle i \rangle}(x)x^2 f_{\langle i \rangle, t}(x)\). Generalizing this to arbitrary \(j\) and summing the respective expressions will yield an expression identical with the right-hand side of (42).

\[a_{s,t} = \sum_{k=0}^{n-1} p_{s,t}(k)r^k\] (48)

where \(r\) is large enough. In fact, as Hesse points out, \(r = 2^{n^2}\), because then the binary representation of \(a_{s,t}\) is merely the concatenation of the binary representations of the \(p_{s,t}(k)\)'s with appropriate padding.

The update formulas for the path counting polynomials of Section 5.1 can be applied directly to compute the updates of their integer representations by computing \(mod r^n\). Now (40) turns into

\[a_{s,t}' = a_{s,t} + a_{s,i} \left( \sum_{k=0}^{n-2} (a_{j,i}r^k) r a_{j,t} \right) mod r^n .\] (49)

Using the result of [6] that the product of \(n^{O(1)}\) numbers in binary of \(n^{O(1)}\) bits can be computed by a TC\(^0\) (or FO[Count]) procedure we show just as Hesse [7] that the update formulas for the integers \(a_{s,t}\) can be computed by a TC\(^0\) program. Together with Lemmas 4 and 1 this completes our proof of Theorem 3.

6 Conclusion

In this paper, we have shown that dynamic LTL model checking is in Dyn-TC\(^0\). This is a first step towards a dynamic treatment of model checking problems from a complexity-theoretic point of view. Other model checking problems should be considered, and, of course, related problems. For instance, we have not dealt with counter-examples (error traces) in this paper, and it is not clear to us how to maintain counter-examples explicitly, say, by a function which on input \(i\) returns the \(i\)th state on an error trace.

References

Coping Polynomially with Numerous but Identical Elements within Planning Problems

Max Kanovich\(^1\) and Jacqueline Vauzeilles\(^2\)

\(^1\) University of Pennsylvania, 200 S. 33rd St, Philadelphia, PA 19104
kanovich@saul.cis.upenn.edu

\(^2\) LIPN, UMR CNRS 7030, Paris 13, 99 Av.J.-B.Clément, 93430 Villetaneuse, France
jv@lipn.univ-paris13.fr

Abstract. Since the typical AI problem of making a plan of the actions to be performed by a robot so that it could get into a set of final situations, if it started with a certain initial situation, is generally exponential (it is even EXPTIME-complete in the case of games ‘Robot against Nature’), the planners are very sensitive to the number of variables, the inherent symmetry of the problem, and the nature of the logic formalisms being used. The paper shows that linear logic provides a convenient tool for representing planning problems. In particular, the paper focuses on planning problems with an unbounded number of functionally identical objects. We show that for such problems linear logic is especially effective and leads to dramatic contraction of the search space (polynomial instead of exponential). The paper addresses the key issue: “How to automatically recognize functions similarity among objects and break the extreme combinatorial explosion caused by this symmetry,” by means of replacing the unbounded number of specific names of objects with one generic name and contracting thereby the exponential search space over ‘real’ objects to a small polynomial search space but over the ‘generic’ one, with providing a more abstract formulation whose solutions are proved to be directly translatable into (optimal) polytime solutions to the original planning problem.

Keywords: linear logic.

1 Motivating Examples

The aim of this paper is to show that the linear logic formalism can automatically exploit peculiarities of the AI systems under consideration, and achieve a significant speedup over the traditional ones by decreasing the combinatorial costs associated with searching large spaces.

There are a number of logical formalisms for handling the typical AI problem of making a plan of the actions to be performed by a robot so that it could get into a set of final situations, if it started with a certain initial situation (e.g., see [17,4,16,9,14,13]). Since the planning problem is generally PSPACE-hard (it is even EXPTIME-complete in the case of games ‘Robot against Nature’ [13]), the planners are very sensitive to the number of variables, functions similarity among objects, and the nature of the logic formalism being used. E.g., the
computer-aided planners run into difficulties caused by the combinatorial costs associated with searching large spaces for the following planning problems in AIPS Planning Competitions [12] (their common sense solutions are obvious!).

Example 1. “Gripper” [1]: There is a robot with two grippers. It can carry a ball in each. The goal is to take $N$ balls from one room to another.

Example 2. “Elevator” [2]: There is an elevator such that only 6 people can be on it at a time. The goal is to move $N$ passengers to their destination.

The situation becomes much more complicated in the case of the knowledge acquisition games where we have to look for winning strategies against Nature, as in the following folklore example.

Example 3. “Shoes”: You have $N$ pairs of shoes in your closet, all identical! And your hands are numb so you can’t tell if the shoes that you grab are right or left handed (footed). The challenge is to pick out enough shoes to make sure you have at least one shoe for each foot.

Our paper focuses on planning problems with an unbounded number of functionally identical objects. To address the key issue: “How to automatically recognize functions similarity among objects and break the combinatorial explosion caused by this symmetry,” we show that a linear logic approach can automatically aid in detecting symmetries in planning domains, with providing a radical reduction of the number of variables, and thereby polynomial solutions to such planning problems.

The main idea is as follows. First, having detected symmetry within a given system, we break it by replacing the unbounded number of specific names of objects with one ‘generic’ name, so that one can solve a mock ‘generic’ problem with a drastically smaller state space (polynomial instead of exponential) but over the ‘generic’ object. Secondly, each of the mock solutions dealing with one ‘generic’ object is proved to be directly translatable into an (optimal) polytime solution to the original planning problem dealing with the unbounded number of ‘real’ objects. In other words, for a problem in which $N$ objects, say $b_1,b_2,...,b_N$, cannot be particularized, first, we treat them as identical copies of a single ‘generic’ object $b$, with the set $\{b_1,b_2,...,b_N\}$ being replaced by the multiset $\{b,b,...,b\}$ (which provides a polynomial search space but over one ‘generic’ $b$). Secondly, having found a solution to the ‘generic’ planning problem, we have to convert it into a solution to the original problem.

1 An excerpt from [1]: “STRIPS representation leads to a combinatorial explosion in the search space. All planners obviously suffer from this explosion, with the exception of HSP that does quite well. Interestingly, HSP plans only transport one ball at a time leading to lots of unnecessary move actions . . . . IPP has been run with RIFO switched on, which excludes one gripper as irrelevant, i.e. non-optimal plans are found using only the left gripper. . . . IPP and BLACKBOX don’t even provide data on the harder problem instances.”
Contrary to traditional, a.k.a. set-theoretical, logics, linear logic is capable of direct handling multiset constructs. The fact that “two copies of $b$ have property $Q$” can be directly expressed as the formula $(Q(b) \otimes Q(b))$ within the LL framework, with $\otimes$ representing the coexistent “and”. But the situation is generally more subtle and messy. E.g., a formula of the form $(P(b) \otimes Q(b))$ can be interpreted (and used!) in two different ways: “one and the same copy of $b$ has both properties $P$ and $Q$”, or “one copy of $b$ has property $P$, and another has property $Q$”. In the case of a single action of the form $(P(z) \otimes Q(z)) \vdash R(z)$ (which presupposes the former interpretation) the above erasing individuality trick fails: a ‘real’ planning problem of the form $(P(b_1) \otimes Q(b_2)) \Rightarrow (R(b_1) \oplus R(b_2))$ is unsolvable, notwithstanding that the ‘generic’ problem $(P(b) \otimes Q(b)) \Rightarrow R(b)$ has a one-step solution. On the other hand, the latter interpretation is more preferable in the case where $(P(b) \otimes Q(b))$ describes a ‘generic’ configuration of the system. To resolve this conflict, we invoke axioms that are monadic with respect to such a $z$. (See Definition 2)

We illustrate our approach with Example 4 which combines basic features of two combinatorially exploded examples “Gripper” and “Elevator” [12].

Example 4. “Briareus” [2]: A robot has $k$ grips. It can carry a ball in each. The goal is to take $N$ balls from one room to another.

The number of total cases to be investigated in the planning process seems to be at least $\Omega(2^N)$, since each of the balls, say $b_1, b_2, ..., b_N$, has at least two states: “in room 1”, or “in room 2”. But this combinatorial explosion stems primarily from the fact that we are dealing with the *set* $\{b_1, b_2, ..., b_N\}$ of $N$ distinct names. Whereas the balls are supposed to be identical and the initial and final configurations are symmetric w.r.t. the ball’s individual names. Therefore, we will rather deal with the multiset $\{b, b, ..., b\}$ consisting of $N$ copies of one ‘generic’ ball $b$, bearing in mind that, because of commutativity of multisets, the number of the corresponding ‘generic’ cases will be polynomial. Another source of a combinatorial explosion $\Omega(2^k)$ here is the unbounded number of grips, say $h_1, h_2, ..., h_k$. Since the grips are also indistinguishable and interchangeable within configurations of the system, the collection of all grips could be thought of as the multiset $\{h, h, ..., h\}$ consisting of $k$ copies of one ‘generic’ grip $h$.

Let $\mathcal{R}(x)$ mean “the robot is in room $x$”, $\mathcal{H}(y, z)$ mean “grip $y$ holds ball $z$”, $\mathcal{E}(y)$ mean “grip $y$ of the robot is empty”, and $\mathcal{F}(x, z)$ mean “ball $z$ is on the floor of room $x$”. Here $x$ is of sort ‘room’, $y$ is of sort ‘grip’, and $z$ is of sort ‘ball’.

The ‘pick up’ action: “Being in room $x$ and having the empty grip $y$, grasp and hold ball $z$ with the $y$”, is formalized by the following Horn clause

$$\text{pick}(x, y, z) := (\mathcal{R}(x) \otimes \mathcal{E}(y) \otimes \mathcal{F}(x, z)) \vdash (\mathcal{R}(x) \otimes \mathcal{H}(y, z))$$ (1)

The ‘put down’ action: “Being in room $x$ and holding ball $z$ in grip $y$, put the $z$ down on the floor, and leave the $y$ empty”, is specified as

$$\text{put}(x, y, z) := (\mathcal{R}(x) \otimes \mathcal{H}(y, z)) \vdash (\mathcal{R}(x) \otimes \mathcal{E}(y) \otimes \mathcal{F}(x, z))$$ (2)

\[2\] During the battle against the Titans, *Briareus*, a hundred-handed giant, took advantage of his one hundred hands by throwing rocks at the Titans. Indeed, *Briareus* would have failed if he had wasted his time to particularize the rocks and hands!
The ‘move’ action: “Move from room $x_1$ to room $x_2$”, is axiomatized as

$$\text{move}(x_1, x_2) := R(x_1) \vdash R(x_2).$$

Within the original planning problem in Example 4

$$\text{In}_{N,k}(h_1, h_2, ..., h_k, b_1, b_2, .., b_N) \Rightarrow \text{Goal}_N(b_1, b_2, .., b_N)$$

we look for a plan of actions (1)-(3) that leads from the initial situation “The robot is in room 1 with $N$ balls $b_1, b_2, ..., b_N$, and its grips $h_1, h_2, ..., h_k$ are empty”:

$$\text{In}_{N,k}(h_1, .., h_k, b_1, .., b_N) = (R(1) \otimes \bigotimes_{j=1}^k E(h_j) \otimes \bigotimes_{i=1}^N F(1, b_i)), $$

into a situation where “All $N$ balls are in room 2”:

$$\text{Goal}_N(b_1, .., b_N) := \bigotimes_{i=1}^N F(2, b_i).$$

Now we mock this original (4) with the following ‘generic’ planning problem

$$\text{In}_{N,k}(h, h, h, h, b, b, b, b) \Rightarrow \text{Goal}_N(b, b, b, b)$$

The advantage of our erasing individuality trick is that the number of all ‘generic’ configurations that can be generated from ‘generic’ $\text{In}_{N,k}(h, h, h, b, b, b, b)$ by means of actions (1)-(3) turns out to be $O(kN)$, which allows us to find (in polytime) the shortest plans but for the ‘generic’ problem (7) (See Fig. 1).

To complete (and justify) our approach, it still remains to prove that the mock ‘generic’ plans found can be easily converted into plans of the actions within the real world.

Suppose we have to translate the following ‘generic’ plan:

1. apply $\text{pick}(1, h, b)$ to $(R(1) \otimes E(h) \otimes E(h) \otimes F(1, b) \otimes F(1, b) \otimes F(1, b))$, resulting in $(R(1) \otimes H(h, b) \otimes E(h) \otimes F(1, b) \otimes F(1, b))$;
2. apply $\text{pick}(1, h, b)$ to $(R(1) \otimes H(h, b) \otimes E(h) \otimes F(1, b) \otimes F(1, b))$, resulting in $(R(1) \otimes H(h, b) \otimes H(h, b) \otimes F(1, b))$;
We individualize the ‘generic’ names, with providing a path through configurations in the real world:

(1) apply pick(1, h_1, b_1) to \((R(1) \otimes E(h_1) \otimes F(1, b_1))\), resulting in \((R(1) \otimes H(h_1, b_1) \otimes F(1, b_2) \otimes F(1, b_3))\);

(2) apply pick(1, h_2, b_2) to \((R(1) \otimes H(h_1, b_1) \otimes E(h_2) \otimes F(1, b_2) \otimes F(1, b_3))\),
resulting in \((R(1) \otimes H(h_1, b_1) \otimes H(h_2, b_2) \otimes F(1, b_3))\);

(3) apply move(1, 2); etc., etc., eventually reaching the real goal \((R(2) \otimes E(h_1) \otimes F(2, b_1) \otimes F(2, b_2) \otimes F(2, b_3))\).

Theorems 1 and 2 provide an easy-to-check syntactical criterion for our erasing individuality technique to be automatically correct within Example 4. Example 5 illustrates confusing subtleties of asymmetric input-outputs even within the same ‘symmetric’ domain of Example 4.

Example 5. There is a ball in each of two rooms, say ball \(b_1\) in room 1, and ball \(b_2\) in room 2. A one-handed robot is to exchange these two balls. The ‘real’ planning problem in Example 5 is as follows:

\[
(R(1) \otimes E(h_1) \otimes F(1, b_1) \otimes F(2, b_2)) \implies (F(2, b_1) \otimes F(1, b_2))
\]  

(8)

Erasing individuality yields the ‘generic’ planning problem

\[
(R(1) \otimes E(h) \otimes F(1, b) \otimes F(2, b)) \implies (F(2, b) \otimes F(1, b))
\]  

(9)

which has a trivial solution “Do nothing”. But such a ‘generic’ solution cannot give any clue to the real planning problem (8).

Lastly, a more complicated case of a knowledge acquisition game ‘Robot against Nature’ is illustrated with the following example.

Example 6. “Rouge et Noir”: A robot deals with \(N\) balls wrapped with paper. Each ball is either red or black. The goal is to unwrap enough balls to be sure to obtain exactly \(n\) one-colored balls.

Let \(W(z)\) mean “ball \(z\) is wrapped with paper”, \(R(z)\) mean “an unwrapped ball \(z\) has turned out to be red”, and \(B(z)\) mean “an unwrapped ball \(z\) is black”. The ‘learn’ action: “Unwrap ball \(z\), with its color being revealed”, is formalized as

\[
\text{learn}(z) := W(z) \lor (R(z) \otimes B(z));
\]  

(10)

the disjunctive \(\lor\)-form of which emphasizes that the effect of this unwrapping is non-deterministic: the robot does not know, in advance, which colour the ball chosen will be of.

For \(N\) balls, say \(b_1, b_2, \ldots, b_N\), let \(\text{In}_N(b_1, \ldots, b_N) = \bigotimes_{i=1}^N W(b_i)\), and \(\text{Goal}_{N,n}(b_1, \ldots, b_N) = \bigoplus_{\{i_1, \ldots, i_n\}} \bigotimes_{j=1}^n R(b_{i_j}) \oplus \bigoplus_{\{i_1, \ldots, i_n\}} \bigotimes_{j=1}^n B(b_{i_j})\).
(here \( \{i_1, \ldots, i_n\} \) ranges over all \( n \)-element subsets of \( \{1, \ldots, N\} \)), then the ‘real’ planning problem in Example 6 is \( \text{In}_N(b_1, b_2, \ldots, b_N) \Rightarrow \text{Goal}_{N,n}(b_1, b_2, \ldots, b_N) \).

Any solution to the problem seems to be at least exponential, since a plan we are looking for here is a winning strategy that should envisage all possible reactions of Nature on the road from the initial position to a final position, and the number of the corresponding ‘red-black’ distributions is \( \Omega(2^N) \).

Our erasing individuality trick yields the mock ‘generic’ planning problem

\[
\text{In}_N(b, b, \ldots, b) \Rightarrow \text{Goal}_{N,n}(b, b, \ldots, b)
\]

with \( \text{Goal}_{N,n}(b, b, \ldots, b) = (\underbrace{\text{R}(b) \otimes \cdots \otimes \text{R}(b)}_{n \text{ times}}) \oplus (\underbrace{\text{B}(b) \otimes \cdots \otimes \text{B}(b)}_{n \text{ times}}) \).

The number of all ‘generic’ configurations in question is \( O(N^3) \). Hence, a mock winning strategy to (11), if any, can be assembled in polytime in a bottom-up manner (see Theorem 4). For \( N=3, n=2 \), the result is shown in Fig. 2. Each vertex \( v \) prescribes that the robot performs a certain action: its outgoing edges show all effects of the action. E.g., at the initial position WWW, learn \((b)\) is applied, with two outgoing edges showing two possible Nature’s reactions: RWW and BWW.

\[\begin{array}{cccc}
\text{WWW} & \text{RWW} & \text{BWW} \\
\text{RRW} & \text{RRB} & \text{RBB} & \text{BBW} & \text{RRW} \\
\end{array}\]

Fig. 2. The ‘generic’ winning strategy. Here \( \text{W} \) stands for \( \text{W}(b) \), \( \text{R} \) for \( \text{R}(b) \), and \( \text{B} \) for \( \text{B}(b) \).

By Theorem 2 we can convert this ‘generic’ winning strategy into a winning strategy over the real world (following this strategy, the robot can never ever be punished by Nature’s choice of the color of the three balls):

(a) At the initial position \( (\text{W}(b_1) \otimes \text{W}(b_2) \otimes \text{W}(b_3)) \), apply learn \((b_1)\), resulting either in \( (\text{R}(b_1) \otimes \text{W}(b_2) \otimes \text{W}(b_3)) \), or in \( (\text{B}(b_1) \otimes \text{W}(b_2) \otimes \text{W}(b_3)) \).
(b) At a position of the form \( (\text{R}(b_1) \otimes \text{W}(b_2) \otimes \text{W}(b_3)) \), apply learn \((b_2)\).
(c) At a position of the form \( (\text{B}(b_1) \otimes \text{W}(b_2) \otimes \text{W}(b_3)) \), apply learn \((b_2)\).
(d) At a position of the form \( (\text{R}(b_{\pi(1)}) \otimes \text{B}(b_{\pi(2)}) \otimes \text{W}(b_{\pi(3)})) \) (for some permutation \( \pi \)), apply learn \((b_{\pi(3)})\).

\section{The Disjunctive Horn Linear Logic}

We will use the following syntactical conventions. A number of sorts \( \tau_1, \ldots, \tau_s \) is assumed. A formula of the form \( P_1(t_{1,1}, \ldots, t_{1,k_1}) \otimes \cdots \otimes P_m(t_{m,1}, \ldots, t_{m,k_m}) \), where \( P_1, \ldots, P_m \) are predicate symbols, \( t_{1,1}, \ldots, t_{m,k_m} \) are variables or constants of the corresponding sorts \( \tau_1, \ldots, \tau_s \), is called an \( \text{LL-monomial} \) of degree \( m \). The
trivial LL-monomial \( V \) of degree 0 is allowed, for which \( X \otimes V = X \). Formulas of the form \( \langle Z_1 \otimes \cdots \otimes Z_k \rangle \), with \( Z_1, \ldots, Z_k \) being LL-monomials, are called \( \text{LL-polynomials} \). LL-monomials and LL-polynomials are taken modulo commutativity and associativity of \( \otimes \) and \( \oplus \). A disjunctive Horn clause is a sequent of the form \( X \vdash (Y_1 \oplus Y_2 \oplus \cdots \oplus Y_\ell) \), where \( X, Y_1, Y_2, \ldots, Y_\ell \) are LL-monomials \( (\ell \geq 1) \). Intuitively, this Horn clause represents a move of a robot based on precondition \( X \) accomplished by one of the possible responds \( Y_1, Y_2, \ldots, Y_\ell \) of Nature. A Horn clause of the form \( X \vdash \emptyset \) will be also called a pure Horn clause. Let \( \#_z(X) \) denote the number of all occurrences of the \( z \) in an LL-monomial \( X \). For LL-monomials \( Y_1, Y_2, \ldots, Y_\ell \), \( \#_z(Y_1 \oplus Y_2 \oplus \cdots \oplus Y_\ell) := \max\{\#_z(Y_j) \mid j = 1, 2, \ldots, \ell\} \).

Given a sort \( \tau \), we will write \( A(z_1, z_2, \ldots, z_n) \) to indicate that the list \( z_1, z_2, \ldots, z_n \) contains all variables of the sort \( \tau \) that occur in the \( A \). By symmetric closure \( A^{\text{Sym}}(z_1, z_2, \ldots, z_n) \) of a given \( A(z_1, z_2, \ldots, z_n) \), we mean the following formula \( \bigoplus_{\pi} A(z_{\pi(1)}, z_{\pi(2)}, \ldots, z_{\pi(n)}) \), where \( \pi \) ranges over all permutations of \( \{1, \ldots, n\} \). A formula \( A(z_1, z_2, \ldots, z_n) \) is symmetric if, for any permutation \( \pi \), a sequent of the form \( A(z_1, z_2, \ldots, z_n) \vdash A(z_{\pi(1)}, z_{\pi(2)}, \ldots, z_{\pi(n)}) \) is derivable in linear logic.

The desired property “generic plans \( \implies \) real plans” is represented as follows.

**Definition 1.** Let \( \text{Ax}_T \) be a set of ‘proper axioms’ specifying the theory \( T \). A given sort \( \tau \) is said to be generic within AL-theory \( T \), if the following holds: whatever variables \( z, z_1, z_2, \ldots, z_n \) of sort \( \tau \), LL-monomial \( W(z_1, z_2, \ldots, z_n) \), in which each of the \( z_i \) has exactly one occurrence, and LL-polynomial \( \tilde{Z}(z_1, z_2, \ldots, z_n) \) such that \( \#_{z_i}(\tilde{Z}) \leq 1 \) for each \( z_i \), we take, if a sequent of the form

\[
W(z, z, \ldots, z) \vdash \tilde{Z}(z, z, \ldots, z) \tag{12}
\]

is derivable from \( \text{Ax}_T \) by the rules of affine logic then a sequent of the form

\[
W(z_1, z_2, \ldots, z_n) \vdash \tilde{Z}^{\text{Sym}}(z_1, z_2, \ldots, z_n) \tag{13}
\]

is also derivable from \( \text{Ax}_T \) by the rules of affine logic.

(a) The condition on the number of occurrences of \( z_i \) makes sense. Let, for instance, \( W(z_1, z_2) := (P(z_1) \otimes Q(z_2)) \), and \( Z(z_1, z_2) := (Q(z_1) \otimes P(z_1)) \). Then \( W(z, z) \vdash Z(z, z) \) is derivable, but \( W(z_1, z_2) \vdash (Z(z_1, z_2) \oplus Z(z_2, z_1)) \) is not.

(b) We have need of the permutations in Definition \( 1 \). Let, for instance, \( W(z_1, z_2) := (P(z_1) \otimes Q(z_2)) \), and \( Z(z_1, z_2) := (Q(z_1) \otimes P(z_2)) \).

Then \( W(z, z) \vdash Z(z, z) \) is derivable, but \( W(z_1, z_2) \vdash Z(z_1, z_2) \) is not.

**Proposition 1.** Let \( \tau \) be a generic sort within theory \( T \). Let \( z_1, z_2, \ldots, z_n \) be variables of sort \( \tau \), and \( W(z_1, z_2, \ldots, z_n) \) be an LL-monomial in which each of the \( z_i \) has exactly one occurrence, and \( \tilde{Z}(z_1, z_2, \ldots, z_n) \) be a symmetric LL-polynomial such that \( \#_{z_i}(\tilde{Z}) \leq 1 \) for each \( z_i \). Then for any constants \( b_1, b_2, \ldots, b_n \) of sort \( \tau \) that have no occurrence in \( \text{Ax}_T \), \( W \), and \( \tilde{Z} \), the sequent

\[
W(b_1, b_2, \ldots, b_n) \vdash \tilde{Z}(b_1, b_2, \ldots, b_n) \tag{14}
\]

\( ^3 \) Affine logic = linear logic + Weakening rule.
is derivable from $\text{Ax}_T$ in affine logic if and only if a sequent of the ‘generic’ form

$$W(b, b, ..., b) \vdash \tilde{Z}(b, b, ..., b) \quad (15)$$

is derivable from $\text{Ax}_T$ by the rules of affine logic.

Thus, the original planning problem (14) dealing with a variety of $n$ ‘real objects’ can be fully sorted out in terms of the ‘generic’ planning problem (15) dealing with only one ‘generic object’.

3 ‘Generic’ Plans $\implies$ ‘Real’ Plans

**Definition 2.** Let $\#_\tau(X)$ denote the number of all occurrences of variables of sort $\tau$ in an LL-monomial $X$.

$$\#_\tau(Y_1 \oplus \cdots \oplus Y_\ell) := \max\{\#_\tau(Y_j) \mid j = 1, 2, ..., \ell\}, \text{ for LL-monomials } Y_1, ..., Y_\ell.$$  

A disjunctive Horn clause $X \vdash (Y_1 \oplus \cdots \oplus Y_\ell)$ is said to be $\tau$-monadic if $\#_\tau(Y_1 \oplus \cdots \oplus Y_\ell) \leq \#_\tau(X) \leq 1$.

We will use the following strong version of the normalization lemma for Horn linear logic (cf. [12]).

**Lemma 1.** Let $\text{Ax}_T$ consist of disjunctive Horn clauses.

Given LL-monomials $W, Z_1, Z_2, ..., Z_k$, let a sequent of the form $W \vdash \tilde{Z}$ where $\tilde{Z} = (Z_1 \oplus \cdots \oplus Z_k)$, be derivable from $\text{Ax}_T$ in affine logic. Then $W \vdash \tilde{Z}$ is derivable from $\text{Ax}_T$ by means of the following rules

$$\frac{(V \otimes Y_1) \vdash \tilde{Z} \quad (V \otimes Y_2) \vdash \tilde{Z} \quad \cdots \quad (V \otimes Y_\ell) \vdash \tilde{Z}}{(V \otimes X) \vdash \tilde{Z}} \quad (16)$$

(for $X \vdash (Y_1 \oplus Y_2 \oplus \cdots \oplus Y_\ell)$ is an instance of a sequent from $\text{Ax}_T$, and $V$ is any LL-monomial) and

$$\frac{(Z_i \otimes Z') \vdash \tilde{Z}}{} \quad (17)$$

($Z'$ is any LL-monomial).

**Theorem 1.** For a given sort $\tau$, let $\text{Ax}_T$ consist only of $\tau$-monadic disjunctive Horn clauses. Then sort $\tau$ is generic within $\text{AL}$-theory $T$, and, in addition to that, whatever $z_1, z_2, ..., z_n$, variables of sort $\tau$, $W(z_1, z_2, ..., z_n)$, an LL-monomial, in which each of the $z_i$ has exactly one occurrence, and $\tilde{Z}(z_1, z_2, ..., z_n)$, an LL-polynomial such that $\#_{z_i}(\tilde{Z}) \leq 1$ for each $z_i$, we take, every AL-proof within theory $T$ for (12) can be translated into an AL-proof within theory $T$ for (13).

**Proof.** Given an AL-proof for (12), first, by Lemma 1 we translate it into a derivation tree with rules (16) and (17). Then we assemble the desired proof for (13) by induction on this derivation. Let us sketch the basic case of rule (16).

Suppose that $X(v) \vdash (Y_1(v) \oplus Y_2(v))$ is a $\tau$-monadic Horn clause from $\text{Ax}_T$, and $v$ is a variable of sort $\tau$, and for some LL-monomial $V(z_2, ..., z_n)$, the sequent
V(z, ..., z) ⊗ X(z) ⊢ \tilde{Z}(z, z, ..., z) is produced by rule (16) from the sequents V(z, ..., z) ⊗ Y_1(z) ⊢ \tilde{Z}(z, z, ..., z) and V(z, ..., z) ⊗ Y_2(z) ⊢ \tilde{Z}(z, z, ..., z). By the hypothesis, two sequents of the form V(u_2, ..., u_n) ⊗ Y_1(u_1) ⊢ \tilde{Z}^{Sym}(u_1, u_2, ..., u_n) and V(v_2, ..., v_n) ⊗ Y_2(v_1) ⊢ \tilde{Z}^{Sym}(v_1, v_2, ..., v_n) are derivable by rules (16) and (17), where u_1, ..., u_n, and v_1, ..., v_n are permutations of z_1, ..., z_n. Since \tilde{Z}^{Sym}(z_1, z_2, ..., z_n) is symmetric, V(z_2, ..., z_n) ⊗ Y_1(z_1) ⊢ \tilde{Z}^{Sym}(z_1, z_2, ..., z_n) and V(z_2, ..., z_n) ⊗ Y_2(z_1) ⊢ \tilde{Z}^{Sym}(z_1, z_2, ..., z_n) are also derivable, and the same rule (16) provides derivability of \(V(z_2, ..., z_n) \otimes X(z_1) \vdash \tilde{Z}^{Sym}(z_1, z_2, ..., z_n)\).

3.1 Plans, a.k.a. Winning Strategies

For the systems with pure deterministic actions, a plan \(P\) is defined as a chain of the actions [17, 4, 16]. In order to cover the general case where the actions with non-deterministic effects are allowed, we extend this definition to graph plans \(P\), in which each vertex \(v\) prescribes the performance of a certain action, with its outgoing edges showing all possible reactions of Nature.

**Definition 3.** Let \(\text{Ax}_T\) be a set of disjunctive Horn clauses specifying actions in a given robot system.

A plan \(P\) for a given problem \(W \Rightarrow (Z_1 \oplus \cdots \oplus Z_k)\) is a finite directed graph having no directed cycles such that

(a) each node \(v\) with exactly \(\ell \geq 1\) sons \(w_1, ..., w_\ell\), is labelled by an instance of a Horn clause from \(\text{Ax}_T\) (representing the action performed at \(v\)) of the form

\[X \vdash (Y_1 \oplus \cdots \oplus Y_\ell)\]  

and by an LL-monomial (representing the position at \(v\)) of the form \((X \otimes V)\);

(b) the outgoing edges \((v, w_1), ..., (v, w_\ell)\) are labeled by pure Horn clauses \(X \vdash Y_1, ..., X \vdash Y_\ell\), respectively;

(c) the position each \(w_i\) is labeled by must be equal to \((Y_j \otimes V)\) (showing all possible effects of action [17A]);

(d) there is exactly one node having no incoming edges (the root); the position at the root is \(W\);

(e) each node \(v\) having no outgoing edges (a terminal node) is labeled only by LL-monomial of the form \((Z_i \otimes Z')\) (representing a final position).

In many cases Definition 3 yields the exponential size of winning strategies by pure technical reasons, even if they have a uniform structure. Therefore, we make Definition 3 more liberal in the following respect. We will allow to label any node \(v\) by a position pattern of the form \((X(z_1) \otimes V(z_2, ..., z_n))\), meaning that being at a position of the form \((X(c_1) \otimes V(c_2, ..., c_n))\), where \(c_1, c_2, ..., c_n\) is a permutation of \(b_1, b_2, ..., b_n\), the robot should apply the corresponding version of the action this node \(v\) is labeled by. (See Example 6)

**Theorem 2.** For a given sort \(\tau\), let \(\text{Ax}_T\) consist only of \(\tau\)-monadic disjunctive Horn clauses. Let \(z_1, z_2, ..., z_n\) be variables of sort \(\tau\), and \(W(z_1, z_2, ..., z_n)\)
be an LL-monomial in which each of the $z_i$ has exactly one occurrence, and $	ilde{Z}(z_1, z_2, ..., z_n)$ be a symmetric LL-polynomial such that $\# z_i(\tilde{Z}) \leq 1$ for each $z_i$. Let $b, b_1, b_2, ..., b_n$ be constants of sort $\tau$ having no occurrence in $\text{Ax}_T$, $W$, and $\tilde{Z}$. Then every cut-free AL-proof within theory $T$ for (15) (dealing with one ‘generic object’ $b$) can be converted (in polytime) into a solution, a winning strategy, to the planning problem (14) dealing with $n$ ‘real objects’.

**Proof Sketch.** A given AL-proof is transformed into a derivation $D$ for (15) with rules from Lemma 1. Because of $\tau$-monadicness, each of the sequents in this $D$ is of the form $U(b, ..., b) \vdash \tilde{Z}(b, ..., b)$, for some LL-monomial $U(z_1, ..., z_n)$.

The desired winning strategy $S$ is constructed as follows.

The left-hand sides $U(b, ..., b)$ of different sequents from $D$ are taken as the nodes of $S$. The position pattern the node $U(b, ..., b)$ must be labeled by is defined as $U(z_1, ..., z_n)$. By $D_U$ we denote one of sub-derivations whose root is of the form $U(b, ..., b) \vdash \tilde{Z}(b, ..., b)$ and its height is minimal. Suppose that in $D_U$ its root is produced by rule (16) from the sequents just above

$$U_1(b, ..., b) \vdash \tilde{Z}(b, ..., b), \ldots, U_\ell(b, ..., b) \vdash \tilde{Z}(b, ..., b),$$

invoking an instance of a sequent from $\text{Ax}_T$ of the form

$$X \vdash (Y_1 \oplus \cdots \oplus Y_\ell)$$

Then we label this node $U(b, ..., b)$ by (19), make arrows from $U(b, ..., b)$ to each of the nodes $U_1(b, ..., b), \ldots, U_\ell(b, ..., b)$, and label these edges by $X \vdash Y_1, \ldots, X \vdash Y_\ell$, respectively. (See Figure 2)

Notice that the size of the plan $S$ constructed is bounded by the number of different sequents in derivation $D$.

**Corollary 1.** Within Example 4, the ‘real’ planning problem $\text{In}_{N,k}(h_1, h_2, ..., h_k, b_1, b_2, ..., b_n) \vdash \text{Goal}_N(b_1, b_2, ..., b_n)$ can be fully sorted out in terms of the ‘generic’ planning problem $\text{In}_{N,k}(h, h, ..., h, b, b, ..., b) \vdash \text{Goal}_N(b, b, ..., b)$.

4 Complexity

4.1 Weighted Balance $\rightarrow$ EXPTIME Decidability

The (propositional) disjunctive Horn linear logic is undecidable [12]. By means of the “mixed balance” we show here that the planning problem is decidable for a reasonably wide class of natural robot systems (cf. [13]).

Suppose that sorts $\tau_1, ..., \tau_m$ are fixed. Given an LL-monomial $X$, we define its mixed weight $\omega_{\tau_1, ..., \tau_m}(X)$ as follows:

$$\omega_{\tau_1, ..., \tau_m}(X) = \#_{\tau_1}(X) + \cdots + \#_{\tau_m}(X) + \text{the number of occurrences of atomic formulas in } X \text{ that do not contain an occurrence of a variable of either of sorts } \tau_1, ..., \tau_m.$$
E.g., for $Y = (R(x) \otimes H(y, z))$, $\omega_{\text{ball.grip}}(Y) = 3$; for $X = (R(x) \otimes E(y) \otimes F(x, z))$, $\omega_{\text{ball.grip}}(X) = 3$.

**Definition 4.** Let $Ax_T$ consist of disjunctive Horn clauses. We will say that $Ax_T$ is well-balanced if one can find sorts $\tau_1, \ldots, \tau_m$ ($m \geq 0$) such that for any sequent $X \vdash (Y_1 \oplus \cdots \oplus Y_\ell)$ taken from $Ax_T$ the following holds:

$$\omega_{\tau_1 \ldots \tau_m}(Y_1) \leq \omega_{\tau_1 \ldots \tau_m}(X), \ldots, \omega_{\tau_1 \ldots \tau_m}(Y_\ell) \leq \omega_{\tau_1 \ldots \tau_m}(X).$$

E.g., the system of axioms in Example 4 turns out to be well-balanced, notwithstanding that this system is not well-balanced in the strict sense of [13].

**Theorem 3.** Let $Ax_T$ be well-balanced, and $D$ be a derivation for a sequent of the form $W \vdash \tilde{Z}$ based on rules (16) and (17) from Lemma 1. Then one can construct (at least in EXPTIME) a solution, a winning strategy, to $W \Rightarrow \tilde{Z}$.

**Proof.** Cf. [13].

### 4.2 Monadic & Balanced $\rightarrow$ Polytime Planning

**Theorem 4.** Let $Ax_T$ be well-balanced, and for a given sort $\tau$, let $Ax_T$ consist only of $\tau$-monadic disjunctive Horn clauses. Let $z_1, z_2, \ldots, z_n$ be variables of sort $\tau$, and $W(z_1, z_2, \ldots, z_n)$ be an LL-monomial in which each of the $z_i$ has exactly one occurrence, and $\tilde{Z}(z_1, z_2, \ldots, z_n)$ be a symmetric LL-polynomial such that $\#_{z_i}(\tilde{Z}) \leq 1$ for each $z_i$. Let $b_1, b_2, \ldots, b_n$ be constants of sort $\tau$ having no occurrence in $Ax_T$, $W$, and $\tilde{Z}$. Then we can determine in polynomial time whether there is a plan for the problem

$$W(b_1, b_2, \ldots, b_n) \vdash \tilde{Z}(b_1, b_2, \ldots, b_n) \tag{20}$$

and, if the answer is positive, make such a plan in polynomial time.

**Proof Sketch.** According to Theorem 2 it suffices to take a derivation $D$ for a “mock” sequent of the form

$$W(b, b, \ldots, b) \vdash \tilde{Z}(b, b, \ldots, b) \tag{21}$$

Let $m$ be the string length of the formula $W(b, \ldots, b)$. By Lemma 11 we can confine ourselves to derivations $D$ in which all sequents are of the form $U(b, \ldots, b) \vdash \tilde{Z}(b, \ldots, b)$ and the degree of $U(b, \ldots, b)$ does not exceed $m$.

The number of all $U(b, \ldots, b)$ whose degree does not exceed $m$ is polynomial. The degree of the polynomial is determined by the number of predicate symbols, their arity, and the number of constants in $Ax_T$, $W(b, \ldots, b)$, and $\tilde{Z}(b, b, \ldots, b)$.

Applying the bottom-up technique, we can construct in polytime a list of all $U(b, \ldots, b)$ used in the proof of Theorem 2 with providing the desired winning strategy. \[\blacksquare\]
5 Yet Another Formalism . . .

What kind of arguments could we offer about linear logic superiority as a modeling formalism for the planning domain considered here? Schematically, our approach involves the following steps:

(a) ‘Generic’ Planning Problem $\implies$ ‘Generic’ Plans
Since the “mock” search space is guaranteed to be polynomial, on the road from a specification of the “mock” generic problem to its solution one can choose among a wide spectrum of techniques: direct searching for the shortest paths, theorem proving, BDDs, SAT, etc. It should be pointed out that we are looking for plans, so that the pure decision procedure is not satisfactory (as a rule, the pure existence of a plan is almost evident). As compared to many existing logic formalisms for planning, the advantage of linear logic here is that there is a clear direct correspondence between proofs and plans.

(b) ‘Generic’ Plans $\implies$ ‘Real’ Plans
The idea that any “mock” generic problem can be transformed to a plan over the real world seems ‘orthogonal’ to traditional set-theoretical logical systems but it can be easily stated, justified and explained in terms of linear logic.

Furthermore, the linear logic approach proposed here is complete in the sense that any planning problems with symmetries can be specified (or reformulated) in terms of the fragment of linear logic consisting of disjunctive Horn clauses.

5.1 Genericness vs. Bisimulation

Theorem 5. For a given sort $\tau$, let $\text{Ax}_\tau$ consist only of $\tau$-monadic pure Horn clauses.
Then for any variables $z, z_1, z_2, \ldots, z_n$ of sort $\tau$ the following holds: whatever LL-monomial $W(z_1, z_2, \ldots, z_n)$ in which each of the $z_i$ has exactly one occurrence,
and LL-polynomial \( \tilde{Z}(z_1, z_2, \ldots, z_n) \) such that \( \#_{z_i}(\tilde{Z}) \leq 1 \) for each \( z_i \), we take, if a sequent of the form

\[
W(z, z, \ldots, z) \vdash \tilde{Z}(z, z, \ldots, z)
\]  

(22)

is derivable from \( \text{Ax}_T \) by the rules of affine logic, then for some permutation \( \pi \), a sequent of the form

\[
W(z_1, z_2, \ldots, z_n) \vdash \tilde{Z}(z_{\pi(1)}, z_{\pi(2)}, \ldots, z_{\pi(n)})
\]  

(23)

is also derivable from \( \text{Ax}_T \) by the rules of affine logic.

Theorem 5 allows us to automatically detect a certain bisimulation symmetry, which can be used to construct a quotient bisimulation partition. There have been similar attempts to automatically detect symmetries in other areas such as SAT checking [3]. In brief, their approach is to take an exponential state space and partition it with respect to the symmetry detected. But the symmetry by itself does not help to reduce the number of variables (any representative still deals with \( z_1, z_2, \ldots, z_n \)). We believe that our approach is more universal, uniform and efficient, since from the very beginning we start from a small “mock” space, and do not invoke the original exponential space at all.

Furthermore, Theorem 5 and the bisimulation caused by it fail in a general case:

**Example 7.** Let \( \text{Ax}_T \) consist of the following axioms:

\[
\begin{align*}
A \otimes P(z) &\vdash B \otimes L(z) \oplus C \otimes R(z) \\
B \otimes P(z) &\vdash D \otimes R(z) \\
C \otimes P(z) &\vdash D \otimes L(z)
\end{align*}
\]

(24)

Then a “mock” sequent of the form \( A \otimes P(b_1) \otimes P(b_2) \vdash D \otimes L(b_1) \otimes R(b_2) \) is derivable from (24), but neither \( A \otimes P(b_1) \otimes P(b_2) \vdash D \otimes L(b_1) \otimes R(b_2) \) nor \( A \otimes P(b_1) \otimes P(b_2) \vdash D \otimes L(b_2) \otimes R(b_1) \) is derivable from (24) by the linear logic rules.

Indeed, Theorem II provides derivability from (24) for a sequent of the ‘symmetric’ form

\[
A \otimes P(b_1) \otimes P(b_2) \vdash D \otimes L(b_1) \otimes R(b_2) \oplus D \otimes L(b_2) \otimes R(b_1).
\]

6 Concluding Remarks

We have shown that the linear logic formalism can automatically exploit peculiarities of the AI systems under consideration, and achieve a significant speedup over the traditional ones by decreasing the combinatorial costs associated with searching large spaces.

We have established a clear and easy-to-check syntactical criterion for detecting symmetry in planning domains, and developed techniques to break it by construction of a more abstract formulation whose solution can automatically
aid in solving the original problem, with providing, in particular, a radical reduction of the number of variables, and thereby \textit{polynomial} solutions to the original planning problems.

These results in some sense are “orthogonal” to traditional logical systems but are easily specified and handled in terms of linear logic, making bridge from human common-sense reasoning and problem solving to \textit{computer-aided planning} and to the ability of the automated systems to reason effectively in complex but natural domains.

References

On Algebraic Specifications of Abstract Data Types

Bakhadyr Khoussainov

Computer Science Department, The University of Auckland, New Zealand
bmk@cs.auckland.ac.nz

Abstract. In this paper we address long standing open problems of Bergstra and Tucker about specifications of abstract data types by means of equations and conditional equations. By an abstract data type we mean the isomorphism type of an algebra. An algebra is algebraically specified if the algebra can be defined uniquely, in a certain precise sense, in terms of a finite number of conditional equations by allowing functions that are not in the original language of the algebra. We provide full solutions to Bergstra and Tucker problems, explain basic ideas, methods, and the logical dependencies between blocks of proofs used in our solutions.

1 Background

In this paper we outline the basic ideas that lead to complete solutions of long standing open problems of Bergstra and Tucker about specifiability of abstract data types by means of finite number of equations and quasiequations. The problems were formulated in the early 80s (e.g. see [1]). The first problem is concerned with finding equational specifications for abstract data types. In 1987 Kasymov solved the problem in [4] by constructing a specific computably enumerable (c.e.) algebra. In [9], the author revisited Kassymov’s proof by providing a natural example of a c.e. algebra, whose construction is based on Kolmogorov complexity, that also solves the problem for equational specifications. However, the second more general problem about specifications of abstract data types by conditional equations (we call them here quasiequations) has since been open. In this paper, we give precise formulations of the problems, a necessary background, and provide basic methods and ideas used in the solutions of the problems.

The modern theory of algebraic specifications has advanced significantly since the early 80s and now possesses its own problems and directions for research. Originally born as a theory for reasoning about abstract data types by means of formal methods and algebraic techniques, the area now covers new methods for specifications and programming concepts (e.g. object-oriented, aspect-oriented, agent-oriented, higher-order logic and functional programming) with the goal of providing foundations for correctness of programs. See for example, recent series of proceedings of the Workshop on Algebraic Development Techniques for the current state of the area. However, in the area there are some open problems of foundational character, and the the problems addressed in this paper are among...
such, that are still needed to be answered. The foundational nature of such problems naturally requires a deep analysis of basic underlying concepts (for this paper these concepts are the precise definitions of abstract data type, equational and quasiequational specification, computation, and deduction) and the use of advanced modern techniques and tools of mathematics for the solutions of the problems (for this paper techniques of modern computability theory and algebra are used). So, we assume that the reader is familiar with standard notions from universal algebra (such as congruence relation, finitely generated and finitely presented algebra, homomorphisms, variety and quasivariety, free algebra), logic (equations, quasiequations, proofs), computability (c.e. sets, simple sets, basics of priority argument constructions), and computable model theory (computable and c.e. algebra, c.e. presentable algebra). All these are in the basic textbooks and introductory chapters from Grätzer \[5\], Goncharov and Ershov \[6\], and Soare \[11\]. Of course, we provide necessary definitions. A good survey on abstract data types and algebraic specifications is Wirsing \[12\]. A basic paper that underlines the logical and algebraic foundations of the problems is Bergstra and Tucker \[2\].

Now we outline this paper. Further in this section, we provide the formulation of the problems and some background. For completeness of our exposition Section 2 briefly outlines the proof for the case of equational specifications (which as mentioned above solved one of the questions of Bergstra and Tucker). Finally, Section 3 is devoted to describing the proof line for the case of quasiequational specifications. Some of the proofs are technical and lengthy, especially the proofs of results in the last section. Therefore we outline basic ideas of the proofs and give appropriate references in which full proofs are presented. We also concentrate on the relationship between the basic blocks of the proofs, and explain methods of algebra and computability that are used in solving the problems.

An algebra is a structure of a finite purely functional language (signature). Thus, any algebra \(A\) is of the form \((A; f_0, \ldots, f_n)\), where \(A\) is a nonempty set called the domain of the algebra, and each \(f_i\) is a functional symbol that names a total operation on the domain \(A\). Often the operation named by \(f_i\) is also denoted by the same symbol \(f_i\). We refer to the symbols \(f_0, \ldots, f_n\) as the signature of the algebra. The Presburger arithmetic \((\omega; 0, S, +)\) is an algebra, so are groups, rings, lattices and Boolean algebras. Fundamental structures that arise in computer science such as lists, stacks, queues, trees, and vectors can all be viewed and studied as algebras. A foundational thesis that connects abstract data types (ADTs) with algebras and forms the basis of the work in algebraic specifications is the following stated as a definition.

**Definition 1.** An abstract data type (ADT) is the isomorphism type of an algebra.

Often abstract data types are defined to be the isomorphism types of many sorted algebras. We note, however, that the specification problems have fully been solved for many sorted algebras (see \[1\]), and hence for the purpose of this paper we omit the case for algebras with many sorts.

Let \(\sigma\) be a functional finite language (signature) with at least one constant symbol. An equation is an expression of the form \(p = q\), and a quasiequation
is an expression of the form \((\&_{i<\omega} p_i = q_i) \rightarrow s = t\), where \(p, q, s, t\) are terms of the signature. These terms may contain variables. An \textit{equational specification} is a \textit{finite} set \(E\) of equations. A \textit{quasiequational specification} is a \textit{finite} set \(Q\) of quasiequations. In this paper the word specification refers to either equational or quasiequational specification unless otherwise stated.

Let \(F(\sigma)\) be the absolutely free algebra generated by the \(\sigma\)-constants. Elements of this algebra are called \textit{ground terms}. Let \(S\) be a specification. Ground terms \(t_1\) and \(t_2\) are \(\equiv_S\)-equivalent if \(t_1 = t_2\) can be proved from \(S\) (within the first order logic). The relation \(\equiv_S\) is a congruence relation on \(F(\sigma)\), and hence one can factorize \(F(\sigma)\) by \(\equiv_S\). We denote this factor algebra by \(F(S)\). In universal algebra \(F(S)\) is called the zero generated free algebra of the quasivariety determined by \(S\).

From a programming point of view \(S\) is thought to be a specification of an abstract data type that is being (or need to be) implemented. One thinks of \(S\) as a set of axioms needed to be satisfied in all implementations of \(S\). Indeed, often when a programmer implements an abstract data type, e.g. stacks or queues or lists, the specification requirements put on the abstract data type are most likely equational or quasiequational in nature. For example, a specification requirement that relates \textit{push} and \textit{pop} operations in a stack abstract data type states that for any stack \(s\) and any item \(i\) the equality \(\text{pop}(\text{push}(s, i), i) = s\) is true.

**Definition 2.** An algebra \(A\) is \textit{specified by a specification} \(S\) if \(A\) is isomorphic to the algebra \(F(S)\). If the specification \(S\) is an equational specification then the algebra \(A\) is \textit{equationally specified}. If \(S\) is a quasiequational specification then the algebra \(A\) is \textit{quasiequationally specified}.

The arithmetic \((\omega, 0, S, +, \times)\) is an equationally specified algebra, and the equations that specify this algebra are the known recursive definitions of + and \(\times\) together with commutativity, associativity, and distributivity axioms. Similarly, the term model for combinatory logic is an equationally specified algebra; this is an algebra of the signature \((K, S, I, \cdot\cdot)\), where \(K\), \(S\), \(I\) are constant symbols and \(\cdot\) is a binary operation all satisfying the following equations: \((K \cdot x) \cdot y = x\), \((S \cdot x) \cdot y \cdot z = (x \cdot z) \cdot (y \cdot z)\), \(I \cdot x = x\). Finally, all finitely presented algebras (such groups or rings) are examples of specified algebras.

In order to study specified algebras we need to employ some notions from computable algebra and model theory. Let \(A = (A, f_0, \ldots, f_n)\) be an algebra. For each element \(a \in A\) introduce a new constant symbol \(c_a\) that names the element \(a\) itself. The \textit{atomic diagram of} \(A\) is the set of all expressions of the type \(f_i(\bar{c}) = c'\) or \(f_i(\bar{c}) \neq c'\) which are true in the algebra \(A\). The \textit{positive atomic diagram of} \(A\) is the set of all expressions of the type \(f_i(\bar{c}) = c'\) which are true in the algebra \(A\). The algebra \(A = (A; f_0, \ldots, f_n)\) is \textit{computable} if its atomic diagram is a computable set. The algebra \(A = (A; f_0, \ldots, f_n)\) is \textit{computably enumerable (c.e.)} if its positive atomic diagram is a computably enumerable set. It is not hard to see that computable algebras are the ones isomorphic to algebras of the type \((\omega, f_0, \ldots, f_n)\), where each \(f_i\) is a computable function on \(\omega\). Clearly, every computable algebra is c.e., and the converse does not generally hold true. Here are some examples of c.e. algebras:
1. Any specified algebra, such as the arithmetic or the term model for combinatorial logic.
2. The Lindenbaum Boolean algebra of any c.e. first order theory, such as the Peano arithmetic.
3. Any finitely presented group, and in fact any finitely presented algebra.

A computably enumerable (c.e.) algebra $A$ can be explained as follows. As the positive diagram of $A$ can be computably enumerated, the set $E = \{(c_a, c_b) \mid a = b \text{ is true in algebra } A\}$ is computably enumerable. Hence the equality relation in $A$ is c.e. Let $f$ be a basic $n$-ary operation on $A$. From the definition of c.e. algebra, the operation $f$ can be thought as a function induced by a computable function, often also denoted by $f$, that is well-behaved on the $E$-equivalence classes in the following sense: for all $x_1, \ldots, x_n, y_1, \ldots, y_n$ if $(x_i, y_i) \in E$, then $(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)) \in E$. Therefore, a natural way to think about $A$ is that the elements of $A$ are $E$-equivalence classes, operations of $A$ are induced by computable operations. This reasoning suggests another equivalent approach to the definition of c.e. algebra explained in the next paragraph.

Let $E$ be a c.e. equivalence relation on $\omega$. A computable $n$-ary function $f$ respects $E$ if for all natural numbers $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ so that $(x_i, y_i) \in E$, for $i = 1, \ldots, n$, we have $(f(x_1, \ldots, x_n), f(y_1, \ldots, y_n)) \in E$. Let $\omega(E)$ be the factor set obtained by factorizing $\omega$ by $E$, and let $f_0, \ldots, f_n$ be computable operations on $\omega$ that respect the equivalence relation $E$. An $E$-algebra is then the algebra $(\omega(E), F_0, \ldots, F_n, c_0, \ldots, c_m)$, where each $F_i$ is naturally induced by $f_i$ and each $c_j$ is a constant symbol. It is now not hard to show that an algebra $A$ is c.e. if and only if $A$ is an $E$-algebra for some c.e. equivalence relation $E$.

An algebra is computably presentable if it is isomorphic to a computable algebra. Similarly, an algebra is c.e. presentable if it is isomorphic to a c.e. algebra. In the literature c.e. algebras are sometimes called semicomputable data structures [1] or positive algebras [6]. Thus, if one thinks of an ADT as an algebra then a computable or c.e. presentation of it can be identified with a machine-theoretic (or program-theoretic) implementation of the ADT. The theorem below, a classic result of universal and computable algebra, outlines the basic properties of specified algebras:

**Theorem A** If the algebra $A$ is specified by specification $S$ then $A$ satisfies the following properties:

1. For ground terms $t_1$ and $t_2$, we have $A \models t_1 = t_2$ iff $S$ proves $t_1 = t_2$. Hence, $A$ is a c.e. algebra.
2. $A$ satisfies $S$.
3. For any algebra $B$ that is generated by $\sigma$-constants and satisfies $S$ there exists a unique homomorphism from $A$ onto $B$.
4. If $B$ is specified by $S$ then $A$ and $B$ are isomorphic.

The first part of the theorem tell us that all the positive facts about the ADT $A$ specified by $S$ can be computably enumerated. The second part tells us that $A$ is a correct implementation of $S$. The third part tells us that $A$
is, in a natural sense, the universal implementation among all data structures that satisfy $S$. Finally, the last part is the uniqueness property of the specified algebra. Therefore, often $A$ is called the **initial algebra** of $S$ and our approach to specification of abstract data types is called the **initial algebra** semantics method.

There is one fundamental difference between equationally specified algebras and quasiequationally specified algebras. For equationally specified algebras the third part of Theorem A can be conversed: if $A$ is equationally specified by $E$ and $B$ is a homomorphic image of $A$ then $B$ is generated by $\sigma$-constants and satisfies $E$. This difference is the main reason why the proof for equational specification problem, outlined in the next section, fails to solve the problem for quasiequational specifications.

Let us consider the following algebra $(\omega, S, 2^x, 0)$. This algebra is computable and finitely generated but does not have an equational specification (see [2]). An important fact is that one can enrich this algebra by expanding its signature and consider the following expanded algebra $(\omega, S, 2^x, +, \times, 0)$. Now this algebra is equationally specified: the natural recursive definition of $+$, the definition of $\times$ via $+$, and the definition of $2^x$ via $\times$ do the job.

The observation above suggests the idea of adding more functions to the original signature. This allows one to possess more flexibility in finding a specification of the algebra in an appropriate expansion. We would like to stress that the use of expansions of the original signature is a common and powerful tool that has been employed in pure model theory, algebra, and real programming practice. For example, in model theory expansions are used in constructing models with different properties, e.g. constructing saturated models, finding expansions that have elimination of quantifiers (e.g. expansions by Skolem functions), etc. In programming practice the use of expansions is a usual and natural routine. For instance, after an ADT is implemented by a code it is often the case that one would like to add more functionality to the code. This essentially amounts to adding extra methods to the program which, by its nature, is an expansion of the ADT implemented.

Let $A$ be a c.e. algebra, and let $f_1, \ldots, f_n$ be computable functions (well behaved with respect to the equality relation on $A$). Then the algebra $B = (A, f_1, \ldots, f_n)$ obtained by adding the operations $f_1, \ldots, f_n$ to $A$ is called an **expansion** of $A$. The signature $\sigma \cup < f_1, \ldots, f_n >$ is an expansion of the original signature. The algebra $A$ is called a **reduct** of $B$. An important result is the following theorem proved by Bergstra and Tucker in [1]:

**Theorem B** Any computable algebra $A$ possesses an expansion that can be equationally specified.

The proof consists of two steps. The first step assumes that all original functions $f$ of $A$ are primitive recursive. Hence each $f$ is defined in terms of a sequence of functions (called a **definition** of $f$) using primitive recursive schemata and the operation of composition successively applied to the basic functions: the successor, projections, and constants. The expansion is then obtained by adding the names of all the functions which participate in the definition of $f$. This ex-
pansion of \( A \) is then specified because the definition of \( f \) can be transformed into a specification of \( f \). In the second step it is assumed that there is a non primitive recursive original operation. This is then reduced to the first step by using a primitive recursive enumeration of the graph of the operation. Bergstra and Tucker have a significant improvement of the result: any computable algebra has an equational specification with 6 new functions and 4 equations only \[3\].

The theorem above tells us that the initial algebra semantics method is sound for the class of all computable algebras. Of course, not every finitely generated c.e. algebra is computable, and by this reason Theorem B does not cover the general case. This leads us to the formulation of the following three problems whose solutions are discussed in the next sections. The first question (called subproblem) is a test case as any answer to the question should exclude computable algebras by the theorem above. The last two questions form the problem of Bergstra and Tucker.

**Subproblem:** Is it true that any c.e. algebra has an equationally specified expansion?

Note that the subproblem is not restricted to finitely generated c.e. algebras.

**Problem 1** \([2]\): Does any finitely generated c.e. algebra have an equationally specified expansion?

**Problem 2** \([2]\): Does any finitely generated c.e. algebra possess a quasiequationally specified expansion?

To the reader the questions may remind the known Higman’s embedding theorem stating that any recursively presented group (that is finitely generated c.e. group) can be embedded into a finitely presented (that is equationally specified) one. However, in Higman’s theorem the underlying domain of the group is allowed to be extended to a bigger domain; this is prohibited in the questions of Bergstra and Tucker above. In addition, in Higman’s theorem the embedding preserves the group structure; this is relaxed in the questions of Bergstra and Tucker as expansions are allowed to be arbitrary computable functions.

Finally, we would like to stress that the specification problems discussed in this paper arose from investigations related to the study of semantics of programs, which is a part of a bigger theme in computer science: correctness of programs (e.g. see \([10]\)). Our methods which solve the specification problems use basics from universal algebra and standard techniques from modern computability theory, thus providing an example of interactions between applied and theoretical computer science, algebra and logic.

## 2 Solution of Problem 1

We first give our solution to the subproblem proved in \([8]\). The proof gives a negative answer to the subproblem. We need two notions. An algebra is **locally finite** if any finite subset in it generates a finite subalgebra. A set \( X \) of natural numbers is **hypersimple** if it is computably enumerated, co-infinite, and there
does not exist a computable function \( f : \omega \to \omega \) such that \( f(i) > a_i \) for all \( i \), where \( a_0, a_1, \ldots \) is the enumeration of the complement of \( X \) in strictly increasing order.

**Theorem 0** There exists an infinite algebra \( A \) any c.e. presentable expansion of which is locally finite.

**Proof (outline).** Let \( X \subset \omega \) be a hypersimple set and \( 1 \not\in X \). Consider now the unary algebra \((A, f)\) that satisfies the following properties: 1. \( f : A \to A \) is a bijection; 2. For every \( x \in X \) there exists an \( n \) such that \( f^n(x) = x \) and all the elements \( x, f(x), \ldots, f^{n-1}(x) \) are pairwise distinct. Call the sequence \( x, f(x), \ldots, f^{n-1}(x) \) an orbit of size \( n \). 3. For every \( n \in X \) the algebra does not contain an orbit of size \( n \). 4. For every \( n \not\in X \) the algebra contains exactly one orbit of size \( n \). Clearly, these properties define \( A \) uniquely.

The algebra \( A \) has a c.e. presentation. Indeed, take a computable algebra \((\omega, f)\) such that for every \( n > 1 \) the algebra has exactly one orbit of size \( n \). Now consider the following equivalence relation \( \eta \) on this algebra: \((x, y) \in \eta \) iff either \( x = y \) or both \( x \) and \( y \) belong to orbits \( O_1 \) and \( O_2 \), respectively such that the of \( O_1 \) and the size of \( O_2 \) are both in \( X \). The equivalence relation is a c.e. congruence of the algebra. Hence the factor algebra is a c.e. presentation of \( A \). Now, assume that \( B \) is an expansion of \( A \) such that \( B \) is a c.e. algebra. Then if there was a finite subset in \( B \) that generated an infinite subalgebra then the set \( X \) would not be hypersimple.

Note that the theorem above gives us a negative answer to the subproblem in a strongest possible way. Indeed, it shows that all c.e. implementations (that is, presentations) of \( A \), and not just one, fail to be equationally specified. The failure is not because c.e. implementations of \( A \) are bad but rather the intrinsic property of the algebra itself.

The next theorem solves Problem 1 in negative. This shows that the initial algebra semantics method with equational specifications is not sound for the class of all finitely generated c.e. algebras. For a detailed proof see [7] or [9].

Recall that a set \( X \) of natural numbers is **simple** if it is co-infinite, computably enumerable, whose complement does not contain an infinite c.e. subset.

**Theorem 1.** There exists a finitely generated c.e. algebra such that every expansion of the algebra is not equationally specified.

**Proof (outline).** Let \( X \subset \omega \) be a set. Consider the factor set \( \omega(X) \) whose elements are the equivalence classes of the equivalence relation \( \eta(X) \) defined as follows: \( \eta(X) = X^2 \cup \{(x, x) \mid x \in \omega\} \). Thus, each element of \( \omega(X) \) either is a singleton or is \( X^2 \). Here is the first lemma which we provide without a proof:

**Lemma 1.** There exists a finitely generated c.e. algebra \( A(X) \) such that the domain of \( A(X) \) is \( \omega(X) \) with \( X \) being a simple set.

The proof can be found either in [7], where \( X \) is constructed directly, or in [9] where the algebra \( A(X) \) is constructed by using Kolmogorov complexity. A
point of note is that $X$ can not be hypersimple which is consistent with the previous theorem.

The next lemma is due to Malcev. We say that an algebra $A$ is **residually finite** if for any $a, b \in A$ with $a \neq b$ there is an onto homomorphism $h: A \to B$ such that $h(a) \neq h(b)$ and $B$ is finite. Here is now the next lemma:

**Lemma 2.** If an algebra $A$ is equationally specified and residually finite then the equality relation in the algebra is decidable.

Note that the equality problem for $A(X)$ is not decidable because $X$ is not a computable set. The next lemma, given without proof, essentially uses the fact that $\omega \setminus X$ contains no infinite c.e. subset (such sets are called immune sets). The lemma is a good example of interactions between two seemingly not related notions: immune set from computability theory on the one hand, and residually finite algebra from universal algebra on the other. The full proof is in [7] or [9].

**Lemma 3.** Any expansion $B$ of $A(X)$ is residually finite.

Thus, the lemmas above show that the algebra $A(X)$ is a desired one thus proving the theorem. \(\square\)

An important comment is that for quasiequational specifications the proof of the theorem above fails because, as has been mentioned, the second lemma stated above fails for algebras specified by quasiequations. Therefore, a new point of view is needed for the study of Problem 2 to which we now turn our attention.

### 3 Solution of Problem 2

Most of the algebras constructed in this section have signature $< c, f, g >$, where $c$ is a constant and $f, g$ are unary function symbols. Therefore the readers can restrict themselves to this signature while reading this section. However, we formulate the results (where possible) for any finite functional signature $\sigma$ with at least one constant symbol.

Let $(a, b)$ be a pair of an algebra $A$. Consider the minimal congruence relation, denoted by $\eta(a, b)$ and called a **principal congruence**, containing the pair. We now give an inductive definition of almost free algebras.

**Definition 3.** The absolutely free algebra $F(\sigma)$ is **almost free**. Assume that $A$ is almost free. Then the factor algebra obtained by factorizing $A$ by $\eta(a, b)$, where $(a, b)$ is any pair in $A$, is **almost free**.

Thus $A$ is almost free if and only if there exists a sequence $A_1, \ldots, A_n$ of algebras and the sequence $(a_1, b_1), \ldots, (a_n, b_n)$ of pairs of elements such that $A_1$ is $F(\sigma)$, each $A_{i+1}$ is obtained by factorizing $A_i$ by $\eta(a_i, b_i)$, and $A$ is obtained from factorizing $A_n$ by $\eta(a_n, b_n)$. We call the sequence $(a_1, b_1), \ldots, (a_n, b_n)$ a **witness sequence** for $A$ to be almost free. The reason in introducing almost free algebras is the following. The desired counterexample that solves Problem 2 is
constructed by stages so that at each stage the construction deals with an almost free algebra, and the construction, if needed, acts by factorizing the algebra with respect to a principal congruence thus again producing an almost free algebra.

In case when $\sigma = \langle c, f, g \rangle$, almost free algebras can be characterized as explained below. Let $\mathcal{A}$ be an algebra of signature $\sigma$ and be generated by $c$. We say that there is an edge from $a$ to $b$ if $f(a) = b$ or $g(a) = b$. Thus, $\mathcal{A}$ is turned into a directed graph. Hence one can employ graph theoretic notions to this algebra viewed as a graph. For example, a distance from $a$ to $b$ is the length of a minimal directed path starting at $a$ and ending at $b$. A level in $\mathcal{A}$ is the set of all elements that are on the same distance from the generator $c$. Thus, if $\mathcal{A}$ is the absolutely free algebra of the signature $\sigma$ then there are $2^n$ number of elements at level $n$. The following is not hard to see. The algebra $\mathcal{A}$ is almost free if and only if either $\mathcal{A}$ is finite or there is a level $L$ such that for all $x \in L$ the subalgebra $\mathcal{A}_x$ generated by $x$ is isomorphic to $\mathcal{F}(\sigma)$, and for all distinct $x, y \in L$ the domains of $\mathcal{A}_x$ and $\mathcal{A}_y$ have no elements in common. Here is a lemma whose proof we omit. The reader can verify the correctness of the lemma in the case (sufficient for our counterexample) when $\sigma$ is $\langle c, f, g \rangle$ using the characterization just explained.

**Lemma 4.** There exists an algorithm that for any almost free algebra $\mathcal{A}$, a witness sequence $(a_1, b_1), \ldots, (a_n, b_n)$ for $\mathcal{A}$, a first order formula $\Phi(x_1, \ldots, x_n)$, and an $m$-tuple $(c_1, \ldots, c_m)$ decides whether or not $\mathcal{A} \models \Phi(c_1, \ldots, c_m)$. In particular, the first order theory of any almost free algebra is decidable.

We use a weaker version of this lemma. First of all, we will deal with the mentioned signature $\langle c, f, g \rangle$. Secondly, we restrict ourselves to formulas $\Phi$ which are quasiequational specifications.

The proof of the next lemma uses a diagonalization argument. The result of this lemma can be obtained without the diagonalization. However, the methods and ideas of this and the next lemmas give a hint towards a solution of the problem.

**Lemma 5.** There exists a finitely generated c.e. algebra that has no quasiequational specification in its own signature.

**Proof (outline).** We construct the desired algebra by stages, where at each stage we deal with the almost free algebra obtained at the previous stage. The signature of the algebra $\mathcal{A}$ that we construct is $\langle c, f, g \rangle$. In order to make $\mathcal{A}$ not quasiequationally specified, our construction needs to make sure that $\mathcal{A}$ is not isomorphic to $\mathcal{F}(C)$ for any given quasiequational specification $C$. Thus, in order to construct $\mathcal{A}$ the construction must guarantee the satisfaction of the following requirements:

$$R_e : \quad \mathcal{A} \text{ is not isomorphic to } \mathcal{F}(C_e),$$

where $\{C_e\}_{e \in \omega}$ is an effective list of all quasiequational specifications (note that each quasiequational specification is finite). The previous lemma is an important
tool guaranteeing that the construction can be carried out effectively. At the initial stage, Stage 0, the construction begins with the algebra is $F(\sigma)$. Denote this algebra by $A_0$. At Stage $n + 1$, the goal of the construction is to guarantee that the requirement $R_n$ is satisfied. Here is a description of Stage $n + 1$.

Stage $n + 1$. Assume that $A_n$ has been constructed, and is almost free. In addition we have its witness for being almost free. Take the quasiequation $C_n$. Check whether or not $A_n \models C_n$. This can be checked due to the previous lemma. If $A_n \models C_n$ then find two distinct elements $a, b$ in $A_n$ such that the factor algebra $B$ obtained by factorization of $A_n$ with respect to the principal congruence $\eta(a, b)$ satisfies $R_0, \ldots, R_n$ and is infinite. Set $A_{n + 1} = B$. Clearly, $A_{n + 1}$ is almost free. If $A_n \models C_n$ then find a tuple $\bar{a}$ that witnesses the fact that $A_n \models \neg C_n$. More precisely, we know that the quasiequational specification $C_n$ is of the form

$$\forall \bar{x}(\forall_{i}(\Phi_i(\bar{x}) \rightarrow \Psi_i(\bar{x}))),$$

where each of $\Phi_i$ and $\Psi_i$ is an equation between terms. Since $A_n$ does not satisfy $C_n$ there exists a tuple $\bar{a}$ and $i$ such that $A_n \models \Phi_i(\bar{a})$ and $A \models \neg \Psi_i(\bar{a})$. The construction then guarantees that $\neg \Psi_i(\bar{a})$ is always true, by making sure that the requirements $C_j$ of lower priority ($j > n + 1$) do not violate the truth value of $\Psi_i(\bar{a})$. In this case $A_{n + 1} = A_n$.

The desired algebra is now the following. Let $\eta$ be the congruence relation such that $(a, b) \in \eta$ iff $a = b$ is true in some $A_n$. Clearly $\eta$ is a c.e. relation. Set $A$ to be the factor algebra obtained by factorizing $F(\sigma)$ with respect to $\eta$. It is not hard to show that $A$ is a desired algebra.

The lemma suggests the idea of constructing the algebra that gives a negative solution to Problem 2 by trying to diagonalize against all possible quasiequational specifications in all possible finite expansions of the signature. Doing this directly seems to be a difficult task because of the following two reasons. Say, the $e$th quasiequational specification contains a new function symbol $\psi$. This means that the requirement $R_e$ is equivalent to an infinite list of requirements as now $\psi$ is a new parameter. Hence the list of requirements that correspond to $C_e$ is now this:

$$(A, \phi_j) \text{ is not isomorphic to } F(C_e),$$

where $\{\phi_j\}_{j \in \omega}$ is an enumeration of all partial computable functions. Secondly, the behaviour of each $\phi_j$ is not under our control as we can only control the algebra $A$ which is being constructed. Therefore it is not clear how to directly construct the desired algebra.

We construct the desired algebra $A$ indirectly. The basic idea of using an indirect way is implicitly suggested by the two lemmas above. If we can construct $A$ in such a way that every possible expansion of $A$ does not produce new functions, in the sense expressed in the definition below, then a modification of the proof of the previous lemma does the job. In order to formally explain this idea we give the following definition. Note that almost all means for all but finite.
Definition 4. A new operation \( \psi : A^n \to A \) on a c.e. algebra \( A \) is termal if there is a term \( t(x) \) in the signature of algebra \( A \) such that for almost all \( (a_1, \ldots, a_n) \in A^n \) the equality \( \psi(a_1, \ldots, a_i, \ldots, a_n) = t(a_i) \) is true. An expansion \( B = (A, \psi_1, \ldots, \psi_m) \) of \( A \) is termal if each new operation \( \psi_i \) is termal.

Note that in case of the signature \( < c, f, g > \), every term is of the form \( f^{n_0} g^{m_0} \ldots n_k g^{m_k}(x) \), where \( n_i, m_i \) are natural numbers, and hence contains one variable only.

Thus, the idea is that we want to construct a finitely generated c.e. algebra \( A \) that incorporates the construction of the previous lemma and, in addition, makes sure that any expansion of \( A \) is termal. We single out the following class of c.e. algebras which, we think, is of independent interest.

Definition 5. We say that an infinite finitely generated computably enumerable algebra \( A \) is computationally complete if any expansion of \( A \) is termal.

The following note is simple. If a finitely generated c.e. algebra is computationally complete then any other c.e. algebra isomorphic to it is also computationally complete. The reason for this is that there is a computable isomorphism between any two finitely generated c.e. algebras. So computational completeness is an isomorphism invariant property for finitely generated c.e. algebras. Thus, for a computational complete algebra every expansion of this algebra, say by a function \( g \), does not give us anything new. The function \( g \) already exists in the algebra as \( g \) can be expressed as a term apart from finitely many values.

We also note the following. If we omit the requirement that the algebra is not finitely generated, then computational completeness becomes non isomorphism invariant. In particular, one can construct a c.e. presentation of the following trivial algebra \( (\omega, id) \), where \( id(x) = x \) for all \( x \in \omega \), such that in that presentation every possible expansion function \( g \) becomes identity or a projection function with respect to the equality relation of the presentation.

Finally, in universal algebra there is a notion of primal algebra defined for finite algebras. A finite algebra is primal if any new function \( f : A \to A \) can be expressed as a term. For example, the two valued Boolean algebra is primal. Computational complete algebras are infinite analogues of primal algebras.

For the next lemma that shows usefulness of computationally complete algebras we need the following definition.

Definition 6. A finitely generated algebra \( A \) is term algebraic if for any element \( b \in A \) the number of ground terms \( t \) in the language of the algebra such that the values of \( t \) in \( A \) equal \( b \) is finite.

Here is now the next lemma:

Lemma 6. If \( A \) is a term algebraic computationally complete algebra then no expansion of \( A \) can be quasiequationally specified.

Proof (outline). The idea is the following. Assume that \( S \) is a quasiequational specification of an expansion \( B = (A, \psi_1, \ldots, \psi_n) \) of the algebra \( A \). Then the
specification $S$ can be replaced with a new specification $S'$ in the original signature so that $S'$ specifies $A$. This can be done because, roughly, any new operation $\psi_i$ that is in $S$ can be replaced with a term $t_i$ in the original signature that equals $\psi_i$ almost everywhere. In the transformation of $S$ into $S'$ one needs to use the fact that the algebra is term algebraic.

Thus, we need to show that computationally complete and term algebraic algebras exist. The next lemma gives a brief outline of the construction. The proof uses a technique borrowed from modern computability theory, a priority construction carried out with a method known as a tree argument construction. For a current full version of the proof see [4].

**Lemma 7.** Computationally complete and term algebraic algebras exist.

**Proof (outline).** We present basic ideas of our construction. The signature of the algebra we construct is our signature $\langle c, f, g \rangle$. The construction of the algebra is a stagewise construction so that at each stage we deal with an almost free algebra of the given signature. Let $\phi_0, \phi_1, \ldots$ be an effective list of all computable partial functions. The algebra $A$ that we want to construct must satisfy the following condition. For every $e \in \omega$, if $\phi_e$ is a total function and expands $A$ then $\phi_e$ must be equal to a term $t$ almost everywhere. Thus, to build the desired algebra $A$, the following list $\{T_e\}_{e \in \omega}$ of requirements must be satisfied:

$$
T_e: \quad \text{If } \phi_e \text{ is total and expands the algebra } A \text{ then } \phi_e \text{ is termal.}
$$

We now describe the basic strategy that satisfies one requirement $T_e$. Start constructing the algebra $A$ by making it isomorphic to the free algebra $F(\sigma)$. While constructing, wait until for some tuples $\bar{a} = (a_1, \ldots, a_n), \bar{b} = (b_1, \ldots, b_n)$ in $A$ each of the following occurs:

1. The values $\phi_e(\bar{a})$ and $\phi_e(\bar{b})$ are defined,
2. There is no term $t$ and no $i$ for which $\phi_e(\bar{a}) = t(a_i)$ and $\phi_e(\bar{b}) = t(b_i)$.
3. Viewing $A$ as a directed graph (described in page 6) there is no directed path neither from $a_1$ to $b_1$ nor from $b_1$ to $a_1$, where $a_1$ and $b_1$ are the first components of the tuples $\bar{a}$ and $\bar{b}$.

If such a pair $(\bar{a}, \bar{b})$ occurs, then the desired algebra is obtained by factorizing $F(\sigma)$ by the principal congruence relation $\eta(a_1, b_1)$. Otherwise, the desired algebra remains isomorphic to $F(\sigma)$.

This strategy satisfies the requirement $T_e$. Indeed, if $A$ is isomorphic to $F(\sigma)$ then either $\phi$ is undefined somewhere or $\phi$ equals to some term $t$ almost everywhere. If $A$ is obtained by factorizing $F$ by $\eta(a, b)$ then $\phi_e$ now does not define an operation on $A$ because there is a tuple at which the value of $\phi$ is not well-defined. A point here is that strategy makes it sure that the function $\phi_e$ that acts improperly (that is $\phi_e$ does not look like a term now), after the action of the construction, does not respect the equality relation of the algebra $A$ being constructed. Clearly, the algebra is term algebraic.
Of course the strategy to satisfy just one requirement is not a difficult one. Problems arise when one tries to satisfy all of the requirement $T_e$. The reason is that the requirements $T_e$ interact with each other. Moreover, while satisfying the requirements $T_e$ the construction needs to guarantee that the algebra being constructed does not collapse into a finite algebra. Now we would like to make a few notes of a general character about the construction and present some technical details of the construction to the reader. For a current version of the full construction see [4].

Naturally, first of all we list all the requirements $T_e$, and say that $T_i$ has a higher priority than $T_j$ if $i < j$. Assume that the requirement $T_e$ is associated with computable partial function $\phi$. While constructing the desired algebra $A$, the construction acts depending on the behavior of $\phi$. At any stage there are elements $x$ in $A$ marked with $\Box_w$ and associated with $\phi$. The index $w$ indicates that $\phi$ is waiting to be defined on these marked $x$. If $\phi$ is not defined on some of these elements then the outcome of $\phi$ at a given stage is that of a waiting state. However, $\phi$ may be defined on the marked elements. Then $\phi$ may exhibit two different behaviors. One is that there is a term $p$ and $i$ such that $\phi(x_1, \ldots, x_i, \ldots, x_n) = p(x_i)$ on all marked elements $x$. The other is that such a term $p$ does not exist. In the former case, the outcome of $\phi$ is that of “$\phi$ looks like a term”, denoted by $t$. In the latter case, the construction should act, by using a version of the basic strategy, in such away that $\phi$ does not respect the equality relation of $A$. If there is a right environment to do this now then $T_e$ is satisfied (Informally, by a right environment we mean that the action of the construction to satisfy $T_e$ does not destroy the work of those requirements of higher priority). In this case the outcome is that of “$\phi$ is destroyed now”, we denote this outcome by $d$. If not, then the construction creates an environment so that $T_e$ will have a chance to destroy $\phi$ at a later stage. In this case the outcome is $w_d$. A point is that no actions performed due to the requirements of lower priority effect the environment created (However, the environment can be destroyed by satisfying requirements of higher priority in which case $T_e$ will have its chance to be satisfied). Therefore when $\phi$ is again defined on all elements of the created environment then the construction is able to act and destroy $\phi$ so that $\phi$ does respect the equality relation of the algebra being constructed.

The desired algebra $A$ is built by using a construction on a priority tree. Constructions on priority trees are often used in pure computability theory (see for example [11]). The alphabet of the outcomes of the priority tree is $O = \{t, w_t, w_d, d\}$. The symbol $t$ corresponds to the outcome saying that the function under consideration is termal, $w_t$ and $w_d$ correspond to waiting states. The outcome $w_t$ expresses the fact that the function under consideration seems to be a term, and the construction is waiting for the function to be defined on some elements of the algebra. The outcome $w_d$ expresses the fact that the function seems not to be a termal function; moreover, the construction has created an environment to make it sure that the function does not respect the equality relation of the algebra and is waiting for the function to be defined on certain elements of the algebra. The outcome $d$ corresponds to the fact that the function
under consideration does not respect the equality relation on $A$. The order on these outcomes is $t < w_t < w_d < d$. Consider the tree $T = \{t, w_t, w_d, d\}^{<\omega}$. To each node $\beta$ of the tree of length $e$ there corresponds a $\beta$-strategy that is devoted to satisfy requirement $T_e$. The $\beta$-strategy is an adaptation of the basic strategy but takes into account the outcomes $\beta(k)$, where $k < e$. In other words, the $\beta$-strategy acts by believing that the outcome of the $\lambda$-strategy ($\lambda$ is the root of the priority tree) to satisfy $T_0$ is $\beta(0)$, the outcome of the $\beta(0)$-strategy to satisfy $T_1$ is $\beta(1)$, the outcome of the $\beta(0)\beta(1)$-strategy to satisfy $R_2$ is $\beta(2)$, etc.

At the end of the construction one defines the true path on the tree by induction. Basically, it is the leftmost path $f$ on $T$ so that every node on the right of $f$ acts finitely often and each node in $f$ acts infinitely often. Then one proves, by induction on the lengths of the nodes on the true path, that the construction succeeds along the true path. The fact that the algebra constructed is term algebraic is guaranteed because of the following. Each requirement $T_e$ determines a level (that is, fixes a distance from the generator) that basically says that all the requirement of lower priority are not allowed to work below that level. In other words, once a distance for $T_e$ is fixed, say is $d$, then all requirements of lower priority than $T_e$ do not affect elements $x$ such that the distance from $x$ to the generator is not greater than $d$. These levels determined by strategies corresponding to $T_e$ may change finitely many times only. Doing this will guarantee that for each element $a$ of the algebra there will be finitely many ground terms whose values equal to $a$ in the algebra.

Now we combine constructions in Lemma 5 and Lemma 7 into one to build an algebra that has no quasiequational specification in any expansion. This is guaranteed by Lemma 6. Here is the result:

**Theorem 2.** There exists a finitely generated c.e. algebra such that every expansion of the algebra is not quasiequationally specified.

**Proof (outline).** Again, the desired algebra is built by using a diagonalization argument. On the one hand, we try to make the algebra computationally complete. On the other, we try ensure that the algebra constructed can not be quasiequationally specified in its own signature. For this we need to satisfy the following lists of requirements:

$$R_e : \quad A \text{ is not isomorphic to } F(C_e),$$

where $\{C_e\}_{e \in \omega}$ is an effective list of all quasiequational specifications in the signature $\langle c, f, g \rangle$, and

$$T_e : \quad \text{If } \phi_e \text{ is total and expands the algebra } A \text{ then } \phi_e \text{ is termal.}$$

where $\{\phi_e\}_{e \in \omega}$ is an effective list of all computable partial functions. Our construction will be a stagewise construction so that even stages are devoted to satisfy the requirements $R_e$, and odd stages $T_e$. Thus, the priority list is the following:

$$R_0, T_0, R_1, T_1, R_2, T_2, \ldots .$$
A point to note here is that the requirements $R_e$ are somewhat independent from the requirements $T_e$. This is because once we acted to satisfy $R_e$ it can be guaranteed that no requirement of lower priority can injure $R_e$. The requirements $T_e$ are satisfied in the manner similar to what is described Lemma \[7\] The requirements $R_e$ are satisfied in the manner as explained in Lemma \[5\] Naturally, the whole construction is put on a priority tree. At nodes of even length, say $2e$, the requirement $R_e$ is met; and at nodes of odd length length the requirement $T_e$ is met. Each node $\alpha$ of even length has exactly one immediate successor $\alpha s$, where $s$ stands for “$R_e$ is satisfied”, while each node $\alpha$ of odd length has four immediate successors $\alpha t$, $\alpha w_t$, $\alpha w_d$, and $\alpha d$ meaning of which are described in the lemma above. □

References

On the Complexity of Existential Pebble Games

Phokion G. Kolaitis and Jonathan Panttaja

Computer Science Department  
University of California, Santa Cruz  
{kolaitis,jpanttaj}@cs.ucsc.edu

Abstract. Existential $k$-pebble games, $k \geq 2$, are combinatorial games played between two players, called the Spoiler and the Duplicator, on two structures. These games were originally introduced in order to analyze the expressive power of Datalog and related infinitary logics with finitely many variables. More recently, however, it was realized that existential $k$-pebble games have tight connections with certain consistency properties which play an important role in identifying tractable classes of constraint satisfaction problems and in designing heuristic algorithms for solving such problems. Specifically, it has been shown that strong $k$-consistency can be established for an instance of constraint satisfaction if and only if the Duplicator has a winning strategy for the existential $k$-pebble game between two finite structures associated with the given instance of constraint satisfaction. In this paper, we pinpoint the computational complexity of determining the winner of the existential $k$-pebble game. The main result is that the following decision problem is EXPTIME-complete: given a positive integer $k$ and two finite structures $A$ and $B$, does the Duplicator win the existential $k$-pebble game on $A$ and $B$? Thus, all algorithms for determining whether strong $k$-consistency can be established (when $k$ is part of the input) are inherently exponential.

1 Introduction and Summary of Results

Combinatorial games are a basic tool for analyzing logical definability and delineating the expressive power of various logics. Typically, a logic $L$ is decomposed into a union $L = \bigcup_{k \geq 1} L(k)$ of fragments according to some syntactic parameter, such as quantifier rank, pattern of quantification, or number of variables. With each fragment $L(k)$, one then seeks to associate a natural combinatorial game $G(k)$ that captures $L(k)$-equivalence. Specifically, the desired game $G(k)$ is played between two players, called the Spoiler and the Duplicator, on two structures $A$ and $B$, and has the following property: the Duplicator has a winning strategy for $G(k)$ on $A$ and $B$ if and only if $A$ and $B$ satisfy the same $L(k)$-sentences. In the case of first-order logic FO, each such fragment is the set FO$(k)$ of all first-order sentences of quantifier rank at most $k$, and the game $G(k)$ is the $k$-move Ehrenfeucht-Fraïssé-game. Moreover, in the case of the infinitary logic $L^\omega_\omega$ with finitely many variables, each such fragment is the infinitary logic $L^k_\omega$ with $k$ variables, $k \geq 1$, and the corresponding game is the $k$-pebble game. As is well known, $k$-pebble games have turned out to be an indispensable tool in the study of logics with fixed-point operators in finite model theory (see [7] for a survey).
Each game $G(k)$ as above gives rise to the decision problem of determining the winner of this game: given two finite structures $A$ and $B$, does the Duplicator have a winning strategy for $G(k)$ on $A$ and $B$? It is easy to show that, for every $k \geq 1$, determining the winner of the $k$-move Ehrenfeucht-Fraïssé-game is in LOGSPACE (this is a consequence of the fact that each equivalence class of FO($k$)-equivalence is first-order definable). The state of affairs, however, is quite different for the $k$-pebble games. Indeed, Grohe [7] established that, for each $k \geq 2$, determining the winner of the $k$-pebble game is a P-complete problem, that is, complete for polynomial-time under logarithmic-space reductions. It is also natural to consider the decision problem that arises by taking the parameter $k$ as part of the input (in addition to the structures $A$ and $B$). Pezzoli [13] investigated the computational complexity of this problem for the Ehrenfeucht-Fraïssé-game and showed that it is PSPACE-complete. In other words, Pezzoli showed that the following problem is PSPACE-complete: given a positive integer $k$ and two finite structures $A$ and $B$, does the Duplicator have a winning strategy for the $k$-move Ehrenfeucht-Fraïssé-game on $A$ and $B$? Thus, when the number of moves is part of the input, an exponential jump occurs in determining the winner of the Ehrenfeucht-Fraïssé-game. It is conjectured that a similar exponential jump in complexity holds for the $k$-pebble game, when $k$ is part of the input. Specifically, the conjecture is that the following problem is EXPTIME-complete: given a positive integer $k$ and two finite structures $A$ and $B$, does the Duplicator have a winning strategy for the $k$-pebble on $A$ and $B$? To date, this conjecture remains unsettled.

In this paper we investigate the computational complexity of the decision problems associated with the class of existential $k$-pebble games (or, in short, ($\exists, k$)-pebble games), which are an asymmetric variant of the $k$-pebble games. These games were introduced in [11] as a tool for studying the expressive power of Datalog and of the existential positive infinitary logic $\exists L^\omega_\omega$ with finitely many variables. More precisely, $\exists L^\omega_\omega$ is the collection of all $L^\omega_\omega$-formulas containing all atomic formulas and closed under existential quantification, infinitary conjunction $\land$, and infinitary disjunction $\lor$. Clearly, $\exists L^\omega_\omega = \bigcup_{k \geq 1} \exists L^k_\omega$, where $\exists L^k_\omega$ is the collection of all $\exists L^\omega_\omega$-formulas with at most $k$ distinct variables. The differences between the ($\exists, k$)-pebble game and the $k$-pebble game played on two structures $A$ and $B$ are that in the ($\exists, k$)-pebble game: (1) the Spoiler always plays on $A$; and (2) the Duplicator strives to maintain a partial homomorphism, instead of a partial isomorphism. The main result of this paper is that determining the winner of the ($\exists, k$)-pebble game, when $k$ is part of the input, is an EXPTIME-complete problem. In contrast, for each fixed $k \geq 2$, determining the winner of ($\exists, k$)-pebble game turns out to be a P-complete problem. Before commenting on the technique used to establish the main result, we discuss the motivation for investigating this problem and the implications of our main result.

Although ($\exists, k$)-pebble games were originally used in database theory and finite model theory, in recent years they turned out to have applications to the study of constraint satisfaction. Numerous problems in several different areas of artificial intelligence and computer science can be modeled as constraint satisfaction problems [4]. In full generality, an instance of the CONSTRAINT SATISFACTION PROBLEM consists of a set of variables, a set of possible values, and a set of constraints on tuples of variables; the question is to determine whether there is an assignment of values to the variables that
satisfies the given constraints. Alternatively, as first pointed out by Feder and Vardi [6], the Constraint Satisfaction Problem can be identified with the Homomorphism Problem: given two relational structures $A$ and $B$, is there a homomorphism $h$ from $A$ to $B$? Intuitively, the structure $A$ represents the variables and the tuples of variables that participate in constraints, the structure $B$ represents the domain of values and the tuples of values that the constrained tuples of variables are allowed to take, and the homomorphisms from $A$ to $B$ are precisely the assignments of values to variables that satisfy the constraints. The Constraint Satisfaction Problem is NP-complete, since it contains Boolean Satisfiability, Colorability, Clique, and many other prominent NP-complete problems as special cases. For this reason, there has been an extensive pursuit of both tractable cases of the Constraint Satisfaction Problem and heuristic algorithms for this problem. In this pursuit, a particularly productive approach has been the introduction and systematic use of various consistency concepts that make explicit additional constraints implied by the original constraints. The strong $k$-consistency property is the most important one among them; intuitively, this property holds when every partial solution on fewer than $k$ variables can be extended to a solution on $k$ variables [5]. Closely related to this is the process of “establishing strong $k$-consistency”, which is the question of whether additional constraints can be added to a given instance of the Constraint Satisfaction Problem in such a way that the resulting instance is strongly $k$-consistent and has the same space of solutions as the original one. Algorithms for establishing strong $k$-consistency play a key role both in identifying tractable cases of constraint satisfaction and in designing heuristics for this class of problems [2,5].

In [12], a tight connection was shown to exist between strong $k$-consistency properties and $(\exists, k)$-pebble games. Specifically, it turns out that strong $k$-consistency can be established for a given instance of the Constraint Satisfaction Problem if and only if the Duplicator has a winning strategy for the $(\exists, k)$-pebble game on the structures $A$ and $B$ forming the instance of the Homomorphism Problem that is equivalent to the given instance of the Constraint Satisfaction Problem. This connection was fruitfully exploited in [3], where it was shown that the tractability of certain important cases of constraint satisfaction follows from the fact that the existence of a solution is equivalent to whether the Duplicator can win the $(\exists, k)$-pebble game for some fixed $k$. Note that, for every fixed $k$, there is a polynomial-time algorithm to determine whether, given two finite structures $A$ and $B$, the Duplicator has a winning strategy for the $(\exists, k)$-pebble game on $A$ and $B$ (this had been already observed in [11]). Nonetheless, since many heuristics for constraint satisfaction require testing whether strong $k$-consistency can be established for arbitrarily large $k$’s, it is important to identify the inherent computational complexity of determining the winner in the $(\exists, k)$-pebble game, when $k$ is part of the input. It is not hard to verify that this problem is solvable in time $O(n^{2k})$, that is, in time exponential in $k$. Moreover, it was conjectured in [12] that a matching lower bound exists, which means that the following problem is EXPTIME-complete: given a positive integer $k$ and two finite structures $A$ and $B$, does the Duplicator have a winning strategy for the $(\exists, k)$-pebble on $A$ and $B$?

In this paper, we prove this conjecture by showing that another pebble game, which was known to be EXPTIME-complete, has a polynomial-time reduction to the $(\exists, k)$-pebble game. Specifically, Kasai, Adachi, and Iwata [8] introduced a pebble game, which
On the Complexity of Existential Pebble Games

we will call the KAI game, and showed that it is EXPTIME-complete via a direct reduction from polynomial-space alternating Turing machines (recall that $\text{APSPACE} = \text{EXPTIME}$ [1]). Our reduction of the KAI game to the $(\exists, k)$-pebble game is quite involved and requires the construction of elaborate combinatorial gadgets. In describing this reduction and establishing its correctness, we will adopt the setup and terminology used by Grohe [7] in showing that, for every $k \geq 2$, the $k$-pebble game is P-complete. Some of the basic gadgets in our reduction already occurred in Grohe’s reduction. However, we will also need to explicitly construct other much more sophisticated gadgets that will serve as “switches” with special properties in the reduction. We note that Grohe also used highly sophisticated gadgets that were graphs with certain homogeneity properties. Grohe’s gadgets, however, have size exponential in $k$ and, hence, they cannot be used in a polynomial-time reduction when $k$ is part of the input (this is also the reason why Grohe’s reduction does not show that the $k$-pebble game is EXPTIME-complete, when $k$ is part of the input). An immediate consequence of our main result is that determining whether strong $k$-consistency can be established, when $k$ is part of the input, is an EXPTIME-complete problem and, thus, inherently exponential. Moreover, this explains why all known algorithms for establishing strong $k$-consistency are exponential in $k$ (even ones considered to be “optimal”, see [2]).

We also address the computational complexity of determining who wins the $(\exists, k)$-pebble game, when $k$ is a fixed positive integer. Kasif [10] showed that determining whether strong 2-consistency can be established is a P-complete problem. From this and the aforementioned connection between strong $k$-consistency and the $(\exists, k)$-pebble game [12], it follows that determining who wins the $(\exists, 2)$-pebble game is a P-complete problem. Here we give a direct proof to the effect that, for every fixed $k \geq 2$, determining who wins the $(\exists, k)$-pebble game is a P-complete problem. This is done via a reduction from the MONOTONE CIRCUIT VALUE PROBLEM, which we present first as a warm-up to the reduction of the KAI game to the $(\exists, k)$-game, when $k$ is part of the input. Due to space limitations, here we present only outlines of these reductions; complete proofs can be found in the full version of the paper, which is available at http://www.cs.ucsc.edu/~kolaitis/papers/.

2 The Existential $k$-Pebble Game

Let $A$ and $B$ be two relational structures over the same vocabulary. A homomorphism $h$ from $A$ to $B$ is a mapping $h : A \rightarrow B$ from the universe $A$ of $A$ to the universe $B$ of $B$ such that, for every relation $R^A$ of $A$ and every tuple $(a_1, \ldots, a_m) \in R^A$, we have that $(h(a_1), \ldots, h(a_m)) \in R^B$. A partial homomorphism from $A$ to $B$ is a homomorphism from a substructure of $A$ to a substructure of $B$.

Let $k \geq 2$ be a positive integer. The existential $k$-pebble game (or, in short, the $(\exists, k)$-pebble game) is played between two players, the Spoiler and the Duplicator, on two relational structures $A$ and $B$ according to the following rules: each player has $k$ pebbles labeled $1, \ldots, k$; on the $i$-th move of a round of the game, $1 \leq i \leq k$, the Spoiler places a pebble on an element $a_i$ of $A$, and the Duplicator responds by placing the pebble with the same label on an element $b_i$ of $B$. The Spoiler wins the game at the end of that round, if the correspondence $a_i \mapsto b_i$, $1 \leq i \leq k$, is not a homomorphism between the substructures of $A$ and $B$ with universes $\{a_1, \ldots, a_k\}$ and $\{b_1, \ldots, b_k\}$, respectively.
Otherwise, the Spoiler removes one or more pebbles, and a new round of the game begins. The Duplicator wins the \((\exists, k)\)-pebble game if he has a \textit{winning strategy}, that is to say, a systematic way that allows him to sustain playing “forever”, so that the Spoiler can never win a round of the game.

To illustrate this game (and its asymmetric character), let \(K_m\) be the \(m\)-clique, that is, the complete undirected graph with \(m\) nodes. For every \(k \geq 2\), the Duplicator wins the \((\exists, k)\)-pebble game on \(K_k\) and \(K_{k+1}\), but the Spoiler wins the \((\exists, k+1)\)-pebble game on \(K_{k+1}\) and \(K_k\). As another example, let \(L_s\) be the \(s\)-element linear order, \(s \geq 2\).

Note that the above description of a winning strategy for the Duplicator in the \((\exists, k)\)-pebble game is rather informal. The concept of a winning strategy can be made precise, however, in terms of families of partial homomorphisms with appropriate properties. Specifically, a \textit{winning strategy for the Duplicator in the existential \(k\)-pebble game on \(A\) and \(B\)} is a nonempty family \(F\) of partial homomorphisms from \(A\) to \(B\) such that:

1. For every \(f \in F\), the domain \(\text{dom}(f)\) of \(f\) has at most \(k\) elements.
2. \(F\) is \textit{closed under subfunctions}, which means that if \(g \in F\) and \(f \subseteq g\), then \(f \in F\).
3. \(F\) has the \(k\)-\textit{forth property}, which means that for every \(f \in F\) with \(|\text{dom}(f)| < k\) and every \(a \in A\) on which \(f\) is undefined, there is a \(g \in F\) that extends \(f\) and is defined on \(a\).

Intuitively, the second condition provides the Duplicator with a “good” move when the Spoiler removes a pebble from an element of \(A\), while the third condition provides the Duplicator with a “good” move when the Spoiler places a pebble on an element of \(A\).

3 The \((\exists, k)\)-Pebble Game Is P-Complete

In this section, we show that, for every \(k \geq 2\), determining the winner of the \((\exists, k)\)-pebble game is a P-complete problem. We do this by constructing a reduction from the \textsc{Monotone Circuit Value Problem} (MCV) in the style of Grohe [7], but with different gadgets. In this reduction, the structures will be undirected graphs with ten unary predicates, called \textit{colors}. So, we actually prove that, for every \(k \geq 2\), the \((\exists, k)\)-pebble game restricted to such structures is P-complete.

The following concepts and terminology come from Grohe [7].

1. In an undirected graph with colors, a \textit{distinguished pair} of vertices is a pair of vertices that are of the same color, and that color is not used for any other vertex in the graph.
2. A \textit{position} of the \((\exists, k)\)-pebble game on \(A\) and \(B\), is a set \(P\) of ordered pairs such that \(P \subseteq A \times B\) and \(|P| \leq k\). Often, we will omit the ordered pair notation and use the shorthand \(ab \in P\) to mean that \((a, b) \in P\).
3. A \textit{strategy} for the Spoiler is simply a mapping from positions to moves which tells the Spoiler how to play given the current position.
4. We say that the Spoiler \textit{can reach a position} \(P'\) from another position \(P\) of the \((\exists, k)\)-pebble game on \(A\) and \(B\) if the Spoiler has a strategy for the \((\exists, k)\)-pebble...
On the Complexity of Existential Pebble Games

Fig. 1. H Gadget based on the one from [7]. $H_S$ is on the left and $H_D$ is on the right.

Fig. 2. I Gadget based on the one from [7]. $I_S$ is on the left and $I_D$ is on the right.

game on $A$ and $B$ such that, starting from position $P$, either he wins the game or after a number of moves the game is in a position $P''$ such that $P' \subseteq P''$.

This concept will be used to combine strategies of the Spoiler on different gadgets in order to construct strategies for the combined game.

5. We say that the Duplicator can avoid a position $P'$ from another position $P$ of the $(\exists, k)$-pebble game on $A$ and $B$ if the Duplicator has a winning strategy for the $(\exists, k)$-pebble game on $A$ and $B$ such that starting from position $P$, position $P'$ never occurs.

For each gadget used in the reduction there will be two pieces, one for the Spoiler’s structure and one for the Duplicator’s structure. For gadget $X$, we call the Spoiler’s side $X_S$, the Duplicator’s side $X_D$, and the pair $(X_S, X_D)$ simply $X$.

3.1 The Gadgets H and I

The graphs $H_D$ and $I_D$, which are both based on gadgets from [7], are going to be used for and nodes and or nodes respectively. $H_D$, as seen in Figure 1, consists of six vertices $h, h', i, i', j, j'$. These six vertices form three distinguished pairs, $(h, h')$, $(i, i')$, and $(j, j')$. There are edges from $h$ to $i$, and $h$ to $j$, and edges from $h'$ to $i'$ and $h'$ to $j'$. This graph has only one non-identity automorphism, which we will call $sw$, that maps any vertex $a$ to $a'$ and any vertex $a'$ to $a$. $H_S$ is simply the subgraph of $H_D$ determined by $h, i, j$. Starting from position $hh'$, the Spoiler can reach both $ii'$ and $jj'$ in the $(\exists, k)$-pebble game on $(H_S, H_D)$.

$I_D$, seen in figure 2 has ten vertices. It contains the three distinguished pairs $(h, h')$, $(i, i')$, and $(j, j')$, plus four additional nodes which we will name by their connections
to the other vertices. These nodes are $hij$, $h'ij'$, $h'i'j$, and $hi'j'$. This graph has three non-identity automorphisms, which we will refer to as $fix_i$, $fix_j$, and $fix_h$. These automorphism fix $i$, $j$, and $h$ respectively, while switching the other two. By playing according to these automorphisms, the Duplicator can avoid either $ii'$ or $jj'$ from $hh'$ but not both in the $(\exists, k)$-pebble game.

### 3.2 Single Input One-Way Switches

The Single Input One-Way Switches are used to restrict the ways in which the Spoiler can win the game. The basic intuition is that the Spoiler can only make progress in one particular direction; moreover, to do so he must use all of his pebbles.

This lemma is similar to Lemma 14 from [7], adapted to the $(\exists, k)$-pebble game.

**Lemma 1.** For every $k \geq 2$ there exists a pair of graphs $O_S^k$ and $O_D^k$, with $O_S^k \subset O_D^k$, $\{x, x', y, y'\} \subset V(O_D^k)$, $xx'$ and $yy'$ distinguished pairs of vertices, and $\{x, y\} \subset V(O_S^k)$, such that:

1. The Spoiler can reach $yy'$ from $xx'$ in the $(\exists, k)$-pebble game on $(O_S^k, O_D^k)$.
2. There exist two disjoint sets of positions of the $(\exists, k)$-pebble game on $(O_S^k, O_D^k)$, called Pretrapped and Trapped positions such that:
   (a) Pretrapped and Trapped positions are partial homomorphisms
   (b) The Duplicator can avoid positions that are not Trapped and not Pretrapped from Pretrapped positions
   (c) The Duplicator can avoid positions that are not Trapped from Trapped positions
   (d) The position $\{xx'\}$ is Pretrapped
   (e) If $P$ is Pretrapped and $|P| < k$, then $P \cup \{yy\}$ is Pretrapped
   (f) The positions $\{yy\}$ and $\{yy'\}$ are Trapped
   (g) If $P$ is Trapped and $|P| < k$, then $P \cup \{xx\}$ is Trapped

We call $(O_S^k, O_D^k)$ the Single Input One-Way Switch.

**Corollary 1.** The Spoiler can reach $\{(y, y')\}$ from $\{(x, x')\}$ in the $(\exists, k)$-pebble game on the Single Input One-Way Switch, but not in the $(\exists, k - 1)$-pebble game on the Single Input One-Way Switch.
Proof. The first part of the Corollary is simply a restatement of condition 1 from Lemma \[1\] Assume for the sake of contradiction that the Spoiler can reach \{\((x, x')\)\} from \{\((y, y')\)\} in the \((\exists, k − 1)\)-pebble game on the Single Input One-Way Switch. Then, from the position \{\((x, x'), (y, y)\)\} of the \((\exists, k)\)-pebble game, the Spoiler could reach the position \{\((y, y'), (y, y)\)\} by ignoring the pebbles on \((y, y)\) and playing the \((\exists, k − 1)\)-pebble game. The position \{\((x, x'), (y, y)\)\} is Pretrapped, but \{\((y, y'), (y, y)\)\} is neither Pretrapped nor Trapped. This is a contradiction because the Duplicator can avoid such positions from Pretrapped positions.

Because of Corollary \[1\] the Spoiler has to use all of his pebbles in order to make progress. The “One-Way” aspect of the Switch lies in the fact that \{\((y, y')\)\} is Trapped, and \{\((x, x')\)\} is not. This means that the Duplicator can avoid \{\((x, x')\)\} from \{\((y, y')\)\}.

3.3 Twisted Switches

The Twisted Switch consists of an \(H\) gadget, an \(I\) gadget, and two Single Input One-Way Switches in parallel. We use a Twisted Switch in the reduction to initialize the game. The construction of the Twisted Switch is the same as that in Grohe \[7\], except that we substitute a One-Way Switch in place of what Grohe calls a Threshold Switch.

The following Lemma introduces a set of positions of the Single Input One-Way Switch, called Switched positions. Within the Twisted Switch, the Duplicator uses Switched positions instead of Trapped and Pretrapped positions on the Single Input One-Way Switch.

Lemma 2. There is a set of positions of the \((\exists, k)\)-pebble game on \(O^k\), called Switched positions, such that

1. \(\{xx', yy\}\) and \(\{xx, yy'\}\) are Switched.
2. If \(P\) is Switched, then either \(P \cup \{xx', yy\}\) or \(P \cup \{xx, yy'\}\) is Switched.
3. The Duplicator can avoid positions that are not Switched from Switched positions in the \((\exists, k)\)-pebble game.

Lemma 3. On the Twisted Switch, the Spoiler can reach \(yy'\) from \(xx'\) in the \((\exists, k)\)-pebble game, and there exists a set of positions of the \((\exists, k)\)-pebble game called Twisted positions such that

1. The Duplicator can avoid non-Twisted positions from Twisted positions
2. \(\{yy\}\), and \(\{yy'\}\), are Twisted positions.
3. If \(P\) is a Twisted position, then \(P \cup \{xx'\}\) is a Twisted position.

The Twisted Switch will be used to initialize the game. In order to do this, \(x \in T_S\) and \(x' \in T_D\) are colored the same color, while \(x \in T_D\) is colored a different color. Thus, if the Spoiler plays on \(x\), then the Duplicator must play on \(x'\). From here the Spoiler can play to \{\((y, y')\)\}. The Twisted positions allow the Duplicator to avoid \{\((x, x)\)\}, which is a losing position.
3.4 Reduction from MCV to $(\exists, k)$-Pebble Game

Theorem 1. For every fixed $k \geq 2$, determining the winner of the $(\exists, k)$-pebble game is $P$-Complete.

Proof. (Outline) Given a monotone circuit $C$, an assignment of values to the input nodes, and a node $v$, we construct a graph $C_D$. For each node $a$ in the circuit, there are three choices.

1. If $a$ is an input node, then $C_D$ contains vertices $a$ and $a'$. If the value of $a$ in $C$ is false, then we color $a'$ black.

2. If $a$ is an and node with parents $b$ and $c$, then $C_D$ contains nodes $a$ and $a'$, a copy of $H_D$, called $H_a$ with $h$ and $h'$ identified with $a$ and $a'$, as well as two copies of $O^k_D$, one of which has $x$ and $x'$ connected to $i$ and $i'$, and $y$ and $y'$ connected to $b$ and $b'$ called $O_{ab}$. The other copy connects $j$ and $j'$ to $c$ and $c'$ in a similar manner and is called $O_{ac}$.

3. If $a$ is an or node with parents $b$ and $c$, then $C_D$ contains nodes $a$ and $a'$, a copy of $I_D$ called $I_a$ with $h$ and $h'$ identified with $a$ and $a'$, as well as two copies of $O^k_D$, one of which has $x$ and $x'$ connected to $i$ and $i'$, and $y$ and $y'$ connected to $b$ and $b'$ called $O_{ab}$. The other copy connects $j$ and $j'$ to $c$ and $c'$ in a similar manner and is called $O_{ac}$.

In any case, $a$ is colored white.

The construction of $C_S$ is similar, except that we do not add $a'$ and we use the Spoiler version of each gadget and switch instead of the Duplicator version.

Also, there is a Twisted Switch $T$, such that $x$ is colored a fresh color in $C_S$, and $x'$ is colored the same color in $C_D$. Also, in $C_S$, $y$ in $T$ is connected to $v$, while in $C_D$, $y$ is connected to $v$ and $y'$ is connected to $v'$.

The Spoiler plays from the top of the Twisted Switch through the graph attempting to reach a false input. Because of the properties of the $H$ and $I$ gadgets and the Single Input One-Way Switch, the Spoiler can do this if the value of the goal node is false. If the value of the goal node is true, then the Duplicator can play along indefinitely. He
does this by choosing a path down the simulated circuit which leads to a true input. If
the Spoiler attempts to depart from the intended mode of play, the Duplicator can use
the properties of Trapped and Pretrapped strategies to arrive at a partial homomorphism
that is a subset of the identity.

4 The \((\exists, k)\)-Pebble Game Is EXPTIME-Complete

Kasai, Adachi and Iwata [8] showed that the following pebble game, which (to avoid
confusion in the terminology) we will call the KAI game, is EXPTIME-complete. The
game is played between two players, called Player I and Player II. An instance of the
KAI game is a quadruple \((X, S, R, t)\), where \(X\) is a set of nodes, \(R \subseteq X^3\) is a set of
rules, \(S \subseteq X\) is the initial position of the game, and \(t \in X\) is the goal. There are as
many pebbles as nodes in the initial position \(S\); at the start of the game, each node in
\(S\) has a pebble on it. The two players take turns and in each turn they slide a pebble as
follows: if \((x, y, z) \in R\) is a rule, then the current player may slide a pebble from \(x\) to \(z\),
if there are pebbles on \(x\) and \(y\), but not \(z\). The first player to place a pebble on the goal
\(t\) wins. The problem is, given \((X, R, S, t)\), does Player I have a winning strategy?

We will show that the \((\exists, k)\)-pebble game is EXPTIME complete by exhibiting a
reduction from the KAI game. The reduction proceeds by constructing an instance of
the \((\exists, k)\)-pebble game such that the Spoiler and the Duplicator simulate the instance of
the KAI game. In particular, the Spoiler can only win by simulating a winning strategy
for Player I in the KAI game. If there is no winning strategy, then the Spoiler does not
gain any advantage by not simulating the KAI game.

In order to perform this simulation, we use a Twisted Switch (Section 3.5) to initialize
the game, and new switches to allow Player I and Player II to choose rules, to apply the
rules, and to force the Spoiler to simulate the game.

4.1 The Gadgets \(H^m\) and \(I^m\)

These gadgets allow the Spoiler and Duplicator to choose rules of the KAI game to use.
In both the \(H^m\) and \(I^m\) gadgets, the nodes \(y^i, y^i_0, y^i_1\) are colored the same color, but the
color of \(y^i\) is different that the color of \(y^j\) for \(i \neq j\).

\[\text{Fig. 5. } H^3 \text{ Gadget.}\]

**Lemma 4.** For every \(k \geq 2\), in the \((\exists, k)\)-pebble game on \((H^m_S, H^m_D)\), from a position
\(\{xx_j\}, j \in \{0, 1\}\), the Spoiler can reach \(\{y^i y^j\}\) for any \(i\).
Lemma 5. For every $k \geq 2$, in the $(\exists, k)$-pebble game on $I^m_S, I^m_D$, from a position $\{x_i j\}, j \in \{0,1\}$, the Duplicator can choose any $1 \leq i \leq m$, and avoid $\{y^l y^j\}$ for $l \neq i$.

4.2 Multiple Input One-Way Switches for the $(\exists, k)$-Pebble Game

The idea of the Multiple Input One-Way Switch is to restrict the Spoiler’s potential winning strategies. We simulate each node $x^i$ in the KAI game by using three nodes in the Duplicator’s graph, $x^i_0, x^i_1, x^i$. These correspond to not having pebble on $x^i$ in the simulated game, having a pebble on $x^i$ in the simulated game, and no information about $x^i$, respectively. In the Multiple Input One-Way Switch, the Spoiler can only make progress if he has information about each node in the simulated game. Also, if the Spoiler attempts to play backwards through the Switch, he will end up with no information about any nodes in the simulated game.

Fig. 6. $I^3$ Gadget.

Fig. 7. A subgraph of $M^4$. 

Fig. 7. A subgraph of $M^4$. 

Lemma 6. For every \( k \geq 2 \), there exists a pair of graphs \( M^k_S \) and \( M^k_D \) such that
\[
\{x^1, x^1_0, x^1_1, \ldots, x^{k-1}, x^{k-1}_0, x^{k-1}_1, y^1, y^1_0, y^1_1, \ldots, y^{k-1}, y^{k-1}_0, y^{k-1}_1\} \subset V(M^k_D),
\{x^1, \ldots, x^{k-1}, y^1, \ldots, y^{k-1}\} \subset V(M^k_S)
\]
and the following properties hold:
1. From a position \( \{x^i x^i_{j_i} | 1 \leq i \leq k - 1, j_i \in \{0, 1\}\} \), the Spoiler can reach the position \( \{y^i y^i_{j_i} | 1 \leq i \leq k - 1, j_i \in \{0, 1\}\} \) in the \((\exists, k)\)-pebble game on \( M^k_S \) and \( M^k_D \).
2. There exist two disjoint sets of positions of the \((\exists, k)\)-pebble game on \((M^k_S, M^k_D)\), called Pretrapped and Trapped positions such that:
   (a) Pretrapped and Trapped positions are partial homomorphisms
   (b) The Duplicator can avoid positions that are not Pretrapped and not Trapped from Pretrapped positions
   (c) The Duplicator can avoid positions that are not Trapped from Trapped positions
   (d) From any position \( P = \{x^i a | 1 \leq i \leq k - 1\} \) where \( |\{x^i x^i_j \in P | j \in \{0, 1\}\}| < k - 1 \), the Duplicator can avoid \( y^i y^i_{j_i} \) for all \( 1 \leq i \leq k - 1, j_i \in \{0, 1\} \).
   (e) All positions that are subsets of positions of the form \( \{x^i x^i_j | 1 \leq i \leq k - 1, j_i \in \{0, 1\}\} \), are PreTrapped.
   (f) If \( P \) is Pretrapped and \( |P| < k \), then \( P \cup \{y^i y^i_i\} \) is Pretrapped for all \( i \)
   (g) Any position in which all of the Spoiler’s pebbles are on nodes \( y^i \), is Trapped.
   (h) If \( P \) is Trapped and \( |P| < k \), then \( P \cup \{x^i x^i\} \) is Trapped for all \( i \)

Moreover, \( |V(M^k_S)| \leq O(k^2) \) and \( |V(M^k_D)| \leq O(k^2) \).

4.3 The Rule Gadget

The Rule gadgets are used to simulate a move of the KAI game. One rule gadget causes the Spoiler to lose if the rule gadget corresponds to a rule that cannot be applied, and another causes the Duplicator to lose if the rule cannot be applied.

Lemma 7. If the rule gadget \( RS^n \) does not correspond to a legal rule, that is, if one of \( xx_1, yy_1, zz_0 \) is not in \( P \), then the Duplicator can avoid \( z^i z^i_1 \) in the \((\exists, k)\)-pebble game on \( RS^n \).

Fig. 8. The Rule Gadget \( RS^n \) that penalizes the Spoiler for choosing a bad rule.
Lemma 8. If the rule gadget $RD^n$ does not correspond to a legal rule, that is, if one of $xx_1$, $yy_1$, $zz_0$ is not in $P$, then the Spoiler can play to $z'z'_0$, which causes the Duplicator to lose.

Fig. 9. The Rule Gadget $RD^n$ that penalizes the Duplicator for choosing a bad rule.

4.4 Winning the Game

For each gadget, we define its boundary to be the set of nodes which are used to connect that gadget to other gadgets. For $M^k$, $H^m$, and $I^m$, the $x$ and $y$ nodes form the boundary. For $RS^n$ and $RD^n$, the boundary consists of all the nodes except for the middle nodes. For the Twisted Switch, the boundary consists of $y$ in $T_S$ and $y, y'$ in $T_D$. In the Twisted Switch, $x$ and $x'$ are never connected to any other gadgets.

Lemma 9. For each gadget other than the Twisted Switch, starting from a position that is a subset of the identity on the boundary, the Duplicator can avoid any position that is not a subset of the identity on the boundary.

By combining this lemma with the properties of the Multiple Input One-Way Switch, we obtain a sufficient condition for the Duplicator to win the $(\exists, k)$-pebble game.

4.5 Reduction from KAI Game to $(\exists, k)$-Pebble Game

Theorem 2. Determining the winner of the $(\exists, k)$-pebble game with $k$ part of the input is EXPTIME-Complete.

Proof. (Outline) We will give a polynomial-time reduction from the KAI Game to the $(\exists, k)$-pebble game. Given an instance $(X, S, R, t)$ of the KAI game, we form an instance of the $(\exists, k)$-pebble game as follows.

The Duplicator’s graph and the Spoiler’s graph each have two sides. One side represents Player I’s turn in the KAI game, while the other side represents Player II’s turn.

First, we build Player I’s side of the graph. For each $x^i \in X$, we form three nodes in $D$, called $xs^i_0, xs^i_1, xs^i_1$. These three nodes correspond to specific information about the simulated KAI game. If there is a pebble on $xs^i_1$, then there is a pebble on $x$ in the KAI game, and $xs^i_0$ corresponds to no pebble on $x$. A pebble on $xs^i$ in the Duplicator’s
graph means that the Spoiler has made a mistake. For each $(x, y, z) \in R$, construct a rule gadget to penalize the Spoiler, connected to each $xs^i, xs^0_0, xs^1_0$ by $H^m$ gadgets. The other end is then connected to a Multiple Input One-Way Switch.

On the other half of the graph, there are another set of nodes $xd^i, xd^0_0, xd^1_1$. Connecting the $xd$ nodes to Duplicator punishing rule gadgets are a set of $I^m$ gadgets where $m = |R|$. Given any two of these $I^m$ gadgets $A$ and $B$, there is an edge between $A.y^i_j$ and $B.y^i_k$ for all $i, j, k$, and an edge between $A.y^i$ and $B.y^i$ for all $i$. We then connect a Multiple Input One-Way Switch to each of the rule gadgets in the obvious way. The outputs of these are connected back around to the first set of state nodes.

We use a Twisted Switch to set up the initial positions. If $x^i \in S$, then there is an edge from $y'$ of the Twisted Switch to $xs^i_1$ otherwise there is an edge from $y'$ to $xs^i_0$. There is an edge from $y$ to $xs^i$ for every $i$. We then color $x'$ a unique color. In addition we give $t_1$ a unique color so that if the Spoiler can reach $tt_1$, then the Duplicator loses.

The Spoiler’s graph is constructed in a similar way. The Spoiler gets $xs^i$ and $xd^i$ for every $x^i \in X$. Also, for each rule and each side, there is a corresponding Multiple Input One-Way Switch followed by a rule gadget. For the Twisted Switch, we color $x$ the same color that we colored $x'$ in the Duplicator’s graph.

The two players now simulate the KAI game by playing the $(\exists, k)$-pebble game on these structures. The Spoiler initializes the game by playing on the Twisted Switch. From here, the Spoiler uses the $H^m$ gadgets to choose a rule, then plays through the $RS^n$
gadget and the Multiple Input One-Way Switch to simulate the move of Player I in the KAI game. Then, the Spoiler continues to play through Player II’s side of the structures, allowing the Duplicator to choose a rule to simulate, and applying that rule. If Player I has a winning strategy for the KAI game, then this simulation process will eventually simulate Player I placing a pebble on \( t \). Because of the coloring of the structures, this causes the Spoiler to win the \((\exists, k)\)-pebble game. The more difficult step is showing that if Player I does not have a winning strategy, then the Duplicator wins the \((\exists, k)\)-pebble game. If the Spoiler plays nice and simulates the KAI game, then the Duplicator can avoid \( t \), by playing a smart strategy for the KAI game. If the Spoiler does not play nice, and departs from the simulation, then, because of the properties of the simulation, the Duplicator can play along indefinitely.

As pointed out in Section 3, the structures used in the reduction of MCV to the \((\exists, k)\)-pebble game with fixed \( k \) were undirected graphs with a fixed number (ten) of colors. In contrast, the structures used in the preceding reduction of the KAI game to the \((\exists, k)\)-game with \( k \) part of the input are undirected graphs with a number of colors that is linear in the size of the input. It is an interesting technical problem to exhibit a reduction of the KAI game to the \((\exists, k)\)-game with \( k \) part of the input in which the structures are undirected graphs with a fixed number of colors.

5 Concluding Remarks

Although in this paper we focused on the \((\exists, k)\)-pebble game because of its connections to constraint satisfaction, in database theory there is also interest in the one-to-one \((\exists, k)\)-pebble game, which is the variant of the \((\exists, k)\)-pebble game in which the Duplicator strives to maintain one-to-one homomorphisms (see [11]). A perusal of the two reductions presented here reveals that in both these reductions the structures constructed have the property that the Duplicator wins the \((\exists, k)\)-pebble game if and only if the Duplicator wins the one-to-one \((\exists, k)\)-pebble game. Consequently, determining the winner in the one-to-one \((\exists, k)\)-pebble is P-complete for every fixed \( k \geq 2 \), and is EXPTIME-complete when \( k \) is part of the input.

Several problems remain open in this area. Kasai and Iwata [9] proved that the number of pebbles used in the KAI game gives rise to a strict hierarchy on the time complexity of that game. Thus, it is natural to ask whether a similar strict hierarchy result can be proved for the \((\exists, k)\)-pebble, for fixed \( k \). This amounts to showing that, for each fixed \( k \geq 2 \), determining the winner of the \((\exists, k)\)-pebble game is not solvable in time \( O(n^s) \) for any \( s < 2k \). Finally, it remains an open problem to establish that the (two-sided) \( k \)-pebble game with \( k \) part of the input is an EXPTIME-complete problem.

References

Computational Aspects of $\Sigma$-Definability over the Real Numbers without the Equality Test

Margarita Korovina

BRICS**, Department of Computer Science, University of Aarhus,
Ny Munkegade, DK-8000 Aarhus C, Denmark,
A. P. Ershov Institute of Informatics Systems,
Lavrent’ev ave., 6, 630090, Novosibirsk, Russia
korovina@brics.dk
http://www.brics.dk/~korovina

Abstract. In this paper we study the expressive power and algorithmic properties of the language of $\Sigma$-formulas intended to represent computability over the real numbers. In order to adequately represent computability, we extend the reals by the structure of hereditarily finite sets. In this setting it is crucial to consider the real numbers without equality since the equality test is undecidable over the reals. We prove Engeler’s Lemma for $\Sigma$-definability over the reals without the equality test which relates $\Sigma$-definability with definability in the constructive infinitary language $L_{\omega_1\omega}$. Thus, a relation over the real numbers is $\Sigma$-definable if and only if it is definable by a disjunction of a recursively enumerable set of quantifier free formulas. This result reveals computational aspects of $\Sigma$-definability and also gives topological characterisation of $\Sigma$-definable relations over the reals without the equality test.

1 Introduction

We start with an informal introduction to problems and solutions considered in this paper. It is well-known that the classical theory of computation, which works with discrete structures, is not suitable for formalisation of computations that operate on real-valued data. Most computational problems in physics and engineering are of this type, e.g. problems relevant to foundation of dynamical and hybrid systems. Since computational processes are discrete in their nature and objects we consider are continuous, formalisation of computability of such objects is already a challenging research problem. Consequently, computability over the reals has become a subject of the great interest and study in Computer Science [4,11,15,18,21,24].

* This research was partially supported by the Danish Natural Science Research Council, Grant no. 21-02-0474, RFFI-DFG Grant no. 01-01-04003 and Grant Scientific School-2112.2003.1.

** Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.
One of the most promising approaches is based on the notion of definability \[18\], where continuous objects and computational processes involving these objects can be defined using finite formulas in a suitable structure. Definability has been a very successful framework for generalised computability theory \[20\], descriptive complexity \[113\] and databases \[6,22\]. One of the most interesting and practically important types of definability is \(\Sigma\)-definability, which generalises recursive enumerability over the natural numbers \[2,10\]. However, the most developed part of definability theory deals with abstract structures with equality (i.e., the natural numbers, trees, automata, etc.). In the case of continuous data types, such as real numbers, real-valued functions and functionals, it is reasonable to consider the corresponding structures without the equality test. This is motivated by the following natural reason. In all effective approaches to extract real number computation via concrete representations \[11,21\], the equality test is undecidable. In order to do any kind of computation or to develop a computability theory, one has to work within a structure rich enough for information to be coded and stored. For this purpose we extend the structure \(\mathbb{R}\) by the set of hereditarily finite sets \(HF(\mathbb{R})\). The idea that the hereditarily finite sets over a structure form a natural domain for computation is discussed in \[2,10,22\].

Note that such or very similar extensions of structures are used in the theory of abstract state machines \[3\], in query languages for hierarchic databases \[6\], and in Web-like databases \[22\]. Since \(\Sigma\)-definability in \(HF(\mathbb{R})\) (i.e., definability by \(\Sigma\)-formulas) is a generalisation of recursive enumerability, we are interested in the language of \(\Sigma\)-formulas as in a suitable language intended to express computability over the reals.

In this paper we address the algorithmic properties of the language of \(\Sigma\)-formulas. For this purpose we study the expressive power of \(\Sigma\)-formulas using a suitable fragment of the constructive infinitary language \(L_{\omega_1\omega}\). Certain fragments of constructive \(L_{\omega_1\omega}\) have been used to study the expressive power of formal approaches to computability such as search computability \[19\], ‘While’-computability \[23\], dynamic logics \[12\] and fixed-point logics \[8\]. One of the most important results in the area is Engeler’s Lemma which relates abstract computability with definability in constructive \(L_{\omega_1\omega}\). Engeler’s Lemma was first proven for formal computability on abstract structures with the equality test (see \[9,10,23\]). The natural question to ask is: “Does Engeler’s Lemma hold for \(\Sigma\)-definability over the reals without the equality test?” In this paper we answer this question positively. Engeler’s Lemma, together with Gandy’s theorem for \(HF(\mathbb{R})\) (recently proved in \[16\]) gives an algorithmic characterisation of \(\Sigma\)-definability. We prove that a relation over the real numbers is \(\Sigma\)-definable if and only if it is definable by a disjunction of a recursively enumerable set of quantifier free formulas. Let us note that the proof of the ‘if’ direction follows from Engeler’s Lemma for \(\Sigma\)-definability and the proof of the ‘only if’ direction uses methods based on Gandy’s Theorem. It is worth noting that both of the directions of this characterisation are important. Engeler’s Lemma gives us an effective procedure which generates quantifier free formulas approximating \(\Sigma\)-relations. This reveals algorithmic aspects of \(\Sigma\)-definability. The converse direction provides tools for
descriptions of the results of effective infinite approximation processes by finite formulas.

The structure of this paper is as follows. In Section 2 we recall the notion of $\Sigma$-definability and provide a background necessary to understand the main results. Section 3 introduces a certain fragment of the constructive infinite logic and presents a characterisation of the expressive power of $\Sigma$-definability which reveals computational aspects of $\Sigma$-definability over the reals without the equality test. We conclude with a discussion of future work.

2 $\Sigma$-Definability over the Real Numbers

2.1 Basic Definitions and Notions

In this paper we consider the standard model of the real numbers

$$\langle \mathbb{IR}, 0, 1, +, \cdot, < \rangle = \langle \mathbb{IR}, \sigma_0 \rangle,$$

denoted also by $\mathbb{IR}$, where $+$ and $\cdot$ are regarded as the usual arithmetic operations on the real numbers. We use the language of strictly ordered rings, so we assume that the predicate $<$ occurs positively in all formulas. In order to develop a computability theory over the reals we extend the structure $\mathbb{IR}$ by the set of hereditarily finite sets $HF(\mathbb{IR})$ which is rich enough for information to be coded and stored. We construct the set of hereditarily finite sets, $HF(A)$ over a structure $A$, as follows:

1. $HF_0(A) \equiv A$,
2. $HF_{n+1}(A) \equiv P_\omega(HF_n(A)) \cup HF_n(A)$, where $n \in \omega$ and for every set $B$, $P_\omega(B)$ is the set of all finite subsets of $B$.
3. $HF(A) = \bigcup_{n \in \omega} HF_n(A)$.

We define $HF(\mathbb{IR})$ as the following model: $HF(\mathbb{IR}) \equiv \langle HF(\mathbb{IR}), U, S, \sigma_0, \emptyset, \in \rangle \equiv \langle HF(\mathbb{IR}), \sigma \rangle$, where the constant $\emptyset$ stands for the empty set and the binary predicate symbol $\in$ has the set-theoretic interpretation. We also add predicate symbols $U$ for urelements (elements from $\mathbb{IR}$) and $S$ for sets. Let $\mathbb{IR}^*$ denote the set of all nonempty finite subsets of the reals.

For our convenience, we use variables subject to the following conventions:

- $r, x, y, z, \ldots$ range over $\mathbb{IR}$ (urelements),
- $i, j, k, l, m, \ldots$ range over $\omega$ (natural numbers),
- $\alpha, \beta, \kappa, \ldots$ range over $HF(\emptyset)$,
- $A, B, C, R, X, Y, Z, \ldots$ range over $\mathbb{IR}^*$ (nonempty finite sets over the reals),
- $U, V, W, \ldots$ range over $HF(\mathbb{IR}^*)$ (finite sets which do not contain real numbers as members) and
- $K, L, M, N, S, \ldots$ range over $HF(\mathbb{IR})$.

We use the same letters as for variables to denote elements from the corresponding structures.

A formula in the form $p_1(x) < p_2(x)$, where $p_1$ and $p_2$ are polynomials with coefficients in $\mathbb{N}$, is called an atomic strict semi-algebraic (s.s.-algebraic) formula.
The set of atomic formulas is the union of the set of atomic s.s.-algebraic formulas and the formulas of the type \( N \in M \), where \( M \) ranges over sets.

The closure of the atomic s.s.-algebraic formulas under finite conjunctions and disjunctions forms the set of \textit{s.s.-algebraic formulas}.

The set of \( \Delta_0 \)-\textit{formulas} is the closure of the set of atomic formulas under \( \wedge, \vee, \neg \), bounded quantifiers \((\exists M \in S)\) and \((\forall M \in S)\), where \((\exists M \in S)\ \Psi\) denotes \(\exists M (M \in S \land \Psi)\), \((\forall M \in S)\ \Psi\) denotes \(\forall M (M \in S \rightarrow \Psi)\) and \( S \) ranges over sets.

We assume that predicates from the language \( \sigma_0 \) can occur only positively in our formulas.

With every atomic s.s.-algebraic formula \( p_1(x_1, \ldots, x_n) < p_2(x_1, \ldots, x_n) \) we associate the following formula, called a \textit{lifted atomic s.s.-algebraic formula}:

\[
p_1(X_1, \ldots, X_n) < p_2(X_1, \ldots, X_n) \iff (\forall x_1 \in X_1) \ldots (\forall x_n \in X_n) p_1(x_1, \ldots, x_n) < p_2(x_1, \ldots, x_n).
\]

The terms \( p_1(\bar{X}) \) and \( p_2(\bar{X}) \) are called \textit{lifted polynomials}. In a similar way, we can also associate with each s.s.-algebraic formula a \textit{lifted s.s.-algebraic formula}, i.e., the lifted s.s.-algebraic formulas are the closure of lifted atomic s.s.-algebraic formulas under finite conjunctions and disjunctions.

The set of \( \Sigma \)-\textit{formulas} is the closure of the set of \( \Delta_0 \)-formulas under \( \wedge, \vee, \neg \), \((\exists M \in S)\), \((\forall M \in S)\) and \( \exists \), where \( S \) ranges over sets.

We are interested in \( \Sigma \)-definability of subsets on \( \mathbb{R}^n \) which can be considered as a generalisation of recursive enumerability. The analogy of \( \Sigma \)-definable and recursive enumerable sets is based upon the following fact. If we consider the structure \( \mathbf{HF} = (\mathbf{HF}(\emptyset), \in) \) with the hereditarily finite sets over \( \emptyset \) as its universe and membership as its only relation, then the \( \Sigma \)-definable sets are exactly the recursively enumerable sets. The notion of \( \Sigma \)-definability has a natural meaning also in the structure \( \mathbf{HF}(\mathbb{R}) \).

\begin{definition}
1. A relation \( B \subseteq \mathbf{HF}(\mathbb{R})^n \) is \( \Delta_0 \) \( (\Sigma) \)-definable, if there exists a \( \Delta_0 \) \( (\Sigma) \)-formula \( \Phi \) such that \( M \in B \leftrightarrow \mathbf{HF}(\mathbb{R}) \models \Phi(M) \).
2. A function is \( \Delta_0 \) \( (\Sigma) \)-definable if its graph is \( \Delta_0 \) \( (\Sigma) \)-definable.
\end{definition}

Note that the sets \( \mathbb{R} \) and \( \omega \) are \( \Delta_0 \)-definable. This fact makes \( \mathbf{HF}(\mathbb{R}) \) a suitable domain for studying subsets of \( \mathbb{R}^n \).

In the following lemma we introduce some \( \Delta_0 \)-definable and \( \Sigma \)-definable predicates that will be used later.

\begin{lemma}
1. The sets \( \mathbb{R}, \mathbb{R}^*, \mathbf{HF}(\mathbb{R}^*) \) and \( \mathbf{HF}(n) \) for all \( n \in \omega \) are \( \Delta_0 \)-definable.
2. The following predicates are \( \Delta_0 \)-definable: \( M = N \), \( M = N \cap S \), \( M = N \cup S \), \( M = (N, S) \), \( M = N \setminus S \), where all variables \( M, N \) and \( S \) range over sets.
3. Every recursive function \( f : \omega^n \rightarrow \omega^m \) is \( \Sigma \)-definable.
4. Every \( B \subseteq \mathbb{N}^n \) is recursively enumerable if and only if it is \( \Sigma \)-definable.
5. All arithmetic operations on ordinals are \( \Sigma \)-definable.
\end{lemma}
Proposition 1 (Σ-reflection principle). Every Σ-formula \( \Phi(M) \) is equivalent to a formula of the type \( \exists N \Psi(N, M) \), where \( \Psi \) is a \( \Delta_0 \)-formula.

In the following lemmas we obtain some properties of s.s.-algebraic and lifted s.s.-algebraic formulas that will be used later.

Lemma 2. If \( B \subseteq \mathbb{R}^n \) is definable by an s.s.-algebraic formula then it is open.

Lemma 3. Let \( \Phi(\bar{x}) \) be an s.s.-algebraic formula and \( \Phi_{\text{lifted}}(\bar{X}) \) be the corresponding lifted s.s.-algebraic formula. If \( R_i = \{r_i\} \) for \( i = 1 \ldots n \), then

\[
\text{HF}(\mathbb{R}) \models \Phi(\bar{r}) \leftrightarrow \Phi_{\text{lifted}}(\bar{R}).
\]

Lemma 4. Let \( \Phi(\bar{x}) \) be an s.s.-algebraic formula and \( \Phi_{\text{lifted}}(\bar{X}) \) be the corresponding lifted s.s.-algebraic formula. Then for all \( \bar{x} \in \mathbb{R}^n \) we have

\[
\text{HF}(\mathbb{R}) \models \Phi(x_1, \ldots, x_n) \leftrightarrow \exists X_1 \ldots \exists X_n \Phi_{\text{lifted}}(X_1, \ldots, X_n) \land \bigwedge_{i \leq n} (x_i \in X_i).
\]

Lemma 5. Let \( \Phi_{\text{lifted}}(\bar{X}) \) be a lifted s.s.-algebraic formula and \( Y_1 \subseteq Z_1, \ldots, Y_n \subseteq Z_n \). If \( \text{HF}(\mathbb{R}) \models \Phi_{\text{lifted}}(Z_1, \ldots, Z_n) \), then \( \text{HF}(\mathbb{R}) \models \Phi_{\text{lifted}}(Y_1, \ldots, Y_n) \).

Lemma 6. Let \( \Phi_{\text{lifted}}(\bar{Y}) \) be a lifted s.s.-algebraic formula. If for some \( X_1, \ldots, X_n \) we have \( \text{HF}(\mathbb{R}) \models \Phi_{\text{lifted}}(X_1, \ldots, X_n) \), then for all \( m > 0 \) there exist \( Y_1, \ldots, Y_m \) of cardinality \( m \) such that for all \( i, j \leq n \) \( Y_i = Y_j \leftrightarrow X_i = X_j \) and \( \text{HF}(\mathbb{R}) \models \Phi_{\text{lifted}}(Y_1, \ldots, Y_n) \).

Lemma 7. Let \( \Phi \) be an existentially quantified s.s.-algebraic formula. Then there exists an s.s.-algebraic formula \( \Psi \) such that \( \text{HF}(\mathbb{R}) \models \Phi(\bar{x}) \leftrightarrow \Psi(\bar{x}) \). Moreover \( \Psi \) can be constructed effectively from \( \Phi \).

Let us note that proofs of all lemmas are straightforward (see [17]) except Lemma 7 which follows from the finiteness theorem [57].

### 2.2 Gandy’s Theorem and Inductive Definitions

Let us recall Gandy’s Theorem for \( \text{HF}(\mathbb{R}) \) which will be essentially used in all proofs of the main results. Let \( \Phi(a_1, \ldots, a_n, P) \) be a Σ-formula, where \( P \) occurs positively in \( \Phi \) and the arity of \( \Phi \) is equal to \( n \). We think of \( \Phi \) as defining an effective operator \( \Gamma : \mathcal{P}(\text{HF}(\mathbb{R}))^n \to \mathcal{P}(\text{HF}(\mathbb{R}))^n \) given by

\[
\Gamma(Q) = \{ \bar{a} \mid \text{HF}(\mathbb{R}), Q \models \Phi(\bar{a}, P) \}.
\]

Since the predicate symbol \( P \) occurs only positively, we have that the corresponding operator \( \Gamma \) is monotone, i.e., for any sets \( B \) and \( C \), from \( B \subseteq C \) follows \( \Gamma(B) \subseteq \Gamma(C) \). By monotonicity, the operator \( \Gamma \) has a least (w.r.t. inclusion) fixed point which can be described as follows. We start from the empty set and apply operator \( \Gamma \) until we reach the fixed point: \( \Gamma^0 = \emptyset \), \( \Gamma^{n+1} = \Gamma(\Gamma^n) \), \( \Gamma^\gamma = \bigcup_{\gamma n < \gamma} \Gamma^n \), where \( \gamma \) is a limit ordinal.

One can easily check that the sets \( \Gamma^n \) form an increasing chain of sets: \( \Gamma^0 \subseteq \Gamma^1 \subseteq \ldots \). By set-theoretical reasons, there exists the least ordinal \( \gamma \) such that \( \Gamma(\Gamma^\gamma) = \Gamma^\gamma \). This \( \Gamma^\gamma \) is the least fixed point of the given operator \( \Gamma \).
Theorem 1 (Gandy’s Theorem for HF(\(I\mathbb{R}\))).  
Let \( \Gamma : \mathcal{P}(HF(I\mathbb{R})^n) \to \mathcal{P}(HF(I\mathbb{R})^n) \) be an effective operator. Then the least fixed-point of \( \Gamma \) is \( \Sigma \)-definable and the least ordinal such that \( \Gamma(\Gamma^\gamma) = \Gamma^\gamma \) is less or equal to \( \omega \).

Proof. See [16].

2.3 Universal \( \Sigma \)-Predicate for s.s.-Algebraic Formulas

In order to obtain a result on the existence of a universal \( \Sigma \)-predicate for the s.s.-algebraic formulas, we first construct a universal \( \Sigma \)-predicate for the lifted s.s.-algebraic formulas. For this purpose we prove \( \Sigma \)-definability of the truth of lifted s.s.-algebraic formulas.

In this section we fix a standard effective Gödel numbering of the terms and formulas of the language \( \sigma \) by finite ordinals which are elements of HF(\( \emptyset \)). Let \( \lceil \phi \rceil \), \( \lceil p \rceil \) denote the codes of a formula \( \phi \) and a term \( p \) respectively. It is worth noting that the type of an expression is effectively recognisable by its code. We also can obtain effectively from the codes of expressions the codes of their subexpressions and vice versa. Since equality is \( \Delta_0 \)-definable in HF(\( \emptyset \)), we can use the well-known characterisation which states that all effective procedures over ordinals are \( \Sigma \)-definable. Thus, for example, the following predicates

\[
\begin{align*}
\text{Code}_{\text{elem}}(n, i, j) & \iff n = [X_i < X_j], \\
\text{Code}_{\text{sum1}}(n, i, j, k) & \iff n = [p + q < f] \land i = [p] \land j = [q] \land k = [f], \\
\text{Code}_{\land}(n, i, j) & \iff n = [\phi \land \psi] \land j = [\phi] \land j = [\psi]
\end{align*}
\]

are \( \Sigma \)-definable. Hence, in \( \Sigma \)-formulas we can use such predicates.

With every element \( A \in HF(I\mathbb{R}) \) we associate an interpretation \( \gamma_A \) of variables \( X_1, X_2, \ldots \) such that

\[
\gamma_A(X) = \begin{cases} 
C & \text{if } \langle \lceil X \rceil, C \rangle \in A \\
\emptyset & \text{for any } \langle \lceil X \rceil, B \rangle \in A, \text{ we have } B = C \\
\emptyset & \text{otherwise.}
\end{cases}
\]

Let \( V \) be a set of variables. An interpretation \( \gamma_A \) is called correct for \( V \) if for all \( X \in V \) we have \( \gamma_A(X) \neq \emptyset \). Let \( \text{Int} \) denote the set of elements \( A \in HF(I\mathbb{R}) \) with the following property: if \( \langle i, C \rangle \) and \( \langle i, B \rangle \) belong to \( A \), then we have \( C = B \). It is easy to see that this set is \( \Delta_0 \)-definable by the following formula:

\[
\text{Int}(A) \iff (\forall U \in A) (\forall W \in A) (\forall V_1 \in U) (\forall V_2 \in U) (\forall i \in V_1) (\forall X \in V_2) \\
(\forall V_3 \in W) (\forall Y \in V_3) ((U = \langle i, X \rangle \land W = \langle i, Y \rangle) \to X = Y).
\]

Theorem 2. There exists a binary \( \Sigma \)-definable predicate \( Tr \) such that for any \( n \in \omega \) and \( A \in HF(I\mathbb{R}) \) we have that \( (n, A) \in Tr \) if and only if \( n \) is the Gödel number of a lifted s.s.-algebraic formula \( \phi \), \( \gamma_A \) is a correct interpretation for free variables of \( \phi \) and \( HF(I\mathbb{R}) \models \phi[\gamma_A] \).
Proof. The predicate $Tr$ is the least fixed point of the operator defined by the following formula:

$$
\Phi(n, U, P) \equiv \Phi_{\text{proper}}(n) \lor \Phi_{\text{elem}}(n, U) \lor \Phi_{\text{sum}}(n, U, P) \lor \\
\Phi_{\text{mult}}(n, U, P) \lor \Phi_{\land}(n, U, P) \lor \Phi_{\lor}(n, U, P),
$$

where $n$, $U$ are free variables and $P$ is a new predicate symbol. The formula $\Phi(n, U, P)$ represents the inductive definition of the truth of the lifted s.s.-algebraic formulas, where the immediate subformulas have the following meaning. The formulas $\Phi_{\text{proper}}(n)$ and $\Phi_{\text{elem}}(n, U)$ define the basis of the inductive definition. In other words, the formula $\Phi_{\text{proper}}(n)$ represents the truth of the proper formulas, i.e., $0 < 1$ and $1 < 0$; the formula $\Phi_{\text{elem}}(n, U)$ represents the truth of the elementary formulas, i.e., the formulas of the type, $X_i < 0, 0 < X_i$, $1 < X_i$ and $X_i < 1$. The formulas $\Phi_{\text{sum}}(n, U, P)$ and $\Phi_{\text{mult}}(n, U, P)$ represent the inductive steps for sum and multiplication. Finally, the formulas $\Phi_{\land}(n, U, P)$ and $\Phi_{\lor}(n, U, P)$ represent the inductive steps for conjunctions and disjunctions. Let us show how to construct these formulas. The $\Delta_0$-formulas $\Phi_{\text{proper}}(n)$ is obvious. The $\Sigma$-formula $\Phi_{\text{elem}}(n, U)$ can be given as follows.

$$
\Phi_{\text{elem}}(n, U) \equiv \exists i \exists j \exists A \exists B \left( (n = [X < Y] \land i = [X] \land j = [Y] \land U = \{\langle i, A \rangle, \langle j, B \rangle\} \land A < B ) \lor \\
(n = [X < 0] \land i = [X] \land U = \{\langle i, A \rangle\} \land A < 0) \lor \\
(n = [0 < X] \land i = [X] \land U = \{\langle i, A \rangle\} \land 0 < A) \lor \\
(n = [X < 1] \land i = [X] \land U = \{\langle i, A \rangle\} \land A < 1) \lor \\
(n = [1 < X] \land i = [X] \land U = \{\langle i, A \rangle\} \land 1 < A) \right).
$$

Now we construct a $\Sigma$-formula $\Phi_{\text{sum}}(n, U, P)$ which represents the case when $n$ is the code of a formula of the type: $p + q < f$, where $p$, $q$ and $f$ are lifted polynomials. Let $\text{nextvar}([\mathcal{F}], l)$ denote the $\Sigma$-definable predicate which means that if $m$ is the maximal index of variables which occur in $\mathcal{F}$, then $l = m + 1$. The formula $\Phi_{\text{sum}}$ can be given as follows.

$$
\Phi_{\text{sum}}(n, U, P) \equiv \exists i \exists j \exists k \exists \ell \exists m \exists \nu \exists \omega \exists A \exists B \exists C \left( n = [p + q < f] \land \\
i = [p] \land j = [q] \land k = [f] \land \\
\text{nextvar}(n, l) \land m = l + 1 \land s = m + 1 \land \\
P([p < X_i], V) \land P([q < X_m], W) \land P([X_s < f], Y) \land \\
\langle i, A \rangle \in V \land \langle j, B \rangle \in W \land \{k, C\} \in Y \land A + B < C \land \\
U = (V \cup W \cup Y) \land \text{Int}(U) \right).
$$

In a similar way, we can produce a $\Sigma$-formula $\Phi_{\text{mul}}(n, U, P)$ which represents the case when $n$ is the code of a formula of the type: $p < q + f$, where $p$, $q$ and $f$ are lifted polynomials. Put $\Phi_{\text{sum}}(n, U, P) \equiv \Phi_{\text{sum}}(n, U, P) \land \Phi_{\text{mul}}(n, U, P)$. In the same way, we can produce the $\Sigma$-formula $\Phi_{\text{mul}}(n, U, P)$ which represents the inductive steps for multiplication.
The $\Sigma$-formula $\Phi$ can be constructed as follows:

$$\Phi(n, U, P) \iff \exists i \exists j n = [\varphi \land \psi] \land i = [\varphi] \land j = [\psi] \land P(i, U) \land P(j, U).$$

In a similar way, we can produce the $\Sigma$-formula $\Phi \land (n, U, P)$ which represents the inductive steps for conjunctions.

From Gandy’s theorem (c.f. Section 2.2) it follows that the least fixed point $Tr$ of the effective operator defined by $\Phi$ is $\Sigma$-definable.

\[\Box\]

**Theorem 3.** For every $n \in \omega$ there exists a $\Sigma$-formula $Univ_n^*(m, X_1, \ldots, X_n)$ such that for any lifted s.s.-algebraic formula $\Phi(X_1, \ldots, X_n)$

$$HF(\mathbb{R}) \models \Phi(R_1, \ldots, R_n) \iff Univ_n^*([\Phi], R_1, \ldots, R_n).$$

**Proof.** It is easy to see that the following formula defines a universal $\Sigma$-predicate for the lifted s.s.-algebraic formulas of arity $n$

$$Univ_n^*(m, X_1, \ldots, X_n) \iff \exists U (U = \{\langle 1, X_1 \rangle, \ldots, \langle n, X_n \rangle\} \land Tr(m, U)).$$

\[\Box\]

**Theorem 4.** For every $n \in \omega$ there exists a $\Sigma$-formula $Univ_n(m, x_1, \ldots, x_n)$ such that for any s.s.-algebraic formula $\Phi(x_1, \ldots, x_n)$

$$HF(\mathbb{R}) \models \Phi(r_1, \ldots, r_n) \iff Univ_n([\Phi], r_1, \ldots, r_n).$$

**Proof.** From the properties of the standard Gödel numbering it follows that the code of an s.s.-algebraic formula can be effectively constructed from the code of the corresponding lifted formula and vice versa. Let $f : \omega \to \omega$ be a recursive function which maps the code of an s.s.-algebraic formula to the code of the corresponding lifted formula. Then the following formula defines a universal $\Sigma$-predicate for the s.s.-algebraic formulas of arity $n$.

$$Univ_n(m, \bar{x}) \iff \exists X_1 \ldots \exists X_n \exists k f(m) = k \land Univ_n^*(k, \bar{x}) \land \bigwedge_{i \leq n} (x_i \in X_i).$$

\[\Box\]

3 Expressive Power of $\Sigma$-Definability over the Reals

3.1 Constructive Infinitary Language $L_{\omega_1 \omega}$

In order to study the expressive power of $\Sigma$-formulas, we will consider a suitable fragment $L_{\omega_1 \omega}^a$ of the constructive infinitary language $L_{\omega_1 \omega}$ (cf. [14]) described below. Informally, the language $L_{\omega_1 \omega}^a$ is obtained by extending the s.s.-algebraic formulas to allow formulas with effective infinite disjunctions but only finitely many variables; that is, formulas of the form $\bigvee_{i \in A} \Phi_i$, where $\{\Phi_i | i \in A\}$ is
an effectively indexed family of s.s.-algebraic formulas, possibly infinite. The meaning of these formulas is as follows: $\text{HF}(\mathbb{R}) \models \bigvee_{i \in A} \Phi_i$ if and only if for at least one $i \in A$ we have $\text{HF}(\mathbb{R}) \models \Phi_i$.

Formally, we propose the inductive definition of formulas as follows. Let $\text{Var}$ be a fixed finite set of variables. The set $L_{\text{Var}}$ of formulas over $\text{Var}$ includes the set of s.s.-algebraic formulas all of the variables of which belong to $\text{Var}$. In addition, if $\{\Phi_i | i \in A\}$ is an indexed family of formulas of $L_{\text{Var}}$ and $A$ is recursively enumerable, then $\bigvee_{i \in A} \Phi_i$ is a formula of $L_{\text{Var}}$. The language $L_{\omega_1}^l$ is the union of all $L_{\text{Var}}$ for all finite sets $\text{Var}$ of variables which range over $\mathbb{R}$.

### 3.2 Engeler’s Lemma for $\Sigma$-Definability

In this section we prove Engeler’s lemma for $\Sigma$-definability which states that if a relation $B \subseteq \mathbb{R}^n$ is $\Sigma$-definable, then it is definable by a formula of $L_{\omega_1}^l$ which can be constructed effectively from the corresponding $\Sigma$-formula.

In order to work effectively with elements of the structures $\text{HF}(\mathbb{R})$ and $\text{HF}(\mathbb{R}^*)$, we represent every element in a regular way. For this, we enrich the language $\sigma$ to $\sigma'$ by the additional functions: singleton $\{U\}$ and binary union $U_1 \cup U_2$. Note that these functions will be eliminated in the resulting formulas of Engeler’s Lemma. Below a term in the language $\{\{\}, \cup\}$ is called a structural term. With every $\alpha \in \text{HF}(n)$ we associate a structured term defined as follows:

$$t_\alpha(N_1, \ldots, N_n) = \begin{cases} N_i & \text{if } \alpha = i \\ \{t_{\alpha_1}(N)\} \cup \ldots \cup \{t_{\alpha_k}(N)\} & \text{if } \alpha = \{\alpha_1, \ldots, \alpha_k\}. \end{cases}$$

**Lemma 8.** 1. For every element $M \in \text{HF}(\mathbb{R})$ there exist a structural term $t_\alpha(N_1, \ldots, N_k)$ and a substitution $\tau : \{N_1, \ldots, N_k\} \to \mathbb{R}$ such that $M$ is represented by $t_\alpha(N)\tau$.

2. For every element $U \in \text{HF}(\mathbb{R}^*)$ there exist a structural term $t_\beta(N_1, \ldots, N_l)$ and a substitution $\nu : \{N_1, \ldots, N_l\} \to \mathbb{R}^*$ such that $U$ is represented by $t_\beta(N)\nu$.

If $\tau(N_i) = r_i$ for $i \leq k$ and $\nu(N_j) = R_j$ for $j \leq l$, then we write $t(r_1, \ldots, r_k)$ and $t(R_1, \ldots, R_l)$ instead of $t_\alpha(N)\tau$ and $t_\beta(N)\nu$ respectively.

**Lemma 9.** Let $t_\beta(X)$ and $t_\gamma(X)$ be structural terms and $\bar{R} \in \text{HF}(\mathbb{R}^*)^n$.

If $t_\beta(\bar{R})$ represents $U_1$, $t_\gamma(\bar{R})$ represents $U_2$ and $\text{HF}(\mathbb{R}) \models U_1 \subseteq U_2$, then we have $\text{HF}(\mathbb{R}) \models t_\beta(\bar{N}) \in t_\gamma(\bar{N})$ for every $\bar{N} \in \text{HF}(\mathbb{R})^n$ such that for all $i, j \leq n$ $N_i = N_j \iff R_i = R_j$.

With every $\Sigma$-formula $\Phi(\bar{x})$ we associate a lifted $\Sigma$-formula $\Phi_{\text{lifted}}(\bar{X})$ obtained from $\Phi$ by replacing every variable ranging over $\text{HF}(\mathbb{R})$ by a variable ranging over $\text{HF}(\mathbb{R}^*)$; every variable ranging over $\mathbb{R}$ by a variable ranging over $\mathbb{R}^*$. For example, $\exists U (\forall Y \in U) p_1(Y, \bar{X}) < p_2(Y, \bar{X})$ is the lifted $\Sigma$-formula corresponding to $\exists M (\forall y \in M) p_1(y, \bar{x}) < p_2(y, \bar{x})$. 
Proposition 2. Let $\Phi_{lifted}(\bar{X})$ be the lifted $\Sigma$-formula corresponding to a $\Sigma$-formula $\Phi(\bar{x})$. Then we have

$$HF(\mathbb{R}) \models \Phi(x_1, \ldots, x_n) \leftrightarrow \exists X_1 \ldots \exists X_n \Phi_{lifted} (\bar{X}) \land \bigwedge_{i \leq n} (x_i \in X_i).$$

Proof. Let $\Phi(\bar{x})$ be a $\Sigma$-formula. By the $\Sigma$-reflection principle (c.f. Section 2.2.), the formula $\Phi(\bar{x})$ can be represented in the form:

$$\Phi(\bar{x}) \iff \exists M \Phi'(M, \bar{x}) \land \bigwedge_{i \leq n} (x_i \in M) \iff \exists M \Phi''(M, \bar{x}),$$

where $\Phi'$ and $\Phi''$ are $\Delta_0$-formulas. The corresponding lifted formula $\Phi_{lifted}(\bar{X})$ is obtained from $\Phi$ by replacing every variable ranging over $HF(\mathbb{R})$ by a variable ranging over $HF(\mathbb{R}^*)$; every variable ranging over $\mathbb{R}$ by a variable ranging over $\mathbb{R}^*$, i.e.,

$$\Phi_{lifted}(\bar{X}) \iff \exists U \Phi'_{lifted}(U, \bar{X}) \land \bigwedge_{i \leq n} (X_i \in U) \iff \exists U \Phi''_{lifted}(U, \bar{X}).$$

$\rightarrow$) Suppose (1) is valid in $HF(\mathbb{R})$ and $N, \bar{r}$ satisfy the formula $\Phi''(M, \bar{x})$. In order to construct some $V$ and $\bar{R}$ which satisfy the corresponding lifted $\Sigma$-formula, we use the operation $up$ defined by induction:

$$up(K) = \begin{cases} \{K\} & \text{if } K \in \mathbb{R} \\ \{up(K_1), \ldots, up(K_l)\} & \text{if } K = \{K_1, \ldots, K_l\}. \end{cases}$$

We put $V = up(N)$, $R_1 = up(r_1), \ldots, R_n = up(r_n)$. By Lemma 3 and Lemma 1, $V$ and $R_1, \ldots, R_n$ satisfy the formula $\Phi''_{lifted}(U, \bar{X})$.

$\leftarrow$) Suppose (2) is valid in $HF(\mathbb{R})$ and $V, R_1, \ldots, R_n$ satisfy the given formula $\Phi''_{lifted}(U, \bar{X})$. By Lemma 3, the set $V$ can be represented by $t_\beta(R_1, \ldots, R_m)$, where $\{R_1, \ldots, R_n\} \subseteq \{R_1, \ldots, R_m\}$. From Lemma 3, it follows that if $V = t_\beta(R_1, \ldots, R_m)$ satisfies the following requirement:

$$\text{there exist } r_1 \in R_1, \ldots, r_m \in R_m \text{ such that } r_i = r_j \iff R_i = R_j \text{ for all } i, j \leq m,$$

then $t_\beta(\bar{r})$ and $r_1, \ldots, r_n$ satisfy $\Phi''(M, \bar{x})$. Let us note that $V$ may not satisfy (3), for example, if $V = \{\{r_1, r_2\}, \{r_1\}, \{r_2\}\}$, where $r_1 \neq r_2 \neq r_3$. The problem here is that the number of elements in the sets is too small to pick different representatives. In this case we construct $R'_1, \ldots, R'_n$ and $V'$ from $R_1, \ldots, R_n$ and $V$ which satisfy the formula $\Phi'_{lifted}(U, \bar{X})$ and the requirement (3). It can be done using Lemma 6. Indeed, there exist $R'_1, \ldots, R'_m$ such that

1. for every $i \leq m$ we have $|R'_i| \geq m$;
2. for every $i, j \leq m$ $R'_i = R'_j$ if and only if $R_i = R_j$;}
3. for every s.s.-algebraic subformula $\phi_{\text{alg}}(\bar{Y}, \bar{X})$ of the formula $\Phi''$ and every substitution $\tau: \{\bar{Y}, \bar{X}\} \rightarrow \{R_1, \ldots, R_m\}$ we have 

$$\text{HF}(\mathbb{I}R) \models \phi_{\text{alg}}(\bar{Y}, \bar{X})\tau \leftrightarrow \text{HF}(\mathbb{I}R) \models \phi_{\text{alg}}(\bar{Y}, \bar{X})\tau', $$

where $\tau': \{\bar{Y}, \bar{X}\} \rightarrow \{R'_1, \ldots, R'_m\}$ is a substitution such that $\tau'(Y_k) = R'_j \leftrightarrow \tau(Y_k) = R_j$ and $\tau'(X_l) = R'_i \leftrightarrow \tau(X_l) = R_i$ for all $i, j, k, l \leq m$. 

By construction, $t_\beta(R'_1, \ldots, R'_m)$ and $R'_1, \ldots, R'_n$ satisfy $\Phi''_{\text{lifted}}(U, \bar{X})$ and (3).

Now every $R'_i$ contains enough elements to choose $r_i$ from $R'_i$ under the condition $r_i = r_j \leftrightarrow R_i = R_j$. It is easy to see that $t_\beta(\bar{r})$ and $r_1, \ldots, r_n$ satisfy $\Phi''(M, \bar{x})$. \hfill $\square$

In the following proposition the lifted s.s.-algebraic atomic formulas and the $\Delta_0$-formulas of the type $X = Y \iff (\forall x \in X) x \in Y \land (\forall y \in Y) y \in X$ and $\neg X = Y \iff (\exists x \in X) x \notin Y \lor (\exists y \in Y) y \notin X$ are considered as basic formulas.

**Proposition 3.** Let $\Phi(X_1, \ldots, X_n)$ be a lifted $\Sigma$-formula. There exists a constructive infinite formula $\Psi \models \bigvee_{i \in \omega} \Psi_i$ such that:

- $\text{HF}(\mathbb{I}R) \models \phi(\bar{X}) \leftrightarrow \Psi(\bar{X})$.
- Every $\Psi_i(\bar{X})$ is a formula of the form $\exists Y_m \ldots \exists Y_n \psi'(\bar{Y}, \bar{X})$, where $\psi'$ is a finite conjunction of basic formulas whose quantifiers range over $\mathbb{I}R^*$.

**Proof.** Let $\Phi(X) \models \exists U \phi'(U, \bar{X})$, where $\phi'$ is a $\Delta_0$-formula. In order to obtain the required formula, we first construct an equivalent infinite formula in the language $\sigma'$ (e.g. $\sigma' = \sigma \cup \{\{\}, \cup\}$) without unbounded quantifiers. Then we prove by induction the existence of an equivalent infinite formula without bounded quantifiers. After that we eliminate $\{\}, \cup, \emptyset$, $\in$ from the obtained formula. By Lemma [3] every $U \in \text{HF}(\mathbb{I}R^*)$ can be represented by $t_\beta(Y_1, \ldots, Y_n)$ for some $Y_1 \in \mathbb{I}R^*, \ldots, Y_n \in \mathbb{I}R^*$. Put 

$$\Phi^*(X) \models \bigvee_{n \in \omega} \bigvee_{\beta \in \text{HF}(n)} \exists Y_1 \ldots \exists Y_n \phi'(t_\beta(Y_1, \ldots, Y_n), \bar{X})$$

$$\models \bigvee_{n \in \omega} \bigvee_{\beta \in \text{HF}(n)} \exists Y_1 \ldots \exists Y_n \phi'_\beta(\bar{Y}, \bar{X}),$$

where every $\phi'_\beta$ is a $\Delta_0$-formula with quantifiers bounded by subterms of $t_\beta$. Since for every $U$ there exists a term $t_\beta$ which codes the structure of $U$, the formula $\Phi^*$ is equivalent to the given one.

Without loss of generality every formula $\phi'_\beta$ can be represented as follows.

$$\phi'_\beta(\bar{Y}, \bar{X}) \models (Q U_1 \in t_{\gamma_1}(\bar{Y})) \ldots (Q U_m \in t_{\gamma_m}(\bar{Y})) \phi_\beta(U, \bar{Y}, \bar{X})$$

$$\models (Q U \in t_{\gamma_k}(\bar{Y})) \phi'_\beta(U, \bar{Y}, \bar{X}),$$

where $Q$ is the quantifier $\exists$ or $\forall$ and $t_{\gamma_i}$ is a subterm of $t_\beta$ for all $i \leq k$. Using induction on the length of the quantifier prefix and the depth of the term which bounds the first quantifier in the quantifier prefix, we show how to obtain an
equivalent quantifier free formula. We proceed by induction on the pairs \(\langle m, n \rangle\) with the lexicographic order, where \(m\) is the length of the quantifier prefix of \(\Phi'_{\beta}\) and \(n\) is the depth of \(t_{\gamma_1}\). Let \(\top\) denote a logical truth which can be represented by the formula \(0 < 1\) and \(\bot\) denote a logical false which can be represented by the formula \(1 < 0\).

The cases \(\langle 0, 0 \rangle\), \(\langle 0, n \rangle\) are obvious. In the case \(\langle m, 0 \rangle\) the formula \(\Phi'_{\beta}\) can be represented in the form \(\Phi'_{\beta} = (Q U_1 \in \emptyset) \phi'_{\beta}\) or \(\Phi'_{\beta} = (Q U_1 \in X) \phi'_{\beta}\). If \(Q\) is the existential quantifier, then put \(\Psi_{\beta} = \top\). If \(Q\) is the universal quantifier, then put \(\Psi_{\beta} = \bot\). Consider the inductive step \(\langle m, n \rangle \rightarrow \langle m, n + 1 \rangle\). The first possibility is that \(\Phi'_{\beta} = (\exists U_1 \in t_{\gamma_1}(\bar{Y})) \phi'_{\beta}(U_1, \bar{Y}, \bar{X})\) and \(t_{\gamma_1}(\bar{Y}) = t'_{\gamma_1}(\bar{Y}) \cup t''_{\gamma_1}(\bar{Y})\). Let us consider the formula \((\exists U_1 \in t'_{\gamma_1}(\bar{Y})) \phi'_{\beta}(U_1, \bar{Y}, \bar{X}) \lor (\exists U_1 \in t''_{\gamma_1}(\bar{Y})) \phi'_{\beta}(U_1, \bar{Y}, \bar{X})\) which is equivalent to \(\Phi'_{\beta}\). The complexity of \(t'_{\gamma_1}\) and \(t''_{\gamma_1}\) is less than \(n + 1\). By inductive hypothesis, there exists a formula \(\Psi_{\beta}\) without quantifiers which is equivalent to \(\Phi'_{\beta}\). The second possibility with a bounded universal quantifier can be considered in a similar way.

Consider the inductive step \(\langle m, n \rangle \rightarrow \langle m + 1, k \rangle\). We have \(\Phi'_{\beta} = (\exists U_1 \in \{t(\bar{Y})\}) \phi'_{\beta}(U_1, \bar{Y}, \bar{X})\). Let us consider \(\phi'_{\beta}(t(\bar{Y}), \bar{Y}, \bar{X})\) which is equivalent to \(\Phi'_{\beta}\). The complexity of the quantifier prefix is less then \(m + 1\). By inductive hypothesis, there exists a formula \(\Psi_{\beta}\) without quantifiers which is equivalent to \(\Phi'_{\beta}\). The case, when \(\Phi'_{\beta} = (\forall U_1 \in \{t(\bar{Y})\}) \phi'_{\beta}(U_1, \bar{Y}, \bar{X})\), is similar.

Now we eliminate \(\{\}, \cup, \emptyset, \in\) from every formula \(\Psi_{\beta}\). It is easy to see that for any terms \(t_\gamma(\bar{Y}, \bar{X})\) and \(t_\kappa(\bar{Y}, \bar{X})\) of the language \(\sigma'\) it is possible to write effectively \(\chi(\bar{Y}, \bar{X})\) and \(\chi(\bar{Y}, \bar{X})\) such that \(\chi(\bar{Y}, \bar{X})\) and \(\chi(\bar{Y}, \bar{X})\) are finite disjunctions of finite conjunctions of basic formulas and the formula \(t_\gamma \in t_\kappa\) is equivalent to \(\chi(\bar{Y}, \bar{X})\) and the formula \(t_\kappa = t_\gamma\) is equivalent to \(\chi(\bar{Y}, \bar{X})\). For example, it is easy to see that the formula \(\{Y_1\} \cup \{Y_2\} \in \{\{Y_3\} \cup \{Y_4\}\} \cup \{\{Y_5\}\}\) is equivalent to \((Y_1 \lor Y_2) \land (Y_3 \lor Y_4)\) and \((Y_1 \lor Y_2) \land Y_3\) is equivalent to \((Y_1 \lor Y_2) \land (Y_3 \lor Y_4)\).

Using these formulas we transform every formula \(\Psi_{\beta}\) into \(\Psi'_{\beta}\) without occurrences of \(\{\}, \cup, \emptyset, \in\).

Put

\[
\Psi(X) = \bigvee_{n \in \omega} \bigvee_{\beta \in \text{HF}(n)} \exists Y_1 \ldots \exists Y_n \Psi'_{\beta}(\bar{Y}, \bar{X}).
\]

By construction, \(\Psi\) has the required form. \(\square\)

Now we are ready to prove Engeler’s Lemma for \(\Sigma\)-definability over the reals.

**Theorem 5 (Engeler’s Lemma for \(\Sigma\)-definability).** If a relation \(B \subset \mathbb{R}^n\) is \(\Sigma\)-definable, then it is definable by a formula of \(L^\omega_{\text{hot}}\). Moreover this formula can be constructed effectively from the corresponding \(\Sigma\)-formula.

**Proof.** Suppose \(B\) is defined by a \(\Sigma\)-formula \(\Phi(\bar{x})\). By Proposition 2 and Proposition 3 there exists an effective sequence \(\{\Psi_i\}_{i \in \omega}\) such that

\[
\text{HF}(\mathbb{R}) \models \Phi(\bar{x}) \iff \bigvee_{i \in \omega} \left( \exists X_1 \ldots \exists X_n \Psi_i(\bar{X}) \land \bigwedge_{j \leq n} x_j \in X_j \right),
\]
where every $\Psi_i(\bar{X})$ is a formula of the form $\exists Y_{m_1} \ldots \exists Y_{m_i} \Psi'_i(\bar{Y}, \bar{X})$, and $\Psi'$ is a finite conjunction of lifted s.s.-algebraic formulas and formulas of the type $X_j = Y_i$, $Y_i = Y_j$, $\neg X_j = Y_i$ and $\neg Y_i = Y_j$.

Now we show that for every $i \in \omega$ there exists an existential quantified s.s.-algebraic formula $\varphi_i$ in the language $\sigma_0 = \{0, 1, +, \cdot, <\}$ such that

$$\text{HF}(\mathbb{R}) \models \varphi_i(\bar{x}) \iff \text{HF}(\mathbb{R}) \models \left( \exists X_1 \ldots \exists X_n \Psi_i(\bar{X}) \land \bigwedge_{j \leq n} x_j \in X_j \right). \quad (4)$$

For this purpose, in every $\Psi_i$ we first eliminate subformulas of the type $X_j = Y_k$ and $Y_j = Y_k$ in the following way. For all $j$ and $k$, such that there exists a subformula $X_j = Y_k$, we replace all occurrences of $Y_k$ by $X_j$, the subformula $X_j = Y_k$ by $\top$ and eliminate the quantifier over $Y_i$. For all $i$ and $k$ such that, there exists $Y_j = Y_k$ for some $j$ and $k > j$, we replace all occurrences of $Y_k$ by $Y_j$, the subformula $Y_j = Y_k$ by $\top$ and eliminate the quantifier over $Y_k$. It is easy to see that the resulting formula $\Psi'_i(\bar{X})$ is equivalent to $\Psi_i(\bar{X})$.

Now if $X_j = X_k$ occurs in $\Psi'_i(\ldots, X_j, \ldots, X_k, \ldots)$ then we replace $\Psi'_i$ by $\Psi'_i(\ldots, X_j, \ldots, X_k, \ldots) \land \Psi'_i(\ldots, X_k, \ldots, X_k, \ldots)$ and remove all $X_j = X_k$ from this formula. Let us argue that the obtained formula $\Psi''_i$ is equivalent to $\Psi'_i$ in the following sense

$$\text{HF}(\mathbb{R}) \models \forall x_1 \ldots \forall x_n \left[ \left( \exists X_1 \ldots \exists X_n \Psi'_i(\bar{X}) \land \bigwedge_{j \leq n} x_j \in X_j \right) \iff \left( \exists X_1 \ldots \exists X_n \Psi''_i(\bar{X}) \land \bigwedge_{j \leq n} x_j \in X_j \right) \right].$$

Implication from left to right is obvious. In order to prove implication from right to left, we need to show that if for $x_1, \ldots, x_n$ there exist $X_1, \ldots, X_n$ such that $\Psi''_i(\bar{X}) \land \bigwedge_{j \leq n} x_j \in X_j$, then there exist $X'_1, \ldots, X'_n$ such that $X'_j = X_k'$ and $\Psi'_i(\bar{X}') \land \bigwedge_{j \leq n} x_j \in X'_j$. For this we can take $X'_j = X_k' = X_j \cup X_k$. In the same way we can eliminate all subformulas of the form $X_j = X_k$ obtaining a formula $\Psi'''_i$. We then construct formulas $\varphi_i$ from $\Psi'''_i$ by replacing $X$ by $x$, $Y_i$ by $y_i$ and subformulas of the type $\neg x = y$ by $x < y \lor x > y$. From the definition of a lifted s.s.-algebraic formula and Proposition \ref{prop:lifting} it follows that the formula $\varphi_i$ satisfies \ref{eq:lifting}. In order to complete the proof, using Lemma \ref{lem:lifting} we construct an effective sequence $\{\varphi'_i\}_{i \in \omega}$ of s.s.-algebraic formulas such that

$$\text{HF}(\mathbb{R}) \models \Phi(x) \iff \bigvee_{i \in \omega} \varphi'_i(x).$$

\[\square\]

### 3.3 Characterisation Theorem for $\Sigma$-Definability

Let us prove the converse statement of Engeler’s Lemma for $\Sigma$-definability.
Theorem 6. If a relation \( B \subset \mathbb{R}^n \) is definable by a formula of \( L^\omega_1 \), then it is \( \Sigma \)-definable. Moreover \( \Sigma \)-formula can be constructed effectively from the corresponding formula of \( L^\omega_1 \).

Proof. Let \( B \subset \mathbb{R}^n \) be definable by \( \bigvee_{\psi \in A} \psi(x) \), where \( A \) is recursively enumerable. By Theorem 4 there exists a universal \( \Sigma \)-predicate \( \text{Univ}_n(m, \bar{x}) \) for s.s.-algebraic formulas with variables from \( \{x_1, \ldots, x_n\} \).

Put
\[
\Phi(\bar{x}) \iff \exists i (i \in A) \land \text{Univ}^n(i, \bar{x}).
\]
It can be shown that \( \Phi \) is a required \( \Sigma \)-formula.

Theorem 7 (Characterisation of \( \Sigma \)-definability). A relation \( B \subset \mathbb{R}^n \) is \( \Sigma \)-definable if and only if it is definable by a formula of \( L^\omega_1 \).

These results reveal algorithmic aspects of \( \Sigma \)-definability. Indeed, suppose \( \Phi(x) \) is a \( \Sigma \)-formula which defines a relation over the reals and we have \( \text{HF}(\mathbb{R}) \models \Phi(\bar{x}) \iff \bigvee_{\psi \in A} \psi(\bar{x}) \). Then each s.s.-algebraic formula \( \psi(\bar{x}) \), such that \( \lfloor \psi \rfloor \in A \), represents a simple approximation of the relation definable by \( \Phi(\bar{x}) \) and there exists a Turing machine that computes these approximations (i.e., enumerates \( \Psi(\bar{x}) \)). A universal Turing machine and a universal \( \Sigma \)-predicate for s.s.-algebraic formulas can then be used to enumerate and check validity of each approximation \( \Psi \).

We also obtain the following topological characterisation of \( \Sigma \)-definability over the reals.

Theorem 8. 1. A set \( B \subset \mathbb{R}^n \) is \( \Sigma \)-definable if and only if it is an effective union of open semi-algebraic sets.

2. A relation \( B \subset \mathbb{R}^n \) is \( \Sigma \)-definable if and only if there exists an effective sequence \( \{A_i\}_{i \in \omega} \) of open semi-algebraic sets such that
   - (a) It monotonically increases: \( A_i \subseteq A_{i+1} \), for \( i \in \omega \);
   - (b) \( B = \bigcup_{i \in \omega} A_i \).

Let \( \Sigma_{\mathbb{R}} \) denote the set of all \( \Sigma \)-definable subsets of \( \mathbb{R}^n \), where \( n \in \omega \).

Corollary 1. 1. The set \( \Sigma_{\mathbb{R}} \) is closed under finite intersections and effective infinite unions.

2. The set \( \Sigma_{\mathbb{R}} \) is closed under \( \Sigma \)-inductive definitions.

3. The set \( \Sigma_{\mathbb{R}} \) is closed under projections.

4 Future Work

In this paper we characterised the expressive power of the language of \( \Sigma \)-formulas over the reals without the equality test. One of the main directions of future work is to develop a logical approach to computability on continuous data types based on this language. So far we have investigated the expressive power of the language of \( \Sigma \)-formulas over the reals. In this respect the following direction of research is of special interest: to propose and study reasonable requirements on the universe and the language of an abstract structure without the equality test under which a similar characterisation can be obtained.
References

1 Introduction

Consider the following imperative programming language. The programs operate on registers storing natural numbers, the input $\vec{x}$ is stored in certain registers, and a number $b$, called the base, is fixed to $\max(\vec{x}, 1) + 1$ before the execution starts. The single primitive instruction $X+$ increases the number stored in the register $X$ by 1 modulo $b$. There are two control structures: the loop while $X \{ P \}$ executing the program $P$ repeatedly as long as the content of the register $X$ is different from 0; the composition $P; Q$ executing first the program $P$, then the program $Q$. This is the whole language. The language is natural, extremely simple, yet powerful. We will prove that it captures LINSPACE, i.e. the numerical relations decidable by such programs are exactly those decidable by Turing machines working in linear space. Variations of the language capturing other important deterministic complexity classes, like e.g. LOGSPACE, P and PSPACE are possible (see Kristiansen and Voda [5]).

We will extend the language without destroying its simplicity with variables of finite types holding (finite) array-like data structures which are naturally coded into numbers. The types are defined in the standard way, i.e. $0$ is a type, and $\sigma \to \tau$ is a type if $\sigma$ and $\tau$ are types. We will prove that the programs compute level by level the functions of the Ritchie hierarchy. A function is Kalmar elementary iff it is computable by a program.

We then study a restricted version of Godel’s functionals primitive recursive in finite types which are obtained by removing the successor function and by replacing the constant 0 by 1. Somewhat surprisingly, the restricted functionals have the same power as our programming language.

Numerous schemes have been introduced to restrict recursion in higher types. So-called ramification techniques (e.g. Simmons [11], Leivant [6], Beckmann and

---

1 The definition is later in the text.
2 The classes in the Ritchie hierarchy add up to the class of Kalmar elementary functions.
Weiermann \([2]\)) restrict higher type recursion to the Kalmár elementary level. By using so-called linearity constraints in conjunction with ramification techniques, higher type recursion can be restricted further down to the “polytime” level (e.g. Bellantoni and Schwichtenberg \([9]\)). This paper suggests a qualitatively different scheme for restricting the recursion in higher types: Modify the set of initial functions. That the power of higher type recursion is severely weakened when the initial functions are sufficiently weak, is hardly surprising, but it is quite surprising that the functions definable by higher type recursion do not degenerate into an unnatural and uninteresting class, even when the only available initial function is the constant 1.

We have recently found out that Tiuryn \([12]\) has investigated the (finite) functions over finite types obtained when the type 0 is finite. He called them generalized arrays and obtained the same space bounds as we by using families of programming languages having both syntax and semantics somewhat more complicated than ours. He did not investigate the connection to Gödel’s functionals and neither did Jones \([4]\) who obtained similar results with an imperative language of functionals over the finite data structure of S-expressions without cons (corresponding to the successor function). The finite version of Gödel’s functionals was investigated by Goerdt \([3]\). We stress here that, in contrast to the approach of Jones and Goerdt who thought it necessary to use also the Cartesian types, we work with the original Gödel’s functionals (which are infinite and without Cartesian types) where we just remove the successor function.

2 Numbers and Types

**Definition.** 0 is a type; \(\sigma \to \tau\) is a type if \(\sigma\) and \(\tau\) are types. We say a type \(\sigma\) has degree \(n\) when \(\text{dg}(\sigma) = n\) where \(\text{dg}(0) = 0\) and \(\text{dg}(\sigma \to \tau) = \max(\text{dg}(\sigma) + 1, \text{dg}(\tau))\). We define the cardinality of type \(\sigma\) at base \(b\), in symbols \(|\sigma|_b\), by recursion on the build-up of \(\sigma\): \(|0|_b = b\) and \(|\rho \to \tau|_b = |\tau|_{|\rho|_b}\). (We use standard conventions and interpret \(\sigma \to \sigma' \to \sigma''\) by associating parentheses to the right, i.e. as \(\sigma \to (\sigma' \to \sigma'')\); further, \(\sigma, \sigma' \to \sigma\) is alternative notation for \(\sigma \to \sigma' \to \sigma''\).) Let \(2^x_0 \overset{\text{def}}{=} x\) and \(2^x_{n+1} \overset{\text{def}}{=} 2^{2^x_n}\).

**Theorem 1.** (i) For any polynomial \(p\) and \(n > 0\) there exists a type \(\sigma\) of degree \(n\) such that \(|\sigma|_{\max(x,1)+1} > 2^{p(x)}_n\). (ii) For every type \(\sigma\) of degree \(n\) there exists a polynomial \(p\) such that \(2^{p(x)}_n > |\sigma|_x\).

**Proof.** (i) We will prove (i) by induction on \(n\). Assume \(n = 1\). Let the types \(\xi_0, \xi_1, \ldots\) be defined by \(\xi_0 = 0\) and \(\xi_{j+1} = 0 \to \xi_j\). We have \(\text{dg}(\xi_j) = 1\) and \(|\xi_j| = x^{(x^j)}\) for every \(j > 0\) (*). (This is easily proved by induction on \(j\).) For an arbitrary polynomial \(p\) we have \(|\xi_k|_{\max(x,2)} > 2^{p(x)}_k\) for a sufficiently large \(k\). Thus, (i) holds when \(n = 1\). We turn to the induction step. Assume that \(|\sigma|_{\max(x,1)+1} > 2^{p(x)}_n\) where \(\text{dg}(\sigma) = n\). Then, we have

\[
2^{p(x)}_{n+1} = 2^{2^{p(x)}_n} < 2^{2^{p(x)}_{\max(x,1)+1}} \leq (|0|_{\max(x,1)+1})^{2^{p(x)}_{\max(x,1)+1}} = |\sigma \to 0|_{\max(x,1)+1}
\]
and $\text{dg}(\sigma \rightarrow 0) = \text{dg}(\sigma) + 1$. (ii) It is obvious if $\sigma$ has degree 0. Assume $\sigma = \rho \rightarrow \tau$ has degree $n + 1$. By the induction hypothesis we have $2^q(\chi) > |\rho|_{\chi}$ and $2^q(\chi) > |\tau|_{\chi}$ for some suitable polynomial $q$. Thus, we have

$$|\sigma|_{\chi} = |\tau|_{\chi} |\rho|_{\chi} < (2^q(\chi))(2^q(\chi)) < (2(2^q(\chi)))(2^q(\chi)) = 2(2^q(\chi))^2 < 2^{(2^q(\chi))} = 2^{p(\chi)}$$

for some suitable polynomial $p$. \hfill \Box

**Definition.** The natural number $a$ is a **number of type** $\sigma$ **at base** $b$, in symbols $a: \sigma_b$, iff $a < |\sigma|_b$. Let $a: (\sigma \rightarrow \tau)_b$. Then $a$ can be uniquely written in the form

$$v_0 + v_1 |\tau|_b^1 + \cdots + v_k |\tau|_b^k$$

where $k = |\sigma|_b - 1$ and $v_j : \tau_b$ for $j \in \{0, \ldots, k\}$. We call $v_0, \ldots, v_k$ the **digits** in $a$, and for any $i: \sigma_b$, we denote the $i$'th digit in $a$ by $a[i]_b$, i.e. $a[i]_b = v_i$. Furthermore, for any $i: \sigma_b$ and $w: \tau_b$, let $[a[i] := w]_b$ denote the number which is the result of setting the $i$'th digit in $a$ to $w$. (Note that $[a[i] := w]_b$ is a number of type $\sigma$ at base $b$.) The notation $a[i_1, \ldots, i_n]_b$, where $n \geq 1$, abbreviates $((a[i_1]_b)[i_2]_b) \ldots [i_n]_b$, and we will call $a[i_1, \ldots, i_n]_b$ a **sub-digit** of $a$. (Thus, every digit is a sub-digit, every digit of a digit is sub-digit, every digit of a digit of digit is a sub-digit, and so on.) Further, let

$$a[i_1, \ldots, i_n+1] := w)_b \overset{\text{def}}{=} a[i_1, \ldots, i_n := a[i_1, \ldots, i_n]_b[i_{n+1} := w]_b$$

for $n \geq 1$. Thus, $a[i_1, \ldots, i_n := w]_b$ is the number which is the results of setting the sub-digit $a[i_1, \ldots, i_n]_b$ in $a$ to $w$. \hfill \Box

### 3 An Imperative Programming Language

We will now define a programming language. An informal explanation of the language follows the definition. It might be a good idea to read the definition and the explanation in parallel.

**Definition.** First we define the syntax of the programming language. The language has an infinite supply of **program variables** $x_0^\sigma, x_1^\sigma, x_2^\sigma, \ldots$ for any type $\sigma$. We will use verbatim Latin letters, uppercase or lowercase, with or without subscripts and superscripts, to denote program variables. Any variable of type $\sigma$ is a **term of type** $\sigma$; $t[X]$ is an is an term of type $\tau$ if $X$ is a variable of type $\sigma$ and $t$ is a term of type $\sigma \rightarrow \tau$. We use $a[i_1, \ldots, i_n]$ to abbreviate $a[i_1][i_2] \ldots [i_n]$. (Note that any term is a variable or has the form $a[i_1, \ldots, i_n]$ where $a, i_1, \ldots, i_n$ are variables.) The primitive instruction $t+$ is a **program in** $\bar{X}$ if $t$ is a term of type $0$ such that every variable in $t$ occurs in the variable list $\bar{X}$; the **loop** $\text{while } t (P)$ is a program in $\bar{X}$ if $P$ is a program in $\bar{X}$, and $t$ is a term of type $0$ such that every variable in $t$ occurs in the variable list $\bar{X}$; the **sequence** $P; Q$ is a program in $\bar{X}$ if $P$ and $Q$ are programs in $\bar{X}$.

We will now define the semantics of the programming language. Let $P$ be a program in $\bar{X} = X_1, \ldots, X_n$. The meaning of $P$ is a $(2n + 1)$-ary **input-output**
Let program relation \( x \) be a list of variables, \( X \) in the set \( \{0,1,\ldots,\sigma|b - 1\} \). The only primitive instruction in the language

\[ a' = a[i_1, \ldots, i_m := (a[i_1, \ldots, i_m]b) \oplus 1]_b. \]

This completes the definition of the programming language.

A program \( P \) in the variables \( X_1, \ldots, X_k \) where \( X_1, \ldots, X_n \) are of type \( 0 \), computes the number-theoretic function \( f(x_1, \ldots, x_n) \) when

\[ \forall x_{n+1}, \ldots, x_k \exists y_1, \ldots, y_{k-1} \left[ x_1, \ldots, x_n, x_{n+1}, \ldots, x_k, P \right] \]  
\[ \text{iff } f(x_1, \ldots, x_n) = z \]

where \( b = \max(x_1, \ldots, x_n, 1) + 1 \)

(Note that the output variable \( Z \) can be of any type.) A program decides a number-theoretic relation if it computes the characteristic function of the relation. A program \( \text{variable } X \) of type \( \sigma \) has degree \( n \) when \( \sigma \) has degree \( n \). A program \( \text{in the variables } X \) has degree \( n \) when every variable in \( X \) has degree \( \leq n \).

Let \( P^n \) denote the set of number-theoretic functions computed by the programs of degree \( n \). The program hierarchy \( P \) is defined by \( P = \bigcup_{n \in \mathbb{N}} P^n \).

Finally, we need some notation to develop, specify and reason about programs. Let \( Z \) be a list of variables, and let \( P \) be a program in \( X_1, \ldots, X_n, Z \). We will use \( x(P)_b y \) to abbreviate \( \forall z \exists u\left[ x, z, x(P)_b y, u \right] \). We will develop programs in an informal mnemonic macro notation. The notation should need no further explanation for readers familiar with standard programming languages. We will always assume that the local variables in the macros are suitably renamed such that we avoid name conflicts. In the macros we will use uppercase Latin letters \( X, Y, Z \), etc. to denote variables of type \( 0 \), and lowercase Latin letters \( u, i, j \), etc. to denote variables of higher types.

In the following we give an explication of the programming language. The execution of program will take place in a certain base \( b \) given by \( b = \max(x, 1) + 1 \) where \( x \in \mathbb{N} \) are the inputs. Program variables of type \( \sigma \) store natural numbers in the set \( \{0,1,\ldots,|\sigma|b - 1\} \). The only primitive instruction in the language
has either the form (i) $a[X_1, \ldots, X_n]$ where $a, X_1, \ldots, X_n$ are variables such that $a[X_1, \ldots, X_n]$ is a type 0 term, or the form (ii) $Y+$ where $Y$ is a type 0 variable. In case (i) the sub-digit $a[X_1, \ldots, X_n]$ in $a$ is increased by 1 modulo $b$, that is, the sub-digit will be increased by 1, except in the case when the sub-digit equals $b-1$, then it is set to 0. In case (ii) $Y$ is increased by 1 modulo $b$. The language has one control structure, the loop while $t$ {...} where $t$ is a term of the same form and type as to those allowed in the primitive instruction. This is a standard while-loop executing its body repeteatedly as long as the value (sub-digit) $t$ refers to is different from 0. The semicolon composing two programs has the standard meaning.

**Definition.** Let $x \oplus_n y \overset{\text{def}}{=} (x + y) \pmod{n}$ and $x \ominus_n y \overset{\text{def}}{=} (x - y) \pmod{n}$. Let $sg(x) = 0$ if $x = 0$; otherwise $sg(x) = 1$.

**Lemma 1.** Let $X, Y$ and $Z$ denote type 0 variables, and assume that $x, y$ and $z$ are assigned to respectively $X, Y$ and $Z$. Let $b \geq 2$ and assume $x, y, z < b$. The following macros can be implemented as type 0 programs.

- $x \{ X:=0 \}_b 0$ (assignment of the constant 0)
- $x \{ X:=1 \}_b 1$ (assignment of the constant 1)
- $x, y \{ Y:=X \}_b x, x$ (assignment)
- $z, x, y \{ Z:=X+Y \}_b x \oplus_b y, x, y$ (addition modulo $b$)
- $z, x, y \{ Z:=X-Y \}_b x \ominus_b y, x, y$ (subtraction modulo $b$)
- $x, y \{ X:=sg Y \}_b sg(y), x$ (converting numbers to boolean values)
- $x, y \{ X:=not Y \}_b x', y$ where $x' = 1$ if $y = 0$; otherwise $x' = 0$. (logical not)
- $z, x, y \{ Z:=X or Y \}_b z', x, y$ where $z' = 0$ if $x = y = 0$; otherwise $z' = 1$. (logical or)
- $z, x, y \{ Z:=X and Y \}_b z', x, y$ where $z' = 1$ if $x = y = 1$; otherwise $z' = 0$. (logical and)

**Proof.** Let $X:=0 \equiv$ while $X \{ X+ \}$ and let $X:=1 \equiv X:=0; X+$. Let $\overline{x}$ denote the complement of $x$ modulo $b$, i.e. $\overline{x}$ is the unique number such that $x \oplus_b \overline{x} = 0$. Let

$$cc(X,Y,Z) \equiv Y:=0; Z:=0; while X \{ X+; Y+; U+ \}.$$ 

Then we have $x, y, z \ cc(X,Y,Z) 0, \overline{x}, \overline{x}$. Note that $\overline{\overline{x}} = x$. Thus, let

$$Y:=X \equiv cc(X,U,V); cc(U,X,Y).$$

Further, let $Z:=X+Y \equiv Z:=X; cc(Y,U,V); while U \{ Z+; Y+; U+ \}$. By using the macros we have defined so far we can also easily define the macro $Z:=X-Y$ since $x \ominus_b y = \overline{\overline{x}} \ominus_b \overline{\overline{y}}$. Let $X:=not Y \equiv Z:=Y; X:=1; while Z \{ X:=0; Z+ \}$ and let $X:=sg Y \equiv U:=not Y; X:=not U$. We see that also $Z:=X or Y$ can be implemented since $sg(sg(x) \oplus_b sg(y)) = 0$ if $x = y = 0$ and 1 otherwise. Use $X:=not Y$ and $Z:=X or Y$ to implement $Z:=X and Y$. \qed

Lemma 1 tells that the type 0 fragment of the programming language is powerful. Indeed the fragment yields full Turing computability if the programs
always are executed at sufficiently high base, i.e. the base \( b \) is set to a sufficiently large number before the execution starts. We will prove that if the base \( b \) is set to \( \max(\vec{x}, 1)+1 \) (where \( \vec{x} \) are the inputs) the type 0 fragment captures the complexity class LINS\( \text{SPACE} \). Programs containing higher types can compute functions beyond LINS\( \text{SPACE} \). When we set the base \( b \) to \( \max(\vec{x}, 1)+1 \), type 0 programs can do arithmetic modulo \( b \), in general, programs of type \( \sigma \) can do do arithmetic modulo \( |\sigma|_b \).

Lemma 2. Let \( b \geq 2 \) and let \( c = |\sigma|_b \). Further, Let \( u, v \) and \( w \) be program variables of type \( \sigma \), and let \( X \) be a program variable of type 0. Assumed that the numbers \( u, v, w \) and \( x \) (where \( u, v, w < c \) and \( x < b \)) are assigned to respectively \( u,v, w \) and \( X \). The following macros can be implemented as type \( \sigma \) programs.

\[
\begin{align*}
- & \ u \{ u:=\pi 0 \}_{b} 0 \ (assignment \ of \ 0 \ to \ a \ variable \ of \ type \ \sigma) \\
- & \ u \{ u+\sigma \}_{b} u \oplus_c 1 \ (successor \ modulo \ |\sigma|_b) \\
- & \ u \{ u-\sigma \}_{b} u \ominus_c 1 \ (predecessor \ modulo \ |\sigma|_b) \\
- & \ u,x \ \{ X:=sg(u) \}_{b} u, sg(x) \\
- & \ u,v,w \ \{ u:=v+w \}_{b} v \oplus_c w, v,w \ (addition \ modulo \ |\sigma|_b) \\
- & \ u,v,w \ \{ u:=v-w \}_{b} v\ominus_c w, v,w \ (subtraction \ modulo \ |\sigma|_b) \\
\end{align*}
\]

Proof. We define the three macros \( u+\pi, u:=\pi 0 \) and \( X:=sg(u) \pi \) simultaneously by recursion over the build up of the type \( \sigma \). By Lemma 1, we can define the macros when \( \sigma \equiv 0 \). Now, assume \( \sigma \equiv \pi \rightarrow \tau \) and that the macros are already defined for the types \( \pi \) and \( \tau \). Then we define \( u:=\pi 0 \) by

\[
i:=\pi 0; X:=1; \ \text{while} \ X \{ u[i]:=\pi 0; i+\pi; X:=sg(i) \pi \}
\]

(Explanation: \( u \) is a \( |\pi|_b \)-digit number in base \( |\tau|_b \). The macro sets each digit to 0.) We define \( X:=sg(u) \pi \) by

\[
i:=\pi 0; U:=1; \\
\ \text{while} \ U \{ X:=sg(u[i]) \tau; i+\pi; Y:=sg(i) \pi; Z:=\neg X; U:=Y \text{and} Z \}
\]

(Explanation: \( u \) is a \( |\pi|_b \)-digit number in base \( |\tau|_b \). The macro sets the type 0 variable \( X \) to 0 if each digit in \( u \) is 0.) We define \( u+\pi \) by

\[
i:=\pi 0; j:=\pi 0; j+\pi; u[i]+\pi; U:=sg(u[i]) \tau; \\
\ \text{while} \ U \{ j+\pi; i+\pi; u[i]+\pi; \\
\ \ X:=sg(u[i]) \tau; Y:=sg(j) \pi; Z:=\neg X; U:=Y \text{and} Z \}
\]

(Explanation: \( u \) is a \( |\pi|_b \)-digit number in base \( |\tau|_b \). The macro increases the digit \( u[0] \) by 1 modulo \( |\tau|_b \). If the digit turns into 0, the macro increase the digit \( u[1] \) by 1 modulo \( |\tau|_b \). If the digit turns into 0, the the digit \( u[2] \) is increased, and so on. Note, when the execution of the loop’s body starts, we have \( i \oplus_d 1 = j \) where \( d = |\tau|_b \). Given the macros \( u+\pi, u:=\pi 0 \) and \( X:=sg(u) \pi \), it is easy to define \( u:=\pi v+w \) and \( u:=\pi v-w \). Implement \( u:=\pi v-w \) and \( u:=\pi v-w \) by computing complements, i.e. take advantage of the equation \( x \oplus_c y = \overline{x} \oplus_c y \) where \( \overline{z} \) denotes the \( c \)-complement of \( z \), i.e. the unique number such that \( z + \overline{z} = c \).
**Definition.** We use standard Turing Machines with a single one-way tape. When computing a number-theoretic function, a Turing machine is assumed to take its input and give its output in binary notation. Let $M$ be a Turing machine computing a number-theoretic function $f(\vec{x})$. We say that $M$ works in space $g(\vec{x})$ if the number of cells visited on $M$'s tape during the computation of $f(\vec{x})$ is bounded by $g(\vec{x})$. A Turing machine works in linear space if there exists a fixed number $k$ such that the number of tape cells visited under a computation on the input $w$ is bounded by $k|w|$ where $w$ denotes the length of the input.

We define the class $R_n$ of number-theoretic functions by

- $f \in R^0$ iff $f$ is computable by a Turing machine working in linear space.
- $f \in R^{n+1}$ iff $f(\vec{x})$ is computable by a Turing machine working in space $g(\vec{x})$ for some $g \in R^n$.

We define the Ritchie hierarchy $R$ by $R = \bigcup_{n \in \mathbb{N}} R^n$. □

**Lemma 3.** (i) For every function $f \in R^i$ there exists a polynomial $p$ such that $2_i^{p(\vec{x})} > f(\vec{x})$. (In particular, any function in $R^0$ is bounded by a polynomial.) (ii) For every polynomial $p$ there exists a function $f \in R_i$ such that $f(\vec{x}) > 2_i^{p(\vec{x})}$. (In particular, any polynomial is bounded by a function in $R^0$.)

**Proof.** (i) We prove (i) by induction on $i$. It is easy to verify that for any $f \in R^0$ there exists a polynomial $p$ such that $f(\vec{x}) < p(\vec{x}) = 2_0^{f(\vec{x})}$. We skip the details. Assume $f \in R^{i+1}$. Then there exist a Turing machine $M$ and $g \in R^i$ such that $M$ computes $f$ in space $g(\vec{x})$. The induction hypothesis yields a polynomial $q(\vec{x})$ such that $M$ computes $f$ in space $2_i^{q(\vec{x})}$, and thus, the number of bits required to represent the value $f(\vec{x})$ is bounded by $2_i^{q(\vec{x})}$. It follows that $2_{i+1}^{p(\vec{x})} > f(\vec{x})$ for some polynomial $p$. (ii) We prove (ii) by induction on $i$. We leave the case $i = 0$ to the reader, and focus on the induction step: Let $p$ be an arbitrary polynomial. The induction hypothesis yields a function $h \in R^i$ such that $h(\vec{x}) > 2_i^{p(\vec{x})}$. Let $f(\vec{x}) = 2^{h(\vec{x})}$. Obviously we have $f(\vec{x}) > 2_i^{p(\vec{x})}$, and we conclude the proof by argue that $f \in R^{i+1}$. A Turing machine $M$ can compute $f$ by the following procedure: First $M$ computes the number $h(\vec{x})$ (this can obviously be done in space $g_0(\vec{x})$ for some $g_0 \in R^i$); then $M$ computes $f(\vec{x})$ by writing down the digit 1 followed by $h(\vec{x})$ copies of the digit 0. Let $g(\vec{x}) = g_0(\vec{x}) + h(\vec{x}) + k$ where $k$ is a sufficiently large fixed number. Then $g \in R^i$ and $M$ works in space $g$. Hence, $f$ can be computed by a Turing machine working in space $g$ for some $g \in R^i$, that is, $f \in R^{i+1}$. □

The hierarchy $R$ was introduced in Ritchie [7]. Note that $R^0 = \mathcal{E}^2$ where $\mathcal{E}^2$ denotes the second Grzegorczyk class; $R = \mathcal{E}^3$ where $\mathcal{E}^3$ denotes the third Grzegorczyk class also known as the class of Kalmár elementary functions.

**Theorem 2.** The program hierarchy and the Ritchie hierarchy match from level 1 and upwards, i.e. $P^{i+1} = R^{i+1}$ for any $i \in \mathbb{N}$.

**Proof.** We first prove $P^{i+1} \subseteq R^{i+1}$. Assume $f \in P^{i+1}$. Then there exists a program $P$ of degree $i+1$ computing $f(\vec{x})$. By Theorem 3 there exists a polynomial $p$
such that no variable exceeds $2^{p(x)}$ during the execution. Thus a Turing machine $M$ representing numbers in binary notation needs $O(2^{p(x)})$ tape cells to simulate the program $P$. Let $c$ be a constant such that $M$ runs in space $c2^{p(x)}$. Then there exists a polynomial $q$ such that $M$ runs in space $2^{q(x)}$, and by Lemma 3(ii) there exists $g \in \mathbb{R}$ such that $M$ runs in space $g(x)$. This proves that $f \in \mathbb{R}^{i+1}$.

We will now prove $\mathbb{R}^n \subseteq \mathbb{P}^n$. The proofs splits into two cases (1) $n = i + 2$, (2) $n = 1$. Case (1). Let $f \in \mathbb{R}^{i+2}$. We can without loss of generality assume that $f$ is a unary function. By Lemma 3 there exist a polynomial $p$ and a Turing machine $M$ computing $f$ such that the number of cells visited on $M$’s tape in the computation of $f(x)$ is bounded by $2^{p(x)}$. We will, uniformly in the Turing machine $M$, construct a program computing $f$. Let $|\Sigma|$ denote the cardinality of $M$’s alphabet (including the blank symbol). If $\max(x, 1) < |\Sigma|$, the program computes $f(x)$ by a hand tailored algorithm, i.e. the program simply consults a finite table. If $\max(x, 1) \geq |\Sigma|$, the program computes $f(x)$ by simulating $M$.

We map each symbol in $M$’s alphabet to a unique value in the set $|0|^\max(x, 1)+1$. (This is possible since $\max(x, 1) \geq |\Sigma|$) By Lemma 1 there exists a type $\tau$ of degree $i+1$ such that $|\tau|^\max(x, 1)+1 > 2^{p(x)}$. Thus, the program can represent $M$’s tape by a variable $a$ of type $\tau \to 0$, and the scanning head by variable $X$ of type $\tau$. The scanning head can be moved one position to the right (left) by executing the program $X+\tau$ ($X-\tau$), and the type $0$-term $a[X]$ yields the scanned symbol. The degree of the program will equal the degree of $a$, i.e. $i + 2$. Thus, $f$ can be computed by a program degree $i + 2$. Hence, $f \in \mathbb{P}^{i+2}$. Case (1). We skip this case.

**Definition.** A function $f$ is **non increasing** when $f(\vec{x}) \leq \max(\vec{x}, 1)$

The reason that the Ritchie hierarchy and the program hierarchy do not match at level 0, is simply that a type 0 programs compute only non increasing functions. The next theorem says that with respect to non increasing functions the hierarchies also match at level 0. The proof of a similar theorem can be found in [5].

**Theorem 3.** We have $f \in \mathbb{P}^0$ iff $f \in \mathbb{R}^0$ for every non increasing function $f$.

The following lemma is needed in the succeeding section. We leave the proof to the reader.

**Lemma 4.** Let $k > 0$, and let $P$ be a program of degree $k$ computing the total function $f(\vec{x})$. There exists a type $\sigma$ of degree $k$ such that the number of steps $P$ uses to compute $f(\vec{x})$ is bounded by $|\sigma|^\max(\vec{x}, 1)+1$. (One step corresponds to the execution of one primitive instruction, i.e. an instruction on the form $t+$.)

## 4 Restricted Functionals in Finite Types

We will introduce functionals in the style of Shoenfield [10]. This style is more convenient for our purposes then the combinator style used e.g. in [1].
Definition. We will now define the constants, variables, terms and defining equations of the system D.

- 1 is a constant of type 0
- For each type \( \sigma \) we have an infinite supply of variables \( x_0^\sigma, x_1^\sigma, x_2^\sigma, \ldots \)
- A variable of type \( \sigma \) is a term of type \( \sigma \). A constant of type \( \sigma \) is a term of type \( \sigma \). If \( t \) is a term of type \( \sigma \to \tau \) and \( u \) is a term of type \( \sigma \), the \( t(u) \) is a term of type \( \tau \).
- Suppose that \( \vec{x} \) are distinct variables of type \( \vec{\sigma} \) respectively, and that \( t \) is a term of type \( \tau \) in which no variable other that \( \vec{x} \) occurs. We introduce a new constant \( F \) of type \( \vec{\sigma} \to \tau \) with the defining equation \( F(\vec{x}) = t \). We will say that \( F \) is defined explicitly (over the constants in \( t \)).
- Suppose that \( x \) is a variable of type \( 0 \), \( G \) is a constant of type \( \sigma \), and \( H \) is a constant of type \( \sigma \to (0 \to \sigma) \). We introduce a new constant \( F \) of type \( 0 \to \sigma \) with the defining equations \( F(0) = G \) and \( F(x + 1) = H(F(x))(x) \).

We will say that \( F \) is defined by primitive recursion (from \( G \) and \( H \)).

The system \( D \) provides a syntax for equations defining functionals in finite types. Each constant in \( D \) corresponds to a functional. Thus, we will also call the constants in \( D \) functionals and we will call constants of type \( 0 \to 0 \) functions. Note that there are no initial constants representing the zero and the successor function in \( D \), and that the left-hand sides in the primitive recursive equations are not terms. We will use the formal system \( Y \) of Shoenfield [10] to discuss and compute the functionals defined in \( D \). The language of \( Y \) has a constant \( k_n \) for each \( n \in \mathbb{N} \) (occasionally we will simply write \( n \) instead of \( k_n \)). Thus, if \( F(0) = G \) and \( F(x + 1) = H(F(x))(x) \) are the defining equations for \( F \), then \( F(k_0) = G \) and all equations \( F(k_{n+1}) = H(F(k_n))(k_n) \) are axioms in \( Y \). Further, \( 1 = k_1 \) is an axiom. (The axioms induced by the explicit definitions in \( D \) are the obvious ones.)

A number-theoretic function \( f \) is definable in \( D \) when there exists a constant \( F \) of type \( 0 \to 0 \) such that \( \vdash_Y F(k_{x_1}, \ldots, k_{x_n}) = k_{f(x_1, \ldots, x_n)} \) (for all \( x_1, \ldots, x_n \in \mathbb{N} \)). A number-theoretic relation \( R \) is definable in \( D \) when there exists a constant \( F \) of type \( 0 \to 0 \) such that \( \vdash_Y F(k_{x_1}, \ldots, k_{x_n}) = k_0 \) iff \( R(x_1, \ldots, x_n) \) holds. \( \square \)

Lemma 5 (Basic functions). The following number-theoretic functions are definable in \( D \). (i) 0, 1 (constant functions); (ii) for each fixed \( k \in \mathbb{N} \) the the function \( C_k(x) \) where \( C_k(x) = k \) if \( x \geq k \), and \( C_k(x) = x \) otherwise (so \( C_k(x) \) is an almost everywhere constant function yielding \( k \) for all but finitely many values of \( x \)); (iii) \( P(x) \) (predecessor) (iv) \( x - y \) (modified subtraction); (v) \( f(x, y, z) \) such that \( f(x, y) = x \oplus y + 1 \) when \( x \leq y \); (vi) \( c(x, y, z) \) where \( c(x, y, z) = x \) if \( z = 0 \) and \( c(x, y, z) = y \) if \( z \neq 0 \); (vii) \( \text{max}(x, y) \).

Proof. It is easy to see that the constant function 1 is definable in \( D \) since it is represented by an initial constant in \( D \). To give equations defining the constant function 0 is slightly nontrivial. Define \( g \) by primitive recursion such that \( g(x, 0) = x \) and \( g(x, y + 1) = y \). It is easy to verify that \( g \) is definable in \( D \). Then we can explicitly define the predecessor \( P \) from \( g \) by \( P(x) = g(x, x) \). Further,
we can define the constant function 0 by 0 = P(1). This proves that (i) and (iii) holds. (iv) holds since we have x−0 = x and x−(y + 1) = P(x−y). (vi) holds since c(x, y, 0) = x and c(x, y, z + 1) = y. (vii) holds since max(x, y) = c(x, y, 1(x−y)). (v) holds since x ⊕_{m+1} 1 = c(0, m−(m−x)−1, m−x). It remains to prove that (ii) holds. Let M^0(z) = 0 and M^{n+1}(z) = M^n(z) ⊕_{z+1} 1. Now, M^n is definable in D for any n ∈ N. Further, M^n(z) = n (mod z + 1). Thus, we have C_k(x) = c(x, M^k(x), P^k(x)) where P^k is the predecessor function repeated k times. Hence, (ii) holds. □

Lemma 6 (Iteration functionals). For all types σ and τ there exists a functional It^τ_σ:0, τ → τ, τ → τ such that It^τ_σ(ℓ, F, X) = F^{[σ|ℓ+1]}(X).

Proof. We prove the lemma by induction on the build-up of σ. Assume σ = 0. It is easy to see that for any τ we can define It^τ_0 such that It^τ_0(ℓ, F, X) = FF^ℓ(X)

Thus, (i) holds since 0|ℓ+1 = ℓ + 1. Assume σ = σ_1 → σ_2. By the induction hypothesis we have It^τ_{σ_2} and It^τ_{σ_1} satisfying the lemma. Define It^τ_σ explicitly from It^τ_{σ_2} and It^τ_{σ_1−τ} such that

It^τ_σ(ℓ, F, X) = It^τ_{σ_1−τ}(ℓ, It^τ_{σ_2}(ℓ), F)(X).

We will prove that we indeed have

It^τ_σ(ℓ, F, X) = F^{[σ|ℓ+1]}(X)  \quad \text{(Goal)}

Let A abbreviate It^τ_{σ_2}(ℓ). Hence, we have

A(F, X) = F^{[σ_2|ℓ+1]}(x)  \quad \text{(†)}

by the induction hypothesis. We need

A^k(F, Y) = F^{[σ_2|ℓ+1]}^k(Y)  \quad \text{(Claim)}

for any Y of type τ. (Goal) follows from (Claim) since

It^τ_σ(ℓ, F, X) = It^τ_{σ_1−τ}(ℓ, A, F)(X)  \quad \text{def. of It^τ_σ}

\begin{align*}
&= A^{[σ_1|ℓ+1]}(F)(X)  \quad \text{ind. hyp. on } σ_1 \\
&= F^{[σ_2|σ_1|ℓ+1]}(X)  \quad \text{(Claim)} \\
&= F^{[σ|ℓ+1]}(X)  \quad \text{def. of } [σ|ℓ+1]
\end{align*}

(Claim) is proved by induction on k. In the inductive case we need

\([A^k(F)](Y) = F^{[σ_2|ℓ+1]}^k(Y)

which is proved by another induction on n. We skip the details. □

Lemma 7 (Transformation functionals). For any finite list of types ⃗ρ there exist a type π and transformation functionals Tr^π_{ρ_i} of type ρ_i → π, Tr^π_{π} of type π → ρ_i such that Tr^π_{ρ_i}(Tr^π_{π}(u)) = u for any u of type ρ_i.
Lemma 8 (List functionals). For any finite list of types $\bar{\sigma}$ there exists a type $\tau$ and a list functional $\langle \ldots \rangle : \bar{\sigma} \rightarrow (0 \rightarrow \tau)$ such that $\Tr^{\bar{\sigma}}_\tau ((X_1, \ldots , X_n)(i)) = X_i$.

Proof. The functional $\langle \ldots \rangle : \bar{\sigma} \rightarrow (0 \rightarrow \tau)$ is easily defined from the basic functionals introduced in Lemma 5 and the transformation functionals $\Tr^{\bar{\sigma}}_\tau$ introduced in Lemma 7.

Definition. For a term $t : \sigma$ we define the value of $t$ at the base $b$ under valuation $v_1, \ldots , v_k$, in symbols $\Val_{b}^{v_1, \ldots , v_k}(t)$. Here the valuation assigns values to the free variables $x_1, \ldots , x_k$ of $t$ such that to $x_i : \sigma$ we assign $v_i < |\sigma|_b$ and we will have $\Val_{b}^{v_1, \ldots , v_k}(t) < |\sigma|_b$. Recall that for numbers $a : (\rho \rightarrow \tau)_b$ and $i : \rho_b$ we denote the $i$th digit in $a$ by $a[i]_b$ (see the definition in Section 2).

- Let $\Val_{b}^{v_1, \ldots , v_k}(x_i) = v_i$ for the variable $x_i$; $\Val_{b}^{v_1, \ldots , v_k}(k_n) = n$ for $n \in \mathbb{N}$; $\Val_{b}^{v_1, \ldots , v_k}(1) = 1$ for the initial function 1.
- Suppose $t \equiv s(u)$. Let $\Val_{b}^{v_1, \ldots , v_k}(t) = \Val_{b}^{v_1, \ldots , v_k}(s)[\Val_{b}^{v_1, \ldots , v_k}(u)]_b$.
- Suppose $t \equiv F$ where $F$ is a constant explicitly defined by $F(\overline{x}) = a$. Then $F$ is of type $\sigma_1 \rightarrow \cdots \sigma_k \rightarrow \tau$ for some $k$, and the variables $x_1, \ldots , x_k$ have types $\sigma_1, \ldots , \sigma_k$ respectively. We define $\Val_{b}^{v_1, \ldots , v_k}(F)$ by recursion on $k$. If $k = 0$, let $\Val_{b}^{v_1, \ldots , v_k}(F) = \Val_{b}^{v_1, \ldots , v_k}(a)$, else let

$$
\Val_{b}^{v_1, \ldots , v_k}(F) = \sum_{i < |\sigma_1|_b}^{m} \Val_{b}^{v_1, \ldots , v_k}(a) \times |\sigma_2 \rightarrow \cdots \rightarrow \sigma_k \rightarrow \tau|_b^i.
$$

- Suppose $t \equiv F$ where $F$ is a constant with defining equations $F(0) = G$ and $F(x + 1) = H(F(x))(x)$. Then let

$$
\Val_{b}^{v_1, \ldots , v_k}(F) = \sum_{i < b}^{m} \Val_{b}^{v_1, \ldots , v_k}(a) \times |0 \rightarrow \sigma|_m^i
$$

where $v_0 = \Val_{b}^{v_1, \ldots , v_k}(G)$ and $v_i + 1 = \Val_{b}^{v_1, \ldots , v_k}(H)[v_i]_b$.

Lemma 9 (Equality functionals). For any type $\sigma$ there exists an equality functional $\Eq_{\sigma} : \sigma, \sigma, \sigma \rightarrow 0$ such that $\Eq_{\sigma}(\ell, t, s) = 0$ iff $\Val_{\ell+1}(t) = \Val_{\ell+1}(s)$ for any variable-free terms $t : \sigma$ and $s : \sigma$.

Proof. Define $0_0 = 0$ and $0 : (\rho \rightarrow \tau)(y) = 0_\tau$ for any types $\rho$ and $\tau$. Then we have $\Val_{b}^{0}(0_\sigma) = 0$ for any $\sigma$ and $b > 1$. Further, we need $\Le_{\sigma} : \sigma, \sigma, \sigma \rightarrow 0$ and $\Suc_{\sigma} : \sigma, \sigma \rightarrow \sigma$ such that

$$
\Le_{\sigma}(\ell, t, s) = 0 \text{ if and only if } \Val_{\ell+1}(t) \leq \Val_{\ell+1}(s)
$$

$$
\Suc_{\sigma}(\ell, t) = s \text{ if and only if } \Val_{\ell+1}(t) + 1 = \Val_{\ell+1}(s) \pmod{|\sigma|_{\ell+1}}
$$

for any variable-free terms $t : \sigma$ and $s : \sigma$. Define the functionals $\Eq_{\sigma}$, $\Le_{\sigma}$ and $\Suc_{\sigma}$ simultaneously by recursion over the build-up of $\sigma$. It follows from Lemma 5.
that the three functionals are definable when $\sigma = 0$. Assume $\sigma = \rho \rightarrow \tau$. Define the "carry predicate" $C : 0, \rho \rightarrow \tau, \rho \rightarrow 0$ such that

$$C(\ell, x, y) = 0 \iff \forall z : \rho \{ \text{Le}_\rho(\ell, z, y) = 0 \rightarrow \text{Eq}_\tau(\ell, x(z)), 0_\tau) = 0 \}.$$  

(Note that although the quantifier ranges over infinitely many functionals $z : \rho$ only their finitely many values at the base $\ell + 1$ matters. The carry predicate is defined from $\text{Le}_\rho, \text{Eq}_\tau, \text{Suc}_\tau, 0_\tau$ using iteration, transformation and list functionals given in Lemma 6, 7 and 8.) Then use the carry predicate to define $\text{Suc}_{\rho \rightarrow \tau}$ such that

$$\text{Suc}_{\rho \rightarrow \tau}(\ell, x, y) = \begin{cases} 
\text{Suc}_\tau(\ell, x(y)) & \text{if } C(m, x, y) = 0 \\
x(y) & \text{otherwise.}
\end{cases}$$

For the definition of $\text{Le}_{\rho \rightarrow \tau}$ define, analogous as $C$, the characteristic function $A : 0, \rho \rightarrow \tau, \rho \rightarrow \tau, \rho \rightarrow 0$ such that

$$A(\ell, x, y, z) = 0 \iff \forall w : \rho \{ \text{Le}_\rho(\ell, z, w) = 0 \rightarrow \text{Eq}_\tau(\ell, x(w)), y(w)) = 0 \}.$$  

Then define $\text{Le}_{\rho \rightarrow \tau}$ to satisfy

$$\text{Le}_{\rho \rightarrow \tau}(\ell, x, y) \iff \exists z : \rho \{ \text{Le}_\tau(\ell, x(z), y(z)) = 0 \land A(\ell, x, y, z) \}.$$  

Finally, define $\text{Eq}_{\rho \rightarrow \tau}$ to satisfy

$$\text{Eq}_{\rho \rightarrow \tau}(\ell, x, y) = 0 \iff \text{Le}_{\rho \rightarrow \tau}(\ell, x, y) = 0 \land \text{Le}_{\rho \rightarrow \tau}(\ell, y, x) = 0.$$  

Lemma 10 (Modification functionals). For any list of types $\vec{\sigma} = \sigma_1, \ldots, \sigma_k$ and type $\tau$ there exists a modification functional $\text{Md}_{\vec{\sigma} \rightarrow \tau} : 0, \vec{\sigma} \rightarrow \tau, \vec{\sigma} \rightarrow \tau$ such that

$$\text{val}_{\ell + 1}(\text{Md}_{\vec{\sigma} \rightarrow \tau}(\ell, F, \vec{a}, v)(\vec{b})) = \begin{cases} 
\text{val}_{\ell + 1}(v) & \text{if } \text{val}_{\ell + 1}(a_i) = \text{val}_{\ell + 1}(b_i) \text{ for } i = 1, \ldots, k \\
\text{val}_{\ell + 1}(F(\vec{b})) & \text{otherwise.}
\end{cases}$$

for all variable-free terms $F, \vec{a}, v, \vec{b}$ of proper types.

Proof. We prove the lemma by induction on the length of $\vec{\sigma}$. If $k = 1$ use the equality functional $\text{Eq}_{\sigma_1}$ to define $\text{Md}_{\vec{\sigma} \rightarrow \tau}$ as required. If $k > 1$, define

$$\text{Md}_{\vec{\sigma} \rightarrow \tau}(\ell, F, \vec{y}, v) = \text{Md}_{\sigma_1 \rightarrow \tau}(\ell, F, y_1, \text{Md}_{\sigma_2 \rightarrow \tau}(\ell, F(y_1), y_2, \ldots, y_k, v)).$$  

Theorem 4. Any relation decidable by a program is also definable in $\mathcal{D}$.  

Assume (vii), \(F\) given by Lemma 5. Further, define \(x\) for all but finitely many values of \(C\) functionals given by Lemma 7; \(\pi\) is the list functional given by Lemma 8; \(\tau\) is the almost everywhere constant function \(T\) be a variable of type \(\sigma_1, \ldots, \sigma_r \rightarrow 0\). Let

\[
F_{m}^{p}(\ell, X) = M_{d_{\sigma_{1}, \ldots, \sigma_{r} \rightarrow 0}}(\ell, X_{m}, X_{i_{1}}, \ldots, X_{i_{r}}, X_{m}(X_{i_{1}}, \ldots, X_{i_{r}}) \oplus_{\ell+1} + 1)
\]

and let \(F_{j}^{p}(\ell, X) = X_{j}\) for \(j \neq m\). The functional \(F_{m}^{p}\) is definable in \(D\) by Lemma 6 and Lemma 10 Assume \(P \equiv Q\); \(R\). Let

\[
F_{i}^{p}(\ell, X) = F_{i}^{q}(\ell, F_{i}^{q}(\ell, X), \ldots, F_{k}^{q}(\ell, X))
\]

for \(i = 1, \ldots, k\). Assume \(P \equiv \text{while} \ t \ Q\). Let \(X\) be a variable of type \(0 \rightarrow \tau\), and let \(F\) of type \(0, (0 \rightarrow \tau) \rightarrow (0 \rightarrow \tau)\) be defined by

\[
F(\ell, X) = \langle F_{i}^{q}(\ell, T_{\pi}^{\sigma_{i}}(X(C_{1}(\ell))), \ldots, T_{\pi}^{\sigma_{k}}(X(C_{k}(\ell)))), \ldots, F_{k}^{q}(\ell, T_{\pi}^{\sigma_{i}}(X(C_{1}(\ell))), \ldots, T_{\pi}^{\sigma_{k}}(X(C_{k}(\ell)))) \rangle
\]

where \(\langle \ldots \rangle\) is the list functional given by Lemma 8. \(T_{\pi}^{\sigma_{i}}\) are the transition functionals given by Lemma 7. \(C_{i}\) are the almost everywhere constant function given by Lemma 5. Further, define \(F' : 0, (0 \rightarrow \tau) \rightarrow (0 \rightarrow \tau)\) such that

\[
F'(\ell, X) = \begin{cases} X & \text{if } T(\ell, X) = 0 \\ F(\ell, X) & \text{otherwise} \end{cases}
\]

where \(T(\ell, X) = T_{\pi}^{0}(X(C_{m}(\ell)))\) if the loop test \(t\) has the form \(t \equiv x_{m}\), and

\[
T(\ell, X) = T_{\pi}^{m}(X(C_{m}(\ell)))(T_{\pi}^{\sigma_{i_{1}}}(X(C_{i_{1}}(\ell))), \ldots, T_{\pi}^{\sigma_{i_{r}}}(X(C_{i_{r}}(\ell))))
\]

if the loop test \(t\) has the form \(t \equiv x_{m}[x_{i_{1}}, \ldots, x_{i_{r}}]\). By Lemma 4 we can pick a type \(\pi\) such that the loop \(P \equiv \text{while} \ t \ Q\) terminates within \(|\pi|_{\max(\bar{x}, 1)+1} \) iterations (where \(\bar{x}\) is the input to the program). Define \(F'' : 0, (0 \rightarrow \tau) \rightarrow (0 \rightarrow \tau)\)
such that $F''(\ell, X) = \mathrm{It}_{\pi}^{0 \rightarrow \tau}(\ell, F'(\ell), X)$ where the iteration functional $\mathrm{It}_{\pi}^{0 \rightarrow \tau}$ is given by Lemma \ref{lemma:iteration_functional}. Finally, let

$$F''_i(\ell, X_1, \ldots, X_k) = \mathrm{Tr}_{\tau}^{\sigma_i}(F''(\ell, \langle X_1, \ldots, X_k \rangle))(C_i(\ell))$$

for $i = 1, \ldots, k$. \hfill \Box

**Theorem 5.** A relation is definable in $D$ if and only if it is decidable by a program.

**Proof.** This follows from the previous theorem and the following argument: Suppose $R$ is definable in $D$. Then there exists a functional $F$ such that $F(\vec{v}) = 0$ iff $R(\vec{v})$ holds. A Turing machine can decide whether $R(\vec{v})$ holds by computing $\mathrm{val}_{\max(\vec{v}, 1)+1}(\vec{F}(\vec{x}))$. Obviously the Turing machine will work in space $|\sigma|_{\max(\vec{v}, 1)+1}$ for some type $\sigma$. Thus, $R$ belongs to the Ritchie hierarchy $\mathcal{R}$. Thus, $R$ is decidable by a program. \hfill \Box

If we adjust the notion of definability in $D$, and say that a function $f$ is definable in $D$ when for some type $\sigma$ there exists a functional $F : \overrightarrow{0} \rightarrow \sigma$ such that $f(\vec{v}) = \mathrm{val}_{\max(\vec{v}, 1)+1}(\vec{F}(\vec{x}))$, then we have the following result: A number-theoretic function is definable in $D$ if and only if it is computable by a program. (The proof is similar to the proof of Theorem \ref{thm:definability_of_relations}.)

**References**

Pebble Games on Trees

Łukasz Krzeszczakowski*

Institute of Informatics, Warsaw University, Poland
lkr@mimuw.edu.pl

Abstract. It is well known that two structures \( A \) and \( B \) are indistinguishable by sentences of the infinitary logic with \( k \) variables \( L^k_{\infty \omega} \) iff Duplicator wins the Barwise game on \( A \) and \( B \) with \( k \) pebbles. The complexity of the problem who wins the game is in general unknown if \( k \) is a part of the input. We prove that the problem is in \( \text{PTIME} \) for some special classes of structures such as finite directed trees and infinite regular trees. More specifically, we show an algorithm running in time \( \log(k)(|A| + |B|)^{O(1)} \).

The algorithm for regular trees is based on a characterization of the winning pairs \((A, B)\) which is valid also for a more general case of (potentially infinite) rooted trees.

1 Introduction

We say that two structures over the same signature are \( \text{FO}_k \)-equivalent iff they satisfy the same formulas of first order logic of quantifier depth at most \( k \). Similarly two structures are \( \text{L}_{\infty \omega}^k \)-equivalent iff they satisfy the same \( \text{L}_{\infty \omega} \) (for finite structures just first order logic) formulas with at most \( k \) variables.

It turns out that \( \text{FO}_k \)-equivalence and \( \text{L}_{\infty \omega}^k \)-equivalence can be expressed in terms of model-theoretic games. Indeed two structures \( A \) and \( B \) are \( \text{FO}_k \)-equivalent iff Duplicator wins \( k \)-moves Ehrenfeucht-Fraïssé game on \( A \) and \( B \); we denote such a game by \( \text{EF}_k(A, B) \). Similarly two structures \( A \) and \( B \) are \( \text{L}_{\infty \omega}^k \)-equivalent iff Duplicator wins \( k \)-pebbles Barwise game on \( A \) and \( B \); we denote such a game by \( \text{Pebble}_k(A, B) \) (J. Barwise introduced these games in [1]). The proofs of these results can be found in [3].

We call the problem of deciding who wins the game \( \text{EF}_k(A, B) \) for two finite structures \( A \) and \( B \) and the number \( k \), the \textit{EF problem} and the problem of deciding who wins the game \( \text{Pebble}_k(A, B) \) – the \textit{pebble problem}. We are interested in determining the complexity of above problems. Martin Grohe in [2] has shown that if the number of pebbles is fixed, the pebble problem is complete for \( \text{PTIME} \). If the number of moves is fixed, then it is easy to see that EF problem is in \( \text{LOGSPACE} \). But Elena Pezzoli has shown in [4] and [5] that if the number of moves is a part of the input then the EF problem is \( \text{PSPACE} \) complete. If the number of pebbles is a part of the input then it can be easily seen that the problem is at least as hard as the Graph Isomorphism problem. Indeed, two

* Supported by Polish KBN grant No. 7 T11C 027 20
graphs with $k$ vertices are isomorphic if Duplicator wins a $k$-pebble Barwise game on these graphs. On the other hand it is easy to see that the problem is in EXPTIME, but the exact complexity is to our knowledge open.

We are mostly interested in finding classes of structures in which the pebble problem is efficiently computable. For a class $\text{CLASS}$ of structures we denote by $\text{Pebble}[\text{CLASS}]$ the following problem: given two structures $A, B \in \text{CLASS}$ and $k \in \mathbb{N}$, decide who wins $\text{Pebble}_k(A, B)$. In this paper we show that the problems $\text{Pebble}[\text{DTREE}]$ and $\text{Pebble}[\text{REGTREE}]$ are in $\text{PTIME}$, where DTREE is the class of finite directed trees and REGTREE is the class of regular trees. Consequently, we can decide in polynomial time if two directed trees, respectively two regular trees, are indistinguishable by $L_{\infty \omega}$ formulas.

2 Preliminaries

2.1 Graphs and Trees

Let $E$ be a binary relation symbol. An (undirected) graph is an $\{E\}$-structure $G = \langle G, E^G \rangle$, where $E^G$ is irreflexive and symmetric relation. A directed graph is an $\{E\}$-structure $G = \langle G, E^G \rangle$, where $E^G$ is arbitrary. Then, its underlying graph $G^u$ is given by $G^u = \langle G, \{(a, b) : a \neq b, E^G(a, b) \text{ or } E^G(b, a)\}\rangle$. A path in a graph $G$ is a finite or infinite sequence $v_1, v_2, \ldots$ of different vertices in $G$ such that $E^G(v_i, v_{i+1})$ for $i = 1, 2, \ldots$. A path in a directed graph is a path in its underlying graph (that is, it need not respect direction of edges).

An (undirected) tree is a connected and acyclic graph. A directed tree is a directed graph whose underlying graph is a tree.

For a directed tree $T$ and a vertex $t \in T$ we have a partial order $\leq^T_t$ on $T$ defined by:

$$v \leq^T_t u \text{ iff } v \text{ appears on the (unique) path from } t \text{ to } u$$

A rooted tree is a directed tree $T$ such that $E^T$ is irreflexive and there is an (unique) vertex $t \in T$ such that the transitive and reflexive closure of $E^T$ coincides with $\leq^T_t$. We denote this vertex $t$ by $r^T$ and call it a root of $T$. We say that $v$ is an (unique) parent of $w$ and $w$ is a child of $v$ if $E^T(v, w)$. The set of children of $v$ is denoted by $\text{children}^T(v)$. We denote by $\text{subtree}^T(v)$ the substructure of $T$ induced by $\{w : v \leq^T_w\}$. We denote by $\text{height}^T(v)$ the height of the subtree of $T$ rooted at $v$, i.e. the length of the longest finite path in $\text{subtree}^T(v)$ started with $v$ (if it exists).

For a rooted tree $T$ we denote by $\text{Dom}(T)$ the set $\{\text{subtree}(t) : t \in T\}$. It is important to note, that we do not distinguish between the isomorphic structures. Then a regular tree is a rooted tree $T$ such that the set $\text{Dom}(T)$ is finite and additionally for every $v \in T$ the set $\{w : E^T(v, w)\}$ is finite.

A pointed directed tree (in short pointed tree) is a structure $T = \langle T, E^T, c^T \rangle$, where $c$ is a constant symbol, such that $S = \langle T, E^T \rangle$ is a directed tree. We denote such a structure by $S(s)$, where $s = c^T$.  

\footnote{The author thanks an anonymous referee for pointing this observation.}

Notice that, for a pointed tree $T(t)$ there exists exactly one rooted tree $S$ such that $T^u = S^u$ and $r^S = t$. Then we write $\text{children}^{T(t)}(v)$ for the set of children of $v$ in $S$ and $\text{height}^{T(t)}(v)$ for the height of the subtree of $S$ rooted at $v$. Furthermore, we denote by $\text{subtree}^{T(t)}(v)$ a pointed tree $T'$, where $T' = \{w \in T : v$ is in the path from $c^T$ to $w\}$, $c^T' = v$ and $E^{T'} = E^T \cap T' \times T'$.

We denote the class of regular trees by $\text{REGTREE}$, the class of finite directed trees by $\text{DTREE}$ and the class of finite pointed directed trees by $\text{PDTREE}$ (recall that we don’t distinguish between isomorphic structures).

When a rooted tree (or a pointed tree) is clear from the context, we omit the superscripts in the denotations $\text{children}(t)$, $\text{subtree}(t)$ and $\text{height}(t)$.

### 2.2 Pebble Games

Let $k \in \mathcal{N}$ and $A, B$ be structures over the same signature. A configuration of the game $\text{Pebble}_k(A, B)$ is a function $c : \{1, ..., k\} \rightarrow A \times B \cup \{\ast\}$, where $\ast$ is a fixed element not in $A \times B$. If $c(i) = (v, w)$ for some $i \in \{1, ..., k\}$, we say that the vertices $v \in A$ and $w \in B$ are marked, or that they are marked by the pebble $p_i$. Additionally we denote $v$ and $w$ by $p_i^A$ and $p_i^B$ respectively. For a fixed configuration $c$, we denote by $\text{Peb}$ the set $\{i : 1 \leq i \leq k, c(i) \neq \ast\}$. The initial configuration of $\text{Pebble}_k(A, B)$ is $c$ such that $c(i) = \ast$ for $i \in \{1, ..., k\}$.

The game is played by two players: Spoiler and Duplicator. The play consists of (potentially) infinite sequence of rounds. In each round Spoiler selects a structure, $A$ or $B$, an element $v \in A$ (or $w \in B$) and $i \in \{1, ..., k\}$. In such a case we say that Spoiler places a pebble $p_i$ on $v$ (on $w$). Then Duplicator selects an element $w \in B$ (or $v \in A$) and we say that he answers $(p_i, w)$ (or $(p_i, v)$), or that the answer of Duplicator is $w$ (or $v$). After the round the configuration is changed by $c(i) := (v, w)$.

Duplicator wins the game if after each round the configuration induces a partial isomorphism between $A$ and $B$, i.e. the relation $\bigcup_{i \in \text{Peb}} c(i) \subseteq A \times B$ is a partial isomorphism.

For a fixed configuration of a play of $\text{Pebble}_k(T_1, T_2)$, where $T_1, T_2 \in \text{REGTREE}$ we denote by $\text{spread}^{T_i}$ for $i \in \{1, 2\}$ a minimal substructure $S$ of $T_i$ such that $S$ includes root of $T_i$ and all marked vertices of $T_i$.

For a fixed configuration of a play of $\text{Pebble}_k(T_1, T_2)$, where $T_1, T_2 \in \text{DTREE}$ and $t_1 \in T_1$, $t_2 \in T_2$, we denote by $\text{spread}^{T_i(t_i)}$ for $i \in \{1, 2\}$ a pointed tree $S$ such that $S = \{v \in T_i : v$ is in the path from $t_i$ to $p_j^{T_i}$ for some pebble $p_j\}$, $c^S = t_i$ and $E^S = E^{T_i} \cap S \times S$.

### 3 Pebble Games on Directed Trees

#### 3.1 Characterisation of Pebble Problem on Directed Trees

Let $k \in \mathcal{N}$. We define the relation $=_{k} \in \mathcal{N} \times \mathcal{N}$ by:

$$m =_{k} n \iff m, n \geq k \text{ or } m = n$$
Definition 1 Let $S(s), T(t) \in PDTREE$, let $R$ be an equivalence relation over $PDTREE$ and $k \in \mathbb{N}$. We denote by $\Delta_k(R, S(s), T(t))$ the following condition:

- $E^S(s, s)$ iff $E^T(t, t)$, and
- for every equivalence class $C \in PDTREE/R$:
  \[
  | \{ s' : E^S(s, s'), \text{ not } E^S(s', s) \text{ and } \text{subtree}(s') \in C \} | =_k \\
  | \{ t' : E^T(t, t'), \text{ not } E^T(t', t) \text{ and } \text{subtree}(t') \in C \} |, \quad \text{and}
  \]

\[
| \{ s' : E^S(s, s'), \text{ not } E^S(s', s) \text{ and } \text{subtree}(s') \in C \} | =_k \\
| \{ t' : E^T(t, t'), \text{ not } E^T(t', t) \text{ and } \text{subtree}(t') \in C \} |, \quad \text{and}
\]

\[
| \{ s' : E^S(s, s'), E^S(s', s), \text{subtree}(s') \in C \} | =_k \\
| \{ t' : E^T(t, t'), E^T(t', t), \text{subtree}(t') \in C \} |, \quad \text{and}
\]

For any set $X$ let $\text{Equiv}(X)$ be the set of equivalence relations on $X$.

Definition 2 For $k \in \mathbb{N}$ let $\Gamma_k : \text{Equiv}(PDTREE) \rightarrow \text{Equiv}(PDTREE)$ be the following mapping:

\[
\Gamma_k(R) = \{ (S, T) : \Delta_k(R, S, T) \}
\]

The definition is correct ($\Gamma_k(R)$ is an equivalence relation for all $k$ and $R$) since $=_k$ is an equivalence relation for all $k$.

Lemma 1 For all $k \in \mathbb{N}$ the mapping $\Gamma_k$ is monotonic.

Proof. Let $R_1, R_2 \in \text{Equiv}(PDTREE)$ and $(S, T) \in PDTREE$. Assume $R_1 \subseteq R_2$ and $(S, T) \in \Gamma_k(R_1)$. Let $X \in PDTREE/R_2$. Since $R_1 \subseteq R_2$, then $X = \bigcup Y$ for some $Y \subseteq PDTREE/R_1$. Since $\Delta_k(R_1, S, T)$, then for all $Z \in Y$

\[
| \{ s : E^S(r^S, s), \text{ not } E^S(s, r^S) \text{ and } \text{subtree}(s) \in Z \} | =_k \\
| \{ t : E^T(r^T, t), \text{ not } E^T(t, r^T) \text{ and } \text{subtree}(t) \in Z \} |
\]

and obviously

\[
| \{ s : E^S(r^S, s), \text{ not } E^S(s, r^S) \text{ and } \text{subtree}(s) \in X \} | =_k \\
| \{ t : E^T(r^T, t), \text{ not } E^T(t, r^T) \text{ and } \text{subtree}(t) \in X \} |
\]

Similar conclusion we get for two other equalities from definition of $\Delta_k$.

Finally $\Delta_k(R_2, S, T)$ and $(S, T) \in \Gamma_k(R_2)$. □

Lemma 2 For any set $A$, the set $\text{Equiv}(A)$ of equivalence relations on $A$ ordered by the inclusion relation is a complete lattice.
Proof. It is enough to show that any subset of $\text{Equiv}(A)$ has a greatest lower bound. Let $X \subseteq \text{Equiv}(A)$, and $R = \bigcap X$. Obviously $R$ is an equivalence relation. We show that $R$ is a greatest lower bound of $X$. Obviously $R$ is an lower bound, since $R \subseteq Y$ for all $Y \in X$. Let $R' \in \text{Equiv}(A)$ be an lower bound of $X$. Then for all $Y \in X$, $R' \subseteq Y$. Then $R' \subseteq R$. □

Note that a least upper bound in $\text{Equiv}(A)$ need not coincide with the set-theoretic union.

By Knaster-Tarski fixed point theorem we get:

**Lemma 3** For all $k \in \mathcal{N}$, $\Gamma_k$ has a least and a greatest fixed point. □

Let $k \in \mathcal{N}$ be fixed. Let $L_0, L_1, \ldots$ and $U_0, U_1, \ldots$ be sequences of equivalence relations on $\text{PDTREE}$ defined by:

- $L_0 := \text{Id}(\text{PDTREE}), U_0 = \text{PDTREE}^2$
- $L_{i+1} = \Gamma_k(L_i), U_{i+1} = \Gamma_k(U_i)$ for $i \in \mathcal{N}$

**Lemma 4** $L_0, L_1, \ldots$ is increasing and $U_0, U_1, \ldots$ is decreasing. □

Denote $\bigcup\{L_i : i \in \mathcal{N}\}$ by $L_\infty$ and $\bigcap\{U_i : i \in \mathcal{N}\}$ by $U_\infty$.

**Lemma 5** Let $R$ be a fixed point of $\Gamma_k$. Then $L_\infty \subseteq R$ and $R \subseteq U_\infty$. □

**Lemma 6** $L_\infty = U_\infty$, consequently $\Gamma_k$ has exactly one fixed point.

Proof. Let $\text{PDTREE}_{h}$ be a set of pointed trees of height at most $h$. By induction w.r.t. $i$ we show that $L_{i+1} \cap (\text{PDTREE})^2 = U_{i+1} \cap (\text{PDTREE})^2$:

- $L_1 \cap (\text{PDTREE})^2 = \text{Id}(\text{PDTREE}) \cap (\text{PDTREE})^2 = U_0 \cap (\text{PDTREE})^2$.
- (Warning $L_0 \cap (\text{PDTREE})^2 = \{(a,a), (b,b)\} \neq \{(a,a), (b,b), (a,b), (b,a)\} = U_0 \cap (\text{PDTREE})^2$, where $a = \langle\{0\}, E^a = i^a = 0\rangle$ and $b = \langle\{0\}, E^b = \{0,0\}, c^b = 0\rangle$)
- If $L_{i+1} \cap (\text{PDTREE})^2 = U_{i+1} \cap (\text{PDTREE})^2$, then for all $S, T \in \text{PDTREE}_{i+1}$, $\Delta_k(L_{i+1}, S, T)$ iff $\Delta_k(U_{i+1}, S, T)$, and then $L_{i+2}(S, T)$ iff $U_{i+2}(S, T)$, so $L_{i+2} \cap (\text{PDTREE})^2 = U_{i+2} \cap (\text{PDTREE})^2$

Since $L_i \subseteq U_i$ for all $i \in \mathcal{N}$ and by lemma 3 we get that $L_{i+1} \cap (\text{PDTREE})^2 = L_\infty \cap (\text{PDTREE})^2$ and $U_{i+1} \cap (\text{PDTREE})^2 = U_\infty \cap (\text{PDTREE})^2$ for all $i$. i.e. $L_\infty \cap (\text{PDTREE})^2 = U_\infty \cap (\text{PDTREE})^2$ for all $i$. Since for every $S \in \text{PDTREE}$ there is $i \in \mathcal{N}$ such that $S \in \text{PDTREE}_i$ we get $L_\infty = U_\infty$. □

By lemma 3 and 5 we get:

**Lemma 7** For all $k \in \mathcal{N}$, $\Gamma_k$ has exactly one fixed point. □

Let us denote this unique fixed point by $\equiv_k$.

**Lemma 8** Let $k \geq 3$, $S, T \in \text{DTREE}$ and $x_1, y_1 \in S, x_2, y_2 \in T$ such that $E^S(x_1, y_1)$ and $E^T(x_2, y_2)$.

Assume that in the play of $\text{Pebble}_k(S, T)$ the vertices $x_1$ and $x_2$ are marked by pebble $p_i$ and $y_1$, $y_2$ by $p_j$. Assume also that the length of a longest path which begins with the vertices $x_1, y_1$ is different from the length of a longest path which begins with $x_2, y_2$. Then Spoiler wins the play.
Proof. By three pebbles - \( p_i, p_j \) and any third - Spoiler can “pass” across the longer path. □

**Theorem 1** Let \( S, T \in DTREE \) and \( k \geq 3 \). Duplicator wins Pebble\(_k\)(\( S, T \)) iff there are \( s \in S, t \in T \) such that \( S(s) \equiv_k T(t) \).

Proof.

\( \Leftarrow \) Suppose that for some \( s \in S \) and \( t \in T \) \( S(s) \equiv_k T(t) \).

**Lemma 9** Suppose that in the play of Pebble\(_k\)(\( S, T \)) the following condition is satisfied:

\( (*) \) There is an isomorphism \( f : spread^{S(s)} \to spread^{T(t)} \) such that \( f(p_i^S) = p_i^T \) for all \( i \in Peb \) and \( subtree^{S(s)}(x) \equiv_k subtree^{T(t)}(f(x)) \) for all \( x \in spread^{S(s)} \).

Then for every move of Spoiler there exists a move of Duplicator such that the requirement \( (*) \) is preserved.

Proof. Assume \(|Peb| < k\) (if not, Spoiler removes one pebble and then requirement \( (*) \) is also preserved). Assume that Spoiler places a pebble \( p_i \) for some \( i \notin Peb \) on \( x \in S \). If \( x \in spread^{S(s)} \) then Duplicator answers \( (p_i, f(x)) \).

Obviously the requirement \( (*) \) is preserved (by the same isomorphism \( f \)). Assume then that \( x \notin spread^{S(s)} \). Let \( a \in S \) be the last vertex in the path from \( s \) to \( x \) that belongs to \( spread^{S(s)} \).

Let \( w \in T \) be a vertex computed by this algorithm:

1. \( v := a, w := f(a) \)
2. while \( v \neq x \)
   - \( v := c_v, \) where \( c_v \in children^{S(s)}(v) \) and \( c_v \) is in the path from \( a \) to \( x \).
   - \( w := c_w, \) where \( c_w \in children^{T(t)}(w) \) such that:
     - \( E^S(v, c_v) \text{ iff } E^T(w, c_w) \) and \( E^S(c_v, v) \text{ iff } E^T(c_w, w) \), and
     - \( subtree^{S(s)}(c_v) \equiv_k subtree^{T(t)}(c_w) \), and
     - \( c_w \notin spread^{T(t)} \).

There exists a vertex \( c_w \in children^{T(t)}(w) \) in a loop which satisfy \( a. \) and \( b. \) because \( subtree^{S(s)}(v) \equiv_k subtree^{T(t)}(w) \) is an invariant of this algorithm (and because \( \equiv_k \) is a fixed point of \( I_k \)). Additionally we can choose a vertex satisfying \( a. \), \( b. \), and \( c. \) because the invariants of the algorithm are:

- \( c_v \notin spread^{S(s)} \)
- every vertex in \( T \) has at most \( k-1 \) children in \( spread^{T(t)} \) (since \( spread^{T(t)} \) is spread by at most \( k-1 \) pebbles)

If Duplicator answers \( w \), then the requirement \( (*) \) will be preserved (\( spread^{S(s)} \) and \( spread^{T(t)} \) will be extended by all values of \( c_v \) and \( c_w \) from the algorithm and \( f \) will be extended by: \( f(c_v) = c_w \) for every \( c_v \) and \( c_w \) computed by the algorithm in the same turn of the loop). □

Initial configuration of Pebble\(_k\)(\( S, T \)) obviously satisfies the requirement \( (*) \). When the requirement \( (*) \) is satisfied then the mapping \( p_i^S \mapsto p_i^T \) is a partial isomorphism. So, by lemma 9 Duplicator wins the game.
\[ \implies \text{Let } s \in S \text{ be a middle vertex of the longest path in } S \text{ (any longest path; if its length is odd, then let it be any of the two middle vertices). For } x \in S \text{ we write } \text{children}(x), \text{height}(x) \text{ and } \text{subtree}(x) \text{ instead of } \text{children}^{S(s)}(x), \text{height}^{S(s)}(x) \text{ and } \text{subtree}^{S(s)}(x) \text{ respectively. Let } h := \text{height}(s). \text{ Notice that there exist } s_1, s_2 \in \text{children}(s), s_1 \neq s_2 \text{ such that } \text{height}(s_1) \geq h - 2 \leq \text{height}(s_2). \]

Let Spoiler place a pebble \( p_1 \) on \( s \). Let \( t \in T \) be an answer of Duplicator. Similarly, for \( x \in T \) we write \( \text{children}(x), \text{height}(x) \text{ and } \text{subtree}(x) \) instead of \( \text{children}^{T(t)}(x), \text{height}^{T(t)}(x) \text{ and } \text{subtree}^{T(t)}(x) \) respectively. If \( \text{height}(t) < h \) then Spoiler places a pebble \( p_2 \) on \( s' \in \text{children}(s) \) such that \( \text{height}(s') = h - 1 \) and then he wins by lemma \( \Box \). If \( \text{height}(t) > h \), Spoiler plays similarly. Assume then that \( \text{height}(t) = h \). If there is no \( t_1, t_2 \in \text{children}(t) \), \( t_1 \neq t_2 \) such that \( \text{height}(t_1) \geq h - 2 \leq \text{height}(t_2) \) then Spoiler places pebbles \( p_2 \) and \( p_3 \) on \( s_1 \) and \( s_2 \) respectively, and then he also wins by lemma \( \Box \). Assume then that such a vertices exist (i.e. \( t \) is a middle vertex of the longest path in \( T \)).

Assume that \( S(s) \neq_k T(t) \).

Let Spoiler play according to this interactive algorithm:

1. \( v := s, \ w := t \).
2. If not \( E^S(v, v) \) iff \( E^T(w, w) \) then Spoiler wins.
3. Let \( X \subseteq S, \ Y \subseteq T \) be such that:
   \[ X = \{ x \in \text{children}(v) : \text{cond}^S(v, x) \text{ and } \text{subtree}(x) \in C \}, \]
   \[ Y = \{ y \in \text{children}(w) : \text{cond}^T(w, y) \text{ and } \text{subtree}(y) \in C \} \]
   for some \( C \in \text{PDTREE} \) — one of the following conditions: 1. \( E^A(a, b) \) and not \( E^A(b, a) \), 2. \( E^A(b, a) \) and not \( E^A(a, b) \), 3. \( E^A(a, b) \) and \( E^A(b, a) \).
   \[ |X| \neq_k |Y| \]

Without loss of generality we can assume that \( |X| < |Y| \). Then obviously \( |X| < k \). Now Spoiler makes a sequence of moves \( (w_1, q_1), \ldots, (w_{|X|+1}, q_{|X|+1}) \), where \( w_1, \ldots, w_{|X|+1} \in Y, \ w_i \neq w_j \) for \( i \neq j \) and \( q_1, \ldots, q_{|X|+1} \in \{ p_1, \ldots, p_k \} \) is any sequence of different pebbles such that neither \( q_1 \) nor \( q_2 \) are placed on \( w \).

Let \( (v_1, q_1), \ldots, (v_{|X|+1}, q_{|X|+1}) \) be an answer of Duplicator. If any of \( v_j \) is not a child of \( v \) or \( v_j = v_i \) for \( i \neq j \) then Duplicator loses the game.

Let \( j \in \{ 1, \ldots, |X| + 1 \} \) be such that \( v_j \notin X \) (such \( j \) obviously exists).

Let \( v := v_j \) and \( w := w_j \) and repeat this step of algorithm.

Notice that ever at the beginning of step 2. of the algorithm we have:
- Vertices \( v \in S \) and \( w \in T \) are marked by the same pebble.
- \( \text{subtree}(v) \neq_k \text{subtree}(w) \)

and by that Spoiler wins the game when he plays according to above algorithm (since a relation of being a child in a tree is well founded). \( \Box \)

### 3.2 Complexity of Pebble Problem on Directed Trees

**Theorem 2** The problem Pebble \( [\text{DTREE}] \) is in \( \text{PTIME} \). More precisely, there exists algorithm that for \( k \in \mathcal{N} \) and \( S, T \in \text{DTREE} \) solves the problem Pebble \( k(S, T) \) in time \( O(\log(k) \times (|S| + |T|)^3) \).
Proof.
The algorithm of deciding whether Duplicator wins $Pebble_k(S, T)$:

- For every middle vertex $s$ of the longest path in $S$ and every middle vertex $t$ of the longest path in $T$ check whether $S(s) \equiv_k T(t)$.

The algorithm of checking if $S(s) \equiv_k T(t)$:

1. For every $v \in \text{children}(s) \cup \text{children}(t)$, $l_v^{S(s)} := 0$ and $l_v^{T(t)} := 0$.
2. For every $v \in \text{children}^X(x)$ and $w \in \text{children}^Y(y)$, where $(x, X), (y, Y) \in \{(s, S(s)), (t, T(t))\}$ such that $E^X(x, v)$ iff $E^Y(y, w)$ and $E^X(v, x)$ iff $E^Y(w, y)$ check recursively whether $\text{subtree}(v) \equiv_k \text{subtree}(w)$ and if so then increase the counters $l_v^X$ and $l_v^Y$ by 1 (any counter increase only if it is less then $k$).
3. If for every $v \in \text{children}(s) \cup \text{children}(t)$, $l_v^{S(s)} = k \land l_v^{T(t)}$ then $S \equiv_k T$. Otherwise $S(s) \not\equiv_k T(t)$.

In any directed tree there may be at most two middle vertices of the longest path. Finding these vertices in $S$ and $T$ has complexity $O((|S| + |T|)^2)$. Complexity of checking if $S(s) \equiv_k T(t)$ is estimated by $O((|S| + |T|)^3 \ast \log(k))$, since for every pair of vertices in $S \cup T$ algorithm is executed at most once (number of such pairs is $(|S| + |T|)^2$) and complexity of one execution is estimated by $O((|S| + |T|) \ast \log(k))$ (without calculating of recursive executions; the factor $\log(k)$ appears because the counters $l_v^X$ are less then $k + 1$ and can be represented by $O(\log(k))$ bits).

Let $R \subseteq \text{PDTREE}^2$ be a relation computed by the algorithm. From the structure of the algorithm after step 2, we have:

$$l_v^X = |\{w \in \text{children}^X(x) : R(\text{subtree}(v), \text{subtree}(w)) \text{ and } E^Y(y, v) \text{ iff } E^X(x, w) \text{ and } E^Y(v, y) \text{ iff } E^X(w, x)\}|$$

for all $v \in \text{children}(y)$, where $(x, X), (y, Y) \in \{(s, S(s)), (t, T(t))\}$.

So $R$ is a fixed point of $\Gamma_k$ and then $R = \equiv_k. \square$

4 Pebble Problem on Rooted Trees

4.1 Characterisation of Pebble Problem on Rooted Trees

For a rooted tree $S$ and $v \in S$ we denote by $\text{childtrees}^S(v)$ (or shortly by $\text{childtrees}(v)$) the set $\{\text{subtree}^S(w) : w \in \text{children}^S(v)\}$.

Definition 3 Let $S, T \in \text{RTREE}$ and $k \in N$. Let $R \subseteq \text{RTREE}^2$ be an equivalence relation. We denote by $\Delta_k(R, S, T)$ the following condition:

- For all $X \in \text{RTREE}/R$:
  - $|\text{childtrees}(r^S) \cap X| = k \land |\text{childtrees}(r^T) \cap X|$
Definition 4 For $k \in \mathbb{N}$ let $\Gamma_k : \text{Equiv}(\text{RTREE}) \to \text{Equiv}(\text{RTREE})$ be the following mapping:

$$\Gamma_k(R) = \{(S, T) : \Delta_k(R, S, T)\}$$

The definition is correct ($\Gamma_k(R)$ is an equivalence relation for all $k$ and $R$) since $=_k$ is an equivalence relation for all $k$.

Lemma 10 For all $k \in \mathbb{N}$ the mapping $\Gamma_k$ is monotonic.

Proof. Let $R_1, R_2 \in \text{Equiv}(\text{RTREE})$ and $S, T \in \text{RTREE}$. Assume $R_1 \subseteq R_2$ and $(S, T) \in \Gamma_k(R_1)$. Let $X \in \text{RTREE}/R_2$. Since $R_1 \subseteq R_2$, then $X = \bigcup Y$ for some $Y \subseteq \text{RTREE}/R_1$. Since $\Delta_k(R_1, S, T)$, then for all $Z \in Y$

$$|\text{childtrees}(r^S) \cap Z| =_k |\text{childtrees}(r^T) \cap Z|$$

and obviously

$$|\text{childtrees}(r^S) \cap X| =_k |\text{childtrees}(r^T) \cap X|$$

Finally $\Delta_k(R_2, S, T)$ and $(S, T) \in \Gamma_k(R_2)$. □

By Knaster-Tarski fixed point theorem, for all $k \in \mathbb{N}$, $\Gamma_k$ has a greatest fixed point. Let us denote it by $\equiv_k$.

Theorem 3 Let $k \geq 3$ and $S, T \in \text{RTREE}$. Duplicator wins Pebble$_k(S, T)$ iff $S \equiv_k T$.

Proof. Since $\equiv_k$ is the greatest fixed point of $\Gamma_k$ then by Knaster-Tarski fixed point theorem, $\equiv_k = \bigvee \{\alpha \in \text{Equiv}(\text{RTREE}) : \alpha \subseteq \Gamma_k(\alpha)\}$. So, to prove that $R = \equiv_k$ for some equivalence relation $R$, it is sufficient to show that $\equiv_k \subseteq R$ and $R \subseteq \Gamma_k(R)$.

Let $R = \{(S, T) : \text{Duplicator wins Pebble}_k(S, T)\}$. Obviously $R$ is an equivalence relation.

• $\equiv_k \subseteq R$

Suppose that $S \equiv_k T$.

Lemma 11 Suppose that in the play of Pebble$_k(S, T)$ the following condition is satisfied:

$(\star)$ There is an isomorphism $f : \text{spread}^S \to \text{spread}^T$ such that $f(p_i^S) = p_i^T$ for all $i \in \text{Peb}$ and subtree$^S(x) \equiv_k$ subtree$^T(f(x))$ for all $x \in \text{spread}^S$.

Then for every move of Spoiler there exists a move of Duplicator such that the requirement $(\star)$ is preserved.
Proof. Assume \(|\text{Peb}| < k\) (if not, Spoiler removes one pebble and then requirement (\(\ast\)) is also preserved). Assume that Spoiler places a pebble \(p_i\) for some \(i \notin \text{Peb}\) on \(x \in S\). If \(x \in \text{spread}^S\) then Duplicator answers \((p_i, f(x))\). Obviously the requirement (\(\ast\)) is preserved (by the same isomorphism \(f\)).

Assume then that \(x \notin \text{spread}^S\). Let \(a \in S\) be the last vertex in the path from \(r^S\) to \(x\) that belongs to \(\text{spread}^S\).

Let \(w \in T\) be a vertex computed by this algorithm:

1. \(v := a, w := f(a)\)
2. while \(v \neq x\) do
   - \(v := c_v\), where \(c_v \in \text{children}^S(v)\) and \(c_v\) is in the path from \(a\) to \(x\).
   - \(w := c_w\), where \(c_w \in \text{children}^T(w)\) such that:
     a. \(\text{subtree}^S(c_v) \equiv_k \text{subtree}^T(c_w)\), and
     b. \(c_w \notin \text{spread}^T\).

There exists a vertex \(c_w \in \text{children}^T(w)\) in a loop which satisfies a. because \(\text{subtree}^S(v) \equiv_k \text{subtree}^T(w)\) is an invariant of this algorithm (and because \(\equiv_k\) is a fixed point of \(\Gamma_k\)). Additionally we can choose a vertex satisfying both a and b. because the invariants of the algorithm are:

- \(c_v \notin \text{spread}^S\)
- every vertex in \(T\) has at most \(k - 1\) children in \(\text{spread}^T\) (since \(\text{spread}^T\) is spread by at most \(k - 1\) pebbles)

If Duplicator answers \(w\), then the requirement (\(\ast\)) will be preserved (\(\text{spread}^S\) and \(\text{spread}^T\) will be extended by all values of \(c_v\) and \(c_w\) from the algorithm and \(f\) will be extended by: \(f(c_v) = c_w\) for every \(c_v\) and \(c_w\) computed by the algorithm in the same turn of the loop). \(\Box\)

Initial configuration of \(\text{Pebble}_k(S, T)\) obviously satisfies the requirement (\(\ast\)). When requirement (\(\ast\)) is satisfied then the mapping \(p_i^S \mapsto p_i^T\) is a partial isomorphism. So, by lemma \(\Box\) Duplicator wins the game.

- \(R \subseteq \Gamma_k(R)\)

Assume that Duplicator wins \(\text{Pebble}_k(S, T)\). Moreover suppose that \((S, T) \notin \Gamma_k(R)\), i.e. there is \(X \in \text{RTREE}/R\) such that \(|\text{childtrees}(r^S) \cap X| \neq_k |\text{childtrees}(r^T) \cap X|\). Let us denote \(\{v \in \text{children}(r^S) : \text{subtree}(v) \in X\}\) by \(A\) and \(\{w \in \text{children}(r^T) : \text{subtree}(w) \in X\}\) by \(B\). Without loss of generality we can assume that \(|A| < |B|\), hence \(|A| < k\). Suppose Spoiler puts \(|A| + 1\) different pebbles on different vertices from \(B\). Then Duplicator has to answer by putting \(|A| + 1\) pebbles on different children of \(r^S\). Let \(v \in \text{children}(r^S), w \in \text{children}(r^T)\) be the vertices marked by the same pebble, such that \((\text{subtree}(v), \text{subtree}(w)) \notin R\). It means that Spoiler has a winning strategy in \(\text{Pebble}_k(\text{subtree}(v), \text{subtree}(w))\). We can assume that in this strategy always some pebble remains present in both trees and that the game starts from positions \(v\) and \(w\) marked by the same pebble. Then this strategy expanded to \((S, T)\) is a winning strategy of Spoiler – contradiction. It shows that \((S, T) \notin \Gamma_k(R)\), that is \(R \subseteq \Gamma_k(R)\). \(\Box\)

### 4.2 Complexity of Pebble Problem on Regular Trees

Let \(k \in \mathcal{N}\) be fixed. Let \(R_0, R_1, ...\) be a sequence of equivalence relations on \(\text{REGTREE}\) defined by:
- \( R_0 = \text{REGTREE}^2 \)
- \( R_{i+1} = \Gamma_k(R_i) \)

**Lemma 12** \( R_0, R_1, \ldots \) is decreasing.

Proof. By induction w.r.t. \( i \):

- \( R_1 \subseteq \text{REGTREE}^2 = R_0 \)
- If \( R_i \subseteq R_{i-1} \) then \( R_{i+1} = \Gamma_k(R_i) \subseteq \Gamma_k(R_{i-1}) = R_i \) (since \( \Gamma_k \) is monotonic).

Let us denote \( \bigcap \{ R_i : i \in \mathbb{N} \} \) by \( R_\infty \)

**Lemma 13** Let \( R \) be a fixed point of \( \Gamma_k \). Then \( R \subseteq R_\infty \).

Proof. By induction w.r.t. \( i \) we show that \( R \subseteq R_i \):

- \( R \subseteq \text{REGTREE}^2 = R_0 \)
- If \( R \subseteq R_i \) then \( R = \Gamma_k(R) \subseteq \Gamma_k(R_i) = R_{i+1} \) (since \( \Gamma_k \) is monotonic).

**Lemma 14** \( R_\infty \) is a fixed point of \( \Gamma_k \).

Proof. Since \( \Gamma_k \) is monotonic and \( R_\infty \subseteq R_i \) for all \( i \in \mathbb{N} \), then \( \Gamma_k(R_\infty) \subseteq \Gamma_k(R_i) = R_{i+1} \) i.e. for all \( i \geq 1 \) \( \Gamma_k(R_\infty) \subseteq R_i \). Since \( \Gamma_k(R_\infty) \subseteq \text{REGTREE}^2 = R_0 \) then \( \Gamma_k(R_\infty) \subseteq R_i \) for all \( i \in \mathbb{N} \) and then \( \Gamma_k(R_\infty) \subseteq R_\infty \).

Let \( (S, T) \in R_\infty \). Then for all \( i \in \mathbb{N} \) \( (S, T) \in R_i \) and then for all \( i \in \mathbb{N} \) \( (S, T) \in R_{i+1} = \Gamma_k(R_i) \) i.e. \( \Delta_k(R_i, S, T) \). Let \( i \in \mathbb{N} \) be such that \( R_i \cap (\text{Dom}(S) \cup \text{Dom}(T))^2 = R_\infty \cap (\text{Dom}(S) \cup \text{Dom}(T))^2 \). Then obviously \( \Delta_k(R_\infty, S, T) \) and then \( (S, T) \in \Gamma_k(R_\infty) \).

By lemmas \([13]\) and \([14]\) we obtain:

**Lemma 15** \( \equiv_k = R_\infty \).

Let \( S, T \in \text{REGTREE} \) be fixed. We denote \( \text{Dom}(S) \cup \text{Dom}(T) \) shortly by \( \text{Dom} \).

**Definition 5** Let \( A, B \in \text{Dom} \) and \( R \in \text{Equiv}(\text{Dom}) \). We denote by \( \Delta'_k(R, A, B) \) the following condition:

- For all \( X \in \text{Dom}/R \):
  - \( |\text{childtrees}(r^A) \cap X| =_k |\text{childtrees}(r^B) \cap X| \)

**Definition 6** Let \( \Gamma'_k : \text{Equiv}(\text{Dom}) \rightarrow \text{Equiv}(\text{Dom}) \) be the following mapping:

\[
\Gamma'_k(R) = \{ (A, B) : \Delta'_k(R, A, B) \}
\]

The definition is correct \( (\Gamma'_k(R) \) is an equivalence relation for all \( k \) and \( R \) since \( =_k \) is an equivalence relation for all \( k \).
Let $R'_0, R'_1, \ldots$ be a sequence of equivalence relations on $Dom$ defined by:

- $R'_0 = Dom^2$
- $R'_{i+1} = \Gamma'_{k}(R_i)$

**Lemma 16** For all $i \in \mathcal{N}$, $R'_i = R_i \cap Dom$.

Proof. By induction w.r.t. $i$:

- $R'_0 = Dom = R_0 \cap Dom$.
- Assume $R'_i = R_i \cap Dom$. Then obviously for all $S, \mathcal{T} \in Dom$, $\Delta_k(R_i, S, T)$ iff $\Delta'_k(R'_i, S, T)$ i.e. $(S, T) \in R'_{i+1}$ iff $(S, T) \in R_{i+1}$. So $R'_{i+1} = R_{i+1}$. □

We represent a regular tree $\mathcal{T}$ by the finite automaton $S$. A state of $S$ is a pair $(t, i)$ such that $t \in Dom(\mathcal{T})$ and $1 \leq i \leq n$, where $n$ is the maximal number of siblings in $\mathcal{T}$ with subtrees $t$. We denote the set of states in $S$ by $S$. We let $(t_1, i_1) \rightarrow (t_2, i_2)$ if the root of $t_1$ has at least $i_2$ children with subtrees $t_2$. The initial state in $S$ is $(\mathcal{T}, 1)$ and we denote it by $p_S$.

In the next theorem we consider the regular trees as representing them automata. In particular the size $|S|$ of regular tree $S$ is the number of states in the automaton representing $S$.

**Theorem 4** The problem Pebble[REGTREE] is in PTIME. More precisely, there exists algorithm that for $k \in \mathcal{N}$ and $S, \mathcal{T} \in$ REGTREE solves problem Pebble$_k(S, \mathcal{T})$ in time $O(\log(k) * (|S| + |\mathcal{T}|)^6)$.

Proof.

Algorithm of deciding whether $S \equiv k \mathcal{T}$:

1. For every $p, q \in S \cup \mathcal{T}$ do $R(p, q) := true$
2. While $\Gamma'_k(R) \neq R$ do $R := \Gamma'_k(R)$
3. If $R(p_S, p_{\mathcal{T}})$ then $S \equiv_k \mathcal{T}$, otherwise $S \not\equiv_k \mathcal{T}$

Algorithm of computing $\Gamma'_k(R)$ in step 2:

- For every $p, q \in S \cup \mathcal{T}$ do
  a. For every $r$ such that $p \rightarrow r$ or $q \rightarrow r$ do $l^p_r := 0$ and $l^q_r := 0$
  b. For every inordered pair $(r_0, r_1)$ such that $x_0 \rightarrow r_0$ and $x_1 \rightarrow r_1$ where $x_0, x_1 \in \{p, q\}$, if $R(r_0, r_1)$ then increase $l^{x_0}_{r_0}$ and $l^{x_1}_{r_1}$ by 1 (increase $l'_{r_i} - i$ only if it is less then $k$; if $x_0 = x_1$ and $r_0 = r_1$ then increase the counter only once).
  c. If for all $r$ such that $p \rightarrow r$ or $q \rightarrow r$, $l^p_r = l^q_r$ then $\Gamma'_k(R)(p, q) := true$, else $\Gamma'_k(R)(p, q) := false$.

Remark that after step b. the value of counter $l^p_r$ is the number of children of $p$ which are in the relation $R$ with $r$.

Correctness of the algorithm follows from lemmas 15 and 16.
Complexity of the algorithm:

- Initialization in step 1. has complexity $O(|S| + |T|)^2$.
- In step 2. there is at most $(|S| + |T|)^2$ turns of the loop, since $\Gamma'_k(R) \subseteq R$.
- Computing of $\Gamma'_k(R)$ has complexity $O(\log(k) \times (|S| + |T|)^4)$. The factor $\log(k)$ appears because the counters $l^p_r$ are less than $k + 1$ and can be represented by $O(\log(k))$ bits.
- Checking whether $\Gamma'_k(R) = R$ has complexity $O(|S| + |T|)^2$.
- Substitution $R := \Gamma'_k(R)$ has complexity $O(|S| + |T|)^2$.

Total complexity of the algorithm is then $O(\log(k) \times (|S| + |T|)^6)$. □

5 Conclusion

We have shown that given a number $k$ and two finite directed trees, or two regular trees, one can decide in polynomial time if Duplicator wins the $k$-pebbles Barwise game on these structures. Directed trees can be viewed as a most general class of tree-shaped directed graphs. (It generalizes undirected trees as well as rooted trees.) We hope that the techniques developed in this paper can be used for extending the result to determine the complexity of pebble problem on wider classes of tree-like structures e.g. graphs of bounded tree-width. A step in this direction was made in [6] where we show that one can decide in polynomial time if Duplicator wins the $k$-pebbles Barwise game (for given number $k$) on given two graphs with explicit tree decompositions (represented by binary relations).

References

Bistability:
An Extensional Characterization of Sequentiality

James Laird
COGS, University of Sussex, UK
jiml@cogs.susx.ac.uk

Abstract. We give a simple order-theoretic construction of a cartesian closed category of sequential functions. It is based on biordered sets analogous to Berry’s bidomains, except that the stable order is replaced with a new notion, the bistable order, and instead of preserving stably bounded greatest lower bounds, functions are required to preserve bistably bounded least upper bounds and greatest lower bounds. We show that bistable cpos and bistable and continuous functions form a CCC, yielding models of functional languages such as the simply-typed $\lambda$-calculus and SPCF. We show that these models are strongly sequential and use this fact to prove universality and full abstraction results.

1 Introduction

A longstanding problem in domain theory has been to find a simple characterization of higher-order sequential functions which is wholly extensional in character. Typically, what is sought is some form of mathematical structure, such that set-theoretic functions which preserve this structure are sequential and can be used to construct a cartesian closed category. Clearly, any solution to this problem is dependant on what one means by sequential. It has been closely associated with the full abstraction problem for PCF, although it is now known that PCF sequentiality cannot be characterized effectively in this sense [8,11]. As an alternative, we have the strongly stable, or sequentially realizable functionals [2,12]. Here the difficulty is perhaps to understand precisely the sense in which these are sequential at higher types, in the absence of direct connections to a sequential programming language or an explicitly sequential model.

Another notion of sequentiality — the observably sequential functionals — was discovered by Cartwright and Felleisen [3]. They observed that if one or more errors are added to a functional language, then the order of evaluation of programs becomes observable by varying their inputs. Thus each function corresponds to a unique evaluation tree or sequential algorithm [4], which can be reconstructed from its graph. The observably sequential functionals do form a cartesian closed category, which contains a fully abstract model of SPCF — PCF with errors and a simple control operator. However, the definitions of observably sequential functions and sequential algorithms are based implicitly or explicitly on (Kahn-Plotkin) sequentiality, and hence they cannot offer a characterization
of it in the above sense. So we may refine our original problem to ask whether there is a simple, order-theoretic characterization of observable sequentiality, as a preservation property for set-theoretic functions.

This paper proposes a solution to this problem. We will construct a cartesian closed category of biorinded sets, and show that it contains universal models of the simply-typed \( \lambda \)-calculus and finitary SPCF, and fully abstract models of SPCF. The basis of our model is a kind of bioorder analogous to Berry’s bido- mains \([1]\), except that the stable order used in the latter has been replaced with a “bistable order”. There are several reasons for this step. Although the bidomain model of PCF is not sequential, even at first order types, the bidomain model of unary PCF is sequential, and universal \([9]\). The proof of this result relies on the fact that there is a “top” element at each type. The connection with observably sequential functions is made by viewing top as an error element. Under this interpretation, the monotone and stable functions on bidomains are not observably sequential, because they are not “error-propagating” (i.e. sequential with respect to \( \top \) as well as \( \bot \)). However, the duality between \( \bot \) and \( \top \) suggests that we “symmetrize” the stable order, which leads to the bistable order, and the notion of bistable function. The latter preserve both the least upper bounds and greatest lower bounds of bistably bounded finite sets and are observably sequential, as we establish by universality and full abstraction results.

2 Bistable Biorders

We shall introduce bistability in the context of a form of bidomain. Although the bistable order can be defined independently of the extensional order, it is the interaction between the extensional order and bistable order in biorders that will be used to characterize sequentiality.

**Definition 1.** A bistable bioorder is a tuple \( \langle |D|, \subseteq_D, \leq_D \rangle \) consisting of a set \( |D| \) with two partial orders (an extensional order \( \subseteq \), and a bistable order \( \leq \)), such that:

- \( \langle |D|, \subseteq \rangle \) has a least element \( \bot \) and a greatest element \( \top \), such that \( \bot \leq \top \), although \( \bot \) and \( \top \) are not in general least and greatest in \( \langle |D|, \leq \rangle \).
- \( a, b \in \langle |D|, \leq \rangle \) are bounded above if and only if they are bounded below (if \( a \) and \( b \) are so bounded, we shall write \( a \uparrow b \)),
- if \( a \uparrow b \) then there are elements \( a \land b, a \lor b \in |D| \) which are (respectively) the greatest lower bound and least upper bound of \( a \) and \( b \) with respect to both orders,
- if \( a \uparrow b \), \( a \uparrow c \) and \( b \uparrow c \), then \( a \lor (b \land c) = (a \lor b) \land (a \lor c) \) (and so \( a \land (b \lor c) = (a \lor b) \lor (a \lor c) \)).

Note that \( \leq \) is included in \( \subseteq \) since if \( a \leq b \) then \( a \land b = a \) is a glb for \( a, b \) with respect to \( \subseteq \).

Thus a bistable bioorder may be presented as a tuple \( \langle |D|, \subseteq_D, \uparrow_D \rangle \) where \( \langle |D|, \subseteq \rangle \) is a partial order with least and greatest elements, \( \uparrow \) is an equivalence relation,
and for each \( x \in |D| \), the equivalence class \( ([x], \sqsubseteq \uparrow [x]) \) is a distributive lattice (having greatest and least elements, in many cases, but not in general). This is equivalent to the above definition, under the interpretation \( x \leq y \) if \( x \downarrow y \) and \( x \sqsubseteq y \).

Simple examples of bistable biorders include the one-point order, \( 1 \) (in which \( \top = \bot \)), and the Sierpinski Space \( \Sigma \), which contains distinct \( \top \) and \( \bot \) elements such that \( \bot \leq \top \) (and so \( \bot \subseteq \top \)). The product and separated sum operations on bistable orders are straightforward (although note that the latter adds two new points, \( \top \) and \( \bot \)).

**Definition 2.** Let \( \{ A_i \mid i \in I \} \) be a family of bistable biorders. The product \( \Pi_{i \in I} A_i \) is defined pointwise:

- \( |\Pi_{i \in I} A_i| = \Pi_{i \in I} |A_i| \),
- \( x \sqsubseteq_{\Pi_{i \in I} A_i} y \) if \( \pi_i(x) \sqsubseteq \pi_i(y) \) for all \( i \in I \),
- \( x \leq_{\Pi_{i \in I} A_i} y \) if \( \pi_i(x) \leq \pi_i(y) \) for all \( i \in I \).

The separated sum \( \Sigma_{i \in I} A_i \) is defined:

- \( |\Sigma_{i \in I} A_i| = \{ \bigcup_{i \in I} |A_i| \} \cup \{ \bot, \top \} \),
- \( x \sqsubseteq_{\Sigma_{i \in I} A_i} y \) if \( x = \bot \) or \( y = \top \) or \( x = \mathfrak{i}_i(z_1) \) and \( y = \mathfrak{i}_i(z_2) \), where \( z_1 \sqsubseteq_i z_2 \),
- \( x \leq_{\Sigma_{i \in I} A_i} y \) if \( x = \bot \) and \( y = \top \), or \( x = \mathfrak{i}_i(z_1) \) and \( y = \mathfrak{i}_i(z_2) \), where \( z_1 \leq_i z_2 \).

### 2.1 Bistable Functions

We shall now construct a CCC of bistable biorders and monotone and bistable functions.

**Definition 3.** A function \( f : D \to E \) is monotone if for all \( x, y \in |D| \), \( x \sqsubseteq_D y \) implies \( f(x) \sqsubseteq_E f(y) \).

\( f \) is bistable if it is monotone with respect to the bistable order, and for all \( x, y \in |D| \), if \( x \downarrow y \) then \( f(x \land y) = f(x) \land f(y) \) and \( f(x \lor y) = f(x) \lor f(y) \).

Thus a monotone function \( f \) is bistable if \( x \downarrow y \) implies \( f(x) \downarrow f(y) \) and for each \( x, f][x]_\uparrow \) is a lattice homomorphism.

**Definition 4.** We shall say that a bistable and monotone function \( f \) is strict if \( f \) preserves the least upper bound and greatest lower bound of the empty set — i.e. \( f(\top) = \top \) and \( f(\bot) = \bot \).

We now construct a bistable biorder of functions, in which the extensional order is standard, and the bistable order is a symmetric version of the stable order.

**Definition 5.** Given bistable biorders \( D, E \), we define the function-space \( D \Rightarrow E \) as follows:
Lemma 1. $D \Rightarrow E$ is a bistable biorder.

Proof. If $f \nleq g$ we can define $f \land g$ and $f \lor g$ by:

$(f \land g)(a) = f(a) \land g(a)$ and $(f \lor g)(a) = f(a) \lor g(a)$ for all $a$.

$f \land g$ and $f \lor g$ are monotone and bistable — e.g. to show that if $a \nleq b$ then $(f \land g)(a) \lor (f \land g)(b) = (f \land g)(a \lor b)$ suppose $f, g \leq h$. Then since $a, b \leq a \lor b$,

$f(a) = f(a \lor b) \land h(a), f(b) = f(a \lor b) \land h(b), g(b) = g(a \lor b) \land h(a), g(b) = g(a \lor b) \land h(b)$ so:

$(f \land g)(a) \lor (f \land g)(b) = (f \land g)(a \lor b)$

Suppose $f, g \leq h$.

Thus if $\mathbb{BBo}$ is the category of bistable biorders and monotone and bistable functions, it is now straightforward to show the following.

Proposition 1. ($\mathbb{BBo}, 1, \times, \Rightarrow$) is cartesian closed.

The separated sum is a co-product in the subcategory of $\mathbb{BBo}$ consisting of bistable biorders and strict functions.

2.2 Bicpos

We shall now extend our notion of bistable biorder with a notions of completeness and continuity.

Definition 6. A bistable bicpo is a bistable biorder $D$ such that $(|D|, \sqsubseteq)$ is a cpo (i.e. every $\sqsubseteq$-directed set has a $\sqsubseteq$-least upper bound), and the following properties hold:

i. If $X, Y \subseteq |D|$ are $\sqsubseteq$-directed sets such that for all $x \in X$ there exists $y \in Y$ such that $x \leq y$, and if $y \in Y$ there exists $x \in X$ such that $x \leq y$ then $\bigsqcup X \leq \bigsqcup Y$.

ii. For any $X \subseteq |D| \times |D|$ such that $\pi_1(X), \pi_2(X)$ are directed and for all $(y, z) \in X$, $y \nleq z$:

$(\bigsqcup_{y \in \pi_1(X)} y) \land (\bigsqcup_{z \in \pi_2(X)} z) = \bigsqcup_{(y, z) \in X} (y \land z)$, and

$(\bigsqcup_{y \in \pi_1(X)} y) \lor (\bigsqcup_{z \in \pi_2(X)} z) = \bigsqcup_{(y, z) \in X} (y \lor z)$. 

Let $\mathcal{BBC}$ be the category of bistable bicpos and (extensionally) continuous and bistable functions.

**Proposition 2.** $\mathcal{BBC}$ is a cartesian closed category.

**Proof.** We use the additional properties to show that for any directed set $F$ of functions from $A$ to $B$, a bistable and continuous least upper bound can be defined pointwise — $(\bigvee F)(a) = \bigvee \{ f(a) \mid f \in F \}$.

E.g. if $a \leq b$ then $(\bigvee F)(a) \leq (\bigvee F)(b)$ by property (i) since for all $f \in F$, $f(a) \leq f(b)$.

$\bigvee F$ is bistable, since if $a \nleq b$ then for each $f \in F$, $f(a) \nleq f(b)$ and hence applying property (ii) to the set $\{ (f(a), f(b)) \mid f \in F \}$, we have:

$$(\bigvee F)(a \wedge b) = \bigvee_{f \in F} (f(a) \wedge f(b)) = (\bigvee_{f \in F} f(a)) \wedge (\bigvee_{f \in F} f(b)),$$

e etcetera.

We must also check that the exponential preserves properties (i) and (ii): e.g. given directed sets of functions $F, G$ such that for all $f \in F$ there exists $g \in G$ such that $f \leq g$ and for all $g \in G$ there exists $f$ such that $f \leq g$, then $\bigvee F \leq \bigvee G$. First, for all $a$, $\bigvee \{ f(a) \mid f \in F \} \leq \bigvee \{ g(a) \mid g \in G \}$ by property (i). Now suppose $a \leq b$. Let $X = \{ (f(b), g(a)) \mid f \in F \land g \in G \land f \leq g \}$. Then $\pi_1(X) = \{ f(b) \mid f \in F \}$ and $\pi_2(X) = \{ g(a) \mid g \in G \}$ and hence by property (ii), $(\bigvee F)(a) = \bigvee_X (f(b) \land g(a)) = (\bigvee F)(b) \land (\bigvee G)(a)$, and so on.

### 2.3 Bistability and Bisequentiality

We may now give a simple proof that the monotone and bistable functions into $\Sigma$ are *bisequential* (i.e. sequential with respect to both $\bot$ and $\top$ elements), and also *strongly sequential* (having a unique sequentiality index).

**Definition 7.** Let $\{ A_i \mid i \in I \}$ and $B$ be bistable biorders. A function $f : \Pi_{i \in I} A_i \to B$ is $i$-strict if $f(x) = \bot$ whenever $\pi_i(x) = \bot$ and $f(x) = \top$ whenever $\pi_i(x) = \top$.

**Lemma 2.** Let $\{ A_i \mid i \in I \}$ be a finite family of bistable biorders. If $f : \Pi_{i \in I} A_i \to \Sigma$ is a strict, monotone and bistable function, then $f$ is $i$-strict for some $i \in I$ (and providing $\Pi_{i \in I} A_i$ has at least two points, $i$ is unique).

**Proof.** Given $j \in I$, let $\bot[\top]_j = \{ x_i \mid i \in I \}$, where $x_i = \top$ if $i = j$, and $x_i = \bot$ otherwise. Similarly $\top[\bot]_j = \{ x_i \mid i \in I \}$, where $x_i = \bot$ if $i = j$, and $x_i = \top$ otherwise.

Then if $\pi_i(x) = \bot$, $x \subseteq \top[\bot]_i$, and if $\pi_i(x) = \top$, $\bot[\top]_i \subseteq x$, and so $f$ is $i$-strict if $f(\top[\bot]_j) = \bot$ and $f(\bot[\top]_i) = \top$. Since $\top[\bot]_j \nleq \bot[\bot]_k$ for all $j, k$, and $\bigvee_{i \in I} \top[\bot]_i = \top$, we have $f(\bigvee_{i \in I} \top[\bot]_i) = f(\top) = \bot$, and so $f(\bot[\bot]_j) = \bot$ for some $i$, and similarly $f(\bigvee_{i \in I} \bot[\top]_i) = \top$, and so $f(\bot[\bot]_j) = \top$ for some $j$. Moreover, if $i \neq j$, then $\bot[\bot]_j \subseteq \top[\bot]_i$, so $\bot[\bot]_j = \bot$ as required.

In the case where $A_i$ is the Sierpinski order, $\Sigma$, there is exactly one $i$-strict function $f : \Pi_{i \in I} A_i \to \Sigma$ — the $i$th projection. Thus for each indexing set $i$, the “biflat” domain $\Sigma_{i \in I} 1$ is isomorphic to $\Sigma^I \Rightarrow I$. (i.e. $(\Pi_{i \in I} \Sigma) \to \Sigma$.}

Lemma 3. For $I \subseteq \omega$, $\Sigma^I \Rightarrow \Sigma \cong \Sigma_{i \in I}1$.

Proof. The map which associates the $i$th projection with $in_i(\top)$ for each $i$ is an isomorphism of bistable biorders. (The proof extends to $I = \omega$ by continuity.)

The extensional and bistable orderings on $\Sigma \times \Sigma \Rightarrow \Sigma$ are depicted in Figure 1.

![Fig. 1. Extensional and bistable orderings on $\Sigma \times \Sigma \Rightarrow \Sigma$](image)

3 Universality and Full Abstraction for SPCF+

We shall now describe a bistable biorder model of SPCF+, which is a version of Cartwright and Felleisen’s SPCF [3] with (infinitary) lifted sums and products, and a single error, $\top$, at each type. This may be thought of either as a simple functional programming language with non-local control, or as a term language for certain bistable biorders. We will show that the finitary fragment of the model is universal — i.e. every element is the denotation of a term. Full abstraction follows from universality for the finite fragment, as we have a complete and order-extensional model. The presence of separated sums is significant for our proof of universality, as we shall show that every finite type-object in the bistable biorder model is a (definable) retract of a type-object built from sums and products.

We can, however, prove universality for restricted fragments of SPCF+ using straightforward syntactic arguments.

Let SPCF+ be the language with types generated by the following grammar (where $I \leq \omega$, and $T_i : i < I$ is a sequence of types):

$$T ::= \Pi_{i < I} T_i \mid \Sigma_{i < I} T_i \mid T_1 \Rightarrow T_2$$

We will sometimes write $1$ for the empty product ($I = 0$), $\Sigma$ for the empty sum, and $\overline{I}$ for $\Sigma_{i < I}1$ (the “flat” domain containing $I$ values). The types of finitary SPCF+ are those generated from finite products and sums ($I < \omega$). Terms are generated from the $\lambda$-calculus with products, lifted sums, and fixpoint operators, plus a catch operator from $\Sigma^I \Rightarrow \Sigma$ to $\overline{I}$ which sends the $i$th projection to the
numeral \( \tilde{i} = \text{in}_i(T) \). From this we can derive a more general form of catch of type \( (\Pi_{i\leq I} A_i \Rightarrow \Sigma) \Rightarrow \tilde{T} \) which sends any \( i \)-strict function to \( \tilde{i} \), as follows:
catch \( M = \text{catch} \lambda x. (M \langle \text{case}_0 \pi_i(x) \rangle \top \mid i < I \rangle) \).

| Table 1. Typing judgements for SPCF+ |
|---------------------------|---------------------------|
| \( \Gamma \vdash T : \top : \tilde{T} \) | \( \Gamma \vdash \lambda x. T : \tilde{T} \) |
| \( \Gamma \vdash x : T : \tilde{T} \) | \( \Gamma \vdash N : S \) |
| \( \forall i < I. \Gamma \vdash M_i : T_i \) | \( \Gamma \vdash \Pi_{i < I} T_i \) |
| \( \Gamma \vdash \text{in}_i(M) : \Sigma_{j < I} T_j \) | \( \Gamma \vdash \text{case}_i M : \Pi_{i < I} (T_i \Rightarrow S) \Rightarrow S \) |
| \( \Gamma \vdash (\Pi_{i < I} \Sigma) \Rightarrow \Sigma \) | \( \Gamma \vdash M : T \Rightarrow \tilde{T} \) |
| \( \Gamma \vdash \text{catch}_i M : \tilde{T} \) | \( \Gamma \vdash \text{Y} M : \tilde{T} \) |

We may give a simple operational semantics for SPCF programs — closed terms of type \( \Sigma \) — using evaluation contexts.

**Definition 8.** Evaluation contexts of SPCF are given by the following grammar:

\[
E[-] ::= [\cdot] \mid E[-] M \mid \text{case} E[-] \mid \pi_i E[-] \mid \text{Y} E[-]
\]

The small-step operational semantics of SPCF programs is as follows:

\[
E[\top] \rightarrow \top
\]
\[
E[\lambda x. M N] \rightarrow E[M[N/x]]
\]
\[
E[\pi_j((M_i \mid i < I))] \rightarrow E[M_j]
\]
\[
E[(\text{case} \text{in}_i(M)) N] \rightarrow E[\pi_i(N) M]
\]
\[
E[\text{catch}_i M] \rightarrow M \langle E[\tilde{\top}] \mid i < I \rangle
\]
\[
E[\text{Y} M] \rightarrow E[M \langle \text{Y} M \rangle]
\]

For a program \( M \) we write \( M \Downarrow \) if \( M \rightarrow \top \).

The interpretation of types as bistable bicpos and terms as bistable and continuous maps is wholly straightforward since we have a cartesian closed category with lifted sums, least fixed points of directed sets with which to interpret the fixed-point combinator, and the isomorphism from \( \Pi_{i \leq I} (\Sigma) \Rightarrow \Sigma \) to \( \Sigma_{i < n} 1 \) (Lemma 3) with which to interpret catch.

**Proposition 3.** \( M \Downarrow \) if and only if \( \llbracket M \rrbracket = \top \).

**Proof.** Standard: Soundness is proved by induction, using the fact that each evaluation context represents a strict map. Adequacy is proved using a computability predicate argument.
To show that the bistable biorder model is universal for finitary types, we shall show that each of the latter is the retract of a \( \Pi \Sigma \) type — i.e a type generated by the grammar:

\[
P ::= \Sigma_{i<l}P_i \mid \Pi_{i<l}P_i
\]

**Lemma 4.** The bistable biorder model of finitary SPCF+ is universal for the \( \Pi \Sigma \) types (i.e. element of a finite \( \Pi \Sigma \) type is the denotation of a term).

**Proof.** This is a straightforward induction on type structure.

We now use the bisequentiality of bistable functions to prove the following key lemma.

**Lemma 5.** For any finite family of bistable biorders \( \{A_{ij} \mid i \in I, j \in J_i\} \), there is an isomorphism

\[
\Pi_{i \in I}\Sigma_{j \in J_i}A_{ij} \Rightarrow \Sigma \cong \Sigma_{k \in I}\Pi_{l \in J_k}((A_{kl} \times \Pi_{i \in I-\{k\}}\Sigma_{j \in J_i}A_{ij}) \Rightarrow \Sigma)
\]

**Proof.** Any monotone and bistable function \( f : \Pi_{i \in I}\Sigma_{j \in J_i}A_{ij} \Rightarrow \Sigma \) is either constant or strict. So by Lemma 2 \( f \) is either \( \top \), \( \bot \), or \( k \)-strict for some unique \( k \in I \), and thus corresponds by currying to a unique strict function \( f_k : \Sigma_{j \in J_k}A_{kj} \Rightarrow (\Pi_{i \in I-\{k\}}\Sigma_{j \in J_i}A_{ij}) \Rightarrow \Sigma \). Because the separated sum is a co-product in the category of strict maps, \( f_i = [\text{in}_i; f_k \mid l \in J_k] \). Finally, \( \text{in}_i; f_k : A_{kl} \Rightarrow (\Pi_{i \in I-\{k\}}\Sigma_{j \in J_i}A_{ij}) \Rightarrow \Sigma \) uncurries to \( g_l : (A_{kl} \times \Pi_{i \in I-\{k\}}\Sigma_{j \in J_i}A_{ij}) \Rightarrow \Sigma \).

Thus there is a bijection from \( \Pi_{i \in I}\Sigma_{j \in J_i}A_{ij} \Rightarrow \Sigma \) to \( \Sigma_{k \in I}\Pi_{l \in J_k}((A_{kl} \times \Pi_{i \in I-\{k\}}\Sigma_{j \in J_i}A_{ij}) \Rightarrow \Sigma) \) which sends \( \top \) to \( \top \), \( \bot \) to \( \bot \), and each \( k \)-strict \( f \) to \( \text{in}_k((g_l \mid l \in J_k)) \), where \( g_l \) is such that if \( \pi_k(x) = \text{in}_l(y_{kl}) \), then \( f(x) = g_i((y_{kl}, \pi_i(x) \mid i \in I - \{k\})) \). Moreover, this is an isomorphism of bistable biorders.

Given two SPCF+ types \( S, T \), we shall say that there is a definable retraction from \( S \) to \( T \) if \( [S] \leq [T] \), and this retraction is the denotation of SPCF+ terms of type \( S \Rightarrow T \) and \( T \Rightarrow S \) (if it is an isomorphism, we say that \( S \) and \( T \) are definably isomorphic).

**Lemma 6.** If SPCF+ is universal at type \( T \), and there is a definable retraction from \( S \) to \( T \), then SPCF+ is universal at type \( S \).

**Proof.** Suppose \( \text{in} : [S] \leq [T] : \text{proj} \). Given an element \( e \in [S] \), we have a term \( M_{\text{in}(e)} : T \) such that \([M_{\text{in}(e)}] = \text{in}(e)\) and thus if \([\text{OUT} : S \Rightarrow T] = \text{proj} \) then \( e = [\text{OUT} M_{\text{in}(e)}] \).

We shall now show that the isomorphism described in Lemma 5 is definable.

**Definition 9.** Given a finitary SPCF+ type \( T = \Pi_{i<n}\Sigma_{j<m_i}T_{ij} \Rightarrow \Sigma \), and \( k < n \), \( l < m_k \), let \( S_{kl} = \Pi_{i<k}\Sigma_{j<m_i}T_{ij} \times T_{kl} \times \Pi_{i<n-k}\Sigma_{j<m_{i+k}}T_{(i+k)j} \).
Lemma 7. \( T \) is definably isomorphic to \( \Sigma_{k<n} \Pi_{l<m_k} (S_{kl} \Rightarrow \Sigma) \).

Proof. The defining terms for the isomorphism of Lemma 5 are, from \( T \) to \( S \):

\[
\lambda f. (\text{case}(\text{catch}(f)) \langle \text{in}_i((M_{i,j} \mid j < m_i)) \mid i < n) \rangle
\]

where \( M_{i,j} = \lambda x. f \langle \pi_0(x), \ldots, \pi_{i-1}(x), \text{in}_j(\pi_i(x)), \pi_{i+1}(x), \ldots, \pi_{n-1}(x) \rangle \), and from \( S \) to \( T \):

\[
\lambda xy. (\text{case}(x) \langle \lambda u. \text{case}(\pi_i(y)) \langle N_{i,j} \mid j < m_i \rangle \mid i < n \rangle \rangle
\]

where \( N_{i,j} = \lambda v. \langle \pi_j(u), \ldots, \pi_{i-1}(y), v, \pi_{i+1}(y), \ldots, \pi_{n-1}(y) \rangle \rangle \).

Lemma 8. If \( T \) is a finitary \( \Pi \Sigma \) type, then \( T \Rightarrow \Sigma \) is definably isomorphic to a \( \Pi \Sigma \) type.

Proof. This is by induction on the size of \( T \) (as a syntax tree), for which the base case is trivial. For the induction step we may assume \( T \) has the form \( \Pi_{i<n} \Sigma_{j<m_i} T_{ij} \). Then \( T \Rightarrow \Sigma \) is definably isomorphic to \( \Sigma_{k<n} \Pi_{l<m_k} (S_{kl} \Rightarrow \Sigma) \) by Lemma 7. Since the size of each \( S_{kl} \) is less than that of \( T \), the inductive hypothesis now applies.

Now let the SPCF types (i.e. types in which the only sums are the atomic types) be those generated by the grammar:

\[
S, T ::= I | S \Rightarrow T
\]

Proposition 4. Every SPCF type is definably isomorphic to a \( \Pi \Sigma \) type.

Proof. By induction on type-structure. For the induction step, suppose \( T = S_0 \Rightarrow \ldots S_{n-1} \Rightarrow \vec{T} \). Then \( T \Rightarrow \Sigma \) is definably isomorphic to \( (\Pi_{i<n} S_i) \times \Sigma \Rightarrow \Sigma \). By induction each \( S_i \) is definably isomorphic to a \( \Pi \Sigma \) type, and by Lemma 8, so is \( T \).

We now use a definable retraction to extend universality to all finitary SPCF+ types.

Lemma 9. For any family of types \( \{ T_i \mid i < I \} \), there is a definable retraction \( \Sigma_{i<I} T_i \leq \vec{T} \times \Pi_{i<I} T_i \)

Proof. The defining terms are, from \( \Sigma_{i<I} T_i \) to \( \vec{T} \times \Pi_{i<I} T_i \):

\[
\lambda x. (\text{case}(x)) \langle \lambda y. (\vec{i}, \perp, \ldots, \perp, y, \perp, \ldots, \perp) \mid i < I \rangle
\]

and from \( \vec{T} \times \Pi_{i<I} T_i \) to \( \Sigma_{i<I} T_i \):

\[
\lambda x. (\text{case} \pi_1(x)) \langle \text{in}_i(\pi_{i+1}(x)) \mid i < I \rangle
\]
So every finitary $\Pi \Sigma$ is the retract of a product of ground types.

**Proposition 5.** Every finitary SPCF+ type is a definable retract of a $\Pi \Sigma$ type.

**Proof.** We use the retraction of Lemma 9 to show that every finitary SPCF+ type is the definable retract of a product of SPCF types, which is definably isomorphic to a $\Pi \Sigma$ type.

(In fact, using a more general version of Lemma 5, we can show that every finitary SPCF+ type is isomorphic to a $\Pi \Sigma$ type.) By Lemma 6 we have the following.

**Corollary 1.** The bistable biorder model of finitary SPCF+ is universal.

We may prove further universality results for fragments of SPCF+, such as (finitary) SPCF, or the simply-typed $\lambda$-calculus over the single ground type $\Sigma$ (the bistable model of the latter is equivalent to the “minimal model” characterized syntactically by Padovani [13]). To do so, it suffices to show that every element of a type-object of the restricted language is definable as a term of which every subterm is in the language. For SPCF and the minimal model, this can be achieved using straightforward Tait-Girard style reducibility arguments.

We could extend universality to all types either by constructing an effectively presented model, or by allowing infinitely deep types and terms. However, universality for the finite fragment is sufficient to prove full abstraction for SPCF+ by showing that each type-object is the limit of a chain of retractions which are SPCF definable.

**Lemma 10.** For each SPCF+ type $S$ there is a sequence of finitary types $S_i : i \in \omega$ and SPCF-definable retractions: $\text{in}_i : [S_i] \subseteq [S] : \text{proj}_i$, such that if $e \in [S]$ then $e = \bigsqcup_{i \in \omega} \text{in}_i(\text{proj}_i(e))$.

**Proof.** We define:

- $(\Sigma_{j < n} T_j)^i = \Sigma_{j < l} (T_j)^i$, if $i < n$, $(\Sigma_{j < n} T_j)^i = \Sigma_{j < n} (T_j)^i$, otherwise.
- $(\Pi_{j < n} T_j)^i = \Pi_{j < l} (T_j)^i$, if $i < n$, $(\Pi_{j < n} T_j)^i = \Pi_{j < n} (T_j)^i$, otherwise.
- $(S \Rightarrow T)^i = S^i \Rightarrow T^i$.

Recall that, given terms $M, N : T$, we say that $M$ observationally approximates $N$ ($M \preceq N$) if for all compatible program contexts $C[\cdot]$, $C[M] \downarrow$ implies $C[N] \downarrow$.

**Proposition 6.** The bistable bicpo model of SPCF+ is fully abstract — i.e. for all terms $M, N : T$, if $M \preceq N$ and only if $[M] \subseteq [N]$.

**Proof.** Equational soundness follows from adequacy in the standard way. To prove completeness — that for all $M, N : S$, $N \preceq M$ implies $[N] \subseteq [M]$, suppose $[N] \not\subseteq [M]$. Then $[N] = \bigsqcup_{i \in \omega} \text{in}_i(\text{proj}_i([N])) \not\subseteq \bigsqcup_{i \in \omega} \text{in}_i(\text{proj}_i([M])) = [M]$. So for some $i$, $\text{proj}_i([N]) = [\text{OUT}_i N] \not\subseteq \text{proj}_i([M]) = [\text{OUT}_i M]$, and so by universality of the finite fragment, there is a term $L : S^i \Rightarrow \Sigma$ such that $[L(\text{OUT}_i N)] \neq \bot$ and $[L(\text{OUT}_i M)] = \bot$ and so $L(\text{OUT}_i N) \downarrow$ and $L(\text{OUT}_i M) \nmid \downarrow$ as required.
4 Conclusions

A natural question concerns the connections between bistable bidomains and more intensional characterizations of the same type structure, such as observably sequential domains, sequential algorithms, and Hyland and Schalk’s “graph games” model \[\text{[7]}\]. It is fairly straightforward to show that sequential algorithms over sequential data structures with a single error act as bistable functions on their arguments, and Curien \[\text{[6]}\] has shown that the category of sequential data structures and sequential algorithms embeds fully in the category of bistable biorders and bistable functions. However, this leaves open the question of how the correspondence works in the opposite direction; what is the image of the embedding, and given an object in that image, can we construct the corresponding sequential data structures and sequential algorithms? The underlying linear structure of the sequential algorithms model also suggests a further question; it is equivalent to the co-Kleisli category of a certain co-monad \[\text{[10,5]}\] on a symmetric monoidal category of games and strategies. Can we provide a parallel “linear decomposition” of a CCC of bistable domains, and describe a purely extensional model of intuitionistic linear logic equivalent to the games model?

To answer to these questions, we have investigated a notion of “locally boolean” domain — a partial order (the extensional order) with an involutive negation, which can be used to give simple definitions of the stable and bistable orders. Our fundamental representation result for these domains is that they can all be generated (up to isomorphism) by taking products and co-products, lifting, and limits of \(\omega\)-chains. In particular, each pointed domain is isomorphic to one of the form \(\Sigma_{i \in I} \Pi_{j \in J} A_{ij}\), which is a more general version of Proposition \[\text{[4]}\] (every type-object of SPCF has this form).

References

Automata on Lempel-Ziv Compressed Strings

Hans Leiß\textsuperscript{1} and Michel de Rougemont\textsuperscript{2}

\textsuperscript{1} Universität München, CIS, D-80538 München, Germany
leiss@cis.uni-muenchen.de

\textsuperscript{2} Université Paris-II & LRI Bâtiment 490, F-91405 Orsay Cedex, France
mdr@lri.fr

Abstract. Using the Lempel-Ziv-78 compression algorithm to compress a string yields a dictionary of substrings, i.e. an edge-labelled tree with an order-compatible enumeration, here called an \textit{LZ}-trie. Queries about strings translate to queries about \textit{LZ}-tries and hence can in principle be answered without decompression. We compare notions of automata accepting \textit{LZ}-tries and consider the relation between acceptable and MSO-definable classes of \textit{LZ}-tries. It turns out that regular properties of strings can be checked efficiently on compressed strings by \textit{LZ}-trie automata.

1 Introduction

We are interested in the \textit{compressed model checking problem}: which properties of strings can be checked given the compressed strings? The challenge is to beat the decompress-and-then-check method. We restrict ourselves to the classical Ziv-Lempel\textsuperscript{8} string compression algorithm \textit{LZ}-78. It compresses a string $w \in \Sigma^+$ to a generally much shorter sequence $LZ(w) \in (\mathbb{N} \times \Sigma)^+$, where the numbers point to previous elements of the sequence.

As usual, we view a string $w$ as a colored finite linear order $S_w = (D, <, U_a)_{a \in \Sigma}$, where $D = \{0, \ldots, |w| - 1\}$ is the set of positions, ordered by $<$ as usual, and $U_a(i)$ means that $a$ occurs at position $i$ of $w$. Properties of strings are expressed in first-order (FO) or second-order (SO) logic of colored linear orders. In a similar way, we give two representations of \textit{LZ}-78-compressed strings $\alpha$ by relational structures, one by node-labelled \textit{LZ}-graphs $G_\alpha$ and another one by \textit{LZ}-tries $T_\alpha$, which are a kind of edge-labelled trees.

A natural approach to the compressed model checking problem is to translate properties of strings to properties of compressed strings. In fact, monadic second-order (MSO) formulas $\varphi$ in the language of strings can be translated to dyadic second-order (DSO) formulas $\varphi^{LZ}$ in the language of \textit{LZ}-graphs $G_\alpha$. One can therefore answer queries $\varphi$ about $S_w$ by evaluating $\varphi^{LZ}$ on the smaller structure $G_{LZ(w)}$. However, since the translation doubles the arity of relation variables, this is not guaranteed to provide an efficient solution.

By Büchi’s well-known theorem (cf. \cite{4}, Theorem 5.2.3), MSO for strings is equally expressive as regular expressions, or finite automata, are. This raises the question whether MSO for \textit{LZ}-graphs leads to a notion of \textit{LZ}-automaton that provides an efficient method for checking a reasonably rich class of properties of compressed strings.
We study this question in the slightly more suitable format of LZ-tries. These come with an enumeration of their nodes and can be viewed as simple acyclic directed graphs. We introduce notions of \textit{LZ-trie automata} by modifying corresponding notions of tree-automata. We show that deterministic bottom-up LZ-trie-automata are less powerful than non-deterministic ones, and that the latter capture \( \exists \)-MSO for LZ-tries, where \( \exists \)-MSO is the set of MSO-formulas of the form \( \exists X \psi \) where \( X \) are set variables and \( \psi \) is an FO-formula. Finally, we show that MSO-properties of strings can be checked efficiently on LZ-compressed strings by deterministic top-down LZ-automata. Thus, regular expression search in strings can be done on the LZ-compressed strings, without decompression.

2 Lempel-Ziv-78 Compression

We fix a finite alphabet \( \Sigma \) and, to avoid structures with empty universe, only consider non-empty strings \( w \in \Sigma^+ \). The classical Lempel-Ziv-78 compression algorithm has many variations (cf. [2], [3]). It decomposes a string \( w \in \Sigma^+ \) into a sequence of substrings or \textit{blocks} \( B_i \in \Sigma^+ \), so that \( w = B_0 \cdots B_m-1 \). The first block \( B_0 \) consists of the first letter of \( w \). Suppose for some \( n > 0 \), we have constructed blocks \( B_0, \ldots, B_{n-1} \) such that \( w = B_0 \cdots B_{n-1}v \) for some \( v \in \Sigma^+ \). Then \( B_n \) is the shortest non-empty prefix of \( v \) that is not among \( \{B_0, \ldots, B_{n-1}\} \), if this exists\(^1\), otherwise \( B_n = v \). The \textit{LZ-compression} \( \text{LZ}(w) \) of \( w \) is the sequence \( p_0 \cdots p_{m-1} \) of pairs \( p_n = (k, a) \) such that \( B_n = B_{k-1}a \) (where \( B_{-1} := \epsilon \)) and \( w = B_0 \cdots B_{m-1} \). The decompression is given by \( B_k = \text{decode}(p_k) \) where \( \text{decode}((0, a)) = a \) and \( \text{decode}((n + 1, a)) = \text{decode}(p_n)a \).

Example 1. The blocks of \( w = abbbababbb \) are \( a, bbb, aa, bba, bbb \), and its compression is \( \text{LZ}(w) = (0, a)(0, b)(2, b)(1, a)(3, a)(3, b). \)

2.1 LZ-Graphs

\textbf{Definition 1.} A \textit{compressed string} \( \alpha = p_0 \cdots p_{m-1} \) is represented as a finite labelled ordered graph

\[ G_\alpha := (D_m, <, U_a, E)_{a \in \Sigma}, \]

where \( m = |\alpha| \) is the number of blocks, \( D_m = \{0, \ldots, m-1\} \), \( < \) the natural order on \( D_m \), \( U_a(i) \) is true iff the last letter of block \( B_i \) is \( a \). The binary relation \( E \) describes the reference to previous pairs: if \( p_i = (k, a) \) for some \( a \in \Sigma \) and \( k > 0 \), (i.e. the \( k \)-th block \( B_{k-1} \) is the longest strict prefix of \( B_i \)), then there is an edge \( E(i, k-1) \) from node \( i \) to \( k-1 \).

\textbf{Example 1 (Cont.).} For \( w = a.bbb.aa.bba.bbb \), \( B_0B_1B_2B_3B_4B_5 \), the graph \( G_{\text{LZ}(w)} \) is

\(^1\) and then \( B_n \) extends one of \( B_0, \ldots, B_{n-1} \) by exactly one letter.
Observe that $S_w$ can be interpreted in $G_{LZ}(w)$ as a binary relation: a position $i$ in $w$ is mapped to the pair $h(i) = (k, j)$ in $LZ(w)$ iff $i$ lies in block $B_k$ and $B_j$ is the nonempty prefix of $B_k$ ending in $i$. Note that in $G_{LZ}(w)$, node $j$ can be reached from $k$ by a path of $E$-edges.

Example 2. If $w = a.aa.ab.aba.aa$, the positions $5, 6, 7$ occurring in block $B_3 = aba$ are represented by $(3, 0), (3, 2), (3, 3)$, because the nonempty prefixes of $B_3$ are $a = B_0$, $ab = B_2$, and $aba = B_3$.

**Theorem 1 (II).** For every MSO-formula $\varphi(x_1, \ldots, x_n, X^{(1)})$ about strings there is a DSO formula $\varphi^{LZ}(x_1, y_1, \ldots, x_n, y_n, X^{(2)})$ about LZ-graphs, such that for each $S_w$ and all $i_1, \ldots, i_n \in S_w$ and $S \subseteq S_w$,

$$S_w \models \varphi[i, S] \iff G_{LZ(w)} \models \varphi^{LZ}[h(i), h(S)].$$

**Proof.** (Sketch) In DSO, we can define $E^*$, the reflexive transitive closure of $E$, and then define $\varphi^{LZ}$ inductively using

\begin{align*}
& (U_a(x_i))^{LZ} := U_a(y_i), \\
& (x_i \leq x_j)^{LZ} := x_i < x_j \lor (x_i = x_j \land y_i \leq y_j), \\
& (\exists x_{n+1}\varphi)^{LZ} := \exists x_{n+1}\exists y_{n+1} (E^*(x_{n+1}, y_{n+1}) \land \varphi^{LZ}), \\
& (\exists X^1\varphi)^{LZ} := \exists X^2(\forall x \forall y (X^2(x, y) \rightarrow E^*(x, y)) \land \varphi^{LZ}).
\end{align*}

(1)

For the atomic cases, note that in $\varphi^{LZ}$ a variable $x_i$ stands for a block of $w$ and $y_i$ for a relative position in this block.

A property $L$ of non-empty strings is definable on strings (resp. on compressed strings or LZ-graphs), if for some formula $\varphi$ of the appropriate language, $L = \{ w \mid S_w \models \varphi \}$ (resp. $L = \{ w \mid G_{LZ(w)} \models \varphi \}$).

**Remark 1.** There are properties of strings that are FO-definable on strings, but not on LZ-graphs, like $\exists x(U_a(x) \land U_b(x + 1))$. There are properties of strings that are FO-definable on LZ-graphs, but not even MSO-definable on strings (cf. II).

**2.2 LZ-Tries**

While compressing $w = B_0 \cdots B_{n-1}v$, the blocks $B_0, \ldots, B_{n-1}$ found are maintained as a dictionary of subwords of $w$ and stored as a tree by sharing common prefixes. The linear order of the blocks in $LZ(w)$ amounts to an enumeration of the nodes of the tree.
Definition 2. A (finite) $\Sigma$-tree $(T, \leq, \xleftarrow{a})_{a \in \Sigma}$ is a (finite) tree $(T, \leq, 0)$ with root 0, where $\{\xleftarrow{a} \subseteq T \times T \mid a \in \Sigma\}$ are pairwise disjoint minimal relations such that $\leq$ is the reflexive transitive closure of their union. A $\Sigma$-tree is a $\Sigma$-trie if to each node $n \in T$ and each $a \in \Sigma$ there is at most one $n' \in T$ such that $n \xleftarrow{a} n'$. A (finite) enumerated $\Sigma$-trie $T = (T, \leq, \xleftarrow{a}, 0, \text{Succ})_{a \in \Sigma}$, or an LZ-trie for short, is a $\Sigma$-trie $(T, \leq, \xleftarrow{a}, 0)$ with a successor relation $\text{Succ}$ on $T$ that is compatible with the partial order $\leq$. We assume that $T = \{0, 1, 2, \ldots, m\}$ and $\text{Succ}(i, j)$ iff $i + 1 = j$ in $\mathbb{N}$.

Example 1 (Cont.). Enumerating the pairs of $LZ(w)$ by 1, 2, etc. in a third component, we obtain a sequence $(0, a, 1)(0, b, 2)(2, b, 3)(1, a, 4)(3, a, 5)(3, b, 6)$ of triples. These represent a tree in which block $B_k$ labels the path from the root 0 to node $k + 1$:

```
    a
   / \   \
  0   1   2
 /   / \
  a   b   \\
 4   3   5
   \
    b
```

A tuple $p_n = (k, a)$ of $LZ(w)$ is drawn as an edge $k \xleftarrow{a} n + 1$.

We write $T_\alpha$ for the enumerated trie representing the compressed word $\alpha$. We always assume that our strings $w$ have a distinguished end symbol; then the final block of $LZ(w)$ is different from the previous ones and the tree of blocks indeed is a trie.

Remark 2. What differs in choosing $G_\alpha$ or $T_\alpha$ is the logical language used to talk about $LZ$-compressed strings $\alpha$. Basically, we have

$$G_\alpha \models E(i, j) \land U_a(i) \iff T_\alpha \models (j + 1) \xleftarrow{a} (i + 1).$$

Modulo the additional root node in the trie, $\geq$ in the trie amounts to $E^*$ in the graph, and $\leq$ in the graph to $\text{Succ}^*$ in the trie. Since $E^*$ resp. $\text{Succ}^*$ is $\exists$- and $\forall$-MSO-definable in the language of $LZ$-graphs resp. $LZ$-tries, $\exists$-MSO-properties of $LZ$-graphs translate to $\exists$-MSO-properties of $LZ$-tries and vice versa. Using [1], quantifiers $Qx$ and $QX$ about strings translate to bounded quantifiers $Q(x, y) \in E^*$ and $QX \subseteq E^*$. So when translating to the language of $LZ$-tries we only quantify over tuples (and relations of such) whose components lie on a path of $T_{LZ(w)}$.

---

2 i.e. a minimal binary relation $\text{Succ}$ whose transitive reflexive closure $\text{Succ}^*$ is a total ordering of $T$. 
3 MSO-Equivalence for LZ-Graphs

For relational structures $A, B$ of the same signature, $A \equiv^r_{MSO} B$ says that $A$ and $B$ satisfy the same MSO-sentences of quantifier rank $\leq r$.

Two facts about MSO for strings imply the existence of finite automata that can check MSO-properties of strings (cf. [4]):

a) for each $r$, there are only finitely many $\equiv^r_{MSO}$-equivalence classes for word structures $S_w$, and

b) for compound strings $wa$, the $\equiv^r_{MSO}$-class of $S_{wa}$ depends only on the $\equiv^r_{MSO}$-classes of $S_w$ and $S_a$.

(The analogous situation holds for trees over $\Sigma$.) LZ-compressed words $\alpha = p_0 \cdots p_{m-1}$ are words over the infinite alphabet $\mathbb{N} \times \Sigma$. Can we check MSO-properties of compressed strings by a kind of finite automaton for LZ-graphs?

Since we deal with a finite relational language, we still have a):

**Proposition 1.** For each $r$, the equivalence relation $\equiv^r_{MSO}$ between LZ-graphs has finite index.

But what about b)? By extending winning strategies for duplicator in the Ehrenfeucht-Fraisse-game $G_r(G_\alpha, G_\beta)$, we can show:

**Lemma 1.** (i) For LZ-compressed words $\alpha(0,a), \beta(0,a')$ over $\Sigma$,

$$G_\alpha \equiv^r_{MSO} G_\beta \land a = a' \implies G_{\alpha(0,a)} \equiv^r_{MSO} G_{\beta(0,a')}.$$  

(ii) For LZ-compressed words $\alpha(k+1,a)$ and $\beta(k'+1,a')$ over $\Sigma$,

$$(G_\alpha, k) \equiv^r_{MSO} (G_\beta, k') \land a = a' \implies G_{\alpha(k+1,a)} \equiv^r_{MSO} G_{\beta(k'+1,a')}.$$  

However, in (ii) one has to assume that the elements $k, k'$ pointed to from the new maximal elements share the same properties. Instead, one would need the stronger claim

$$G_\alpha \equiv G_\beta \land G_\alpha|k \equiv G_\beta|k' \land a = a' \implies G_{\alpha(k+1,a)} \equiv G_{\beta(k'+1,a')}.$$  

where $G_\alpha|k$ is the restriction of $G_\alpha$ with $k$ as its maximal element. The equivalence class of $G_\alpha|k$ would be the automaton state assigned to $k$. (Notice that $G_\alpha|k$ is a LZ-graph, but $(k+1,a)$ is not a LZ-compressed word.) The problem is that duplicator’s winning strategies for $G_r(G_\alpha, G_\beta)$ and $G_r(G_\alpha|k, G_\beta|k')$ may pick different elements to answer spoilers playing of some element of, say, $G_\alpha|k$.

Thus, unlike in the case of strings or trees, for compound compressed words $\alpha(k,a)$ we have component LZ-graphs $G_\alpha$ and $G_\alpha|k$ that are not disjoint, and winning strategies in games for these do not combine to winning strategies for composed LZ-graphs.

From this we conclude that we cannot use a Büchi-Mylhill-Nerode construction to obtain from the $\equiv^r_{MSO}$-classes a finite sequential automaton for LZ-graphs, and likewise for LZ-tries.
4  **LZ-Trie-Automata**

If we view LZ-tries as trees with an additional edge $Succ$ between nodes, we obtain directed acyclic graphs of a special kind: the successor child may be equal to some descendant with respect to the $\leftarrow$-child-relations. Since these are still very close to trees, it is natural to use a variation of tree-automata as an approximative notion of LZ-automaton for checking properties of LZ-tries.

**Definition 3.** Let $n \in G$ be a node in a graph $(G, E)$. For $m \in \mathbb{N}$, the sphere of radius $m$ around $n$, $s_m(n)$, is the set of nodes $k \in G$ such that there is an $E$-path of length $\leq m$ from $n$ to $k$ or vice versa. The hemisphere of radius $m$ around $n$, $hs_m(n)$, is the set of nodes $k$ such that there is an $E$-path of length $\leq m$ from $n$ to $k$.

**Definition 4.** Let $n \in T$ be a node in the LZ-trie $T$. The bottom-up LZ-hemisphere of radius $m$ around $n$, $bu-hs^T_m(n)$, is the restriction of $T$ to the $m$-hemisphere around $n$ in the graph $(T, E)$, where

$$ E := \bigcup \{ \leftarrow^a \mid a \in \Sigma \} \cup \{Succ\}. $$

The top-down LZ-hemisphere of radius $m$ around $n$, $td-hs^T_m(n)$, is the restriction of $T$ to the $m$-hemisphere around $n$ in the graph $(T, \bar{E})$, where $\bar{E}$ is the converse of $E$. An LZ-hemisphere is an LZ-hemisphere of some radius around some node in some LZ-trie $T$.

**Example 1 (Cont.).** An LZ-trie and the bottom-up resp. top-down 2-hemispheres of node 3 (with dashed edges for $Succ$ resp. $Pred$):

![Diagram](image-url)

**Definition 5.** A finite bottom-up (resp. top-down) $m$-LZ-automaton $A = (Q, \Sigma, \delta, q_m, F)$ consists of a finite set $Q$ of states, sets $I, F \subseteq Q$ of initial and final states, a finite alphabet $\Sigma$, a finite transition relation $\delta$ consisting of pairs $(P, q)$, written $P \rightarrow q$, where $q \in Q$ and $P$ is a bottom-up (resp. top-down) LZ-hemisphere of radius $m$ whose nodes except the root are labelled by elements of $Q$.

A run of $A$ on an LZ-trie $T$ is a function $r : T \rightarrow Q$ where $r(\text{max}) \in I$ (resp. $r(0) \in I$) and for each $n \in T$ there is some $(P, q) \in \delta$ such that $bu-hs^T_m(n)$
(resp. $\text{td-hs}_m^T(n)$), expanded by the labelling of nodes given by $r$, is isomorphic to $P$ with label $q$ at its root. $A$ accepts $T$ if there is a run $r$ of $A$ on $T$ such that $r(0) \in F$ (resp. $r(\text{max}) \in F$). Let $L(A) := \{ T \mid A$ accepts $T \}$ be the class of LZ-tries accepted by $A$.

$A$ is deterministic if $|I| = 1$ and $q = q'$ when $(P,q),(P,q') \in \delta$.

While an $m$-LZ-automaton $A$ sequentially follows the enumeration of a trie $T_\alpha$, it can access some states reached at suffixes (resp. prefixes) of $\alpha$. Strictly speaking, it does not have a ‘finite memory’.

### 4.1 Bottom-Up LZ-Trie-Automata

A 1-LZ-automaton working bottom-up the LZ-trie towards the root has transitions that determine the state at a node from the states at the node’s $\Sigma$-children and successor. But it also has to distinguish which of the $\Sigma$-children is the successor of the node, if any.

**Example 2.** Consider the class $K$ of LZ-tries over $\Sigma = \{ a, b \}$ which have a node whose successor and $a$-child agree, i.e. which satisfy the sentence $\varphi := \exists x \exists y [y = x + 1 \land (x \leftarrow y)]$. We give a bottom-up 1-LZ-automaton $A$ accepting $K$. We write a transition in the form

$$(q_a, q_b, q_{\text{succ}}, i) \rightarrow p,$$

where $q_a, q_b, q_{\text{succ}}$ are the states of the $a$-, $b$- and $\text{Succ}$-child or $\bot$, when there is no such child, and $i \in \{ 1, 2, 3, \bot \}$ says which of the children is equal to the successor node, if any. Thus, $(p, q, p, 1) \rightarrow q'$ corresponds to the transition $P \rightarrow q'$ which can be shown as

$$\begin{align*}
 & q'(n) \\
 \text{p(n+1)} & \quad \quad \text{q(m)} \\
 & a \quad \quad b
\end{align*}$$

$A$ has a final state $q_1$, which is assigned to all ancestors of the root of a subtrie satisfying $\varphi$, and an initial state $q_0$, which is assigned to all other nodes of the input trie. Letting $q,p,q'$ range over $\{ q_0, q_1 \}$, the transition table is

- $a) (\bot, \bot, \bot, \bot) \rightarrow q_0$
- $b) (q, \bot, q, 1) \rightarrow q_1$
- $c) (q,p,q,1) \rightarrow q_1$
- $d) (q,p,q',2) \rightarrow q'$
- $e) (q,p,q',3) \rightarrow q'$
- $f) (q,\bot,q',3) \rightarrow q'$
- $g) (\bot,q,q',3) \rightarrow q'$
- $h) (\bot,\bot,q',3) \rightarrow q'$.

Rule $a)$ means that if there is no successor-node, $A$ is in state $q_0$. Rules $b)$ and $c)$ say that if the successor-node is the $a$-child, then $A$ goes to $q_1$ as we just saw the pattern $\varphi$. Rules $d) - f)$ say that if the successor node differs from the $a$-child, $A$ remains in the state of the successor node. Similar for $g)$ and $h)$, when there is no $a$-child.
**Theorem 2.** There is a property of LZ-tries that is recognized by a non-deterministic bottom-up 1-LZ-automaton but not by any deterministic bottom-up m-LZ-automaton.

**Proof.** (Sketch) Let $\Sigma = \{a, b, c\}$ and consider the following property $\varphi$ of enumerated $\Sigma$-tries:

There are two subsequent nodes $i - 1$ and $i$ such that some node $j$ is both the \textit{a}-predecessor of $i$ and a \textit{b}-ancestor of $i - 1$.

An LZ-trie satisfies $\varphi$ iff it contains nodes linked as follows (in the trie and in the LZ-graph, respectively), where $j < i - 1$:

When finding nodes connected like $i - 1$ and $i$, a bottom-up 1-LZ-automaton can (non-deterministically) guess that a $b$-ancestor of $i - 1$ will be the $a$-parent of $i$ and check this while proceeding. But no deterministic bottom-up $m$-LZ-automaton can check the property $\varphi$ because, intuitively speaking, the $m$-hemisphere of a node $j$ is not big enough to see if the predecessor of its $a$-child is one of its $b$-descendants. The proof details are not quite obvious but have to be omitted for lack of space.

### 4.2 Top-Down LZ-Trie-Automata

G. Navarro [5] has shown how to do regular expression search on LZ-78-compressed texts by simulating an automaton reading the original text, beating the decompression-and-search approach by a factor of 2. Actually, this simulation is a deterministic top-down 1-LZ-automaton on the LZ-compressed text:

**Theorem 3.** The LZ-compression $\{T_{LZ}(w) \mid w \in R\}$ of any regular set $R \subseteq \Sigma^+$ is accepted by a deterministic top-down 1-LZ-automaton.

**Proof.** Let $A = (Q, \Sigma, q_0, \delta, F)$ be a deterministic finite automaton accepting $R$. Define a deterministic top-down 1-LZ-automaton $A' = (Q', \Sigma, \delta', q'_0, F')$ by $Q' := Q \times (Q \to Q)$, $q'_0 := (q_0, \lambda q. q)$, $F' := \{(q, f) \mid q \in F\}$ and $\delta'$ according to the following transitions:

$$0 \cdot (q_0, \lambda q. q)$$

$$\delta(f'(p), a), \lambda q. \delta(f'(q), a)$$
for each $a \in \Sigma$ (including the case $i = k$).

Suppose $r'$ is a run of $A'$ on $T_{LZ}(w)$ for a compressed word $LZ(w) = p_0 \cdots p_{m-1}$ where $p_k$ represents block $B_k$ of $w = B_0 \cdots B_{m-1}$. By induction we see that for each $k < m$,

$$r'(k) = (\delta(q_0, B_0 \cdots B_{k-1}), \lambda p. \delta(p, B_{k-1})).$$

For $k > 0$, suppose $r'(i) = (p, f) = (\delta(q_0, B_0 \cdots B_{i-1}), f)$ and $p_i = (k, a)$. Then $B_i = B_{k-1}a$, and with $r'(k) = (p', f') = (p', \lambda p. \delta(p, B_{k-1}))$

$$r'(i + 1) = (\delta(f'(p), a), \lambda q. \delta(f'(q), a))
\begin{align*}
  &= (\delta(p, B_{k-1}a), \lambda q. \delta(q, B_{k-1}a)) \\
  &= (\delta(\delta(q_0, B_0 \cdots B_{i-1}), B_i), \lambda q. \delta(q, B_i)) \\
  &= (\delta(q_0, B_0 \cdots B_i), \lambda q. \delta(q, B_i)).
\end{align*}$$

Hence $A'$ accepts $\{T_{LZ}(w) \mid w \in R\}$, because

$$w \in R \iff \delta(q_0, B_0 \cdots B_{m-1}) \in F \iff r'(m) \in F'.$$

### 4.3 Graph Acceptors and $\exists$-MSO for LZ-Tries

If we consider the states $q$ of an automaton $A$ as monadic predicates on nodes of tries, the condition “there is an accepting $A$-run” can be expressed by an $\exists$-MSO-sentence $\exists q \psi$ in the language of LZ-tries:

**Proposition 2.** For every $m$-LZ-automaton $A$ there is an $\exists$-MSO-sentence $\varphi_A$ defining the class of LZ-tries accepted by $A$.

For sentences, this improves on the translation of Theorem $[1]$.

**Corollary 1.** Every property of strings which is definable in MSO on strings is definable in $\exists$-MSO on LZ-tries.


The converse of Corollary $[1]$ is wrong: $\{a.b. \cdots .b^n.a.b^n.b \mid n \in \mathbb{N}\}$ is not a regular language, but has an FO-definable class of LZ-tries.

To show the converse of Proposition $[2]$ we use the graph acceptors for directed acyclic graphs presented by W. Thomas $[7]$.

On graphs $G = (V, E)$, a tile $\delta$ over a set $Q$ is an $m$-neighbourhood of a node with a labelling in $Q$, i.e. a finite node-labelled graph. A graph acceptor is a triple $A = (Q, \Delta, Occ)$ where $\Delta$ is a finite set of tiles over the finite set $Q$ and $Occ$ is a boolean combination of conditions $\chi_{p, \delta} := there\ are\ \geq p\ occurrences\ of\ tile\ \delta$. A run of $A$ on $G$ is a function $r : V \rightarrow Q$ such that each $m$-neighbourhood of $G$ becomes a tile $\delta \in \Delta$. Then $A$ accepts a graph $G$ if there exists a run of $A$ on $G$ whose tiles satisfy $Occ$.

Note that the existence of an accepting run is non-constructive. The main result of $[7]$ says that a class of graphs of bounded degree is definable in $\exists$-MSO iff it is accepted by a graph acceptor.
Theorem 4. A class $\mathcal{K}$ of LZ-tries is $\exists$-MSO-definable iff for some $m$, $\mathcal{K}$ is accepted by an $m$-LZ-automaton.

Proof. Note that each LZ-trie $T$ satisfies the following conditions:

(i) each node of $T$ has a degree $\leq |\Sigma| + 1$,
(ii) $T$ is an acyclic (directed) graph with a designated out-edge (the successor) for each node,
(iii) $T$ has a node that is reachable from any node by a path.

Using (i), by Theorem 3 of [7], $\mathcal{K}$ is $\exists$-MSO-definable iff $\mathcal{K}$ is recognizable by a graph acceptor. Moreover, from (i)-(iii) and Proposition 6 of [7], it follows that $\mathcal{K}$ is recognizable by a graph acceptor iff it is recognizable by a graph acceptor without occurrence constraints.

$\Leftarrow$: Suppose $\mathcal{K}$ is accepted by some $m$-LZ-automaton $A$. We may assume that final states of $A$ do not occur in the $m$-hemispheres of transitions $(P, q)$ of $A$. Consider the transitions of $A$ as a tiling system. Then an accepting run of $A$ is a tiling that uses at least one of the tiles $(P, q)$ where $q \in F$. Thus, $\mathcal{K}$ has a graph acceptor.

$\Rightarrow$: By the above remarks, $\mathcal{K}$ is recognizable by a graph acceptor without occurrence constraints. On the LZ-tries, the tiles have a root node, so we can view each tile as a transition rule saying that the automaton enters a state at the root depending on the $m$-sphere of the root and the states at the descendants in the sphere. Let each state be final; then the automaton accepts iff there is a tiling.

A graph $G = (V, E)$ is $k$-colorable if there is a partitioning of the set $V$ of nodes into at most $k$ classes (colors) such that any two adjacent nodes belong to different classes.

Clearly, $k$-colorability can be expressed by an $\exists$-MSO sentence, and hence has a graph acceptor. While 3-colorability on arbitrary graphs is an NP-complete problem, it is trivial on LZ-tries, because the maximal node $m$ is connected to at most two nodes:

Proposition 3. Every LZ-trie is 3-colorable.

Potthoff e.a. [6] have shown that on the class of directed acyclic graphs whose edges are uniquely labelled with a bounded number of labels, 2-colorability is not recognizable by a deterministic graph acceptor, i.e. one with at most one accepting run on each graph. For the subclass of LZ-tries, however, it is (actually by a 4-state deterministic bottom-up-LZ-automaton):

Proposition 4. On the class of LZ-tries, 2-colorability is recognizable by a deterministic graph acceptor.

4.4 Strictness of the Automata Hierarchy

The bottom-up resp. top-down hierarchies of $m$-LZ-automata are strict, and no level exhausts the MSO-properties of tries:
Theorem 5. For every $m$, there is an MSO-sentence defining a class of LZ-tries that is not accepted by any $m$-LZ-automaton.

Proof. For $m = 1$, let $L = \{b^1b^2 \cdots b^nab^1ab^2a \cdots b^{n-1}a \mid n \in \mathbb{N}\} \subseteq \{a, b\}^+$. Each $w \in L$ has a LZ-block decomposition as indicated by

$$w_n = b^1.b^2.\cdots.b^n.a.b^1.a.b^2.\cdots.b^{n-1}a$$

and a compression

$$LZ(w_n) = (0, b)(1, b) \cdots (n-1, b)(0, a)(1, a) \cdots (n-1, a).$$

The corresponding tries $T_{LZ}(w_n)$ look like

![Diagram](image)

where the successor relation is given by the node numbers.

The class of enumerated tries in $LZ(L)$ can be defined by the MSO-sentence $\varphi$ saying that the set $B$ of nodes that are the root 0 or a $b$-child, has the following properties (of which (i) is neither $\exists$-MSO nor $\forall$-MSO):

(i) $B \supseteq \{0\}$ is the smallest set being closed under $b$-children,

(ii) a node is not in $B$ iff it is an $a$-child of a node in $B$,

(iii) for all nodes $x, y$, we have $y = x + 1$ iff one of the following holds:

(a) $y$ is the $b$-child of $x$,

(b) $y$ is the $a$-child of the $b$-child $y'$ of some $x'$ whose $a$-child is $x$,

(c) $y$ is the $a$-child of 0 and $x$ the member of $B$ that has no $b$-child.

Claim. $LZ(L)$ is not accepted by a 1-bottom-up-LZ-automaton.

Suppose that $A$ is a 1-bottom-up-LZ-automaton that accepts the class defined by $\varphi$. Let $m > |Q|$ and $w = w_{2m}$. An accepting run of $A$ on $T_{LZ}(w)$ assigns the same state, say $q$, to at least two different $a$-childs, say nodes $n + k + 2$ and $n + 2$. The state of their predecessors is determined by a rule

$$(\bot, \bot, q, 3) \rightarrow p.$$ 

Let $T'$ be like $T_{LZ}(w)$, except that nodes $n + 1$ and $n + k + 1$ are switched in the ordering. Then $A$ will also accept the modified structure, as indicated by the states assigned to nodes as follows:
But the enumeration of $a$-children in $T'$ does not conform to $\varphi$, so $T' \notin T(A) \setminus LZ(L)$.

For the case $m > 1$, we modify the example as follows: along the $B$-part, between two nodes that have both an $a$-child and a $b$-child, we add $m - 1$ nodes that have no $a$-child, i.e. the subgraphs

The $LZ$-tries arising this way are the compressions of the words

\[ w_{n,k} = b^1.b^2.b^3.\ldots.b^{nm+k}.a.b^m.a.b^2m.a.\ldots.b^{nm}a. \]

Then \( \{T_{LZ(w_{n,k})} \mid n, k \in \mathbb{N}, k \leq m \} \) is accepted by a $(m+1)$-bottom-up-$LZ$-automaton, hence MSO-definable. But it is not accepted by any $m$-bottom-up-$LZ$-automaton: intuitively, the $m$-hemisphere of a node $k$ does not tell whether the path from $k$ following $a$ and successor and the path following $b^m a$ end at the same node.

A similar argument shows that the class defined by $\varphi$ is also not accepted by any $m$-top-down-$LZ$-automaton with the same $m$.

## 5 Conclusion

We have shown some relations between properties of strings, properties of their Lempel-Ziv-78-compressions, and automata accepting compressed strings, but the picture is not complete yet. For example:

**Conjecture 1.** On the class of all $LZ$-tries, not every MSO-formula is equivalent to an $\exists$-MSO-formula.
It seems likely that the class of acceptable $LZ$-tries is not closed under complement (cf. Theorem 2). We also believe that deterministic bottom-up-$LZ$-automata are much weaker than deterministic top-down-$LZ$-automata (cf. Theorem 3):

**Conjecture 2.** There is no deterministic bottom-up $m$-$LZ$-automaton that can check on $LZ(w)$ whether $w \in \{a, b\}^+$ has a subword $ab$.

References

Abstract. Complexity of provability and satisfiability problems in many non-classical logics, for instance, in intuitionistic logic, various systems of modal logic, temporal and dynamic logics was studied in [3, 5, 9, 22–24]. Ladner [9] proved that the provability problem is \textsc{PSPACE}-complete for modal logics K, T and S4 and \textsc{coNP}-complete for S5. Statman [24] proved that the problem of determining if an arbitrary implicational formula is intuitionistically valid is \textsc{PSPACE}-complete.

We consider the complexity of some properties for non-classical logics which are known to be decidable. The complexity of tabularity, pre-tabularity, and interpolation problems in extensions of the intuitionistic logic and of the modal logic S4 is studied, as well as the complexity of amalgamation problems in varieties of Heyting algebras and closure algebras.

1 Main Results

A superintuitionistic logic is a set of modality-free formulas containing the set \textsc{Int} of all intuitionistically valid formulas and closed under substitutions and modus ponens. We consider the families \( E(\text{Int}) \) of all superintuitionistic logics and \( \text{NE}(\text{S4}) \) of the normal extensions of S4. Logics in \( \text{NE}(\text{S4}) \) are, in addition, closed under the necessitation rule \( A/\Box A \) and contain all formulas valid in S4. For any logic \( L \), we denote by \( L + A \) the extension \( L \) by an extra axiom scheme \( A \). In particular,

\[
\begin{align*}
\text{Cl} &= \text{Int} + (p \lor \neg p), \\
\text{For} &= \text{Int} + \bot, \\
\text{KC} &= \text{Int} + (\neg p \lor \neg \neg p), \\
\text{LC} &= \text{Int} + (p \rightarrow q) \lor (q \rightarrow p), \\
\text{S4} &= \text{K} + (\Box p \rightarrow p) + (\Box p \rightarrow \Box \Box p), \\
\text{S4.1} &= \text{S4} + (\Box \Box p \rightarrow \Box p), \\
\text{S4.2} &= \text{S4} + (\Diamond \Box p \rightarrow \Box \Diamond p), \\
\text{S4.3} &= \text{S4} + (\Box (\Box p \rightarrow q) \lor \Box (\Box q \rightarrow p)), \\
\text{S5} &= \text{S4} + (\Diamond p \rightarrow \Box \Diamond p), \\
\text{Grz} &= \text{S4} + (\Box (\Box (p \rightarrow \Box p) \rightarrow p) \rightarrow p), \\
\text{Grz.2} &= \text{Grz} + \text{S4.2}, \\
\text{Grz.3} &= \text{Grz} + \text{S4.3}.
\end{align*}
\]

A logic is called tabular if it can be characterized by finitely many finite models; and pre-tabular if it is maximal among the non-tabular logics. A logic \( L \) is called locally
If for any finite set $P$ of propositional variables there exist only finitely many formulas of variables in $P$ non-equivalent in $L$. It is known that each tabular logic is locally tabular.

A logic $L$ is said to have $Craig’s$ interpolation property, denoted CIP, if for every formula $(A \rightarrow B) \in L$ there exists a formula $C$ such that (i) both $(A \rightarrow C) \in L$ and $(C \rightarrow B) \in L$, and (ii) every variable of $C$ occurs in both $A$ and $B$. A logic $L$ is said to have the interpolation property, denoted IPD, if $A \vdash_L B$ implies that exists a formula $C$ such that (i) both $A \vdash_L C$ and $C \vdash_L B$, and (ii) every variable of $C$ occurs in both $A$ and $B$.

We assume that the language contains the propositional constants $\top$ or $\bot$. A size of a formula $A$, denoted $|A|$, is the number of occurrences of variables and logical symbols in $A$.

Let $L$ be Int or some modal logic. By the tabularity (pretabularity etc.) problem over $L$ we mean the problem of determining for an arbitrary formula $A$, whether $L + A$ is tabular (pretabular, etc.), and consider its complexity with respect to the size of $A$.

One can find the necessary definitions from Complexity Theory in [7, 21]. In Section 3 we prove the following result.

**Theorem 1** (i) The tabularity problems over both Int and S4 are NP-complete.
(ii) The pretabularity problems over both Int and S4 are both NP-hard and coNP-hard, and belong to in $\Delta^p_2$.
(iii) The local tabularity problem over S4 is NP-complete.

It is not known whether the local tabularity problem over Int is decidable.

The interpolation problem is considered in Section 4. We prove the following results.

**Theorem 2** (i) The interpolation problems over both Int and Grz are PSPACE-complete.
(ii) The problem of determining whether Int + $A$ is a tabular logic with CIP is NP-complete.
(iii) Both CIP and IPD problems over S4 are in coNEXP and PSPACE-hard.
(iv) The problems of determining whether S4 + $A$ is a tabular (pretabular or locally tabular) logic with CIP (or IPD) are both NP-hard and coNP-hard, and belong to $\Delta^p_2$.

There exists a well-known duality between $E(Int)$ and the family of varieties of Heyting algebras, and also between $NE(S4)$ and the family of varieties of closure algebras. If $Ax$ is a set of formulas and $L = Int + Ax$ then $\{A = T \mid A \in Ax\}$ forms a base of identities for the variety $V(L)$ associated with $L$. The definitions for $L \in NE(S4)$ are analogous. On the other hand, with any identity $A = B$ one can associate a formula $A \leftrightarrow B$ which is valid in an algebra if and only if the identity $A = B$ holds in this algebra. By rewriting Theorem 2 in the algebraic language, we obtain the following result.

**Theorem 3** (i) The amalgamability problem for finitely based varieties of Heyting algebras is PSPACE-complete.
(ii) The amalgamability and super-amalgamability problems for finitely-based varieties of closure algebras are \(\text{PSPACE}\)-hard and belong to \(\text{coNEXP}\). 

2 Complexity and Decision Problems

We assume the knowledge of the standard material in complexity theory which can be found, for example, in [21]. In this paper we will use complexity classes \(\text{P}, \text{NP}, \text{EXP}, \text{NEXP}, \text{PSPACE}, \text{coNP}, \text{coNEXP}, \text{DP},\) and \(\Delta^p_2\).

If we take two superintuitionistic logics \(L_1\) and \(L_2\) then \(L_1 + L_2\) and \(L_1 \cap L_2\) are superintuitionistic logics, too. The former logic can be axiomatized by all extra axioms of \(L_1\) and \(L_2\). For an axiomatization of the intersection one can use Miura’s theorem [20]. To formulate it, we recall some definitions.

If \(A\) and \(B\) are two formulas, we denote by \(A \lor' B\) a disjunction \(A \lor B'\), where \(B'\) is obtained from \(B\) by renaming variables so that \(A\) and \(B'\) have no variables in common.

It was proved by Miura (see [20]) that the intersection \(L_1 \cap L_2\) of two logics \(L_1, L_2 \in E(\text{Int})\) is axiomatizable by the set of formulas \(A \lor' B\), where \(A\) ranges over the set of extra axioms of \(L_1\) and \(B\) over the set of extra axioms of \(L_2\). For logics in \(\text{NE}(S4)\) one should take the set of formulas \(\Box A \lor' \Box B\) for all extra axioms \(A\) of \(L_1\) and \(B\) of \(L_2\) [19].

With each logic \(L\) one can associate the following decision problems.

1. Given a formula \(A\), is this formula provable in \(L\) (\(L\)-provability);
2. The dual problem of unprovability in \(L\);

In addition, with each pair of logic \(L_0, L\) one can associate the following decision problem.

3. Given a formula \(A\), is the logic \(L_0 + A\) equivalent to \(L\) (equivalence to \(L\) over \(L_0\));

The equivalence problem between logics will play an important role in this paper. In general, the equivalence problem is undecidable. When we restrict ourself to considering particular families of logics, for instance, superintuitionistic logics, then the problem of equivalence to a particular logic \(L\) may be undecidable too. On the other hand, this problem is decidable if we take into consideration only superintuitionistic logics and deal with their equivalence to one of logics \(\text{Cl}, \text{KC},\) or \(\text{LC}\).

For arbitrary fixed \(L_1 \in E(\text{Int})\) we consider two inclusions \(\text{Int} + A \subseteq L_1\) and \(L_1 \subseteq \text{Int} + A\). The former inclusion is the same as provability of \(A\) in \(L_1\), and the latter is the same as provability of extra axioms of \(L_1\) in \(\text{Int} + A\). It is evident that for any logic \(L_1 \in E(\text{Int})\), the logic \(\text{Int} + A\) is equivalent to \(L_1\) if and only if both inclusions \(\text{Int} + A \subseteq L_1\) and \(L_1 \subseteq \text{Int} + A\) hold. On the other hand, we have

**Proposition 4.** Let \(L_0\) be a logic in \(E(\text{Int})\) or in \(\text{NE}(S4)\) and \(L_1\) be a finitely axiomatizable extension of \(L_0\). Then each of the inclusion problems \(L_0 + A \subseteq L_1\) and \(L_1 \subseteq L_0 + A\) is linearly reducible to equivalence to \(L_1\) over \(L_0\).

**Proof.** Let us consider the case \(L_0 \in E(\text{Int})\). Let \(B\) be a single axiom of \(L_1\) w.r.t \(L_0\), i.e. \(L_1 = L_0 + B\). Evidently, \(L_0 + A \subseteq L_1\) if and only if \(L_0 + (A \& B)\) is equivalent
to \(L_1\), so the first inclusion problem is polynomial-time reducible to equivalence to \(L_1\) over \(L_0\).

Further, \(L_1 \subseteq L_0 + A\) if and only if \(L_1 = L_1 \cap (L_0 + A)\). By Miura’s theorem, 
\(L_1 \cap (L_0 + A)\) is \(L_0 + (B \lor A)\), so \(L_1 \subseteq L_0 + A\) if and only if \(L_1 = L_0 + (B \lor A)\),
which gives us a reduction of the second inclusion.

The proof for the case \(L_0 \in NE(S4)\) is analogous, but one should use the formula
\(\Box B \lor \Box A\) instead of \(B \lor A\).

If \(S\) is a set of extensions of a given logic \(L_0\), we can consider the problem of membership in \(S\) (over \(L_0\)), i.e. the problem of recognizing whether \(L_0 + A\) is in \(S\).

We say that a logic \(L\) has the Hallden Property, denoted \(HP\) if for any formulas \(A\) and \(B\) without common variables, \(A \lor B \in L\) implies \(A \in L\) or \(B \in L\).

**Proposition 5** Let \(L_0\) be Int or a logic in \(NE(S4)\) and \(S\) be a family of extensions of \(L_0\) containing a finitely axiomatizable logic \(L_1 = L_0 + A\) satisfying the conditions:

1. \((\exists L' \supset L_1)(\forall L)((L_1 \subseteq L \subseteq L' \text{ and } L \in S) \Rightarrow L = L_1)\),
2. \(L_1\) has the Hallden property.

Then the problem of \(L_1\)-provability is linearly reducible to the problem of membership in \(S\) over \(L_0\).

**Proof.** Suppose that \(L_1 = \text{Int} + A, A'\) is any formula in \(L' - L_1\), and \(B\) is an arbitrary formula. Consider the formula \(A \& (A' \lor B)\).

If \(L_1 \vdash B\), then \(L_1 \vdash A' \lor B\), so \(\text{Int} + A \& (A' \lor B) = L_1 + (A' \lor B) = L_1 \subseteq S\).

If \(L_1 \not\vdash B\), then \(L_1 \not\vdash A' \lor B\) by the Hallden property, so \(L_1 + (A' \lor B) \not= L_1\). On the other hand, \(L_1 + (A' \lor B) \subseteq L'\). By condition (1) we get \(\text{Int} + A_1 \& (A' \lor B) = L_1 + (A' \lor B) \not\in S\), so we have

\[L_1 \vdash B \iff (\text{Int} + A_1 \& (A' \lor B)) \in S.\]

The proof for modal logics is carried out analogously, by inserting \(\Box\) in formulas.

Let us now prove some lower bounds on complexity.

**Proposition 6** Let \(L_0\) be a logic in \(E(\text{Int})\) or in \(NE(S4)\), \(S\) be a family of extensions of \(L_0\), and \(L_1, L_2\) be two finitely axiomatizable extensions of \(L_0\) such that \(L_1 \subset L_2\).

1. If \(L_1 \not\in S\) and \(L_2 \in S\) then the problem of membership in \(S\) over \(L_0\) is \(NP\)-hard.
2. If \(L_1 \in S\) and \(L_2 \not\in S\) then the problem of membership in \(S\) over \(L_0\) is \(coNP\)-hard.

**Proof.**

1. Take, for example, \(L_0 = \text{Int}\). Let \(L_1 = \text{Int} + A_1, L_2 = \text{Int} + A_2\). For an arbitrary formula \(B\) we have (by Glivenko’s theorem)

\[B\text{ is satisfiable} \Rightarrow \text{Int} + \neg B\text{ is inconsistent, i.e. }\text{Int} + \neg B = \text{For};\]

\[B\text{ is unsatisfiable} \Rightarrow Cl \vdash \neg B \Rightarrow \text{Int} \vdash \neg B, \text{ i.e., }\text{Int} + \neg B = \text{Int}.\]
We immediately obtain

\[ \text{Int} + (A_1 \& (A_2 \lor' \neg B)) = L_1 + (A_2 \lor' \neg B) = (L_1 + A_2) \cap \text{For} = L_2 \in S. \]

If \( B \) is unsatisfiable, then

\[ \text{Int} + (A_1 \& (A_2 \lor' \neg B)) = (L_1 + A_2) \cap (\text{Int} + A_1 + \neg B) = L_1 \notin S. \]

If \( L_0 \) is a modal logic in \( NE(S4) \), then for arbitrary modality-free \( B \) we have \( \text{Cl} \vdash B \) if and only if \( L_0 \vdash B \), so the prove for \( L_0 = \text{Int} \), we only must replace \( A_2 \lor' \neg B \) by \( \Box A_2 \lor' \Box \neg B \).

2. Similar to the previous case, but replace \( S \) by its complement. \( \square \)

We immediately obtain

**Corollary 7** Let \( L_0 \) be \( \text{Int} \) or \( S4 \), \( P \) a property of logic non-trivial on the class of finitely axiomatizable extensions of \( L_0 \). Then the problem of determining whether \( L_0 + A \) has the property \( P \) is either \( \text{NP-hard} \) or \( \text{coNP-hard} \).

**Proposition 8** Let \( L_0 \) be \( \text{Int} \) or \( S4 \) and \( S \) be a family of extensions of \( L_0 \). Let there exist finitely axiomatizable extensions \( L, L', L'' \) of \( L_0 \) such that \( L' \subset L \subset L'' \), \( L \in S \), and \( L', L'' \notin S \). Then the problem of membership in \( S \) over \( L_0 \) is \( \text{DP-hard} \).

**Proof.** Assume \( L_0 = \text{Int} \), \( L = \text{Int} + A \), \( L' = \text{Int} + A' \), and \( L'' = \text{Int} + A'' \). For each pair \( \varphi, \psi \) of boolean formulas define

\[ f(\varphi, \psi) = A' \& (A'' \lor' \neg \varphi \lor' (A\&\neg \psi)). \]

We will prove that \( \varphi \) is satisfiable and \( \psi \) is unsatisfiable if and only if \( (\text{Int} + f(\varphi, \psi)) \in S \). To this end consider three cases.

1. \( \varphi \) is satisfiable and \( \psi \) is unsatisfiable. Then there exists a substitution \( s \) of \( \bot \) or \( \top \) for the variables of \( \varphi \) such that \( \text{Int} \vdash s(\varphi) \leftrightarrow \top \). So in the logic \( \text{Int} + f(\varphi, \psi) \) one can derive \( A' \& (A'' \lor' \neg \top \lor' (A\&\neg \psi)) \), that is \( A' \& (A'' \lor' (A\&\neg \psi)) \). It is clear that \( \text{Int} + A' \& (A'' \lor' (A\&\neg \psi)) \) contains \( \text{Int} + f(\varphi, \psi) \), hence these two calculi are equivalent. Further, \( \text{Cl} \vdash \neg \psi \), so \( \text{Int} \vdash \neg \psi \) by Glivenko’s theorem. It follows that \( \text{Int} + A' \& (A'' \lor' (A\&\neg \psi)) \) is equivalent to \( \text{Int} + A' \& (A'' \lor' A) \), hence also to \( \text{Int} + A = L \) by the condition \( L' \subset L \subset L'' \). Therefore, \( \text{Int} + f(\varphi, \psi) \in S \).

2. Both \( \psi \) and \( \varphi \) are satisfiable. Since instances of \( \varphi \) and \( \psi \) in \( f(\varphi, \psi) \) have no variables in common, there exists a substitution \( s \) of \( \top \) or \( \bot \) for the variables of \( \varphi \) and \( \psi \) in \( f(\varphi, \psi) \) such that \( \text{Int} \vdash s(\varphi) \leftrightarrow \top \) and \( \text{Int} \vdash s(\psi) \leftrightarrow \top \). As a consequence, \( \text{Int} + f(\varphi, \psi) \) is equivalent to \( \text{Int} + (A' \& A'') \), which in turn is equivalent to \( \text{Int} + A'' = L'' \), and \( \text{Int} + f(\varphi, \psi) \notin S \).

3. \( \varphi \) is unsatisfiable. We have \( \text{Cl} \vdash \neg \varphi \). By Glivenko’s theorem \( \text{Int} \vdash \neg \varphi \). Then \( \text{Int} + f(\varphi, \psi) \) is equivalent to \( \text{Int} + A' = L' \), and \( \text{Int} + f(\varphi, \psi) \notin S \).
The proof for normal extensions of $S4$ is analogous. We only should redefine

$$f(\varphi, \psi) = A' \& (\Box A'' \lor \neg \varphi \lor (\Box A \& \neg \psi)).$$

**Proposition 9** Let $L_0$ be in $E(\text{Int})$ or in $NE(\text{S4})$ and $L$ be any finitely axiomatizable extension of $L_0$.

(a) If $L$ is consistent, then the problem of provability in $L$ is $\text{coNP}$-hard and the problem of unprovability $\text{NP}$-hard.

(b) If $L \neq L_0$, then the problem of equivalence to $L$ over $L_0$ is $\text{NP}$-hard.

(c) If $L$ is consistent, then the problem of equivalence to $L$ over $L_0$ is $\text{coNP}$-hard.

(d) If $L \neq L_0$ and $L$ is consistent, then the problem of equivalence to $L$ over $L_0$ is $\text{DP}$-hard.

**Proof.** (a) $B$ is a satisfiable boolean formula iff $\neg B$ is not provable in $L_0$.

(b) and (c) follow from Proposition 6 by $S = \{L\}$.

(d) In Proposition 8 take $S = \{L\}$, $L' = L_0$, $L'' = \text{For} + \bot$, $L = L_0 + A$. Then $\varphi$ is satisfiable and $\psi$ is unsatisfiable if and only if $(L_0 + (\neg \varphi \lor (A \& \neg \psi)) = L \in S$.

**3 Tabularity and Related Properties**

To find out the complexity of the tabularity and related problems we need some computational characteristics of particular logics and their models. Although there exist logics in $E(\text{Int})$ and $NE(\text{S4})$ which are not Kripke-complete, our main results can be proved in terms of Kripke structures. We recall some definitions.

A frame $W = (W, R)$ is a non-empty set with a binary relation $R$. As we will consider only normal extensions of $S4$, all our frames are supposed to be reflexive and transitive. A Kripke structure $M = (W, R, \models)$ is a frame with a truth-relation $\models$, where the boolean operations are defined in the usual way, and $x \models \Box A$ iff $\forall y(xRy \Rightarrow y \models A)$.

For extensions of Int we need a different definition. An intuitionistic Kripke structure is a tuple $M = (W, \leq, \models_i)$ which is (a) partially ordered by $\leq$, (b) satisfies the monotonicity condition: for each variable $p$,

$$(x \models_i p \text{ and } x \leq y) \Rightarrow y \models_i p$$

and (c)

$$x \models_i A \rightarrow B \text{ iff } \forall y(x \leq y \Rightarrow (y \models_i A \Rightarrow y \models_i B)),

x \models_i \neg A \text{ iff } \forall y(x \leq y \Rightarrow y \models_i A).$$

The relations $x \models_i (A \& B)$ and $x \models_i (A \lor B)$ are defined as usual.

If $M = (W, \leq, \models)$ is a (modal or intuitionistic) Kripke structure, we say that a formula $A$ is true in $M$ if $x \models A$ for all $x \in W$; $A$ is satisfiable in $M$ if $x \models A$ for some $x \in W$; $A$ is refutable in $M$ if $x \not\models A$ for some $x \in W$. We say that a formula $A$ is valid in a frame $W$ and $W$ validates $A$ if $A$ is true in any Kripke structure $M = (W, \leq, \models)$ based on $W$; $A$ is refutable in $W$ if it is not valid in $W$. 


Let $L$ be a modal logic. A frame $W$ is called an $L$-frame if all formulas provable in $L$ are true in each Kripke structure based on $W$. If $L$ is a superintuitionistic logic, a frame $W$ is called an $L$-frame if all formulas provable in $L$ are true in each intuitionistic Kripke structure based on $W$. For intuitionistic or modal logic $L$, we say that a formula $A$ is $L$-valid if $A$ is valid in all $L$-frames; $A$ is $L$-refutable if it is not $L$-valid.

A logic $L$ is called Kripke-complete if provability in $L$ is equivalent to $L$-validity. A logic $L$ is said to have the finite model property (FMP) if provability in $L$ is equivalent to validity in all finite $L$-frames. It is clear that any logic with FMP is Kripke-complete. Each tabular logic $L$ can be characterized by a suitable finite $L$-frame. A logic $L$ is polynomially (exponentially) approximable if any formula $A$ unprovable in $L$ is refutable in some finite $L$-frame whose cardinality is bounded by a polynomial (respectively, exponential) function of the size of $A$.

**Remark.** For the aim of this paper, the notion of refutability is more important than satisfiability. In the case of modal logics one can easily see that a formula $A$ is satisfiable in a structure if and only if $\neg A$ is refutable in the same structure. There is no such a straightforward direct reduction for intuitionistic structures.

In our calculation we use a lemma whose proof is given in Proposition 3.1 of [5]:

**Lemma 10** (See [5].) Given a structure $M$ and a formula $A$, there is an algorithm for calculating the value of $A$ in $M$ that runs in time $O(||M|| \times |A|)$, where $||M||$ is the sum of the number of elements in the frame and the number of pairs in $R$.

As a consequence, we easily obtain the following result.

**Lemma 11** For each finite (intuitionistic or modal) frame $W$, refutability in $W$ is NP-complete and validity in $W$ is coNP-complete.

It is known that any property of frames expressible in first order logic with one binary relation is recognizable on a finite frame in polynomial time w.r.t. the size of the frame (e.g., [21], Theorem 5.1). By Lemma 10 we obtain the following result.

**Lemma 12** If a logic $L$ in $E$(Int) or in $NE$(K) is polynomially (or exponentially) approximable by a class of frames definable by finitely many first order formulas, then the $L$-refutability problem is in NP (resp. in NEXP) and the $L$-provability problem is in coNP (respectively, in coNEXP).

By Proposition 9(a) we conclude the following.

**Lemma 13** If a consistent logic $L$ in $E$(Int) or in $NE$(K) is polynomially approximable by a class of frames definable by finitely many first order formulas, then the $L$-refutability problem is NP-complete and the $L$-provability problem is coNP-complete.

The NP-completeness of refutability problem for many superintuitionistic and modal logics was proved in [2].

**Proposition 14** (a) For each consistent tabular logic $L$ in $E$(Int) or in $NE$(S4) the problem of equivalence to $L$ over Int (respectively, over S4) is DP-complete.

(b) The consistency problems over both Int and S4 are coNP-complete.
They are characterized, respectively, by the classes of frames $X$, $W$, $Int$ extensions of $z$.

It means that for every tabular $L \in E(\text{Int})$ there exist finitely many tabular logics $L_1, \ldots, L_k$ in $E(\text{Int})$ such that for every $L' \in E(\text{Int})$,

$$L' \supseteq L \text{ iff } L' \not\subseteq L_i \text{ for } i = 1, \ldots, k.$$ 

We see from Lemma 11 that the inclusion problem $\text{Int} + A \supseteq L$ is in $\text{NP}$. By the same Lemma, the problem $\text{Int} + A \subseteq L$ (which is the problem of $L$-validity of $A$) is $\text{coNP}$-complete, so the problem of equivalence to $L$ over $\text{Int}$ is in $\text{DP}$. By Proposition 9 this problem is $\text{DP}$-hard, so it is $\text{DP}$-complete. The proof for modal logics in $\text{NE}(\text{S4})$ is analogous. An analog of mentioned Kuznetsov’s result for extensions of $\text{S4}$ was proved in [19].

(b) It is known that $\text{Int} + A$ is consistent iff $A$ is a two-valued tautology. Also $\text{S4} + A$ is consistent iff $A$ is valid in the one-element reflexive frame.

Let us define the following sequences of frames for $n \geq 1$:

- $S_n$ is the set $\{1, \ldots, n\}$ with the natural ordering relation;
- $U_{n+1}$ is the set $\{0, 1, \ldots, n+1\}$, where $0RxR(n+1)$ for all $x$ and $\neg xRy$ for $1 \leq x, y \leq n, x \neq y$;
- $V_n$ is the subframe of $U_{n+1}$ obtained by deleting $(n+1)$;
- $X_n$ is $\{1, \ldots, n\}$, where $xRy$ for all $x, y$;
- $Y_{n+1}$ is the set $\{1, \ldots, n+1\}$, where $xRy$ for all $x, y \leq n$ and $xR(n+1)$ for all $x$.

In order to prove Theorem 1, we remind [10] that there are exactly three pretabular extensions of $\text{Int}$, namely, the logics

$$\text{LC} = \text{Int} + (p \to q) \lor (q \to p),$$

$$\text{LP}_2 = \text{Int} + (p \lor (p \to q \lor \neg q)),$$

$$\text{LQ}_3 = \text{Int} + ((r \lor (r \lor (p \lor (p \lor q \lor \neg q))) \lor \neg p \lor \neg \neg p).$$

The logic $\text{LC}$ is characterized by the finite linearly ordered intuitionistic frames, $\text{LP}_2$ by the finite frames satisfying the condition $\forall x \forall y \forall z(x \leq y \leq z \Rightarrow (x = y \lor y = z))$. At last, $\text{LQ}_3$ is characterized by the finite frames with the least and the greatest elements and with the chains of length not more than 3.

The family $\text{NE}(\text{S4})$ contains exactly five pretabular logics ([12]):

$$\text{PM}1 = \text{Grz.3},$$

$$\text{PM}2 = \text{Grz.2} + (\Box p \lor \Box(\Box p \to \Box q \lor \Box(\Box q \to \Box r \lor \Box \Diamond \neg r))),$$

$$\text{PM}3 = \text{S4} + (\Box(p \to \Box p) \lor \Box(\Diamond p \to p)),$$

$$\text{PM}4 = \text{S4.1} + (\Box q \lor \Box(\Box q \to \Box r \lor \Box \Diamond \neg r)),$$

$$\text{PM}5 = \text{S5}.$$

They are characterized, respectively, by the classes of frames $\{S_n | n > 0\}$, $\{U_n | n > 0\}$, $\{V_{n+1} | n > 0\}$, $\{Y_{n+1} | n > 0\}$, $\{X_n | n > 0\}$.

It means that provability in $L$ is equivalent to $L$-validity for all these logics, and unprovability is equivalent to $L$-refutability.
LEMMA 15 Each pretabular logic in $E(\text{Int})$ or $E(S4)$ is linearly approximable.

**Proof.** One can show that if a modal formula $A$ is refutable in a frame $S_n$ (respectively, $V_n, U_n, Y_n, X_n$), then it is refutable in $S_m$ (respectively, $V_m, U_m, Y_m, X_m$), where $m \leq |A| + 3$. Also if a modal-free formula $A$ is refutable in an intuitionistic structure based on $S_n, V_n$ or $U_n$ then it is refutable in, respectively, $S_m, V_m$ or $U_m$ for $m \leq |A| + 3$. \(\square\)

NP-completeness of the satisfiability problem was proved for $S5$ in [9] and for Grz.3 in [23]. We state

LEMMA 16 For each pretabular logic $L$ in $E(\text{Int})$ or in $NE(S4)$, refutability problem of $L$ is NP-complete, and provability problem of $L$ is coNP-complete.

**Proof.** By Lemmas 15 and 12 $L$-refutability problem is in NP for each logic under consideration. On the other hand, refutability problem is NP-hard for any pretabular logic by Proposition 9(a). \(\square\)

To find out the complexity of pretabularity problem, we need one more proposition.

PROPOSITION 17 Let $L_0$ be Int or S4 and $L$ any its pretabular extension. Then the inclusion problem $L_0 + A \supseteq L$ over $L_0$ is NP-complete, and the problem of equivalence to $L$ over $L_0$ is DP-complete.

**Proof.** For the inclusion problem we use the following well-known criteria [6, 13, 14] (refutability is considered in intuitionistic structures based on the above-mentioned frames):

- $\text{Int} + A \supseteq \text{LC}$ iff $A$ is refutable in both $V_2$ and $U_3$,
- $\text{Int} + A \supseteq \text{LP}_2$ iff $A$ is refutable in $S_3$,
- $\text{Int} + A \supseteq \text{LQ}_3$ iff $A$ is refutable in both $V_2$ and $S_4$.

Below we mean refutability in modal Kripke structures.

- $\text{S4} + A \supseteq \text{PM}1$ iff $A$ is refutable in all $V_2, U_3, X_2, Y_3$,
- $\text{S4} + A \supseteq \text{PM}2$ iff $A$ is refutable in all $V_2, S_4, X_2, Y_3$,
- $\text{S4} + A \supseteq \text{PM}3$ iff $A$ is refutable in all $S_3, X_2, Y_3$,
- $\text{S4} + A \supseteq \text{PM}4$ iff $A$ is refutable in all $S_3, V_2$ and $X_2$,
- $\text{S4} + A \supseteq \text{PM}5$ iff $A$ is refutable in $S_2$.

We see that for every pretabular logic $L$ we only should verify refutability of $A$ in finitely many fixed finite frames, so our inclusion problem for $L$ is in NP by Lemma 11. Further, $L_0 + A$ is equivalent to $L$ if and only if $A$ is valid in $L$ and $L_0 + A \supseteq L$. The problem of $L$-validity is coNP-complete by Lemma 16. It follows that the problem of equivalence to $L$ over $L_0$ is in DP, so it is DP-complete by Proposition 9(d). \(\square\)

Now we are in a position to prove Theorem 1:

**Proof** (of Theorem 1). (i) **The tabularity problems over both Int and S4 are NP-complete.** Kuznetsov [8] proved that a logic is tabular if and only if it is not contained in $L_1$ for any pretabular logic $L_1$. So a logic $\text{Int} + A$ is tabular if and only if $A$ is
refutable in each of \( LC, LP_2 \) and \( LQ_3 \). Thus the tabularity problem over \( Int \) is in \( NP \) by Lemma 16. On the other hand, this problem is \( NP \)-hard by Proposition 6(a) (one can take \( L_1 = Int \) and \( L_2 = For \)).

The family \( NE(S4) \) contains five pretabular logics, and one can prove the statement by analogy.

(ii) The pretabularity problems over both \( Int \) and \( S4 \) are both \( NP \)-hard and \( coNP \)-hard, and belong to \( \Delta^p_2 \). The pretabularity problem over \( Int \) is the problem of membership in the set \( \{ LC, LP_2, LQ_3 \} \) which is in \( \Delta^p_2 \) by Proposition 17. On the other hand, it is \( DP \)-hard by Proposition 8. The proof for modal logics is analogous.

(iii) The local tabularity problem over \( S4 \) is \( NP \)-complete. A logic in \( NE(S4) \) is locally tabular if and only if it is not contained in \( Grz_3 \). So local tabularity of \( S4 + A \) is equivalent to refutability of \( A \) in \( Grz_3 \) which is \( NP \)-complete by Lemma 16.

\[ \square \]

### 4 Interpolation and Amalgamation

In this section we prove Theorems 2 and 3. Theorem 2(i) is an immediate corollary of Propositions 18(i) and 19. Theorem 2(ii) is proved in Proposition 18(ii) and the rest of Theorem 2 in Proposition 24. Theorem 3 follows from Theorem 2 and Proposition 25.

We start with

**Proposition 18**  
(i) The interpolation problem over \( Int \) is \( PSPACE \)-complete;  
(ii) The problem of determining whether \( Int + A \) is a tabular logic with CIP is \( NP \)-complete.

**Proof.** In [13] the first author proved that in \( E(\text{Int}) \) there are exactly eight logics with Craig’s interpolation property CIP, which is equivalent to IPD in the extensions of \( Int \). The list of these logics consists of \( Int, KC, LC, LP_2, Cl, For, LC + LP_2 \) and \( LP_2 + (p \rightarrow q) \lor (q \rightarrow p) \lor (p \leftrightarrow \neg q) \). Note that all these logics have the Hallden Property.

The logics \( Cl, LC + LP_2, LP_2 + (p \rightarrow q) \lor (q \rightarrow p) \lor (p \leftrightarrow \neg q) \) are characterized, respectively, by the frames \( S_1, S_2 \) and \( V_2; LC \) by \( \{ S_n | n > 0 \} \), \( LP_2 \) by \( \{ V_n | n > 0 \} \) and \( KC \) by finite partially ordered frames having the greatest element. In addition, we have the following criterion for inclusion ([13]):

\[
\begin{align*}
\text{Int} + A \supseteq For & \quad \text{iff } A \text{ is refutable in } S_1, \\
\text{Int} + A \supseteq Cl & \quad \text{iff } A \text{ is refutable in } S_2, \\
\text{Int} + A \supseteq (LC + LP_2) & \quad \text{iff } A \text{ is refutable in both } V_2 \text{ and } S_3, \\
\text{Int} + A \supseteq LP_2 + (p \rightarrow q) \lor (q \rightarrow p) \lor (p \leftrightarrow \neg q) & \quad \text{iff } A \text{ is refutable in both } V_3 \text{ and } S_3, \\
\text{Int} + A \supseteq LC & \quad \text{iff } A \text{ is refutable in both } V_2 \text{ and } U_3, \\
\text{Int} + A \supseteq LP_2 & \quad \text{iff } A \text{ is refutable in } S_3, \\
\text{Int} + A \supseteq KC & \quad \text{iff } A \text{ is refutable in } V_2.
\end{align*}
\]

So for all proper extensions of \( Int \) with CIP, the inclusion problem \( \text{Int} + A \supseteq L \) is equivalent to refutability of \( A \) in one or two finite frames which is in \( NP \) by Lemma 11.
The provability problem is \textsc{coNP}-complete for tabular logics Cl, LC + LP_2, LP_2 + (p → q) ∨ (q → p) ∨ (p ↔ ¬q), and also for pretabular logics LC and LP_2. It is clear that the equivalence problem is in \textsc{DP} for each of these logics.

Further, since the validity problem in \textsc{Int} is \textsc{PSPACE}-complete [24], so is the problem of equivalence to \textsc{Int} over \textsc{Int}. To find the complexity bounds for validity in \textsc{KC}, we note that

\[ \text{KC} ⊨ A \text{ iff } \text{Int} ⊨ (¬P_1 ∨ ¬P_1) & \ldots \& (¬P_n ∨ ¬P_n) → A, \]

where \( P_1, \ldots, P_n \) are all variables of \( A \), so validity in \textsc{KC} is in \textsc{PSPACE}. On the other hand, \textsc{KC} contains the implicational fragment of \textsc{Int} [25] which is \textsc{PSPACE}-hard [24], so validity in \textsc{KC} is \textsc{PSPACE}-complete. It follows that equivalence to \textsc{KC} is in \textsc{PSPACE}, and it is \textsc{PSPACE}-complete by Proposition 5. Thus equivalence to \( L \) is in \textsc{PSPACE} for each logic \( L \) with CIP in \( E(\text{Int}) \), so the interpolation problem over \textsc{Int} is in \textsc{PSPACE}. To use Proposition 5, we take, for instance, \( L_1 = \text{Int} \) and \( L_2 = \text{KC} \cap LP_2 \). Therefore the interpolation problem over \textsc{Int} is \textsc{PSPACE}-complete.

To prove (ii), we note that \textsc{Int} + \( A \) is a tabular logic with CIP if and only if \textsc{Int} + \( A \) \( \supseteq \) \textsc{Int} + (\( p → q \)) ∨ (\( q → p \)) ∨ (\( p ↔ ¬q \)), and we already know that this inclusion problem is \textsc{NP}-complete.

\[ \square \]

Note. \textsc{PSPACE}-completeness of the provability problem in \textsc{KC} was proved in [2]. Tableau calculi and cut-free sequent calculi for superintuitionistic logics with CIP are constructed in [1].

Boolos [4] proved Craig’s Interpolation Property CIP for the Grzegorczyk logic \textsc{Grz}. The complete list of normal extensions of \textsc{Grz} with the interpolation property IPD contains seven logics (see [18]), namely, \textsc{Grz}, \textsc{Grz}2, \( PM_3 = \text{Grz} + (\square(p → \square p) \lor \square(\Diamond p → p)) \), \( Gr_4 = PM_3 + (☐□p & ☐q & ⊢ r → ☐□(p & q) \lor ☐□(p & r) \lor ☐□(q & r)) \), \( Gr_5 = \text{Grz}2 + Gr_4 \), \( Gr_6 = \text{Grz} + (p → \square p) \), \( Gr_7 = Fm \). All of them have CIP, so IPD and CIP are equivalent in \textsc{NE} (\textsc{Grz}).

It is known [2] that provability problem for \textsc{Grz} is \textsc{PSPACE}-complete. Using this result, we assert

\[ \text{PROPOSITION 19} \quad \text{The interpolation problem over } \textsc{Grz} \text{ is } \textsc{PSPACE}-\text{complete}. \]

\[ \text{PROOF.} \quad \text{We note that the logics } Gr_4 - Gr_7 \text{ are tabular and } PM_3 \text{ is pretabular, so for each of these logics the problem of equivalence is in } \textsc{DP} \text{ by Propositions 14 and 17.} \]

Further, we note that

\[ \text{Grz}2 ⊨ B \text{ iff } \text{Grz} ⊨ A(p_1) & \ldots & A(p_n) → B, \]

where \( A(p) = \square ◻ p → ◻ \square p \) and \( p_1, \ldots, p_n \) are all variables of \( B \), so the provability problem of \textsc{Grz}2 is linearly reducible to that of \textsc{Grz}, and is in \textsc{PSPACE}.

In addition, we have ([14]):

\[ S_4 + B \supseteq \text{Grz}2 \text{ iff } B \text{ is refutable in all } X_2, Y_3 \text{ and } V_2, \]

so the inclusion problem \( S_4 + B \supseteq \text{Grz}2 \) is in \textsc{NP}, and the problem of equivalence to \textsc{Grz}2 is in \textsc{PSPACE}.
Taking into account that the problem of equivalence to Grz over Grz is PSPACE-complete, we conclude that the interpolation problem over Grz is in PSPACE. On the other hand, the problem of provability in Grz is reducible to the interpolation problem by Proposition 5 (it is sufficient to take $L_1 = \text{Grz}$ and $L_2 = \text{Grz.2} \cap \text{PM3}$).

\textbf{Note.} One can reduce the provability problem of KC to the provability problem of Grz.2, so the latter problem as well as the problem of equivalence to Grz.2 is PSPACE-complete.

Let us now consider the interpolation properties CIP and IPD over S4. It is known that CIP implies IPD in extensions of S4. The first author proved in \cite{14, 17} that there exist not more than forty nine logics with IPD in NE(S4), among them at most thirty seven have CIP, and at least twelve logics have IPD and do not possess CIP. At the present time, it is known that at least thirty one logics have CIP and at least forty three logics have IPD (\cite{16, 18}). Although the interpolation problem in NE(S4) is not yet completely solved, nevertheless, it is decidable since for each logic $L$ in the mentioned list of forty nine logics, the problem of equivalence to $L$ is decidable over S4 \cite{14}.

The list $L$ of the forty nine logics containing all logics with IPD in NE(S4) consists of thirty one locally tabular logics (for which the interpolation problem is completely solved) and of eighteen logics which are not locally tabular. All logics in $L$ are finitely axiomatizable, and have FMP and Hallden property. Each logic in $L$ was described in detail in \cite{14}. Using this description, we can state two lemmas:

\textbf{Lemma 20} Let $L$ be a logic with IPD in NE(S4), $L \neq S4$. Then the inclusion problem $S4 + A \supseteq L$ is NP-complete.

\textbf{Lemma 21} Let $L$ be a consistent and locally tabular logic with IPD in NE(S4). Then $L$ is polynomially approximable, the $L$-provability problem is coNP-complete, and the problem of equivalence to $L$ over S4 is DP-complete.

In order to find complexity bounds for logics in $L$ which are not locally tabular, we recall some definitions.

In \cite{12} the first author introduced two characteristics $\mu_1$ and $\mu_2$ of frames and logics in NE(S4) as follows. With any element $a$ of a frame $W$ we associate its cluster, i.e., the set $C(a) = \{x \in W | aRxRa\}$. A cluster of $a$ is internal if there is $y$ such that $aRy$ and $\neg yRa$, and external otherwise. We define $\mu_1(W)$ and $\mu_2(W)$ as the suprema of cardinalities of external and of internal clusters in $W$ respectively. If $L$ is a calculus, we define $\mu_i(L) = \sup \{\mu_i(W) | W$ is a finite $L$-frame$\}$.

It was proved in \cite{12} that $\mu_1(L) = \omega$ iff $L \subseteq S5$, and $\mu_2(L) = \omega$ iff $L \subseteq PM4$, where $PM4$ is one of five pretabular logics in NE(S4).

Also we need two sequences of formulas $\xi_n$ and $\eta_{n+1}$, defined as follows:

- $\xi_n = \varphi_n \rightarrow \neg p_1$, where $\varphi_n$ is the conjunction of formulas
- $\Box(p_i \rightarrow \neg p_j)$ for all $i \neq j$;
- $\Box(p_i \rightarrow \Box p_j)$ for all $i, j \leq n$;
- $\Box(p_1 \lor \ldots \lor p_n)$;
\( \eta_{n+1} = \psi_n \rightarrow \neg p_1 \), where \( \psi_n \) is the conjunction of formulas
\( \Box (p_i \rightarrow \neg p_j) \) for all \( i \neq j \);
\( \Box (p_i \rightarrow \Diamond p_j) \) for all \( i \leq n \) and \( j \leq n + 1 \);
\( \Box (p_{n+1} \rightarrow \neg \Diamond p_j) \) for all \( j \leq n \);
\( \Box (p_1 \lor \ldots \lor p_{n+1}) \).

The following Lemma was proved in [15].

**Lemma 22** Let \( L = S4 + A \). Then for all \( n > 0 \)
(a) \( L \vdash \xi_{n+1} \) iff \( \mu_1(L) \leq n \);
(b) \( L \vdash \eta_{n+2} \) iff \( \mu_2(L) \leq n \).

Now we turn to logics in the list \( \mathcal{L} \) which are not locally tabular.

**Lemma 23** Let \( L \) be a non-locally-tabular logic with IPD in \( NE(S4) \). Then both problems of \( L \)-provability and of equivalence to \( L \) over \( S4 \) are in \( \text{coNEXP} \) and \( \text{PSPACE-hard} \).

**Proof.** Our list \( \mathcal{L} \) contains eighteen logics which are not locally tabular. They are of the form \( L, L + \xi_{i+1}, L + \eta_{j+2}, L + \xi_i + 1 + \eta_j + 2 \) for \( L \in \{S4, S4.2\} \) and \( 1 \leq i, j \leq 2 \). It was proved in [15, 14] that all logics \( S4 + \xi_{i+1}, S4 + \eta_{j+2}, S4 + \xi_i + 1 + \eta_j + 2 \) have the finite model property. They are characterized, respectively, by all finite frames \( W \) with \( \mu_1(W) \leq i \) and/or \( \mu_2(W) \leq j \). The classes of finite frames for \( S4.2, S4.2 + \xi_{i+1}, S4.2 + \eta_{j+2}, S4.2 + \xi_i + 1 + \eta_j + 2 \) should, in addition, satisfy the condition:
\( (xRy \text{ and } xRz) \Rightarrow \exists u(yRu \text{ and } zRu) \).

Moreover, if a formula \( A \) is unprovable in some logic \( L \) of this list, then there exists a finite frame of a size not more than \( 2^{14|A|} \) which validates \( L \) and refutes \( A \) [15, 14]. It follows by Lemma 12 that the refutability problem of \( L \) is in \( NEXP \), and \( L \)-provability problem is in \( \text{coNEXP} \). The inclusion problem \( S4 + A \supseteq L \) is in \( NP \) for all these logics by Lemma 20, so the problem of equivalence to \( L \) over \( S4 \) is in \( \text{coNEXP} \).

On the other hand, there exists the well-known Gödel-Tarski translation \( T \) from \( \text{Int} \) to \( S4 \) defined as follows:
\[
\begin{align*}
T(p) &= \Box p \text{ for any variable } p; \\
T(A \& B) &= T(A) \& T(B); \\
T(A \lor B) &= T(A) \lor T(B); \\
T(A \rightarrow B) &= \Box (T(A) \rightarrow T(B)); \\
T(\neg A) &= \Box \neg T(A).
\end{align*}
\]

This function \( T \) also reduces \( \text{Int} \) to the logics \( S4 + \xi_{i+1}, S4 + \eta_{j+2}, S4 + \xi_i + 1 + \eta_j + 2 \), and \( \text{KC} \) to each of logics \( S4.2, S4.2 + \xi_{i+1}, S4.2 + \eta_{j+2}, S4.2 + \xi_i + 1 + \eta_j + 2 \). It follows that \( L \)-provability problem is \( \text{PSPACE-hard} \) for all logics of this lemma. \( \Box \)

**Proposition 24** (i) Both CIP and IPD problems over \( S4 \) are in \( \text{coNEXP} \) and \( \text{PSPACE-hard} \).
(ii) The problems of determining whether \( S4 + A \) is a tabular (pretabular or locally tabular) logic with CIP (or IPD) are in \( \Delta^p_2 \) and \( \text{DP-hard} \).
Proof. It was proved in [16, 18] that the family of logics with IPD in $NE(S4)$ contains the thirty one locally tabular logics (of which one is inconsistent and thirty are consistent) and also the logics $L, L + \eta_2, L + \eta_3, L + \xi_{i+1} + \eta_3, \xi_{i+1} + \eta_3, \eta_1$ where $L \in \{S4, S4.2\}$ and $1 \leq i \leq 2$, which are not locally tabular. These non-locally-tabular logics and, in addition, nineteen locally tabular logics in $NE(S4)$ have CIP. We do not know whether the six logics $L + \eta_4, L + \xi_2 + \eta_4, L + \xi_3 + \eta_4$ have IPD (or CIP) or not.

The problem of equivalence over $S4$ to the inconsistent logic $FM$ is the same as the inconsistency problem, so it is NP-complete by Proposition 14(b). For each consistent and locally tabular logic $L$ with IPD the problem of equivalence to $L$ over $S4$ is $\text{PSPACE}$-hard by Proposition 5. Thus, we proved (i).

Pretabular logics with IPD are $PM2 - PM5$, each of them has CIP and is locally tabular. The problem of membership in the set $\{PM2, \ldots, PM5\}$ is in $\Delta^p_2$ and $\text{NP}$-hard by Propositions 17 and 8.

The problem of determining whether $S4 + A$ is a tabular (or locally tabular) logic with IPD (or CIP) is in $\Delta^p_2$ by Lemma 21. On the other hand, this problem is $\text{NP}$-hard by Proposition 8: it is sufficient to note that the logic $L$ determined by the three-element frame $\{0, 1, 2\}$, where $0R1R2R1$, is tabular and has CIP, $L' = S4$ is not locally tabular, and a tabular logic $L''$ determined by two two-element frames $S_2$ and $X_2$ (see Section 4) includes $L$ and does not possess IPD.

It is well-known that there is a duality between $E(\text{Int})$ and the family of varieties of Heyting algebras, and also between $NE(S4)$ and the family of varieties of closure algebras. If $Ax$ is a set of formulas and $L = \text{Int} + Ax$ then $\{A = \top|A \in Ax\}$ forms a base of identities for the variety $V(L)$ associated with $L$. The definitions for the logics $L \in NE(S4)$ are analogous. On the other hand, with any identity $A = B$ one can associate a formula $A \leftrightarrow B$ which is valid in some algebra if and only if the identity $A = B$ holds in this algebra.

We remind that a class $\mathcal{X}$ is amalgalamable if it satisfies the following condition for all algebras $A, B, C$ in $\mathcal{X}$.

$$ (\text{AP}) \quad \text{For all monomorphisms } \alpha : A \to B, \beta : A \to C \text{ there exist an algebra } D \text{ in } \mathcal{X} \text{ and monomorphisms } \gamma : B \to D, \delta : C \to D \text{ such that } \gamma\alpha = \delta\beta. $$

We say that $\mathcal{X}$ is super-amalgalamable, if for any algebras $A, B, C$ in $\mathcal{X}$, the condition AP is satisfied and, in addition, for each $x \in B, y \in C$ the following equivalences hold:

$$ \gamma(x) \leq \delta(y) \iff (\exists z \in A)(x \leq \alpha(z) \& \beta(z) \leq y) \text{ and } \gamma(x) \geq \delta(y) \iff (\exists z \in A)(x \geq \alpha(z) \& \beta(z) \geq y). $$

We will describe the complexity of the amalgamability problem for a variety with respect to the sum of the lengths of its base of identities.

The following Proposition is an immediate corollary of Theorem 1 in [13] and of Theorems 1 and 2 in [14].
Proposition 25 Let $L$ be in $E(\text{Int})$ or in $NE(S4)$. Then
(i) $L$ has CIP iff $V(L)$ is super-amalgamable;
(ii) $L$ has IPD iff $V(L)$ is amalgamable.

Using Theorem 2 and Proposition 25, we immediately obtain

Theorem 3 (i) The amalgamability problem for finitely based varieties of Heyting algebras is $\text{PSPACE}$-complete.
(ii) The amalgamability and super-amalgamability problems for finitely based varieties of closure algebras are in $\text{coNEXP}$ and $\text{PSPACE}$-hard.

References

Goal-Directed Calculi for Gödel-Dummett Logics

George Metcalfe\(^1\), Nicola Olivetti\(^2\), and Dov Gabbay\(^1\)

\(^1\) Department of Computer Science, King’s College London, Strand, London WC2R 2LS, UK
{metcalfe,dg}@dcs.kcl.ac.uk
\(^2\) Department of Computer Science, University of Turin, Corso Svizzera 185, 10149 Turin, Italy
olivetti@di.unito.it

Abstract. In this work we present goal-directed calculi for the Gödel-Dummett logic \(LC\) and its finite-valued counterparts, \(LC_n\) \((n \geq 2)\). We introduce a terminating hypersequent calculus for the implicational fragment of \(LC\) with local rules and a single identity axiom. We also give a labelled goal-directed calculus with invertible rules and show that it is co-NP. Finally we derive labelled goal-directed calculi for \(LC_n\).

Keywords: Gödel Logics, Intermediate Logics, Fuzzy Logics, Hypersequents, Goal-Directed Calculi.

1 Introduction

Matrices for the finite-valued Gödel logics \(LC_n\) \((n = 2, 3, \ldots)\) were introduced by Gödel in [14]; an infinite-valued version was presented and axiomatised by Dummett in [7] and is known as the Gödel-Dummett logic \(LC\).

\(LC\) is important as a logic intermediate between intuitionistic logic \(LJ\) and classical logic \(LK\). It has been used in investigations of relevance logics [8] and the provability logic of Heyting algebras [22], and has attracted renewed interest recently as one of the fundamental t-norm based fuzzy logics [16]. Moreover, the three-valued Gödel logic \(LC_3\) is of interest as the logic of “here and there” [18] used to check the strong equivalence of logic programs.

A proof calculus for \(LC\) extending Gentzen’s calculus for \(LJ\) has been provided by Avron [3] using hypersequents, a generalisation of Gentzen-style sequents. Although very natural, this calculus is not particularly suitable for proof search since its rules are both non-invertible and non-terminating. The first single sequent calculus for \(LC\) was given by Sonobe in [21]; terminating and contraction-free versions with invertible rules were subsequently developed by Avellone et al. [2], Dykhoff [9] and Fiorino [10]. These calculi all possess rules with an arbitrary number of premises however. Responding to this problem, Avron and Koniskowa [4] and Larchey-Wendling [17] have presented systems where all rules are local. Indeed Avron and Koniskowa have introduced three new systems; two single sequent calculi and a hypersequent calculus, the latter turning out to be identical under translation to the sequents of relations calculus given by Baaz and Fermüller in [5]. Systems for the finite-valued Gödel logics can be obtained...
from these calculi by adding extra axioms. More general approaches to auto-
mated deduction in finite-valued logics are described in [15,16]. We also men-
tion here the calculus of Aguzzoli and Gerla [1] which uses the fact that a formula is
valid in $\text{LC}$ if it is valid in $\text{LC}_n$ where $n$ is a function of the size of the formula.

In this work we develop goal-directed calculi for the implicational fragments
of $\text{LC}$ and $\text{LC}_n$ following the methodology of Gabbay and Olivetti [12]. The goal-
directed approach is a generalisation of the logic programming style of deduction
particularly suitable for proof-search. It has been applied to a wide range of logics
including classical and intuitionistic logics, intermediate logics, modal logics and
substructural logics [12,20]. Goal-directed methods have also been developed for
$\text{LC}_n$ [12] but the systems obtained are not particularly suitable for proof search.

We proceed as follows. In Sections 2 and 3 we introduce Gödel-Dummett
logics and goal-directed methods respectively. Then in Sections 4 and 5 we give
goal-directed hypersequent and labelled goal-directed calculi for $\text{LC}_n$. Finally in
Section 6 we present goal-directed labelled calculi for $\text{LC}_n$.

2 Gödel-Dummett Logics

The language is built inductively in the usual way from propositional variables
$p_1, p_2, \ldots$, binary connectives $\rightarrow, \land$ and $\lor$, and the constants $\bot$ and $t$.

**Definition 1 (Gödel-Dummett Logics).** $\text{LC}_n = \langle \{0, 1/(n - 1), \ldots, (n - 2)/(n - 1), 1\}, \rightarrow, \land, \lor, \bot, t \rangle$ and $\text{LC} = \langle \{0, 1\}, \rightarrow, \land, \lor, \bot, t \rangle$ where $A \land B = min(A, B)$, $A \lor B = max(A, B)$, $\bot = 0$, $t = 1$ and $A \rightarrow B = 1$ if $A \leq B$, $B$ otherwise. We write $\models A \ (\models_n A)$ iff $1 \leq v(A)$ for every valuation $v$ in $\text{LC}$ ($\text{LC}_n$).

$\text{LC}$ is characterised by linearly ordered Kripke models and $\text{LC}_n$ by linearly
ordered Kripke models with $n - 1$ elements (see eg [11] for details).

**Definition 2 (Axiomatisations for $\text{LC}^\rightarrow$ and $\text{LC}_n^\rightarrow$).** The axiomatisation
of $\text{LC}^\rightarrow$ consists of modus ponens and the following axioms [7], adopting the
convention of writing $A_1 \rightarrow \ldots \rightarrow A_n \rightarrow B$ for $A_1 \rightarrow (A_2 \rightarrow (\ldots (A_n \rightarrow B) \ldots)$:

1. $A \rightarrow B \rightarrow A$
2. $(A \rightarrow B \rightarrow C) \rightarrow (A \rightarrow B) \rightarrow A \rightarrow C$
3. $((A \rightarrow B) \rightarrow C) \rightarrow ((B \rightarrow A) \rightarrow C) \rightarrow C$

$\text{LC}_n^\rightarrow$ is obtained for $n \geq 2$ by adding the axiom:

$(A_1 \rightarrow B) \rightarrow ((A_1 \rightarrow A_2) \rightarrow B) \rightarrow (\ldots \rightarrow ((A_{n-1} \rightarrow A_n) \rightarrow B) \rightarrow B$

3 Goal-Directed Methods

In this paper we adopt the following “logic programming style” goal-directed
paradigm of deduction. For a given logic let us denote by $\Gamma \vdash ? A$ the query

\footnote{Goal-directed calculi for the full propositional language have been developed but are
not presented here due to lack of space.}
“does A follow from Γ?” where Γ is a database (collection) of formulae and A is a goal formula. The deduction is goal-directed in the sense that the next step in a proof is determined by the form of the current goal: a complex goal is decomposed until its atomic constituents are reached, an atomic goal q is matched (if possible) with the “head” of a formula G → q in the database, and its “body” G asked in turn. This can be viewed as a sort of resolution step, or generalised Modus Tollens.

This model of deduction can be refined in several ways: (1) by putting constraints/labels on database formulae, restricting those available to match an atomic goal, (2) by adding more control to ensure termination, either by loop-checking or by “diminishing resources” ie removing formulae “used” to match an atomic goal, (3) by re-asking goals previously occurring in the deduction using restart rules. Note however that for applications such as deductive databases and logic programming, a terminating proof procedure is not always essential. We might want to get a proof of the goal from the database quickly if one exists, but be willing to have no answer otherwise (and preempt termination externally).

Goal-directed procedures have been proposed for a variety of logics, in some cases as refinements of sequent calculi called “Uniform proof systems” [20]. Here we illustrate the goal-directed methodology by presenting the diminishing resources with bounded restart algorithm given in [12] for the implicational fragment of intuitionistic logic.

Definition 3 (GDLJ→). Queries have the form Γ ⊢ ? G, H where Γ is a multiset of formulae and H (called the history) is a sequence of atomic goals. The rules are as follows (where H ∗ (q) is q appended to H):

(success) Γ ⊢ ? q, H succeeds if q ∈ Γ


(reduction) From Γ, A₁ → ⋯ → Aₙ → q ⊢ ? q, H step to
Γ ⊢ ? Aᵢ, H ∗ (q) for i = 1, ⋯ , n

(bounded restart) From Γ ⊢ ? q, H step to
Γ ⊢ ? q₁, H ∗ (q) if q₁ follows q in the sequence H

Example 1. Consider the following proof, observing that (bounded restart) is needed at (2) to compensate for the removal of p → q at (1):

\[\begin{align*}
\Gamma & \vdash \top \,(p \rightarrow q) \rightarrow p \rightarrow q, \emptyset \\
(p \rightarrow q) & \rightarrow p \vdash \top (p \rightarrow q) \rightarrow q, \emptyset \\
(p \rightarrow q) & \rightarrow p \rightarrow q \vdash \top q, \emptyset \quad (1) \\
(p \rightarrow q) & \rightarrow p \vdash \top p, (q) \\
\vdash p \rightarrow q, (q,p) \\
p & \vdash \top q, (q,p) \quad (2) \\
p & \vdash \top p, (q,p,q) \text{ by bounded restart} \\
success
\end{align*}\]

2 In fact the calculus is an O(n log n)-space decision procedure for this fragment [13].
4 A Hypersequent Calculus for LC\(\rightarrow\)

We begin by generalising the goal-directed approach used for intuitionistic logic in Section 3 analogously to Avron’s generalisation of Gentzen sequents, here defining a hypersequent as a set of sequents with restarts.

**Definition 4 (Hypersequent).** A component is a structure \(S = \Gamma \vdash B, R\) where \(\Gamma\) is a set of formulae, \(B\) is a formula and \(R = ()\) or \(R = (q)\) for some atom \(q\). A hypersequent is a set of components written \(S_1|\ldots|S_n\).

We interpret a hypersequent as a disjunction, each restart being interpreted as an extra formula on the left hand side of the appropriate component.

**Definition 5 (Interpretation of Hypersequents for LC).** The interpretation of \(S = A_1,\ldots, A_n \vdash B, R\) is \(\phi^S = A_1 \rightarrow \ldots \rightarrow A_n \rightarrow (B \rightarrow \phi^R) \rightarrow B\) where \(\phi^l = t\) and \(\phi^{(C)} = C\). The interpretation of a hypersequent \(G = S_1|\ldots|S_n\) is \(\phi^{S_1} \lor \ldots \lor \phi^{S_n}\) and we write \(\models G\) iff \(\models \phi^G\).

We present the following hypersequent calculus\(^3\) for LC\(\rightarrow\).

**Definition 6 (GDLC\(_6\)).** GDLC\(_h\) has the following rules:

- **(success)** \(G|\Gamma, q \vdash q, R\)
- **(implication)** \(G|\Gamma, A \vdash B, R\)
  \[G|\Gamma \vdash A \rightarrow B, R\]
  for \(i = 1 \ldots n\) \((n > 0)\)
- **(reduction1)** \(G|\Gamma, q \vdash R| \vdash A_i, (q)\)
  \[G|\Gamma, A_1 \rightarrow \ldots \rightarrow A_n \rightarrow q \vdash q, R\]
  for \(i = 1 \ldots n\) \((n > 0)\)
- **(reduction2)** \(G|\Gamma_1, q \vdash p, R_1| \vdash q, R_2\)
  \[G|\Gamma_1, A_1 \rightarrow \ldots \rightarrow A_n \rightarrow q \vdash p, R_1| \vdash q, R_2\]
- **(mingle)** \(G|\Gamma_1 \vdash q, R_1| \vdash R_2\)
  \[G|\Gamma_1, q \vdash q, R_1| \vdash q, R_2\]
- **(restart)** \(G|\Gamma_1 \vdash p, R\)
  \[G|\Gamma_1, q \vdash q, R| \vdash p, (q)\]

**Example 2.** The prelinearity axiom for LC\(\rightarrow\) is proved as follows.

\[\vdash r, ()| \vdash p \vdash p, (r)\]
\[\vdash r, ()| \vdash p \vdash q, (r)| \vdash q \vdash p, (r)\]  
(mingle)
\[\vdash r, ()| \vdash p \rightarrow q, (r)| \vdash q \vdash p, (r)\]  
(implication)
\[(p \rightarrow q) \rightarrow r| \vdash r, ()| \vdash q \vdash p, (r)\]  
(reduction1)
\[(p \rightarrow q) \rightarrow r| \vdash r, (q \rightarrow p) \rightarrow r| \vdash r, ()\]  
(implication)
\[(p \rightarrow q) \rightarrow r| \vdash ((q \rightarrow p) \rightarrow r)| \vdash r, ()\]  
(implication)
\[\vdash ((p \rightarrow q) \rightarrow r)| \vdash ((q \rightarrow p) \rightarrow r)| \vdash r, ()\]

\(^3\) It is easy to transform this calculus into a more goal-directed style of presentation as in Section 3; here the intention is rather to facilitate comparison with other hypersequent calculi developed for LC.
Read upwards $\text{GDLC}_h^\rightarrow$ is goal-directed for each component. Both reduction rules look for a formula on the left hand side of a component $S_1$, with a head that matches an atomic formula on the right hand side of a component $S_2$. (reduction1) deals with the case where $S_1 = S_2$ and (reduction2) with the case where $S_1 \neq S_2$. The (mingle) and (restart) rules are also goal-directed, combining the left hand sides of two components into a single new component. Only the choice of components is not goal-directed.

**Theorem 1.** If a hypersequent $G$ is derivable in $\text{GDLC}_h^\rightarrow$ then $\models G$.

Termination of $\text{GDLC}_h^\rightarrow$ is guaranteed since the set of subformulae occurring in each premise of the rules is strictly smaller than the set of subformulae occurring in the conclusion.

**Theorem 2.** $\text{GDLC}_h^\rightarrow$ is terminating.

To prove completeness we distinguish two functions of $\text{GDLC}_h^\rightarrow$: (1) to decompose complex formulae using invertible rules, and (2) to determine the validity of a hypersequent. We show that (1) allows us to reduce the provability of a hypersequent to the provability of hypersequents containing only atoms and irrelevant formulae (in the sense that they do not affect the validity of the hypersequent), and that (2) allows us to prove all valid atomic hypersequents (and thus also valid atomic hypersequents with irrelevant formulae).

**Definition 7 (Invertible).** A rule is invertible if whenever the conclusion of the rule is valid then all its premises are valid.

**Proposition 1.** (implication), (reduction1) and (reduction2) are invertible.

Applying the (implication), (reduction1) and (reduction2) rules exhaustively to a hypersequent results in hypersequents where for each component the right hand side is an atom that fails to match the head of a formula on the left hand side of any component.

**Definition 8 (Irreducible).** Let $\text{Head}(\Gamma) = \{q\mid A_1 \rightarrow \ldots \rightarrow A_n \rightarrow q \in \Gamma\}$. A hypersequent $\Gamma_1 \vdash A_1, R_1 \ldots | \Gamma_n \vdash A_n, R_n$ is irreducible if $A_i$ is atomic for $i = 1 \ldots n$ and there exist $\Pi_1, \ldots, \Pi_n$ and $\Delta_1, \ldots, \Delta_n$ such that $\Gamma_i = \Delta_i \cup \Pi_i$ and $\Delta_i$ is atomic for $i = 1 \ldots n$ and $\text{Head}(\Pi_1 \cup \ldots \cup \Pi_n) \cap \{A_1, \ldots, A_n\} = \emptyset$.

**Proposition 2.** For a valid hypersequent $G$ we can apply the rules of $\text{GDLC}_h^\rightarrow$ to obtain a multiset of valid irreducible hypersequents.

If an irreducible hypersequent is valid then by removing non-atomic formulae we obtain an atomic hypersequent that is also valid.

**Proposition 3.** If $\models \Pi_1, \Delta_1 \vdash q_1, R_1 \ldots | \Pi_n, \Delta_n \vdash q_n, R_n$ where $\Delta_i$ is atomic for $i = 1 \ldots n$ and $\text{Head}(\Pi_1 \cup \ldots \cup \Pi_n) \cap \{q_1, \ldots, q_n\} = \emptyset$ then $\models \Delta_1 \vdash q_1, R_1 \ldots | \Delta_n \vdash q_n, R_n$.

We now turn our attention to proving that all valid atomic hypersequents are derivable in $\text{GDLC}_h^\rightarrow$. We introduce for this purpose the notion of a (strict) G-cycle for a hypersequent $G$ and show that if a G-cycle exists then $G$ is provable.
in \( \text{GDLC}_h^\gamma \). We then make use of a proposition by Avron and Koniskowa \cite{4} which states that every valid atomic hypersequent \( G \) contains a \( G \)-cycle.

**Definition 9 (\( \leq_G, <_G, \triangleleft_G \)).** For a hypersequent \( G \) and atomic \( p,q \) we define \( \leq_G, <_G \) and \( \triangleleft_G \) as follows: (1) \( p <_G q \) iff \( \Gamma,q \vdash p, R \) is a component of \( G \), (2) \( p \leq_G q \) iff \( \Gamma \vdash p, (q) \) is a component of \( G \), (3) \( p \triangleleft_G q \) iff \( p <_G q \) or \( p \leq_G q \).

**Definition 10 ((Strict) \( G \)-cycle).** A \( G \)-cycle is a sequence of atoms \( p_1, \ldots, p_n \) such that \( p_i <_G p_{i+1} \) for \( i = 1 \ldots n-1 \), \( p_i <_G p_{i+1} \) for some \( 1 \leq i \leq n-1 \) and \( p_1 = p_n \). A strict \( G \)-cycle is a \( G \)-cycle where \( p_i \neq p_j \) for \( 1 \leq i < j \leq n-1 \).

**Proposition 4 (Avron and Koniskowa \cite{4}).** An atomic hypersequent \( G \) is valid iff there exists a \( G \)-cycle.

**Proposition 5.** If there is a \( G \)-cycle then there is a strict \( G \)-cycle.

**Proof.** Let \( C \) be a \( G \)-cycle \( p_1, \ldots, p_n \). We proceed by induction on \( r_C = |\{p_i|p_i = p_j, 1 \leq i < j \leq n-1\}| \). If \( r_C = 0 \) then \( C \) is strict and we are done. For \( r_C > 0, p_i = p_j \) for some \( i < j \); if \( C' = p_i, \ldots, p_j \) is a \( G \)-cycle (ie if there exists \( i \leq k \leq j - 1 \) such that \( p_k <_G p_{k+1} \)) then \( r_{C'} < r_C \) and we are done, otherwise \( C' = p_1, \ldots, p_i, p_j, \ldots, p_n \) must be a \( G \)-cycle where \( r_{C'} < r_C \). □

**Proposition 6.** For a hypersequent \( G \) if there exists a strict \( G \)-cycle then \( G \) is provable in \( \text{GDLC}_h^\gamma \).

**Proof.** By induction on the length \( l \) of a strict \( G \)-cycle. If \( l = 2 \) then we have \( p <_G p \) which means that \( \Gamma, p \vdash p, R \) is a component of \( G \) so \( G \) is derivable by \( (success) \). For a strict \( G \)-cycle \( C = p_1, \ldots, p_l \) where \( l > 2 \) we have a subsequence \( q_1, q_2, q_3 \) which must have one of the following patterns: (1) \( q_1 <_G q_2 <_G q_3 \), (2) \( q_1 \leq_G q_2 <_G q_3 \), (3) \( q_1 <_G q_2 \leq_G q_3 \).

For (1) we have that \( S_1 = \Gamma_1, q_2 \vdash q_1, R_1 \) and \( S_2 = \Gamma_2, q_3 \vdash q_2, R_2 \) are components of \( G \). If they are the same component then \( q_1 = q_2 \) contradicts the strictness of \( C \). By applying \( (mingle) \) to \( G \) we obtain \( G' \) where \( S_1 \) and \( S_2 \) have been replaced by \( S_3 = \Gamma_1, \Gamma_2, q_3 \vdash q_1, R_1 \). We claim that by removing \( q_2 \) from \( C \) we obtain a \( G' \)-cycle of length \( l - 1 \) whence by the induction hypothesis \( G' \) is derivable and we are done. We have that \( q_1 <_G q_3 \) since \( S_3 \) is a component of \( G' \), and that the only inequalities in \( G \) not in \( G' \) are of the form \( q_2 <_G p \) or \( p <_G q_2 \) which cannot occur since \( C \) is strict. (2) follows similarly to (1).

For (3) we first consider the subcase where \( l = 3 \) so we have \( q_1 = q_3 \). Here we have \( S_1 = \Gamma_1, q_2 \vdash q_1, R \) and \( S_2 = \Gamma_2 \vdash q_2, (q_1) \) are distinct components of \( C \); by applying \( (restart) \) to \( G \) we obtain \( G' \) where \( S_1 \) and \( S_2 \) have been replaced by \( S_3 = \Gamma_1, \Gamma_2, q_2 \vdash q_2, R \) which is derivable by \( (success) \). We now consider the possible patterns for \( l > 3 \): (3a) \( q_1 <_G q_2 \leq_G q_3 \leq_G p \), (3b) \( q_1 <_G q_2 \leq_G q_3 <_G p \), (3c) \( p \leq_G q_1 <_G q_2 \leq_G q_3 \). (3d) \( p <_G q_1 <_G q_2 \leq_G q_3 \). (3b) and (3c) are covered by pattern (2) and (3d) is covered by pattern (1) so we only need to consider (3a). \( S_1 = \Gamma_1 \vdash q_2, (q_3) \) and \( S_2 = \Gamma_2 \vdash q_3, (p) \) are
components of $G$. Applying $(\text{restart})$ to $G$ we obtain $G'$ where $S_1$ and $S_2$ have been replaced by $S_3 = \Gamma_1, \Gamma_2 \vdash q_2, (p)$. Now removing $q_3$ from $C$ we get a strict $G'$-cycle of length $l - 1$, and by the induction hypothesis $G'$ is derivable. \hfill \Box

Since it follows from Propositions 2 and 3 that the rules of $\text{GDLC}_h^\to$ can be applied to a valid hypersequent to obtain a set of valid atomic hypersequents, and from Propositions 4, 5 and 6 that all valid atomic hypersequents are provable in $\text{GDLC}_h^\to$, we get that $\text{GDLC}_h^\to$ is complete.

**Theorem 3.** For a hypersequent $G$, if $\models G$ then $G$ is derivable in $\text{GDLC}_h^\to$.

## 5 A Labelled Goal-Directed Calculus for $\text{LC}^\to$

$\text{GDLC}_h^\to$ has some nice properties for proof search; it is terminating, contraction-free and has invertible reduction rules that operate on the principal connective of one formula at a time. As a goal-directed calculus it is not quite satisfactory however. For a start the $(\text{mingle})$ and $(\text{restart})$ rules are not invertible so backtracking may be required to complete a proof. Alternative rules are:

\[
\begin{align*}
\text{(mingle') } & G[\Gamma_1, \Gamma_2 \vdash p, R_2 | \Gamma_1 \vdash q, R_1] \\
& \frac{G[\Gamma_1 \vdash q, R_1 | \Gamma_2 \vdash p, R_2]}{G[\Gamma_1 \vdash q, \Gamma_2 \vdash p, R_1]} \quad \text{(restart') } & G[\Gamma_1, \Gamma_2 \vdash p, (q)] & \frac{G[\Gamma_2 \vdash q, R]}{G[\Gamma_1 \vdash q, R | \Gamma_2 \vdash p, (q)]}
\end{align*}
\]

Replacing $(\text{mingle})$ and $(\text{restart})$ with $(\text{mingle'})$ and $(\text{restart'})$ gives a sound and complete system with invertible rules but to keep termination the condition $\Gamma_1 \not\subseteq \Gamma_2$ is needed for both cases, thereby losing the logical purity of the calculus.

A further problem for the goal-directedness of $\text{GDLC}_h^\to$ is the creation of components with just goals and restarts. Consider the following derivation:

\[
\begin{align*}
& \frac{p, q \vdash q, (r)}{p, q \vdash q, (r)} \\
& \frac{p, q \vdash r, (r)}{p \vdash p} \quad \frac{p \vdash p}{p \vdash q, (r)} \quad \frac{p \vdash q, (r)}{p \vdash p} \quad \frac{p, q \vdash r, (r)}{p, q \vdash r, (r)} \\
& \frac{p \vdash p}{p \vdash p} \quad \frac{p \vdash q, (r)}{p \vdash p} \quad \frac{p \vdash q, (r)}{p \vdash p} \quad \frac{p \vdash q, (r)}{p \vdash p}
\end{align*}
\]

Multiple uses of $(\text{restart})$ seem unwarranted for such a simple theorem. The problem is that when a reduction rule is applied to a formula $p \rightarrow q$ a new component is created with goal $p$, restart $q$ and no antecedent formulae; hence a formula with head $p$ can only be reduced using $(\text{reduction2})$, which adds an extra branch, and $(\text{restart})$. Alternative rules for $(\text{reduction1})$ and $(\text{reduction2})$ can be given that avoid this problem by keeping the same database formulae for the new component. However this loses the nice property of having the size of hypersequents bounded linearly by the size of the original formula and may require an exponential number of copies of the same formula to be reduced.

\[\text{In practice the goal-directedness of rules can reduce the amount of backtracking required dramatically; hence for proof search the non-invertible rules without subset conditions might be preferable.}\]
Our solution here is to maintain a global database using labels to keep track of which formulae are accessible to which goals. We can then give rules that reduce copies of a formula occurring in different components at the same time. To this end we define and interpret goal-directed queries as follows:

**Definition 11 (Goal-Directed Query).** A goal-directed query (query) is a structure of the form $\Gamma \vdash^? \delta_1 : G_1, R_1, H$ where:
- $\Gamma$ is a set of labelled formulae called the database.
- $H = \{(\delta_2 : G_2, R_2), \ldots, (\delta_n : G_n, R_n)\}$ is called the history.
- $\delta_1, \ldots, \delta_n$ are sets of labels.
- $G_1, \ldots, G_n$ are formulae called goals.
- $R_i = ()$ or $R_i = (q)$ ($q$ atomic) for $i = 1 \ldots n$ are called restarts.

**Definition 12 (Interpretation of Goal-Directed Queries).** Let $\text{Acc}(\Gamma, \delta) = \{A[x : A \in \Gamma \text{ and } x \in \delta]\}$ and for a query $Q = \Gamma \vdash^? \delta_1 : G_1, R_1, H$ where $H = \{(\delta_2 : G_2, R_2), \ldots, (\delta_n : G_n, R_n)\}$, define $\text{GQ} = \text{Acc}(\Gamma, \delta_1) \vdash G_1, R_1 | \ldots | \text{Acc}(\Gamma, \delta_n) \vdash G_n, R_n$ and write $\models Q$ iff $\models \text{GQ}$.

We introduce the following labelled goal-directed calculus for $\text{LC}^\rightarrow$.

**Definition 13 (GDLC$_1^\rightarrow$).** GDLC$_1^\rightarrow$ has the following rules:

- **(success)** $\Gamma \vdash^? \delta : q, R, H$ succeeds if $x : q \in \Gamma$ and $x \in \delta$
- **(implication)** From $\Gamma \vdash^? \delta : A \rightarrow B, R, H$ step to:
  $\Gamma, x : A \vdash^? \delta \cup \{x\} : B, R, H$ (x new)
- **(reduction1)** From $\Gamma, x : A_1 \rightarrow \ldots \rightarrow A_n \rightarrow q \vdash^? \delta : q, R, H$
  where: $x \in \delta$ and $n > 0$ step to:
  $\Gamma \vdash^? \delta : A_i, (q), H \cup \{\delta : q, R\}$ for $i = 1 \ldots n$
- **(reduction2)** From $\Gamma, x : A_1 \rightarrow \ldots \rightarrow A_n \rightarrow q \vdash^? \delta : q, R_1, H$
  where: $H = H' \cup \{(\gamma : p, R_2)\}$, $n > 0$ and $x \in \gamma$ step to:
  $\Gamma, x : q \vdash^? \gamma \cup \delta : p, R_2, H' \cup \{\delta : q, R_1\}$ and:
  $\Gamma \vdash^? \delta : A_i, (q), H \cup \{\delta : q, R_1\}$ for $i = 1 \ldots n$
- **(mingle)** From $\Gamma, x : q \vdash^? \delta : q, R_1, H$
  where: $H = H' \cup \{(\gamma : p, R_2)\}$ and $x \in \gamma$ step to:
  $\Gamma, x : q \vdash^? \delta \cup \gamma : p, R_2, H' \cup \{\delta : q, R_1\}$
- **(restart)** From $\Gamma \vdash^? \delta : q, R, H$
  where: $H = H' \cup \{(\gamma : p, (q))\}$ step to:
  $\Gamma \vdash^? \delta \cup \gamma : p, (q), H' \cup \{\delta : q, R_1\}$
- **(switch)** From $\Gamma \vdash^? \delta : q, R_1, H$
  where: $H = H' \cup \{(\gamma : p, R_2)\}$ step to:
  $\Gamma \vdash^? \gamma : p, R_2, H' \cup \{\delta : q, R_1\}$

\textsuperscript{5} Labels are also used in \textsuperscript{[12]} to give a non-terminating goal-directed system for $\text{LC}^\rightarrow$.
The \((\text{success})\) and \((\text{implication})\) rules are straightforward translations of the corresponding hypersequent rules. \((\text{reduction}1)\) and \((\text{reduction}2)\) match the current goal with a labelled formula in the database; the former for a label accessible from the current goal, the latter for a label accessible from a goal in the history. Both remove the formula from the database so that the formula is no longer accessible to \textit{any} goal in the query. The \((\text{mingle})\) and \((\text{restart})\) rules perform similar functions to the revised \((\text{mingle}')\) and \((\text{restart}')\) rules in Section 4. \((\text{switch})\) which has no counterpart in the hypersequent calculus, allows the computation to move from the current goal to one in the history.

\textbf{Example 3.} We show below how GDLC\textsubscript{1} avoids the repeated applications of \((\text{restart})\) characteristic of GDLC\textsubscript{h}:

\[
\begin{align*}
& x : p, y : p \rightarrow q, z : q \rightarrow r \vdash \{x, y, z : r, (,), \emptyset \} \\
& x : p, y : p \rightarrow q \quad \vdash \{x, y, z : (q, (r)), \{H1\} \\
& x : p \quad \vdash \{x, y, z : (p, q), \{H1, H2\} \}
\end{align*}
\]

\textit{success}

\textbf{Example 4.} A proof of the formula \((((p_1 \rightarrow p_2) \rightarrow q) \rightarrow ((p_2 \rightarrow p_3) \rightarrow q) \rightarrow \ldots \rightarrow ((p_{n-1} \rightarrow p_n) \rightarrow q) \rightarrow ((p_n \rightarrow p_1) \rightarrow q) \rightarrow q, (), \emptyset \)

\[
\begin{align*}
x : (p_1 \rightarrow p_2) \rightarrow q, y : (p_2 \rightarrow p_3) \rightarrow q, z : (p_3 \rightarrow p_1) \rightarrow q \vdash \{x, y, z : q, (,), \emptyset \} \\
y, z \quad \vdash \{x, y, z : p_1 \rightarrow p_2, (q), \{H1\} \quad H_1 = \{(x, y, z) : q, ()\} \\
(1) y, z, u : p_1 \quad \vdash \{x, y, z, u : p_2, (q), \{H1\} \} \\
y, z, u \quad \vdash \{x, y, z : (q, ()), \{H2\} \quad H_2 = \{(x, y, z, u) : p_2, (q)\} \\
z, u \quad \vdash \{x, y, z : p_2 \rightarrow p_3, (q), \{H1, H2\} \} \\
(2) z, u, v : p_2 \quad \vdash \{x, y, z, v : p_3, (q), \{H1, H2\} \} \\
z, u, v \quad \vdash \{x, y, z : (q, ()), \{H2, H3\} \quad H_3 = \{(x, y, z, v) : p_3, (q)\} \\
u, v \quad \vdash \{x, y, z : p_3 \rightarrow p_1, (q), \{H1, H2, H3\} \} \\
(3) u, v, w : p_3 \quad \vdash \{x, y, z, w : p_1, (q), \{H1, H2, H3\} \} \\
(4) u, v, w \quad \vdash \{x, y, z, u, w : p_2, (q), \{H1, H3, H4\} \quad H_4 = \{(x, y, z, w) : p_1, (q)\} \\
u, v, w \quad \vdash \{x, y, z, u, v, w : p_3, (q), \{H1, H2, H3\} \quad H'_2 = \{(x, y, z, u, w) : p_2, (q)\} \quad \text{success}
\end{align*}
\]

At \((1)\) and \((2)\) \(q\) fails to match anything in the database so \((\text{switch})\) is applied. At \((3)\) and \((4)\) the current goal matches an atom in the database accessible to a goal in the history, so \((\text{mingle})\) is applied.

\textbf{Theorem 4.} \textit{If a query} \(Q\) \textit{succeeds in GDLC}\textsubscript{1} then \(\models Q\).
We now turn our attention to the termination of \( \text{GDLC}_1 \). Unlike hypersequents, queries focus on one goal at a time and require the rule (\( \text{switch} \)) to move to a goal recorded in the history. This can cause unwanted looping, eg:

\[
\emptyset \vdash ? \emptyset : p_1,() \quad \{ \emptyset : p_2,() \} \quad \text{use (\( \text{switch} \)) to step to:}
\]

\[
\emptyset \vdash ? \emptyset : p_2,() \quad \{ \emptyset : p_1,() \} \quad \text{then use (\( \text{switch} \)) again to step to:}
\]

\[
\emptyset \vdash ? \emptyset : p_1,() \quad \{ \emptyset : p_2,() \} \quad \text{and so on...}
\]

Our solution is to divide the history into two parts, the left and right history. Goal-directed queries are written as \( \Gamma \vdash ? \delta_1 : G_1,R_1,LH,RH \) with the same properties as Definition 11 except that both \( LH \) and \( RH \) are sets of goals with sets of labels and restarts. Definition 12 is revised accordingly. The calculus \( \text{GDLC}_1 \) remains almost the same with the union of the left history and the right history replacing the history. Goal-directed queries resulting from applications of all rules except (\( \text{switch} \)) move everything from the right history into the left history, leaving the former empty. The (\( \text{switch} \)) rule is changed as follows:

(\( \text{switch} \)) From \( \Gamma \vdash ? \delta : q,LH,RH \) where: \( LH = LH' \cup \{ (\gamma : p,R_2) \} \) step to:

\[
\Gamma \vdash ? \gamma : p,R_2,LH',RH \cup \{ (\delta : q,R_1) \}
\]

Clearly (\( \text{switch} \)) can be applied successively only as many times as there are items in the left history.

Looping may also occur as a result of (\( \text{restart} \)) and (\( \text{mingle} \)) which both add labels to the set available for a particular goal. If the labels added already occur in the set then clearly nothing has been gained. We therefore add the subset condition (\( \delta \not\subseteq \gamma \)) to both rules. These changes do not affect the soundness of the calculus and allow us to prove the following termination theorem for \( \text{GDLC}_1 \).

**Theorem 5.** \( \text{GDLC}_1 \) is terminating.

The completeness of \( \text{GDLC}_1 \) is proved similarly to Theorem 3, noting that dividing the history only causes the calculus to terminate when no rule is applicable in any component of the corresponding hypersequent.

**Theorem 6.** For a query \( Q \), if \( \models Q \) then \( Q \) succeeds in \( \text{GDLC}_1 \).

To show that \( \text{GDLC}_1 \) is co-NP we give the following algorithm to decide whether a query fails or not for \( \text{GDLC}_1 \) and show that it is NP.

**Definition 14 (fail(\( Q \))).** For a rule \( R \) and query \( Q \), \( \text{applies}(R,Q) \) is true if \( R \) can be applied to \( Q \), \( \text{result}(R,Q) \) is the set of queries produced by applying \( R \) to \( Q \) and choose(\( S \)) for a set \( S \) is a member of \( S \) chosen non-deterministically\(^6\).

\[
\text{fail(query } Q)\{
\text{do } Q = \text{reduce}(Q);
\text{while } (Q != “Irreducible” && Q != “Success”);
\text{if } (Q == “Success”) \text{ return REJECT};
\text{else } \{
\text{do } Q = \text{expand}(Q);
\text{while } (Q != “Fail” && Q != “Success”);
\text{if } (Q == “Fail”) \text{ return ACCEPT};
\text{else return REJECT} \}\}
\]

\(^6\) Only (\( \text{reduction1} \)) and (\( \text{reduction2} \)) can result in more than one query to be chosen.
reduce(query Q)\{
    if (applies(success, Q))
        Q = “Success”;
    else if (applies(implication, Q))
        Q = choose(result(implication, Q));
    else if (applies(reduction1, Q))
        Q = choose(result(reduction1, Q));
    else if (applies(reduction2, Q))
        Q = choose(result(reduction2, Q));
    else if (applies(switch, Q))
        Q = choose(result(switch, Q));
    else Q = “Irreducible”;
    return Q
\}

expand(query Q)\{
    if (applies(success, Q))
        Q = “Success”;
    else if (applies(mingle, Q))
        Q = choose(result(mingle, Q));
    else if (applies(restart, Q))
        Q = choose(result(restart, Q));
    else if (applies(switch, Q))
        Q = choose(result(switch, Q));
    else Q = “Fail”;
    return Q
\}

Theorem 7. \[\text{fail}(Q) \in NP\]

Proof. We show that every execution of \(\text{fail}(Q)\) is polynomially bounded in the size \(s\) of \(Q\). \(\text{fail}\) has two distinct stages; applying \(\text{reduce}\) until \(Q\) succeeds or is irreducible, then for the latter case applying \(\text{expand}\) until \(Q\) succeeds or fails. We examine the length of these two steps separately, starting with \(\text{reduce}\). Since each application of (\(\text{implication}\)), (\(\text{reduction1}\)) and (\(\text{reduction2}\)) reduces the multiset complexity of \(Q\) we have that there can be at most \(O(s)\) applications of these rules. However, before each application of (\(\text{implication}\)), (\(\text{reduction1}\)) and (\(\text{reduction2}\)) there may be a number of applications of (\(\text{switch}\)). Now since each application of (\(\text{switch}\)) reduces the size of the right history of \(Q\) which is of order \(O(s)\) we can limit this number to \(O(s)\). Hence we get that there will be \(O(s.s) = O(s^2)\) \(\text{reduce}\) steps. To apply \(\text{reduce}\) requires checking the applicability of (\(\text{success}\)), (\(\text{implication}\)), (\(\text{reduction1}\)), (\(\text{reduction2}\)) and (\(\text{switch}\)) which is \(O(s^2)\) so our first step is \(O(s^2.s^2) = O(s^4)\). We turn our attention now to \(\text{expand}\). Since each (\(\text{mingle}\)) and (\(\text{restart}\)) step decreases the measure \(\sum_{i=1}^{n} (|\sum_{j=1}^{n} \delta_j| - |\delta_i|)\) there can only be \(O(s^2)\) applications of these rules. Again however we may require \(O(s)\) applications of (\(\text{switch}\)) giving \(O(s^2.s) = O(s^3)\) \(\text{expand}\) steps. Checking the applicability of (\(\text{success}\)), (\(\text{implication}\)) and (\(\text{switch}\)) is \(O(s^2)\) but checking the applicability of (\(\text{mingle}\)) and (\(\text{restart}\)) is \(O(s^4)\) due to the subset conditions so each \(\text{expand}\) step is \(O(s^4)\). Hence our second stage is \(O(s^4.s^3) = O(s^7)\) and we have that \(\text{fail}\) is \(O(s^7)\).

6 Finite-Valued Gõdel-Dummett Logics

\(\text{LC}_n\) is obtained from \(\text{LC}\) by restricting the number of truth values to \(n\). This can be achieved for our calculi by adding further axioms eg to \(\text{GDLC}_n^\perp\): \(\text{(n-success)} G|\Gamma_1 \vdash q_1, R_1|\Gamma_2, q_1 \vdash q_2, R_2| \ldots |\Gamma_n, q_{n-1} \vdash q_n, R_n\)

\(^7\) Note that the bound \(O(s^7)\) is worst-case. In practice there are many improvements that could be made to reduce the complexity of the algorithm.
A more elegant approach however is to check such a restriction as a proof progresses. Each application of the (mingle) rule restricts the set of truth values taken by the new goal, thus for example in LC$_3$ we have that $q \leq p$ or $p \leq r$ iff $q \leq p$ or $p \leq r$ or $r > 0$. Hence we can add a counter for each goal (current and in the history) interpreted as restrictions on the truth values for that goal. Validity for goal-directed queries in LC$_n$ is then defined as follows:

Definition 15 (Validity of Goal-Directed Queries for LC$_n$). $\Gamma \vdash^n_\Rightarrow_n \delta_1 : G_1, a_1, R_1, \{ (\delta_2 : G_2, a_2, R_2), \ldots, (\delta_m : G_m, a_m, R_m) \} \text{ is valid in }$ LC$_n$ if for all valuations $v$ for LC$_n$ there exists $i, 1 \leq i \leq m$, such that either $v(G_i) \geq 1 - (a_i/(n-1))$ or $v(G_i) \geq v(\land \text{Acc}(\Gamma, \delta_i) \land (G_i \rightarrow \phi^{R_i}))$ where $\phi^l = t$ and $\phi^{(C)} = C$.

We present the following goal-directed calculus for LC$_n$:

Definition 16 (GDLC$_1^+(n)$). GDLC$_1^+(n)$ consists of the rules (implication), (reduction1), (reduction2) and (switch) of GDLC$_1^+$ (with the same counters in both premises and conclusions) and also:

(success) $\Gamma \vdash^n_\Rightarrow_n \delta : q, a, R, H$ succeeds if either $x : q \in \Gamma$ and $x \in \delta$ or $a = n-1$

(mingle) From $\Gamma, x : q \vdash^n_\Rightarrow_n \delta : q, a, R_1, H$

where: $H = H' \cup \{ (\gamma : p, b, R_2) \}$ and $x \in \gamma$ step to:

$\Gamma, x : q \vdash^n_\Rightarrow_n \delta \cup \gamma : p, \max(a+1, b), R_2, H' \cup \{ (\delta : q, a, R_1) \}$

(restart) From $\Gamma \vdash^n_\Rightarrow_n \delta : q, a, R, H$

where: $H = H' \cup \{ (\gamma : p, b, (q)) \}$ step to:

$\Gamma \vdash^n_\Rightarrow_n \delta \cup \gamma : p, \max(a, b), (q), H' \cup \{ (\delta : q, a, R) \}$

Example 5. Consider the following proof of the characteristic axiom for LC$_3$:

$\vdash_3 \emptyset : (p_1 \rightarrow q) \rightarrow ((p_2 \rightarrow p_1) \rightarrow q) \rightarrow ((p_2 \rightarrow p_3) \rightarrow q) \rightarrow 0, ()$, $\emptyset$

$x : p_1 \rightarrow q, y : (p_1 \rightarrow p_2) \rightarrow q, z : (p_2 \rightarrow p_3) \rightarrow q \vdash^n_3 \{ x, y, z \} : q, 0, (), \emptyset$

$y, z \vdash_3 \{ x, y, z \} : p_1, 0, (), \{ H_1 \}$

$H_1 = \{ \{ x, y, z \} : q, 0, () \}$

$y, z \vdash_3 \{ x, y, z \} : q, 0, (), \{ H_2 \}$

$H_2 = \{ \{ x, y, z \} : p_1, 0, () \}$

$z \vdash_3 \{ x, y, z \} : p_1 \rightarrow p_1, 0, (), \{ H_1, H_2 \}$

$x, u : p_1 \vdash_3 \{ x, y, z, u \} : p_2, 0, (), \{ H_1, H_2 \}$

$z, u \vdash_3 \{ x, y, z \} : q, 0, (), \{ H_2, H_3 \}$

$H_3 = \{ \{ x, y, z, u \} : p_2, 0, () \}$

$u \vdash_3 \{ x, y, z \} : p_2 \rightarrow p_3, 0, (), \{ H_1, H_2, H_3 \}$

$u, v : p_2 \vdash_3 \{ x, y, z, v \} : p_3, 0, (), \{ H_1, H_2, H_3 \}$

$u, v \vdash_3 \{ x, y, z \} : p_1, 0, (), \{ H_1, H_3, H_4 \}$

$H_4 = \{ \{ x, y, z, v \} : p_3, 0, () \}$

$v \vdash_3 \{ x, y, z, u \} : p_2, 1, (q), \{ H_1, H_2, H_4 \}$

$v \vdash_3 \{ x, y, z, u, v \} : p_3, 2, (q), \{ H_1, H_2, H_3 \}$

success

Theorem 8. A query $Q$ succeeds in GDLC$_1^+(n)$ iff $\models_n Q$. 
7 Discussion

In this paper we have presented goal-directed systems for the implicational fragments of the finite-valued and infinite-valued Gödel-Dummett logics, $\text{LC}_n$ and $\text{LC}$, including terminating hypersequent calculi with simple axioms and local rules, and co-NP labelled calculi with invertible rules.

The first achievement of this work is to have extended the goal-directed approach to Gödel-Dummett logics. Goal-directed methods have been developed for a wide range of logics including classical, intuitionistic, substructural, intermediate and basic modal logics [12], but not as yet for many-valued or fuzzy logics. This work provides a starting point for an investigation of goal-directed methods for these logics. In particular we intend to look at Lukasiewicz logic $\text{L}$, recently given first sequent and hypersequent calculi [19].

Goal-directed systems are particularly suitable for proof search since they focus on a goal formula and do not decompose unnecessary database formulae. Our calculi have other advantages over others given in the literature however. Avron’s original hypersequent calculus [3] is clearly the most natural proof system for $\text{LC}$ but is unsuitable for proof search, while Sonobe’s calculus and related approaches [21, 29, 10] suffer from the problem of having non-local rules. In our opinion the state of the art is provided by Avron and Koniskowa’s calculi $\text{GLCR}_{\text{RS}}$, $\text{GLC}_{\text{RS}}^*$ and $\text{GLC}_{\text{RS}}^*$ [4], and also Baaz and Fermüller’s sequents of relations calculus $\text{RG}_{\infty}$ [5]. One immediate advantage that $\text{GDLC}_h^{\rightarrow}$ possesses over all these systems is that it has just one basic axiom and no expansion rules. Moreover, $\text{GDLC}_h^{\rightarrow}$, unlike Avron and Koniskowa’s calculi, has rules that operate only on principal connectives. $\text{RG}_{\infty}$ also has this property but uses two types of components. A more important advantage of $\text{GDLC}_h^{\rightarrow}$ over these calculi however is the fact that the size of hypersequents is linear in the size of the original formula. Even better from the point of view of proof search, is the labelled calculus $\text{GDLC}_l^{\rightarrow}$, which by maintaining a global database, ensures that subformulae are reduced only once in each branch of a proof, without removing accessibility to goals. Moreover, by imposing subset conditions $\text{GDLC}_l^{\rightarrow}$ becomes co-NP, although we note that using rules which rely on removing formulae from the database may prove to be more efficient.

Finally we comment on some proposed extensions. First, we note that it is straightforward to extend our calculi to cope with negation and conjunction, the former by introducing a constant $\bot$, and the latter by defining a simple normal form. Disjunction may be defined as $A \lor B = ((A \rightarrow B) \rightarrow B) \land ((B \rightarrow A) \rightarrow A)$ but at the cost of an exponential blow-up in the size of formulae. A better approach is to define a normal form and refine the rules of our calculi accordingly. To extend goal-directed methods for $\text{LC}$ and $\text{LC}_n$ to the first-order case, a good starting point is the universal fragment, a goal-directed calculus having been obtained for this fragment of intuitionistic logic using unification and run-time skolemisation [12].

---

8 Goal-directed calculi for $\text{LC}$ have been presented in [12] but they are non-terminating and significantly inferior to those given here.
References

A Logic for Probability in Quantum Systems

Ron van der Meyden and Manas Patra

School of Computer Science and Engineering
University of New South Wales, Sydney 2052, Australia
{meyden,manasp}@cse.unsw.edu.au

Abstract. Quantum computation deals with projective measurements and unitary transformations in finite dimensional Hilbert spaces. The paper presents a propositional logic designed to describe quantum computation at an operational level by supporting reasoning about the probabilities associated to such measurements: measurement probabilities, and transition probabilities (a quantum analogue of conditional probabilities). We present two axiomatizations, one for the logic as a whole and one for the fragment dealing just with measurement probabilities. These axiomatizations are proved to be sound and complete. The logic is also shown to be decidable, and we provide results characterizing its complexity in a number of cases.

1 Introduction

Quantum computing promises to open fresh vistas for computer science – almost 100 years after quantum mechanics revolutionized physics. We expect that logical methods will play a key role in the development of quantum computer systems, much like the role they have played in classical computing. (Pratt [Pra92] has also argued that logical support for reasoning about quantum effects may become essential in the next ten years to designers of classical hardware.) In this paper, we study what is potentially one of the building blocks of modal logics for quantum computing, by developing a logic for expressing properties of quantum states, in the same way that propositional logic, which expresses properties of classical states, provides the basis for modal logics of computation.

A new propositional basis is required for a logic of quantum computation because quantum mechanics introduced a new paradigm for the description of microscopic phenomena, radically revising the classical notions of state and observable. Both the notion of quantum state and the predictions quantum mechanics makes about measurements in the laboratory are inherently probabilistic in nature. This suggests that a logic for reasoning about quantum computations should support reasoning about the values of measurement probabilities.

There already exists an extensive body of literature on logics for reasoning about probability [AH94, Bac90, Car50, FHM90, Nil86]. Of these works, relatively few concern explicit statements about probability values; the most expressive

* On leave from Department of Physics, S. V. College, Univrsity of Delhi, Delhi-110021, India.
and most fully analyzed framework that does admit such statements is that of Fagin et al. [FHM90], which permits statements such as $\sum_{i=1}^{k} a_i P(\phi_i) \leq a_{k+1}$, where the $\phi_i$ are propositional formulas and the $a_i$ are integer constants.

However, these existing results cannot be directly applied to reasoning about probabilities in the context of quantum computation. One of the obstacles is that the space of propositions in quantum mechanics has a significantly more complicated structure than the space of classical propositions. Classical physics is based on a notion of measurement of observables that leaves the state of the system invariant, so that the order in which measurements are performed is irrelevant. By contrast, measurement in quantum mechanics changes the state, and the order of measurements matters.

Two measurements that commute are said to correspond to compatible observables, and it can be shown that such observables are co-measurable, i.e. their values may be inferred from a single measurement of another observable. While compatible observables interact classically, incompatible observables do not. Moreover, the probabilities related to incompatible observables are connected by a non-linear equation over the complex number field. The subject of “quantum logic” [BvN36] studies the algebraic structure of the subspaces of the quantum state space that are associated with measurements. This algebra turns out to have a non-boolean structure in which the distributive law fails. “Quantum probability” theory [Gud89] goes on to define a revised notion of probability space based on these non-boolean algebraic spaces. Quantum logic has been motivated by concerns for the philosophical foundations of quantum mechanics, seeking in particular to justify the use of Hilbert spaces in quantum mechanics.

Our approach in this paper will be somewhat more pragmatic: we will take the Hilbert space formalization for granted, and focus on the operationally observable probabilities of quantum measurement outcomes. We take finite dimensional Hilbert spaces as our semantic basis; the restriction to finite dimensions is precisely that used in quantum computation. We consider two probability operators: a monadic operator $P$ and a dyadic operator $T$. The operator $P$ resembles a standard probability operator. Unlike quantum logic, which permits boolean combinations of incompatible propositions, we restrict the application of this probability operator to boolean combinations of a set of compatible propositions. A consequence of this is that the logic of this probability operator is similar to the logic of the classical probability operator. The second type of probability operator $T$ that we consider resembles a classical conditional probability operator: in the quantum case, this notion corresponds to the probability of a transition between subspaces taking place when a measurement is performed.

The main results of the paper are completeness theorems for two axiomatizations. The first deals with the fragment of the logic containing only the operator $P$, and permitting the expression of linear inequalities over probability values. The complete axiomatization in this case is wholly propositional (i.e. does not require quantification), and consists of a set of axioms concerning the interaction of the probability operators and boolean logic, together with a set of axioms
for reasoning about linear inequalities. The second axiomatization is for the full language, with both operators \( P \) and \( T \). When axiomatizing reasoning about classical conditional probabilities, one needs an axiomatization of real closed fields \[\text{Tar51}\]. For our logic, we need an extended theory of algebraically closed fields with that of real closed fields embedded in it. Further, we show that the satisfiability problem is decidable for both languages we consider, and we characterize its complexity. The complexity classes we identify for the satisfiability problem turn out to be the same classes as in the complexity results of Fagin et al. \[\text{FHM90}\] for the classical case, viz., \( \text{NP complete} \) for the language with just the operator \( P \) and linear constraints, and in exponential space for the language with both operators \( P \) and \( T \) and with polynomial equations and quantification over numbers.

The structure of the paper is as follows. Section 2 provides an overview of the basic definitions of quantum mechanics. We introduce the syntax of the language in section 3.1 and provide its semantics in Section 3.2. Section 4 provides a few illustrations of what can be expressed in the logic. The axiomatizations and completeness results are presented in Section 5.1 (for the logic based on \( P \)), and Section 5.2 (for the logic based on both \( P \) and \( T \)). Section 6 concludes by discussing some questions raised by these results and sketching future work.

2 Quantum Theory

The most complete description of a physical system is given by its state. According to quantum theory the state of physical system \( S \) is a unit vector in a Hilbert space. Associated with \( S \) is a Hilbert space, that is complex vector space with an inner product. We shall restrict ourselves to finite-dimensional spaces, which are adequate for quantum computing and our logic. We assume that all states are pure states, as described below. However, we could easily extend the syntax and semantics to “mixed” states which are positive, normalised combination of pure states.

Postulate 1 Associated to \( S \) is a finite-dimensional complex Hilbert space \( \{H, \langle \cdot, \cdot \rangle\} \) called the state space. The dimension \( n \) is determined by the system.

We identify \( H \) with \( C^n \) with standard inner product; if \( |\alpha\rangle = (x_1 \ldots x_n), \ |\beta\rangle = (y_1 \ldots y_n) \) then \( \langle \alpha|\beta\rangle = \sum x_i y_i \) where \( \langle \alpha\rangle \) is the conjugate vector \( (\bar{x_1}, \ldots, \bar{x_n}) \). The inner product satisfies \( \langle \alpha|\beta_1 + \beta_2\rangle = \langle \alpha|\beta_1\rangle + \langle \alpha|\beta_2\rangle \) and \( \langle \alpha|\beta\rangle = \langle \beta|\alpha\rangle \) and \( \langle \alpha|\alpha\rangle \geq 0 \). The quantity \( \|\alpha\| = \sqrt{\langle \alpha|\alpha\rangle} \) is called the length of \( |\alpha\rangle \). If it is 1 then \( \alpha \) is called a unit vector. Note that the length or more generally \( |\langle \alpha|\beta\rangle| \) is invariant w.r.t. multiplication of \( |\alpha\rangle \) and \( |\beta\rangle \) by arbitrary complex constants of modulus 1. With this notation we extend Postulate 1.

Postulate 2 The state of the system is given by a unit vector, determined up to a scalar multiple of modulus 1. Moreover, each such vector is realisable as a state.
Thus $|\alpha\rangle$ and $|\beta\rangle$ represent the same state iff $|\alpha\rangle = e^{ic}|\beta\rangle$ $c$ real.

A basis $b = \{\alpha_1, \ldots, \alpha_n\}$ of $H$ is a linearly independent set of vectors such that every vector in $H$ is a (unique) linear combination of the $|\alpha_i\rangle$'s. It is orthonormal iff $\langle\alpha_i|\alpha_j\rangle = \delta_{ij}$ where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. From any set of $n$ linearly independent vectors we can construct an orthonormal basis.

It is the set of orthonormal vectors which correspond to the classical notion of states. Their occurrence in any test can be considered as mutually exclusive events. Henceforth basis will mean an orthonormal one.

**Postulate 3** Any orthonormal basis represents a realisable maximal test.

Let $n$ be the maximum number of different outcomes possible in a given system for any test. For example we may test for the value of the $z$-component of spin of an electron or polarisation of a photon. We imagine we have large number of similarly prepared systems called an ensemble and we test for the values of different measurable quantities like spin etc.

For a spin-$1/2$ system we always get a maximum of 2 outcomes (‘up’ and ‘down’) for any test. So $n = 2$. This number is a property of the system and according to the postulate equals the dimension of the state space. In general, we postulate that for an ensemble in an arbitrary state, it is always possible to devise a test that yields the $n$ outcomes corresponding to an orthonormal basis with definite probability.

**Postulate 4** If the system is prepared in state $|\alpha\rangle$ and a maximal test corresponding to a basis $b = \{|\beta_i\rangle\mid i = 1, \ldots, n\}$ is performed, the probability that the outcome will correspond to $|\beta_i\rangle$ is given by $p_i = |\langle\alpha|\beta_i\rangle|^2$.

Since one of the outcomes must occur, $\sum p_i = 1$. In the relative frequency interpretation of probability this means that if we have an ensemble of $N$ systems and perform a maximal test corresponding to $\{|\beta_i\rangle\}$ then if the frequency of outcome corresponding to $|\beta_i\rangle$ is $n_i$, we have $p_i = \lim_{N \to \infty} \frac{n_i}{N}$. If the system is known to be in one of the states in a basis $\{|\beta_i\rangle\}$, say $|\beta_1\rangle$, then $p_1 = 1$ and $p_i = 0$ for $i \neq 1$. That is, we can predict the outcome with certainty for this maximal test. This is the case that corresponds to the classical theory. However, if we choose a different maximal test corresponding to a different basis then the outcomes become random.

We note that if $H$ is a Hilbert space with inner product $\langle,\rangle$, then $H$ is isomorphic to the dual space of linear functionals (i.e. complex valued functions) on $H$. Thus for each $|\alpha\rangle$ let $\langle\alpha|$ denote its image under this isomorphism, called the dual. Then $\langle\langle\alpha|\beta\rangle\rangle = \langle\alpha|\beta\rangle$ by definition. Further, $\langle\langle\alpha|\beta\rangle|\gamma\rangle \overset{\text{def}}{=} \langle\beta|\gamma\rangle|\alpha\rangle$ is a linear operator on $H$. In particular, $|\alpha\rangle\langle\alpha|$ is identified as the projection operator on $|\alpha\rangle$. Let $\mathcal{L}(H)$ denote the space of linear operators on $H$. An operator $A \in \mathcal{L}(H)$ is hermitian if $\langle\alpha|A\beta\rangle = \langle A\alpha|\beta\rangle$ for all $|\alpha\rangle$ and $|\beta\rangle$. An operator $U$ is called unitary if $\langle U\alpha|U\beta\rangle = \langle\alpha|\beta\rangle$ for all $|\alpha\rangle$ and $|\beta\rangle$. In matrix notation let $B^\dagger$ denote the transposed conjugate of a square matrix $B$. Then $B$ is hermitian if $B = B^\dagger$ and $U$ is unitary if $U^{-1} = U^\dagger$. Hermitian and unitary matrices play a crucial role in quantum theory.
Let the system be in a state \(|\alpha_i\rangle\) where \(\{|\alpha_j\rangle\mid j = 1, \ldots, n\}\) is an orthonormal basis. Let \(\{|\beta_k\rangle\mid k = 1, \ldots, n\}\) be another orthonormal basis. Then if we do a maximal test with respect to the later basis the probability of obtaining result \(|\beta_k\rangle\) is \(p_{ik} = |\langle\alpha_i|\beta_k\rangle|^2\). \(p_{ik}\) can also be written as \(\text{Tr}(\langle\alpha_i|\langle\alpha_i|(|\beta_k\rangle\langle\beta_k|))\) where the trace \(\text{Tr}\) is the sum of the diagonal elements of a square matrix, which is independent of the representation. The \(p_{ij}\) are called the transition probabilities. Let \(U = (u_{ij} = \langle\alpha_i|\beta_j\rangle)\) be a matrix. Then \(U\) is unitary. It is the matrix which expresses the change of basis and \(p_{ij} = |u_{ij}|^2\). We thus see that the transition probability matrix is doubly stochastic, i.e., \(\sum_i p_{ij} = \sum_j p_{ij} = 1\). But an arbitrary doubly stochastic matrix (for example appearing in classical Markov processes) may not correspond to transition probability matrix in quantum theory because it may not satisfy \(p_{ij} = |u_{ij}|^2\) for some unitary \(U = (u_{ij})\). Such matrices \((p_{ij})\) are called orthostochastic. Thus the \(p_{ij}\) must satisfy some relations. We thus see an important difference with classical probability theory. We can not make arbitrary probability assignments (satisfying of course the usual probability constraints) but the probabilities must satisfy certain nonlinear inequalities. This is also true of the probabilities \(p_i\) introduced earlier.

3 Syntax and Semantics

We now present the syntax and semantics of our logic of quantum probabilities.

3.1 Syntax

For each dimension \(n \geq 1\) we define two languages interpreted with respect to \(n\)-dimensional Hilbert Space: \(\mathcal{L}_n(P)\) and \(\mathcal{L}_n(P, T)\). (We write \(\mathcal{L}_n(X)\) when making assertions that apply to both languages.) We begin by describing the language \(\mathcal{L}_n(P)\), which is based on the probability operator \(P\).

Maximal measurements correspond to orthonormal bases of the Hilbert space, which are related by unitary transformations, as discussed above. Bases are represented in both languages by means of basis variables \(b, c, d, \ldots\). A basis component of \(\mathcal{L}_n(X)\) is an expression of the form \(b_i\) where \(1 \leq i \leq n\) and \(b\) is a basis variable.1 Semantically, basis components correspond to the elements of an orthonormal basis of the \(n\)-dimensional Hilbert space.

The probability operators in our language will apply to formulas expressing properties of the outcome of a maximal measurement. We capture these properties by \(b\)-formulas, where \(b\) is a basis variable representing the measurement. A \(b\)-formula is a Boolean combination of \(b\)-components, i.e., an expression of the form \(b_i\), or \(\alpha \land \alpha'\), or \(\neg \alpha\), where \(\alpha\) and \(\alpha'\) are \(b\)-formulas. We define \(\alpha \lor \alpha'\) as \(\neg (\neg \alpha \land \neg \alpha')\). Note that all the basis components in a \(b\)-formula must be constructed from the same basis variable \(b\); if \(t\) and \(t'\) are distinct basis formulas then

\[\text{Note that we do not use subcripting to distinguish basis variables: } b_1 \text{ and } b_2 \text{ always denote components of the same basis } b, \text{ rather than two distinct basis variables. We use superscripting to denote distinct basis variables when use of distinct letters for the basis variables do not suffice.}\]
b_1 \land b'_1 is not a b-formula. Intuitively, this restriction ensures that b-formulas describe the outcomes of measurements compatible with the basis b, and prevents construction of formulas combining results of incompatible measurements.

A probability term is an expression of the form P(\alpha) where \alpha is a b-formula for some basis variable b. A linear probability atom is an expression of the form a_1 \cdot P(\alpha_1) + \ldots + a_k \cdot P(\alpha_k) \leq c, where each a_i is an integer, c is an integer, and each \alpha_i is a b'-formula for some basis constant b'. The formulas of the language \mathcal{L}_n(P) are all the boolean combinations of linear probability atoms i.e., each linear probability atom is a formula of \mathcal{L}_n(P), and if \phi_1 and \phi_2 are formulas of \mathcal{L}_n(P) then so are \neg \phi_1, \phi_1 \land \phi_2, \phi_1 \Rightarrow \phi_2, \phi_1 \Leftrightarrow \phi_2 may be defined in this language as usual. Expressions such as X < Y, X = Y, where X,Y are linear combinations of probability terms are also definable in this language.

For the language \mathcal{L}_n(P,T), we add transition probability terms, which are expressions of the form T(\alpha, \beta), where \alpha is a b-formula for some basis variable b and \beta is a c-formula for some basis variable c. Intuitively, these are a kind of conditional probability, expressing the probability that a measurement in basis c will have outcome satisfying \beta, given that the current state satisfies \alpha. A transition probability atom is an expression of the form p \leq 0, where p is a polynomial expression with integer coefficients over probability terms and transition probability terms. Note that linear probability atoms are a special case of transition probability atoms. As above, comparison operators other than \‘\leq‘ are definable. Note that quantum probabilities are inherently quadratic. Define a simple transition probability formula to be a boolean combination of transition probability atoms. We need quantification over both real and complex numbers and a new type of term, denoting a complex number, to represent the entries of unitary matrices.

The proof theory for \mathcal{L}(P,T) presented below will make use of results concerning the first order language \mathcal{L}_{\{+,=,\}} with equality the only predicate symbol and function symbols representing addition and multiplication. The language \mathcal{L}_{\{+,=,\}} leads to a recursively axiomatizable (in fact, decidable) theory both when interpreted with respect to the real field \mathbb{R} and the complex field \mathbb{C}. For our logic, it is convenient to first define the sorted first order language \mathcal{L}_{\mathbb{RC}} with two sorts \mathbb{R} and \mathbb{C}, with \mathbb{R} a subsort of \mathbb{C}. The language \mathcal{L}_{\mathbb{RC}} has equality as its only predicate symbol, a constant symbol c (of sort \mathbb{R}) for each integer, and the (infix) binary functions + and \cdot as its only function symbols. Both function symbols are overloaded: if t_1, t_2 are terms of sort \mathbb{R} (respectively, \mathbb{C}), then so are t_1 + t_2 and t_1 \cdot t_2. We write \forall x : \mathbb{R}.(\phi) and \forall x : \mathbb{C}.(\phi) for universal quantification over the reals and complex numbers respectively. In this language, we may define operations such as complex conjugation x + iy = x - iy, and the modulus of a complex number |x + iy| = \sqrt{x^2 + y^2} (where x and y are real), so we use such operations freely. The inequality x \leq y on real terms x, y is also definable (by \exists z : \mathbb{R}.(y = x + z^2)).

The language \mathcal{L}_{n}(P,T) is defined to be the extension of \mathcal{L}_{\mathbb{RC}} in which we add probability terms and transition probability terms as terms of sort \mathbb{R}, as well as
the terms $m_{ij}(b, c)$, of sort $C$, for $1 \leq i, j \leq n$ and basis variables $b, c$. Intuitively, $m_{ij}(b, c)$ denotes the $ij$-th entry of the unitary matrix that transforms the basis denoted by $b$ into the basis denoted by $c$.

3.2 Semantics

We now present the semantics for the language $L_n(P, T)$ (and consequently for the sublanguage $L_n(P)$). As remarked above, the language has been designed to allow it to serve as the propositional fragment of a modal logic. Thus, although there are no explicit modal operators, the semantics has some resemblances to Kripke semantics. In particular, we interpret formulas at a state within a collection of states, with respect to an interpretation of the atomic symbols.

A structure for $L_n(P, T)$ is an $n$-dimensional Hilbert space $H$. For notational convenience we use plain Greek letters ($\psi$ not $|\psi\rangle$ of $\mathbb{2}$) in this section. A state within this structure is a unit vector $\psi$ in $H$. An interpretation of $L_n(P, T)$ in a structure $H$ is a function $\pi$, such that

1. for each basis variable $b$, $\pi(b)$ is an orthonormal basis $\psi_1, \ldots, \psi_n$ of $H$; (we write $\pi(b)_i$ for $\psi_i$)
2. for each real variable $x$, $\pi(x)$ is a real number;
3. for each complex variable $X$, $\pi(X)$ is a complex number.

If $M = (m_{ij})$ is an $n \times n$ unitary matrix and $B = \psi_1, \ldots, \psi_n$ is a sequence of vectors of $H$, we write $MB$ for the sequence of vectors $\psi'_1, \ldots, \psi'_n$, where $\psi'_i = \sum_{k=1}^{n} m_{ik} \psi_i$. If $B$ is an orthonormal basis of $H$ then so is $MB$.

We extend the interpretation $\pi$ to terms $t$ of various sorts as follows. Given the term $t$, a state $\psi$ and an interpretation $\pi$, we define the interpretation $[t]_{\pi, \psi}$ of $X$ with respect to $\pi$ and $\psi$ as follows. Basis variables are interpreted as bases:

1. $[b]_{\pi, \psi} = \pi(b)$, when $b$ is a basis variable.

When $b$ is a basis variable, we interpret $b$-formulas as projection operators on $H$ (these may also be understood as representing the subspaces of $H$ onto which they project):

2. $[b]_{\pi, \psi} = |\psi\rangle \langle \psi|$, where $\psi = \pi(b)_i$;
3. $[\alpha_1 \land \alpha_2]_{\pi, \psi} = [\alpha_1]_{\pi, \psi} \cdot [\alpha_2]_{\pi, \psi}$ (this is the projection operator projecting onto the intersection of the subspaces of $H$ that are the images of the projection operators $[\alpha_1]_{\pi, \psi}$ and $[\alpha_1]_{\pi, \psi}$ which could be written as the product of these projectors);
4. $[-\alpha]_{\pi, \psi} = [\alpha]_{\pi, \psi}^\perp$ is the projection operator projecting onto the orthogonal complement of the image of $H$ under $[\alpha]_{\pi, \psi}$.

(The reason we have taken $\pi(b)$ to be a basis rather than sequence of projection operators is to provide semantics for the terms $m_{ij}(b, c)$.) Terms of sort $R$, including probability terms and transition probability terms, are interpreted as real numbers, and terms of sort $C$, including the unitary matrix entry terms $m_{ij}(b, c)$, are interpreted as complex numbers:
5. \([x]_{\pi,\psi} = \pi(x)\), when \(x\) is real or complex variable;
6. \([k]_{\pi,\psi} = k\), when \(k\) is an integer;
7. \([P(\alpha)]_{\pi,\psi} = \|\alpha\|_{\pi,\psi}^2\);
8. \([T(\alpha, \beta)]_{\pi,\psi} = \text{Tr}([\beta]_{\pi,\psi}[\alpha]_{\pi,\psi})\);
9. \([m_{ij}(b, c)]_{\pi,\psi} = c_{ij}\), where \(M = (c_{ij})\) is the \(n \times n\) (unitary) complex array such that \(M\pi(b) = \pi(c)\);
10. \([X \cdot Y]_{\pi,\psi} = [X]_{\pi,\psi} \cdot [X]_{\pi,\psi}\);
11. \([X + Y]_{\pi,\psi} = [X]_{\pi,\psi} + [Y]_{\pi,\psi}\).

To give semantics to formulas of \(L_n(P, T)\), we define a relation of satisfaction of a formula \(\phi\) at a state \(\psi\) in a structure \(H\), with respect to an interpretation \(\pi\), denoted by \(H, \pi, \psi \models \phi\). The definition is by the following induction:

1. \(H, \pi, \psi \models X = Y\) if \([X]_{\pi,\psi} = [Y]_{\pi,\psi}\) (in case of \(L_n(P)\), this clause is replaced by \(H, \pi, \psi \models X \leq c\) if \([X]_{\pi,\psi} \leq c\));
2. \(H, \pi, \psi \models \neg \phi\) if not \(H, \pi, \psi \models \phi\);
3. \(H, \pi, \psi \models \phi_1 \land \phi_2\) if \(H, \pi, \psi \models \phi_1\) and \(H, \pi, \psi \models \phi_2\);
4. \(H, \pi, \psi \models \exists x : \mathbb{R}(\phi)\) if there is a real number \(r\) such that \(H, \pi[r/x], \psi \models \phi\);
5. \(H, \pi, \psi \models \exists x : \mathbb{C}(\phi)\) if there is a complex number \(c\) such that \(H, \pi[c/x], \psi \models \phi\);

A formula \(\phi\) of \(L_n(P, T)\) is satisfiable (in the \(n\)-dimensional Hilbert space \(H\)) if there exists an interpretation \(\pi\) and a state \(\psi\) such that \(H, \pi, \psi \models \phi\). A formula \(\phi\) is valid (in \(H\)) if \(H, \pi, \psi \models \phi\) for all interpretations \(\pi\) and states \(\psi\).

4 Examples

We give some examples of formulas in our language which express important concepts of quantum mechanics.

**Superposition:** A vector \(|\alpha\rangle\) is a superposition of two vectors \(|\beta_1\rangle\) and \(|\beta_2\rangle\) if it is a linear combination of the two, i.e., \(|\alpha\rangle = c_1|\beta_1\rangle + c_2|\beta_2\rangle\) for some complex numbers \(c_1, c_2\). Consider the formula \(T(b_1, b_1' \lor b_2') = 1\). If \(\pi(b_1) = |\alpha\rangle\), \(\pi(b_1') = |\beta_1\rangle\) and \(\pi(b_2') = |\beta_2\rangle\) then \(H, \pi, \psi \models T(b_1, b_1' \lor b_2') = 1\) iff \(\text{Tr}(|\alpha\rangle\langle\beta_1| + |\beta_2\rangle\langle\beta_2|) = 1\). This is equivalent to \(|\langle\beta_1|\alpha\rangle|^2 + |\langle\beta_2|\alpha\rangle|^2 = 1\), which is true iff the state \(|\alpha\rangle\) is a superposition of the states \(|\beta_1\rangle\) and \(|\beta_2\rangle\). That is, the formula expresses that “\(b_1\) is a superposition of \(b_1'\) and \(b_2'\).”

**Phase Relations:** Let \(b_0, \ldots, b_k\) be \(k + 1\) bases. Then the following formula states a relation between \(b_0\) and the \(b_j\), for \(j = 1 \ldots k\).

\[
\text{MP}_k \quad \forall x_1 \ldots x_n : \mathbb{R}(\bigwedge_{i=1}^n P(b_0^i) = x_i^2 \Rightarrow \exists z_1 \ldots z_n : \mathbb{C}(\bigwedge_{i=1}^n |z_i| = 1 \land \bigwedge_{j=1}^k \bigwedge_{i=1}^n P(b_i^j) = |\Sigma_{r=1}^n m_{ir}(b_0^j, b_i^j)x_rz_r^2|)
\]

**Proposition 1.** The formula \(\text{MP}_k\) is valid for all \(k \geq 1\).
Proof. Let $\pi$ be any interpretation and $|\psi\rangle$ any vector in $H_n$. If $[P(b_i^0)]_{\pi,|\psi\rangle} = \pi(x_i)^2$, then we may write $|\psi\rangle = \sum_{i=1}^{n} c_i \pi(x_i)\pi(b_0)i$, where the $c_i$ are complex numbers with $|c_i| = 1$. Define $\pi(z_i) = c_i$. We can then calculate the probabilities with respect to other bases as follows:

$$[P(b_j^0)]_{\pi,|\psi\rangle} = |\langle \pi(b_j^0) | \psi\rangle|^2$$

$$= |\langle \sum_{r=1}^{n} m_{ir}(b_0^0, b_j^0) \pi(b_0^0) | \psi\rangle \cdot \pi(x_r) \cdot \pi(b_0^0)||^2$$

from which it can be seen that $MP_k$ holds.

**Quantum State Tomography:** Suppose we are given a collection of identically prepared systems (an ensemble), which corresponds to an unknown quantum state. Quantum state tomography (QST) addresses the problem of determining this unknown state. By measuring the ensemble in a single basis, we may determine a probability distribution over the outcomes associated to the basis elements. This distribution does not suffice to determine the state of the system. However, we may also divide the original collection of systems into subcollections and subject each subcollection to maximal measurement corresponding to an appropriately chosen orthonormal bases. We get sets of probability distributions. For an appropriately chosen set of measurements, this set of distributions suffices to compute the state. The following formula expresses this fact.

Let $u$ be the sequence of variables $u_{ij}^k$ where $1 \leq i, j, k \leq n$.

$$\exists u : C \left[ \bigwedge_{1 \leq i, j, k \leq n} m_{ij}(b, c^k) = u_{ij}^k \right.$$ 

$$\Rightarrow \forall z_1 \ldots z_n : C(\right.$$ 

$$\bigwedge_{1 \leq i, k \leq n} P(c_i^k) = |\sum_j u_{ij}^k z_j \sqrt{P(b_j)}/P(b_j)|^2$$

$$\Rightarrow \bigwedge_i P(b_i') = |\sum_j m_{ij}(b, b') z_j \sqrt{P(b_j)/P(b_j)}|^2 \right].$$

This formula is valid. It expresses the fact that there is a “pattern of inter-relation” between a set of bases $b$ and $c^1, c^2, \ldots c^n$, captured by the values $u$, such that for any vector $\psi$, the probabilities of measurements associated to a set of bases related in this pattern provide sufficient information to calculate the probabilities of measurements with respect to any other basis. Note that by the discussion of the formula $MP_k$ above, it is always possible to find values for $z_1, \ldots z_k$ such that

$$\bigwedge_{1 \leq i, k \leq n} P(c_i^k) = |\sum_j u_{ij}^k z_j \sqrt{P(b_j)/P(b_j)}|^2$$

is satisfied. Thus the universally quantified formula in the conclusion is never true vacuously. The formula therefore expresses that given an appropriately related set of bases $b$ and $c^1, c^2, \ldots c^n$, it is possible, given any basis $b'$, to calculate the values $P(b_i')$ from the values $u_{ij}^k$, $P(b_i)$, $P(c_i^k)$ and $m_{ij}(b, b')$. We do this by first solving the equation (1) for the phase values $z_1, \ldots, z_n$, and then computing $P(b_i')$ as $|\sum_j m_{ij}(b, b') z_j \sqrt{P(b_j)/P(b_j)}|^2$. 


5 Axiomatization

We now present axiomatizations and state the completeness results for the languages $\mathcal{L}_n(P)$ and $\mathcal{L}_n(P,T)$.

5.1 Axiomatizing $\mathcal{L}_n(P)$

The axiomatization of $\mathcal{L}_n(P)$ consists of a number of parts, each dealing with one of the syntactic constructs of the language.

The first fragment of our axiomatization deals with the boolean logic of the $b$-formulas. As this is slightly richer than propositional logic, we identify for each basis variable $b$ a fragment of the proof theory that deals only with $b$-formulas.

The axioms of this fragment can be taken to be any complete axiomatization of propositional logic over the atomic formulas $b_1, \ldots, b_n$, with, e.g., Modus Ponens as the proof rule, plus the following axioms that capture the fact that we are dealing with an $n$-dimensional Hilbert space:

\begin{align*}
\text{B1} & \quad b_1 \lor \ldots \lor b_n \\
\text{B2} & \quad \neg(b_i \land b_j) \quad \text{for } i \neq j
\end{align*}

We say that a $b$-formula $\phi$ is a $b$-tautology, and write $\vdash_b \phi$ if it can be derived from these axioms alone. Note that these definitions isolate reasoning about $b$-formulas from reasoning about $c$-formulas when $b$ and $c$ are distinct.

Next, we have some axioms capturing the properties of the probability operator. The following axioms correspond very closely to the axioms W1-W4 of Fagin et al. [FHM90], but with the difference that we need to be careful to respect the syntactic constraints on probability terms. In the following, we require that there exists a basis term $b$ such that $\phi, \phi_1$ and $\phi_2$ are $b$-formulas:

\begin{align*}
\text{P1} & \quad 0 \leq P(\phi) \leq 1 \\
\text{P2} & \quad P(\phi) = 1 \quad \text{if } \phi \text{ is a } b\text{-tautology} \\
\text{P3} & \quad P(\phi_1 \land \phi_2) + P(\phi_1 \land \neg \phi_2) = P(\phi_1) \\
\text{P4} & \quad P(\phi_1) = P(\phi_2) \quad \text{if } \phi_1 \Leftrightarrow \phi_2 \text{ is a } b\text{-tautology}
\end{align*}

Note that in P2 and P4, we deal with $b$-tautologies where Fagin et al have tautologies of propositional logic. Fagin et al note that in their axiomatization, there is no axiom corresponding to the fact that probability measures are countably additive. This does not make their axiomatization incomplete, because the countable additivity is not expressible in the logic. In our logic, we also do not have a countable additivity axiom, but the reason is somewhat simpler: the only measurable properties with respect to a basis denoted by $b$ are those corresponding to the $2^n$ inequivalent $b$-formulas. Semantically, these formulas correspond to the linear subspaces generated by subsets of the $n$ basis vectors.

In addition to the above axioms, we also need a set of axioms that capture reasoning about linear inequalities. That is, we need to be able to derive formulas such as $(2P(\phi_1) \leq 3 \land 4P(\phi_2) \leq 1) \Rightarrow P(\phi_1) + 2P(\phi_2) \leq 2$, the validity of which follows just from the meaning of these operations on real numbers, rather than the meaning of the probability terms. We refer the reader to Fagin et al. [FHM90] for such an axiomatization $\text{AX}_{\text{INeq}}$ of such reasoning, and leave it as an exercise to
construct an axiomatization $\text{AX}_{\text{INEQ}}'$ suited to $\mathcal{L}_n(P)$ (the difference is to replace all variable occurrences by probability terms).

Let $\text{AX}_n(P)$ be the abovementioned axioms and rules of inference. Then we have the following:

**Theorem 1.** $\text{AX}_n(P)$ is a sound and complete axiomatization of $\mathcal{L}_n(P)$.

The axiomatization $\text{AX}_n(P)$ is almost identical to Fagin et al’s axiomatization of for linear inequalities over classical probabilities — the main difference is the syntactic restrictions relating to $b$-formulas. Thus, from the point of view of the language $\mathcal{L}_n(P)$, quantum probabilities behave similarly to classical probabilities. The similarity is also reflected in the complexity of the logic:

**Theorem 2.** Satisfiability of a formula of $\mathcal{L}_n(P)$ in $n$-dimensional Hilbert space (with $n \geq 2$) is NP-complete.

The same complexity was obtained by FHM for their logic of classical probabilities. Interestingly, the proof of Theorem 1 shows that probabilities with respect to different bases act independently. More precisely, the completeness proof can be used to establish the following:

**Proposition 2.** Let $\phi_1, \ldots, \phi_m$ be formulas and $b^1, \ldots, b^m$ be distinct basis variables such that for each $j = 1, \ldots, n$, the only probability terms occurring in $\phi_j$ are $b^j$ probability terms. Then $\phi_1 \land \ldots \land \phi_m$ is satisfiable iff each $\phi_j$ is satisfiable.

Thus, unlike quantum logic, $\mathcal{L}_n(P)$ is unable to express constraints on the ways that incompatible propositions are “pasted together”. To capture such constraints, we need to turn to our richer logic, dealing with transition probabilities.

### 5.2 Axiomatizing $\mathcal{L}_n(P, T)$

We now present the axiomatization of $\mathcal{L}_n(P, T)$.

To capture the properties of the probability operator $P$, the axiomatization contains the axiomatization of $b$-formulas used above, and the probability axioms P1-P4. We need a similar set of axioms for the transition probabilities. In the following, $\phi, \phi_1, \phi_2$ are $b$-formulas for some basis variable $b$, and $\phi', \phi'_1, \phi'_2$ are $b'$-formulas for some basis variable $b'$. For a fixed $b$-formula $\phi$, the probabilities of a transition to a $b'$-formula satisfy four properties directly analogous to P1-P4 above:

- **T1** $T(\phi, \phi') \geq 0$
- **T2** $T(\phi, \phi') = 1$ when $\phi'$ is a $b'$-tautology
- **T3** $T(\phi, \phi'_1 \land \phi'_2) + T(\phi, \phi'_1 \land \neg \phi'_2) = T(\phi, \phi'_1)$
- **T4** $T(\phi, \phi'_1) = T(\phi, \phi'_2)$ if $\phi'_1 \iff \phi'_2$ is a $b'$-tautology

These properties allow us to decompose a transition probability term into an equivalent expression of transition probability terms in which the second argument contains only atomic basis formulas. A similar decomposition with respect to the first argument can be obtained using the following property:
\textbf{T5} \quad T(\phi, \phi') = T(\phi', \phi)

Additionally, we have the following properties concerning a number of special cases of transitions. The first concerns transitions within a given basis:

\textbf{T6a} \quad T(b_i, b_j) = 0 \quad \text{when } i \neq j

\textbf{T6b} \quad T(b_i, b_i) = 1

Next, note that the formula \(P(b_i) = 1\) can be understood as saying that the state \(\psi\) at which the formula is being evaluated is equal to the \(i\)-th vector in the basis \(b\). The following property can be understood as stating that transition probabilities for transitions from the current state reduce to simple probabilities:

\textbf{T7} \quad P(b_i) = 1 \Rightarrow T(b_i, \phi) = P(\phi)

For \(L_n(P)\) our axiomatization used a set of axioms for reasoning about linear inequalities. In the case of \(L_n(P, T)\), we need to reason about non-linear polynomials. Moreover, to capture the quantum nature of the probabilities, we need to reason about complex numbers. This leads us to include in the axiomatization a set of axioms for reasoning about real and complex numbers. The following proposition follows from the fact that the complex numbers may be represented as pairs of real numbers, that the operations of complex addition and multiplication may be defined as operations on these pairs, and that the language \(L_{\{+, \ldots =\}}\) has an axiomatizable theory with respect to the standard model \(R\) \([\text{Tar51]}\).

\textbf{Proposition 3.} The set of valid formulas of \(L_{\text{RC}}\) has a recursive axiomatization \(\text{AX}_{\text{RC}}\).

The rules of inference of \(\text{AX}_{\text{RC}}\) are the usual rules for first order logic with equality. For reasons of space we do not list the axioms of \(\text{AX}_{\text{RC}}\) here. We include these axioms and rules of inference in our axiomatization. The language \(L_n(P, T)\) contains the terms \(m_{ij}(b, c)\), of complex number sort, to represent the unitary operators associated with basis transformations. The following properties are direct from the definition of these terms:

\textbf{M1} \quad m_{ij}(b, c) = m_{ji}(c, b)

The fact that the transformation from a basis to itself corresponds to the identity matrix, and that consecutive basis transformations correspond to matrix multiplication, are captured by the next two properties:

\textbf{M2a} \quad m_{ij}(b, b) = 1 \text{ if } i = j

\textbf{M2b} \quad m_{ij}(b, b) = 0 \text{ if } i \neq j

\textbf{M3} \quad m_{ij}(b, d) = \sum_{k=1}^{n} m_{ik}(b, c)m_{kj}(c, d)

We note that the following property, expressing unitarity of the transformation, follows from \(\text{M1-\text{M3}}\):

\textbf{M4} \quad \sum_{k=1}^{n} m_{ik}(b, c)m_{jk}(b, c) = 1

These matrices are connected to probabilities by the following axiom:

\textbf{MT} \quad T(b_i, c_j) = |m_{ij}(b, c)|^2

We note that \(\text{M1-\text{M3}}\) and \(\text{MT}\) imply some of the properties of transition probabilities noted above. In particular, \(T1\), \(T2\) and \(T6\) become derivable, as does the case \(T(b_i, b_j) = T(b_j, b_i)\) of \(T5\).

Let the axiomatization \(\text{AX}_{\text{n}(P, T)}\) consist of the propositional component with \(\text{B1-\text{B2}}\) for the \(b\)-formulas, the axioms \(\text{P1-P4}, \text{T1-T7}\), the axioms and rules of \(\text{AX}_{\text{RC}}\) (including the usual rules of inference for first order logic), \(\text{M1-\text{M3}}, \text{MT}\)
and the axiom $\text{MP}_k$ (see Section 4) for all $k \leq n^2 - n + 1$. Then we have the following:

**Theorem 3.** $\text{AX}_n(P, T)$ is a sound and complete axiomatization for the language $\mathcal{L}_n(P, T)$.

Note that although we have shown that $\text{MP}_k$ is sound for all $k \geq 1$, we have only included its instances for $k \leq n^2 - n + 1$ in the set of axioms. Indeed, as part of the completeness proof we show that it is not necessary to include $\text{MP}_k$ for larger $k$ since it already follows:

**Theorem 4.** For every number $k \geq 1$, the formula $\text{MP}_k$ is a theorem of $\text{AX}_n(P, T)$.

We have stated this result as a theorem because we feel that it is of significance for physics as well as the logic of quantum probabilities. It shows that to determine whether a set of numbers can have arisen as the probabilities associated to a set of $k$ bases in $n$-dimensional Hilbert space, it suffices to check the probabilities associated to every subset of size $n^2 - n + 1$.

As for the language $\mathcal{L}_n(P)$, we also can also obtain from the completeness proof some complexity bounds for $\mathcal{L}_n(P, T)$.

**Theorem 5.** Satisfiability of a formula in $\mathcal{L}_n(P, T)$ can be decided in exponential space. If the formula is quantifier free, then its satisfiability can be decided in polynomial space.

The proof of this result makes use of results of Ben-Or, Kozen and Reif [BKR86] for the full language, and of Canny [Can88] for the quantifier free case.

### 6 Conclusion

A topic that has been of some interest in the quantum mechanics literature is the extent to which it is possible to eliminate the use of complex numbers, and to reason about quantum probabilities purely as real numbers. This requires the characterization of the relationships between the quantum probabilities that follow from their Hilbert space definition. These relationships have been characterized in some low dimensions [Per95], but their characterization in general remains an open problem. Our work may provide an avenue to address this problem, by applying quantifier elimination to our axiomatization.

We have shown our logic to be to be decidable. An interesting topic for further research is to determine the extent to which it is possible to further enrich the logic while retaining decidability/axiomatizability. Extensions that suggest themselves are temporal logic, dynamic logic and the logic of knowledge. Even before embarking on a study of such modal extensions, a variety of constructs dealing only with a single state are worthy of study. Constructs such as quantification over bases and unitary transformations, can also be added while keeping
the language decidable. One construct that is of critical significance for quantum computing is the tensor product. We can already handle this to some extent simply by applying our language to the case where the dimension $n$ is a product $n_1 \cdot n_2$, but it is desirable to have the tensor product as a more integral part of the language. We plan to study such extensions in future work.

References


A Strongly Normalising Curry-Howard Correspondence for IZF Set Theory

Alexandre Miquel
Laboratoire de Recherche en Informatique
Université Paris-Sud, 91405 Orsay Cedex, France
Alexandre.Miquel@lri.fr

Abstract. We propose a method for realising the proofs of Intuitionistic Zermelo-Fraenkel set theory (IZF) by strongly normalising \( \lambda \)-terms. This method relies on the introduction of a Curry-style type theory extended with specific subtyping principles, which is then used as a low-level language to interpret IZF via a representation of sets as pointed graphs inspired by Aczel’s hyperset theory.

As a consequence, we refine a classical result of Myhill and Friedman by showing how a strongly normalising \( \lambda \)-term that computes a function of type \( \mathbb{N} \to \mathbb{N} \) can be extracted from the proof of its existence in IZF.

1 Introduction

In this paper, we revisit the work of Myhill [12], Friedman [4], McCarty [9] and Krivine [6, 7] related to the study of the computational contents of the proofs of set theory. Unlike the former approaches that are based on variants of Kleene’s realisability, we propose a different framework that combines ideas coming from hyperset theory to the theory of type systems [2, 5, 8].

Technically, our work relies on the introduction of a type system which is used as an assembly language to interpret the axioms of intuitionisitic set theory in terms of pointed graphs. Conceptually, this translation is the natural reformulation of Aczel’s model of hyperset theory [1] (originally achieved in set theory) in a type-theoretical framework. Of course, building this model in type theory allows us to benefit from the representation of proofs as \( \lambda \)-terms. Moreover, since the type system we are using enjoys the strong normalisation property, any proof of IZF is thus realised by a strongly normalising \( \lambda \)-term via our translation.

Apart from the strong normalisation property for IZF—which is the main contribution of the paper—there are several interests in using a type-theoretical framework as an intermediate language for studying set theory.

On the practical side, one may benefit from the use of many proof-assistants based on type theory to build the corresponding \( \lambda \)-terms explicitly: since the proofs of the axioms of IZF tend to be quite large, writing them down would be almost impossible by hand. By using the method described in this paper, the
author could explicitly construct (and check) all the proof-terms of IZF axioms with the help of the Coq proof-assistant [3].

On a more theoretical side, this decomposition gives new insights about the relationship between set theories and type theories. As far as we know, there is currently no type system whose proof theoretical strength reaches the one of ZF. Despite its non-standard subtyping rules, the extended type system we will present in section 5 deserves the credit of enjoying this property.

2 The Logical Framework

In this section, we introduce the core part of the logical framework we will use in this paper. The initial system, called $F\omega.2$, will be first extended in paragraph 3.2 and then in section 5 to reach the proof theoretical strength of IZF.

2.1 System $F\omega.2$

System $F\omega.2$—or system $F\omega$ with one universe—is organised in two syntactic categories: a syntactic category of object-terms in order to represent mathematical objects, and a syntactic category of proof-terms in order to represent mathematical proofs.

Object terms actually form an autonomous type system (the ‘higher part’ of system $F\omega.2$) which is completely independent from proof-terms. (Unlike the calculus of constructions, system $F\omega.2$ has no proof-dependent types.) This type system can be seen as an extension of Martin-Löf’s logical framework

\[
M, N, T, U, A, B ::= \text{Type}_1 | \text{Type}_2 | \Pi x: T. U \\
\text{Object terms} \\
| x | \lambda x: T. U | M(N) \\
| \text{Prop} | A \Rightarrow B \\
| \forall x: T. B | \exists x: T. B
\]

with a primitive type $\text{Prop}$ of propositions (below the first universe $\text{Type}_1$), plus extra constructions to represent implication and universal/existential quantification. In this presentation, the letters $M, N$ denote arbitrary object terms, whereas the letters $T, U$ are reserved for types (i.e. the terms of type $\text{Type}_1$ or $\text{Type}_2$) and the letters $A, B$ for propositions (i.e. the terms of type $\text{Prop}$).

As in Martin-Löf’s logical framework [8], we make a distinction between a universe $\text{Type}_1$ of small data-types and a (top) universe $\text{Type}_2$ of large data-types. For convenience, we also consider a cumulativity rule $\text{Type}_1 \subset \text{Type}_2$. Notice that at the level of object terms, $\text{Prop}$ is not a sort, and propositions are not

---

1 The corresponding proof-scripts can be downloaded on the author’s web page at http://pauillac.inria.fr/~miquel.

2 “$F\omega.2$” means: “system $F$ with higher-order twice”.

3 To preserve simplicity, we do not address here the problem of propagating cumulativity through dependent products.
Formation of signatures: \( \Sigma \vdash \)

\[
\begin{align*}
\emptyset & \vdash \\
\Sigma ; [x : T] & \vdash \ x \notin DV(\Sigma)
\end{align*}
\]

Typing rules of object terms: \( \Sigma \vdash M : T \)

\[
\begin{align*}
\Sigma \vdash \text{Prop} : \text{Type}_1 & \quad \Sigma \vdash \text{Type}_1 : \text{Type}_2 \\
\Sigma \vdash \text{Type}_2 & \quad \Sigma \vdash M : \text{Type}_i \\
\Sigma \vdash T : \text{Type} & \quad \Sigma \vdash \Pi x : T. U : \text{Type}_i^{i \in \{1, 2\}} \\
\Sigma \vdash \lambda x : T. M : \text{Type} & \quad \Sigma \vdash \Pi x : \text{Type}_i \vdash N : T \\
\Sigma \vdash M : T & \quad \Sigma \vdash M(N) : U \{x := N\} \\
\Sigma \vdash M \Rightarrow N : T & \quad \Sigma \vdash M \Rightarrow N \vdash N : T \\
\Sigma \vdash \exists x : T. B : \text{Prop} & \quad \Sigma \vdash \forall x : T. (A \Rightarrow B) : \text{Prop}
\end{align*}
\]

Formation of logical contexts: \( \Sigma \vdash \Gamma \text{ ctx} \)

\[
\begin{align*}
\Sigma \vdash \emptyset & \text{ ctx} \\
\Sigma \vdash \Gamma \text{ ctx} & \quad \Sigma \vdash [\xi : A] \text{ ctx} \quad \xi \notin DV(\Gamma)
\end{align*}
\]

Typing rules of proof-terms: \( (\Sigma) \Gamma \vdash t : A \)

\[
\begin{align*}
(\Sigma) \Gamma \vdash \xi : A & \quad (\xi : A) \in \Gamma \\
(\Sigma) \Gamma \vdash \lambda \xi . t : A \Rightarrow B & \quad A =_\beta A' \\
(\Sigma) \Gamma \vdash [\xi : A] t : B & \quad (\Sigma) \Gamma \vdash t : B \\
(\Sigma) \Gamma \vdash \forall \xi . t : A \Rightarrow B & \quad (\Sigma) \Gamma \vdash [x : T] t : B \\
(\Sigma) \Gamma \vdash \Pi x : T. B & \quad x \notin FV(\Gamma) \\
(\Sigma) \Gamma \vdash \forall x : T. B & \quad (\Sigma) \Gamma \vdash t : B \{x := N\} \\
(\Sigma) \Gamma \vdash \exists x : T. B & \quad (\Sigma) \Gamma \vdash t : \exists x : T. B \\
(\Sigma) \Gamma \vdash t : (\exists x : T. A) \Rightarrow B \\
(\Sigma) \Gamma \vdash t : (\forall x : T. B) \Rightarrow (\forall x : T. (A \Rightarrow B)) \\
(\Sigma) \Gamma \vdash t : (\Pi x : T. U) \Rightarrow (\Pi x : T. U)
\end{align*}
\]

Fig. 1. Typing rules of system \( F\omega.2 \)
data-types. (Of course, propositions will be considered as data-types, but only at the level of proof-terms.)

Formally, the type system of object terms is based on a judgement $\Sigma \vdash M : T$ which expresses that the term $M$ has type $T$ under the assumptions in $\Sigma$. The corresponding typing assumptions are regrouped in *signatures*

\[
\Sigma ::= [x_1 : T_1; \ldots; x_n : T_n]
\]

that are finite ordered lists of declarations of the form $(x : T)$. In order to ensure that the terms $T$ involved in such declarations are well-formed types, we also need a judgement $\Sigma \vdash$ which expresses that the signature $\Sigma$ is well-formed. The typing rules for both judgements are then defined by mutual recursion thanks to the rules given in Fig. 1.

The proof system (i.e. the ‘lower part’) of system $\text{F}_\omega.2$ follows the typing discipline à la Curry, so that proofs terms are actually pure $\lambda$-terms:

\[
t, u ::= \xi \mid \lambda \xi . t \mid tu
\]

By this, we mean that the universal and existential quantifications are treated as infinitary intersection and union types respectively, so that in practice the corresponding introduction and elimination rules have no impact on proof-terms. On the other hand, implication is introduced and eliminated as usual, by the means of $\lambda$-abstraction and application.

Proof-terms depend on assumptions that are declared in *logical contexts*:

\[
\Gamma ::= [\xi_1 : A_1; \ldots; \xi_k : A_k].
\]

Since the object-terms $A_i$ involved in such declarations cannot depend on proof-variables, the order of these declarations is irrelevant. To ensure that each $A_i$ is a well-formed proposition (in a given signature), we introduce a judgement $\Sigma \vdash \Gamma \text{ ctx}$ which expresses that the logical context $\Gamma$ is well-formed in the signature $\Sigma$. Finally, the last judgement $(\Sigma)\Gamma \vdash t : A$ expresses that in the signature $\Sigma$, the proof-term $t$ is a proof of the sequent $\Gamma \vdash A$. The rules of inference for both judgements are given in Fig. 1.

Although the logic of system $\text{F}_\omega.2$ is ultimately built on implication and quantifiers, the connectives $\land$, $\lor$ and units $\top$, $\bot$ are definable by the means of standard second-order encodings, as well as Leibniz equality.

The **Primitive Existential Quantifier**. Although the existential quantifier could have been defined purely in terms of $\forall$ and $\Rightarrow$ (by the mean of the standard second-order encoding of the existential quantifier), system $\text{F}_\omega.2$ introduces a primitive form of existential quantification that behaves exactly as an infinitary union type constructor. (See the last two proof-typing rules of Fig. 1.)

In practice, the primitive form of existential quantifier—which restores a symmetry between quantifiers in a spirit which is very close to the one of standard realisability—tends to produce more compact proof-terms than the second-order
encoding. But the main reason for using the primitive form of existential quantification is that denotations of propositions become much more readable in the normalisation model, a point which is important when studying intuitionistic forms of the axiom of choice as we will do in paragraphs 4.3 and 4.4.

From the point of view of provability of course, both forms of existential quantification are logically equivalent, and the full system with primitive form of existential quantification is nothing else but a conservative extension of the system restricted to $\forall$ and $\Rightarrow$ only.

3 Interpreting Zermelo’s Set Theory

In this section, we briefly explain how to encode Intuitionistic Zermelo’s set theory in system $F\omega.2$. We only give the basics of the encoding whose implementation details can be found in the author’s PhD thesis [11].

Formally, Intuitionistic Zermelo’s set theory (IZ) is the intuitionistic first-order theory based on two binary predicates ‘=’ (equality) and ‘∈’ (membership) whose axioms are given in Fig. 2.

<table>
<thead>
<tr>
<th>Equality &amp; Compatibility axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Refl) $\forall x \ x = x$</td>
</tr>
<tr>
<td>(Sym) $\forall x, y \ x = y \Rightarrow y = x$</td>
</tr>
<tr>
<td>(Trans) $\forall x, y, z \ x = y \Rightarrow y = z \Rightarrow x = z$</td>
</tr>
<tr>
<td>(Compat-L) $\forall x, y, z \ x = y \Rightarrow y \in z \Rightarrow x \in z$</td>
</tr>
<tr>
<td>(Compat-R) $\forall x, y, z \ x \in y \Rightarrow y = z \Rightarrow x \in z$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zermelo’s axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Ext) $\forall a, b \ (\forall x \ x \in a \iff x \in b) \Rightarrow a = b$</td>
</tr>
<tr>
<td>(Pair) $\forall a, b \exists c \forall x \ x \in c \iff x = a \lor x = b$</td>
</tr>
<tr>
<td>(Select) $\forall a \exists b \forall x \ x \in b \iff x \in a \land \phi$</td>
</tr>
<tr>
<td>where $\phi$ is any formula such that $b \notin FV(\phi)$</td>
</tr>
<tr>
<td>(Power) $\forall a \exists b \forall x \ x \in b \iff (\forall y \ y \in x \Rightarrow y \in a)$</td>
</tr>
<tr>
<td>(Union) $\forall a \exists b \forall x \ x \in b \iff (\exists y \ y \in a \land x \in y)$</td>
</tr>
<tr>
<td>(Infinity) $\exists a \ \emptyset \in a \land (\forall x \ x \in a \Rightarrow x \cup {x} \in a)$</td>
</tr>
</tbody>
</table>

Fig. 2. Axioms of Zermelo’s set theory

3.1 Sets as Pointed Graphs

A set is represented in system $F\omega.2$ as a pointed graph, that is, as a triple $(X, A, a)$ where:
1. $X : Type_1$ is the carrier (i.e. the type of vertices)
2. $A : X \to X \to Prop$ is the edge relation (or local membership)
3. $a : X$ is the root of the pointed graph.

Intuitively, a pointed graph $(X, A, a)$ can be thought as a kind of transitive closure of a set whose structure is given by the relation $A(x, y)$ (that can be read as: ‘$x$ is a local element of $y$’) and by the root $a$ (i.e. the entry point in the transitive closure).

Notice that unlike the terms $X$, $A$ and $a$, the triple $(X, A, a)$ is not a real object of system $F\omega.2$ (which does not provide any pairing mechanism) but only an informal notation to group related components. Similarly, we also introduce the following shorthands

$$\begin{align*}
\forall (X, A, a). \phi & \triangleq (X : Type_1 . \forall A : (X \to X \to Prop) . \forall a : X . \phi) \\
\exists (X, A, a). \phi & \triangleq (X : Type_1 . \exists A : (X \to X \to Prop) . \exists a : X . \phi) \\
\lambda (X, A, a). M & \triangleq (X : Type_1 . \lambda A : (X \to X \to Prop) . \lambda a : X . M)
\end{align*}$$

to denote the universal and existential quantifications as well as the $\lambda$-abstraction over the class of pointed graphs.

Extensional equality of set theory is interpreted as bisimilarity in the class of pointed graphs. This relation denoted by $(X, A, a) \approx (Y, B, b)$ is defined by:

$$(X, A, a) \approx (Y, B, b) \triangleq \exists R : (X \to Y \to Prop) .
\begin{align*}
(\forall x, x' : X . \forall y : Y . \ A(x', x) \land R(x, y) & \Rightarrow \exists y' : Y . \ B(y', y) \land R(x', y')) \land \\
(\forall y, y' : Y . \forall x : X . \ B(y', y) \land R(x, y) & \Rightarrow \exists x' : X . \ A(x', x) \land R(x', y')) \land \\
R(a, b).)
\end{align*}$$

Membership is then interpreted as shifted bisimilarity, namely:

$$(X, A, a) \in (Y, B, b) \triangleq \exists b' : B . \ B(b', b) \land (X, A, a) \approx (Y, B, b').$$

Once the interpretation of equality and membership has been defined, it is straightforward to translate any formula $\phi$ of set theory as a proposition $\phi^*$ in system $F\omega.2$. For that, we consider a fixed mapping that associates three distinct variables $X_i : Type_1$, $A_i : X_i \to X_i \to Prop$ and $a_i : X_i$ of the type system $F\omega.2$ to each variable $x_i$ of the first-order language of set theory, and we set:

$$\begin{align*}
(x_i = x_j)^* & \triangleq (X_i, A_i, a_i) \approx (X_j, A_j, a_j) \\
(x_i \in x_j)^* & \triangleq (X_i, A_i, a_i) \in (X_j, A_j, a_j) \\
(\phi \diamond \psi)^* & \triangleq \phi^* \diamond \psi^* \quad (\diamond \in \{\Rightarrow; \land; \lor\}) \\
(Q x \phi)^* & \triangleq Q(X, A, a) . \phi^* \quad (Q \in \{\forall; \exists\})
\end{align*}$$

It is straightforward to check that this translation validates the equality and compatibility axioms of Fig. 2.
3.2 Soundness of Zermelo’s Axioms

The main interest of interpreting sets as pointed graphs and equality as bisimilarity is that it automatically validates the axiom of extensionality:

**Proposition 1 (Extensionality)** — The translation of the extensionality axiom is provable in system $F\omega.2$.

As pointed out by [6, 9, 12], the interpretation of the extensionality axiom is the cornerstone of any computational interpretation of (the proofs of) set theory. On the other hand, proving that the bisimilarity relation is extensional w.r.t. the shifted bisimilarity relation is quite easy, and the formalisation of this proof in type-theory gives the corresponding $\lambda$-term for free.

The other axioms of Zermelo express the possibility of constructing new sets from other sets by several means. Except for the axiom of infinity, all the corresponding constructions have a natural translation in terms of pointed graphs that can be formalised in system $F\omega.2$, so that:

**Proposition 2 (Finitary Zermelo axioms)** — The translation of (Pair), (Power), (Union) and of each instance of (Select) is provable in $F\omega.2$.

However, the axiom of infinity poses another problem, since its interpretation in terms of pointed graphs requires an infinite small data-type whose existence cannot be proved in the logical framework we presented in section 2.4.

As proposed in [11], a way to solve this problem is to add an extra universe below the universe $Type_1$ (i.e. a universe $Type_0$) from which one easily reconstructs a type of numerals by suitable encodings (so that the full construction of a model of IZ actually takes place in system $F\omega.3$).

In this paper, we consider a simpler solution by extending our logical framework with primitive numerals. For that, we introduce a small type $Nat : Type_1$ with two constructors $0 : Nat$ and $S : Nat \to Nat$, as well as two primitive functions $\text{pred} : Nat \to Nat$ and $\text{null} : Nat \to Prop$ with the computational rules

$$\text{pred}(0) \to_\beta 0 \quad \text{pred}(S(M)) \to_\beta M \quad \text{null}(0) \to_\beta \top \quad \text{null}(S(M)) \to_\beta \bot$$

from which we easily derive that $S$ is injective, and that $S(n) \neq 0$ for any $n : Nat$ (where $\neq$ denotes the negation of Leibniz equality). In this framework, the induction principle comes for free provided we restrict all the quantifications with the predicate $\text{wf} \_\text{nat}$ defined by

$$\text{wf} \_\text{nat}(n) \triangleq \forall P : (Nat \to Prop). \ P(0) \Rightarrow (\forall p : Nat. \ P(p) \Rightarrow P(S(p)))) \Rightarrow P(n).$$

Using this, it is then easy to build a pointed graph that represents the set $\omega$ of von Neumann numerals, so that:

---

4 A simple counter-model is the obvious extension of the finitary boolean model of Church’s theory of simple types to system $F\omega.2$, in which small types are interpreted by hereditarily finite sets whereas large types are interpreted by the elements of a fixed set-theoretical universe.
Proposition 3 (Infinity) — The translation of the axiom of infinity is provable in system $F\omega.2$ extended with primitive numerals.

3.3 Beyond Zermelo

It is natural to ask whether the former soundness result can be extended to IZF, which is obtained by adding the *collection scheme* to IZ:

$$(\text{Coll}) \quad \forall a \ (\forall x \in a \ \exists y \ \phi) \ \Rightarrow \ \exists b \ \forall x \in a \ \exists y \in b \ \phi$$

(where $\phi$ is an arbitrary formula such that $b \notin FV(\phi)$).

Unfortunately, the answer is negative, since the soundness of (Coll) would entail the relative consistency of ZF w.r.t. $F\omega.2$ with primitive numerals (via Gödel’s negation translation, which maps provable formulas of ZF to provable formulas of IZF). On the other hand, our type system can be seen as a subsystem of the calculus of constructions with universes whose strong normalisation property (and logical consistency) has been proved in ZF [10], so that its proof-theoretical strength is actually less than the one of ZF$^5$.

4 The Normalisation Model $\mathcal{M}$

This section is devoted to the construction of a strong normalisation model $\mathcal{M}$ of $F\omega.2$. The main interest of such a model is not only that it constitutes the main device for proving the strong normalisation property of system $F\omega.2$ but that it naturally validates more propositions than the syntax, and thus suggests extensions of it.

4.1 Interpreting Object-Terms

The model $\mathcal{M}$ is defined in classical set theory with axiom of choice (ZFC). To interpret type-theoretical universes, we also assume the following axiom:

Axiom 4 — There exists two nested ZF-universes.

By ZF-universe (or set-theoretical universe), we mean any transitive set $\mathfrak{U}$ that fulfils the following conditions:

1. $A \in \mathfrak{U} \ \Rightarrow \ \Psi(A) \in \mathfrak{U}$
2. $A \in \mathfrak{U} \land (B_x)_{x \in A} \in \mathfrak{U}^A \ \Rightarrow \ (\bigcup_{x \in A} B_x) \in \mathfrak{U}$
3. $\omega \in \mathfrak{U}$

ZF-universes are closed under all the operations that can be defined in ZFC, among which the generalised cartesian product that will be used to interpret dependent products:

$A \in \mathfrak{U} \land (B_x)_{x \in A} \in \mathfrak{U}^A \ \Rightarrow \ \left( \prod_{x \in A} B_x \right) \in \mathfrak{U}$.

$^5$ We conjecture that system $F\omega.2$ with primitive numerals has the same proof-theoretical strength as higher-order Zermelo’s set theory.
In the following, we assume that $\mathcal{U}_1$ and $\mathcal{U}_2$ are two ZF-universes such that $\mathcal{U}_1 \subseteq \mathcal{U}_2$. Since we want to interpret any signature, we must prohibit empty types by setting $\{ \text{Type}_i \} = \mathcal{U}_i \setminus \{ \emptyset \}$ for $i \in \{ 1; 2 \}$. (Notice that these sets are still closed under generalised cartesian products, thanks to the axiom of choice.)

Propositions are interpreted as saturated sets [10]. The set of all saturated sets, denoted by $\mathfrak{SA}_\Sigma$, is closed under arbitrary (non-empty) unions and intersections, and forms a complete lattice whose top element is $SN$, and whose bottom element is the set $Neut$ of neutral terms. Moreover, $\mathfrak{SA}_\Sigma$ is closed under the construction $S \rightarrow T$ defined by

$$S \rightarrow T \triangleq \{ t \in A; \forall u \in S \; tu \in T \} \in \mathfrak{SA}_\Sigma.$$ 

Let $M = \mathcal{U}_2$. A valuation is a function $\rho : \mathcal{V} \rightarrow M$, where $\mathcal{V}$ denotes the set of object-variables. The interpretation $(M, \rho) \mapsto [M]_\rho$ is defined by

$$[\text{Type}]_\rho = \mathcal{U}_i \setminus \{ \emptyset \} \quad [\text{Prop}]_\rho = \mathfrak{SA}_\Sigma \quad [x]_\rho = \rho(x)$$

$$[\Pi x : T \cdot U]_\rho = \prod_{v \in [T]_\rho} [U]_{(\rho; x \leftarrow v)} \quad [\lambda x : T \cdot M]_\rho = (v \in [T]_\rho \mapsto [M]_{(\rho; x \leftarrow v)})$$

$$[M(N)]_\rho = [M]_\rho([N]_\rho) \quad [A \Rightarrow B]_\rho = [A]_\rho \Rightarrow [B]_\rho \quad [\forall x : T \cdot A]_\rho = \bigcap_{v \in [T]_\rho} [A]_{(\rho; x \leftarrow v)} \quad [\exists x : T \cdot A]_\rho = \bigcup_{v \in [T]_\rho} [A]_{(\rho; x \leftarrow v)}$$

whereas the constants $\text{Nat}$, $0$, $S$, $\text{pred}$ and $\text{null}$ are interpreted in the obvious way. Notice that the right-hand side of the equation which gives the interpretation of the application may be undefined, so that the function $\rho \mapsto [M]_\rho$ is partial.

Each signature $\Sigma$ is interpreted as the set $[\Sigma]$ of all valuations $\rho$ such that $\rho(x) \in [T]$ for each declaration $(x : T) \in \Sigma$ (i.e. the set of adapted valuations). We finally get by a straightforward induction:

**Proposition 5 (Soundness of typing)** — If $\Sigma \vdash M : T$, then for all $\rho \in [\Sigma]$, the denotations $[M]_\rho$ and $[T]_\rho$ are well-defined, and $[M]_\rho \in [T]_\rho$.

As for any strong normalisation model, our model enjoys the crucial property that the denotation of a (well-formed) type is always inhabited. For this reason, any well-formed signature admits at least an adapted valuation.

### 4.2 The Normalisation Invariant

As well as we interpreted signatures as sets of adapted valuations, each logical context $\Gamma = [\xi_1 : A_1; \ldots; \xi_n : A_n]$ is now interpreted as a set of adapted substitutions. Formally, $[\Gamma]_\rho$ is defined (for a given valuation $\rho$) as the set of all substitutions $\sigma = [\xi_1 := u_1; \ldots; \xi_n := u_n]$ such that $u_i \in [A_i]_\rho$ for all $i = 1..n$.

We can now express the strong normalisation invariant of (Curry-style) system $F\omega.2$ as follows:
Proposition 6 (Strong normalisation invariant) — If \( \langle \Sigma \rangle \Gamma \vdash t : A \) is derivable, then for all \( \rho \in [\Sigma] \) and for all \( \sigma \in [\Gamma]_{\rho} \) one has \( t[\sigma] \in [A]_{\rho} \).

By instantiating this result to an arbitrary adapted valuation \( \rho \in [\Sigma] \) and to the identity substitution (adapted to \( \Gamma \)) we conclude that:

Corollary 7 (Strong normalisation) — In \( \text{F} \omega.2 \), all the typable proof-terms are strongly normalising \( \lambda \)-terms.

Truth in the Model. Although the strong normalisation model \( \mathcal{M} \) is quite close of a realisability model (think of the interpretation of \( \forall, \exists \) and \( \Rightarrow \)), an important difference with the standard realisability approach is that in \( \mathcal{M} \), the denotation of a proposition is never empty, since a saturated set contains at least all the neutral terms. To define a suitable notion of truth in the model, one has to exclude such ‘paraproofs’ by only considering closed terms.

Formally, we will say that a proposition \( A \) is true in the model \( \mathcal{M} \) if its denotation \( [A] \in \mathbb{S}[\Sigma] \) contains at least a closed proof-term. Notice that the proposition \( \bot = \forall p : \text{Prop}.p \) (whose denotation is \( \text{Neut} \)) is false in the model, which shows that system \( \text{F} \omega.2 \) is logically consistent.

4.3 Axiom of Choice and Uniform Collection

An example of a true but non-provable proposition is the following formulation of the axiom of choice in \( \text{F} \omega.2 \):

\[
(\forall x : T . \exists x : U. A(x, y)) \Rightarrow \exists f : T \rightarrow U . \forall x : T. A(x, f(x))
\]

where \( T \) and \( U \) are arbitrary data-types. Although this proposition is not provable in \( \text{F} \omega.2 \), it is straightforward to check that in the model \( \mathcal{M} \), its denotation contains the closed proof-term \( \lambda \xi . \xi \) since both members of the corresponding implication have the very same denotation, namely:

\[
\bigcap_{x \in T} \bigcup_{y \in U} A(x, y) = \bigcup_{f \in U^T} \bigcap_{x \in T} A(x, f(x)).
\]

(In the model of course, the left-to-right inclusion relies on the axiom of choice.)

This fact suggests that we can add the following proof principle

\[
\langle \Sigma \rangle \Gamma \vdash t : \forall x : T. \exists y : U. A(x, y)
\]  

\[
\langle \Sigma \rangle \Gamma \vdash t : \exists f : (T \\rightarrow U) . \forall x : T. A(x, f(x))
\]

to our system without breaking the normalisation invariant.

The Uniform Collection Scheme. Coming back to our translation of set theory, it is interesting to notice that this additional rule is actually sufficient for proving a weak form of collection scheme—that will be called here the uniform collection scheme—whose statement is the following:

\[
(\forall x \ \exists y \ \phi(x, y)) \Rightarrow \forall a \ \exists b \ \forall x \in a \ \exists y \in b \ \phi(x, y)
\]
This statement is weaker than the collection scheme since it relies on a totality assumption which is stronger than that of collection, by requiring that the binary relation \( \phi(x, y) \) should be total not only on \( a \), but on the whole universe. (Of course, the uniform collection is classically equivalent to the collection scheme.)

### 4.4 An Intuitionistic Choice Operator

The difficulty in realising the full collection scheme is that for any \( x \in a \), the implicit witness \( y \) given by the proof of \( \exists y \, \phi(x, y) \) does not only depend on \( x \), but also on the proof of the relativisation \( x \in a \). To overcome this difficulty, we propose to express this dependency by using a trick inspired by Krivine’s interpretation of the denumerable axiom of choice [6, 7].

For that, we extend system \( \text{F}_\omega.2 \) with an intuitionistic choice operator that associates to any predicate \( A(x) \) a sequence of objects \( \epsilon x : T . A(x) : \text{Nat} \rightarrow T \) whose intended meaning is: if \( A \) holds for some \( x : T \), then \( A \) holds for some element of the sequence \( \epsilon x : T . A(x) \).

Formally, the interpretation of this new construction in the model \( M \) relies on a fixed enumeration \( (t_n)_{n \in \omega} \) of all the \( \lambda \)-terms. The construction \( \epsilon x : T . A(x) \) is then interpreted as a function \( f \in T^\omega \) (defined by using the axiom of choice) such that for all \( n \in \omega \), either:

1. \( f(n) \) is some \( x \in T \) such that \( t_n \in A(x) \) if such an element exists; or
2. \( f(n) \) is an arbitrary element of \( T \) otherwise.

Once the function \( f \in T^\omega \) that interprets the construction \( \epsilon x : T . A(x) \) has been defined, it is straightforward to check that \( \bigcup_{x \in T} A(x) \subset \bigcup_{n \in \omega} A(f(n)) \).

(The converse inclusion also holds, but is not of interest.) Coming back to our logical framework, this inclusion means that the model validates the typing rule

\[
\begin{align*}
\langle \Sigma \rangle \Gamma \vdash t : \exists x : T . A(x) \\
\langle \Sigma \rangle \Gamma \vdash t : \exists n : \text{Nat} . A(\epsilon x : T . A(x))(n)
\end{align*}
\]

which can thus be added to our type system without harm for the strong normalisation property.

## 5 The Extended Framework \( \text{F}_\omega.2^{++} \)

Using the material we presented above, we can now define an extended Curry-style framework called \( \text{F}_\omega.2^{++} \) that actually contains enough proof principles to allow the (translation of the) collection scheme to be derived.

### 5.1 Syntax and Typing Rules

**Object-Terms and Signatures.** The syntax and typing rules of object-terms and signatures of system \( \text{F}_\omega.2^{++} \) are the one of system \( \text{F}_\omega.2 \) extended with primitive numerals and the intuitionistic choice operator discussed in paragraph 4.4 (see Fig. 3). The corresponding reduction rules are the usual \( \beta \)-rule and the reduction rules of the constants \( \text{pred} \) and \( \text{null} \) that we gave in paragraph 3.2.
Judgements $\Sigma \vdash$, $\Sigma \vdash M : T$ and $\Sigma \vdash \Gamma \ ctx$

Rules of Fig. 1 p. 443 + constants $\text{Nat}$, $0$, $S$, $\text{pred}$, $\text{null}$

$$\Sigma; [x : T] \vdash A : \text{Prop}$$

$$\Sigma \vdash \epsilon x : T . A : \text{Nat} \rightarrow T$$

Propositional subtyping: $\Sigma \vdash A \leq B$

$$\Sigma \vdash A : \text{Prop} \quad \Sigma \vdash A' : \text{Prop} \quad \Sigma \vdash A \leq A'$$

$$\Sigma \vdash A \leq B \quad \Sigma \vdash B \leq C \quad \Sigma \vdash A \leq C$$

$$\Sigma; [x : T] \vdash A \leq B \quad \Sigma \vdash \forall x : T . B \quad x \notin \text{FV}(A)$$

$$\Sigma; [x : T] \vdash A : \text{Prop} \quad \Sigma \vdash N : T \quad \Sigma \vdash A \leq B \quad \Sigma \vdash \exists x : T . A \leq B \quad x \notin \text{FV}(B)$$

$$\Sigma; [x : T] \vdash A : \text{Prop} \quad \Sigma \vdash B : \text{Prop} \quad \Sigma \vdash \forall x : T . (A \Rightarrow B) \leq (\exists x : T . A) \Rightarrow B$$

$$\Sigma; [x : T ; y : U] \vdash A : \text{Prop} \quad \Sigma \vdash \forall x : T . \exists y : U . A \leq \exists f : (\Pi x : T . U) . \forall x : T . A(y := f(x))$$

$$\Sigma; [x : T] \vdash A : \text{Prop} \quad \Sigma \vdash \exists x : T . A \leq \exists n : \text{Nat} . A(x := (\epsilon x : T . A)(n))$$

Typing rules of proofs-terms: $\langle \Sigma \rangle \Gamma \vdash t : A$

$$\langle \Sigma \rangle \Gamma \vdash \ctx \quad (\xi : A) \in \Gamma \quad \langle \Sigma \rangle \Gamma \vdash [\xi : A] \vdash t : B \quad \langle \Sigma \rangle \Gamma \vdash \lambda \xi . t : A \Rightarrow B$$

$$\langle \Sigma \rangle \Gamma \vdash t : A \Rightarrow B \quad \langle \Sigma \rangle \Gamma \vdash u : A \quad \langle \Sigma \rangle \Gamma \vdash tu : B$$

$$\langle \Sigma ; [x : T] \rangle \Gamma \vdash t : A \quad x \notin \text{FV}(\Gamma) \quad \langle \Sigma \rangle \Gamma \vdash t : A \quad \Sigma \vdash A \leq A'$$

Fig. 3. Rules of inference of system $F_{\omega . 2^{++}}$.  

452  Alexandre Miquel
Logical Contexts and Proof-Terms. Logical contexts have the same syntax and formation rules as in system $F\omega.2$, and proof-terms are still pure $\lambda$-terms.

The main novelty of system $F\omega.2^{++}$ is the introduction of a new form of judgement called propositional subtyping and written $\Sigma \vdash A \leq B$. This judgement—whose logical meaning is a direct implication—is intended to capture the different inclusions between saturated sets that we pointed out throughout section 4. In particular, this judgement (whose rules of inference are given in Fig. 3) now incorporates the introduction and elimination rules for both quantifiers, as well as a very natural rule that expresses the contravariance of the domain of implication (and the covariance of its codomain) w.r.t. subtyping. The last two subtyping rules express both intuitionistic forms of axiom of choice discussed in paragraphs 4.3 and 4.4. (Notice that the latter is actually sufficient to derive the collection scheme.)

In this framework, the proof-typing rules become simpler than in system $F\omega.2$, for that many logical rules have been incorporated in the propositional subtyping judgement, and are now accessed via a standard subsumption rule.

5.2 Strong Normalisation

From the results of section 4, it is clear that $M$ is still a strong normalisation model for $F\omega.2^{++}$. The proofs of propositions 5 and 6 are easily adapted to system $F\omega.2^{++}$ by interpreting the new construction $\epsilon x : T . A$ as explained in 4.4. Notice that in order to prove the strong normalisation invariant for system $F\omega.2^{++}$, we first have to check the soundness of propositional subtyping:

Proposition 8 — If $\Sigma \vdash A \leq B$, then for all $\rho \in [\Sigma]$ one has $[A]_\rho \subset [B]_\rho$.

From this, we easily deduce that all the well-formed proof-terms of system $F\omega.2^{++}$ are strongly normalising, and that the system is logically consistent.

5.3 Deriving the Collection Scheme

Of course, the main interest of using system $F\omega.2^{++}$ is that we can now realise the collection scheme, by the means of the intuitionistic choice operator:

Proposition 9 (Collection scheme) — The translation of each instance of (COLL) is provable in system $F\omega.2^{++}$.

Notice that this result entails that the proof-theoretical strength of system $F\omega.2^{++}$ is at least the one of IZF/ZF.

5.4 Extracting Functions from Proofs

In this paragraph, we aim to show that from any proof (in IZF) of a statement $\phi$ of the form

$$\phi \equiv \forall x \in \omega \exists y \in \omega \psi(x, y)$$
(where $\psi$ is an arbitrary formula s.t. $FV(\psi) = \{x; y\}$) we can extract a strongly normalising $\lambda$-term that computes the corresponding function.

The extraction process relies on the fact that, internally, the set $\omega$ of von Neumann numerals is implemented [11] as a pointed graph $(X, A, a)$ equipped with an injection $i : \text{Nat} \rightarrow X$ that associates to any object $n : \text{Nat}$ such that $\text{wf} \text{nat}(n)$ the vertex $i(n) : X$ which represents the corresponding von Neumann numeral in the graph $(X, A)$. Using this, it is easy to derive from a proof of $\phi$ (in IZF) a proof-term (in $F\omega.2^{++}$) of the statement:

$$\forall n : \text{Nat}. \text{wf} \text{nat}(n) \Rightarrow \exists p : \text{Nat}. \text{wf} \text{nat}(p) \wedge \phi^*(X, A, i(n), X, A, i(p))$$

(where $\psi^*$ denotes the translation of the binary relation $\psi$ in $F\omega.2^{++}$). By dropping the second component of the conjunction, we thus get a proof-term

$$\lambda\xi. t'(\lambda\xi_1, \xi_2. \xi_1) : (\forall n : \text{Nat}. \text{wf} \text{nat}(n)) \Rightarrow (\exists p : \text{Nat}. \text{wf} \text{nat}(p))$$

that obviously computes the desired function, since:

**Fact 10** — The closed inhabitants of the saturated set $[[\text{wf} \text{nat}(x)]_{x \leftarrow n}$ (for a given $n \in \omega$) are the SN-terms whose normal form is Church numeral $\lceil n \rceil$.

**References**

Hilbert’s ε-calculus [12] is based on an extension of the language of predicate logic by a term-forming operator $\varepsilon_x$. This operator is governed by the critical axiom

$$A(t) \rightarrow A(\varepsilon_x A(x)),$$

where $t$ is an arbitrary term. Within the ε-calculus, quantifiers become definable by $\exists x A(x) \Leftrightarrow A(\varepsilon_x A(x))$ and $\forall x A(x) \Leftrightarrow A(\varepsilon_x \neg A(x))$. (The expression $\varepsilon_x A(x)$ is called an ε-term.)

The ε-calculus was developed in the context of Hilbert’s program of consistency proofs. It is of independent interest, however, and a study from a computational and proof-theoretic point of view is worthwhile. As a simple example of the advantages of the ε-calculus note that by encoding quantifiers on the term-level, formalizing (informal) proofs is sometimes easier in the ε-calculus, compared to formalizing proofs in, e.g., sequent calculi.

In this tutorial we will restrict our attention to predicate logic. After formally introducing the relevant concepts we present some conservativity results for the ε-calculus. In particular we will prove the first and second ε-theorem.

Let $T$ denote a finitely axiomatized open theory with axioms $P_1, \ldots, P_t$ based on the ε-calculus. We suppose some usual formalization $S$ of predicate logic, e.g. Gentzen’s LK. Then the first ε-theorem expresses that any formula (without quantified variables) deducible in $T$ is also deducible from $P_1, \ldots, P_t$ in the quantifier-free fragment of $S$. The second ε-theorem expresses that any formula (free of ε-terms) deducible in $T$ is also deducible from $P_1, \ldots, P_t$ in $S$.

As an easy consequence of the first ε-theorem one obtains: Any consequence $F$ of a finitely axiomatized open theory $T$, where $F$ is quantifier-free is provable from the axioms of $T$ by the quantifier-free fragment of predicate logic. Another consequence is Herbrand’s theorem, while the converse of Herbrand’s theorem follows from the second ε-theorem.

References


* Supported by a Marie Curie fellowship, grant number HPMF-CT-2002-015777
Modular Semantics and Logics of Classes

Bernhard Reus*

School of Cognitive and Computing Sciences
University of Sussex
bernhard@cogs.susx.ac.uk

Abstract. The semantics of class-based languages can be defined in terms of objects only [1, 7, 8] if classes are viewed as objects with a constructor method. One obtains a store in which method closures are held together with field values. Such a store is also called “higher-order” and does not come for free [13]. It is much harder to prove properties of such stores and as a consequence (soundness of) programming logics can become rather contrived (see [2]).

A simpler semantics separates methods from the object store [4, 12]. But again, there is a drawback. Once the semantics of a package of classes is computed it is impossible to add other classes in a compositional way. Modular reasoning principles are therefore not obtainable either.

In this paper we improve a simple class-based semantics to deal with extensions compositionally and derive modular reasoning principles for a logic of classes. The domain theoretic reasoning principle behind this is fixpoint induction.

Modularity is obtained by endowing the denotations of classes with an additional parameter that accounts for those classes added “later at linkage time.”

Local class definitions (inner classes) are possible but for dynamic classloading one cannot do without higher-order store.

1 Introduction and Motivation

In our quest for a complete denotational understanding of the semantics and logics of object-oriented programming languages we have treated object-based languages [13] in terms of Abadi and Cardelli’s object calculus [1] and class-based languages [12] in terms of a simple sublanguage of sequential Java.

Class-based languages use the class concept to describe a collection of objects that are the instances of the class. In [8] two denotational models for various languages (including classes and inheritance) are given. One uses fixpoint closures, the other one self-application but in both models objects contain fields and methods (method closures). This leads to what is called “higher-order store”, the domain of which is defined by a mixed variant equation similar to the one below:

\[ \text{St} = \text{Rec}_{\text{Loc}}(\text{Rec}_{\mathcal{F}}\text{Val}) \times \text{Rec}_{\mathcal{M}}(\text{St} \rightarrow \text{Val} \times \text{St}). \]

* partially supported by the EPSRC under grant GR/R65190/01 and by the Nuffield Foundation under grant NAL/00244/A

© Springer-Verlag Berlin Heidelberg 2003
For such stores invariance properties are hard to show.

A simpler approach aims at separating methods from objects. Methods do not have to be stored in the heap but are gathered in an environment that contains a method suite for each class. The class name acts as a link between the objects in the store and the method suites and is used for method dispatch. There are several variations of such a denotational semantics in the literature, [3, 4, 12], but in all of them only a fixed set of classes can be interpreted. Logics based on operational semantics which address these problems were presented in [11, 14]. A denotational semantics, however, provides easier handling of mutual recursive modules as well as better tools for analysing and extending the language and logic. It also allows us to study the differences between logics for languages with higher-order store (object-based languages) and class-based languages more thoroughly.

In this paper we introduce a denotational semantics that is “open” with respect to extensions of classes and thus allows for a modular programming logic. The main contributions are:

- modular (compositional) denotational semantics for classes without higher-order store
- modular logic for classes (in terms of denotations)
- semantics for local classes
- explanation of modularity in class-based languages and relationship w.r.t. object-based languages.

Correctness of programs is usually shown using Hoare-like calculi. But it is the design, analysis, and soundness proof of these calculi that can be simplified using denotational semantics.

In the next section we briefly present the syntax of a simple class-based language. Then a (standard) denotational semantics is provided. Section 4 improves this semantics to obtain modularity and compositionality. Finally, modular proof rules are discussed. The paper end with a short summary and outlook (Section 6).

2 Syntax

The syntax and semantics of the class-based language is a refinement of the one in [12]. Let \( \mathcal{F} \) be the set of field names, \( \mathcal{M} \) the set of method names, and \( \mathcal{C} \) the set of class names. Usually, \( f \) stands for a field name, \( m, n \) stand for method names, and \( C, D \) for class names.

\[
\begin{align*}
a, b ::= & \quad x \quad \text{variables} \\
| & \quad \text{new } C() \quad \text{object creation} \\
| & \quad a.f \quad \text{field selection} \\
| & \quad a.f = b \quad \text{field update} \\
| & \quad a.m() \quad \text{method call} \\
| & \quad \text{let } x=a \text{ in } b \quad \text{local def.}
\end{align*}
\]
The “self” reference \textbf{this} is just considered a special (predefined) variable name, therefore \textbf{this} \in \Var. Note that we do not distinguish between expressions and statements (like in the object calculus). There is no extra operator for sequential composition as it can be expressed using \texttt{let}, i.e. \texttt{a;b} \equiv \texttt{let } x = a \texttt{ in } b \texttt{ where } x \texttt{ does not occur freely in } a \texttt{ and } b. Methods always have to return a result.

Additionally, there is a syntax for class definitions:

\[
c ::= \texttt{class } C \{ [\texttt{inherits } D] \begin{array}{l} f_i = 1,\ldots,n \\ m_j = b_j \quad j = 1,\ldots,m \\ C() = b \end{array} \}
\]

\[
eC ::= \epsilon \mid c . eC
\]

A class definition is similar to one in Java with all type declarations omitted. It contains a list of field declarations, a list of method declarations and a constructor. For the sake of simplicity all functions and the constructor are nullary and shadowing of field variables by inheritance is disallowed, i.e. it is not possible to declare a field in a subclass that was already declared in a superclass\footnote{A more sophisticated treatment shadowing the duplicate fields of the superclass as used in Java can be modelled by adding the duplicate fields with appropriately changed field names.}.

3 Denotational Semantics

The denotational semantics is supposed to live in the category of predomains – i.e. complete partially ordered sets – and continuous maps. Equivalently one can work in a category with domains (that have a least element) and strict, continuous maps. Note that the fixpoint operator fix is only defined on domains with least element.

Indexed collections, like stores and method suites, can be nicely modelled as records.

3.1 Record Types

Let \mathbb{L} be a (countable) set of labels and \mathcal{A} a predomain then the type of records with entries from \mathcal{A} and labels from \mathbb{L} is defined as follows:

\[
\text{Rec}_{\mathbb{L}}(\mathcal{A}) = \Sigma_{L \in \mathcal{P}\text{fin}(\mathbb{L})} \mathcal{A}^L
\]

where \mathcal{A}^L is the set of all total functions from \mathbb{L} to \mathcal{A}. It is easily seen that \text{Rec}_{\mathbb{L}} is a locally continuous functor on \text{PreDom}. A record with labels \ell_i and corresponding entries \nu_i (1 \leq i \leq k) is written \{\ell_1 = \nu_1, \ldots, \ell_k = \nu_k\}. Notice that \text{Rec}_{\mathbb{L}}(\mathcal{A}) is always non-empty as it contains the element \langle \emptyset, \emptyset \rangle. Records are ordered as follows:

\[
\langle L_1, f_1 \rangle \sqsubseteq \langle L_2, f_2 \rangle \iff L_1 = L_2 \land \forall \ell \in L_1. f_1(\ell) \sqsubseteq_A f_2(\ell)
\]
Therefore, a record and its extension are incomparable. Note that records do not have a least element, only minimal ones. This can be remedied by lifting the record type to attach a new least element, usually called $\bot$: $\text{Rec}_L(A)$ is a complete poset if $A$ is so, thus the lifted $\text{Rec}_L(A)_\bot$ is always a domain.

Basic record operations like selection, update, extension, and deletion are defined below.

**Definition 1.** Let $r \in \text{Rec}_L(A)$ such that $r = \langle L, f \rangle$ with $L \subseteq L$, and $f \in A^L$. Definedness of label $l$ in record $r$ is as follows: $l \in \text{dom} \ r \iff l \in L$.

Selection of a label $l \in L$ in record $r$, short $r.l$, is defined if $l \in \text{dom} \ r$ and yields $f(l) \in A$.

An update function for records is defined in Table 1. It is undefined for labels which do not appear in the argument record.

| $\{ l_i = f_i \}_{i=1}^n$ | undefined if $l \not\in \text{dom} \{ l_i = f_i \}_{i=1}^n$ | $l_i = f_i$ if $l = l_i$ |
| $\{ l_1 = f_1, \ldots, l_i = f, \ldots, l_n = f_n \}$ | |

If $r_1 = \langle L_1, f_1 \rangle$ and $r_2 = \langle L_2, f_2 \rangle$ are two records in $\text{Rec}_L(A)$ then the extension $r_1 + r_2$ is defined as follows:

$$\langle L_1, f_1 \rangle + \langle L_2, f_2 \rangle = \langle L_1 \cup L_2, \lambda l. \text{if } l \in L_1 \text{ then } f_1(l) \text{ else } f_2(l) \rangle$$

Note that in $r_1 + r_2$ the values of the fields in $r_2$ which are also used in $r_1$ are lost.

Finally, if $r = \langle L, f \rangle$ and $l \in L$, then the deletion operation which erases $l$ in $r$, $r \setminus l$, is defined as follows:

$$r \setminus l = \begin{cases} r & \text{if } l \not\in \text{dom} \ r \\ \langle L \setminus l, f \rangle & \text{otherwise} \end{cases}$$

For a nested record $r \in \text{Rec}_L(\text{Rec}_L(A))$ we abbreviate $r[l_1 := r.l_1[l_2 := a]]$ by the simpler and more intuitive $r[l_1.l_2 := a]$.

### 3.2 Semantic Domains

We assume flat predomains for basic values (like booleans or integers) $\text{Val}$ and a flat predomain of locations $\text{Loc}$.

The class-based language introduced above finds its interpretation within the following non-recursive system of domains:
Definition 2. Let the semantic counterparts of objects (1), stores (2), closures (3), method suites (4), and class descriptors (5) be defined as below.

\[
\begin{align*}
\text{Ob} &= \text{Rec}_F(\text{Val}) \times \mathcal{C} \\
\text{St} &= \text{Rec}_{\text{Loc}}(\text{Ob}) \\
\text{Cl} &= \text{Loc} \times \text{St} \to \text{Val} \times \text{St} \\
\text{Ms} &= \text{Rec}_\mathcal{M}(\text{Cl}) \times (\text{Loc} \times \text{St} \to \text{St}) \times \mathcal{C} \\
\text{Mss} &= \text{Rec}_\mathcal{C}(\text{Ms})
\end{align*}
\]

An object consists of a record of field values plus the name of its class type. This information is stored in order to be able to do the dynamic dispatch for method calls (due to subtype polymorphism). A store is a record of objects indexed by locations. A closure is a partial function mapping a location, representing the callee object, and (the old) store to a value, the result of the method, and a new store which accounts for the side effects during execution of the method in question. A method suite (Ms) is a record \(\text{Rec}_\mathcal{M}(\text{Cl}) \times (\text{Loc} \times \text{St} \to \text{St}) \times \mathcal{C}\) which contains all methods of a class, the initialisation code for the constructor, and the name of the superclass. We assume that there is always a superclass, i.e. there is a root class \(\text{Root}\) that all classes inherit from\(^2\). The constructor takes a location in which to create the object. This reference is created freshly by the semantics of \text{new} and then passed to the constructors of the superclass. A class descriptor (Mss) is the collection of method suites for all declared classes. The function type : Ob \to \mathcal{C} returns an object’s class, i.e. type(o) = o.2, which is the object’s dynamic type. For interpreting variables we need an environment \(\text{Env}\) which maps variables to values, i.e. Env = Var \to Val.

3.3 Semantic Equations

The interpretation of the syntax is given by the following semantic equations:

Definition 3. Given an environment \(\rho \in \text{Env}\), an environment of method suites \(\mu \in \text{Mss}\) and an expression \(a\), its interpretation \([a] \rho : \text{Mss} \to \text{St} \to \text{Val} \times \text{St}\) is defined in Table 2 (page 461). Note that the semantic let on the right hand side of the definitions is strict in its first argument.

Given an environment of method suites \(\mu \in \text{Mss}\) a class definition \(c\) is defined via \([c] : \text{Mss} \to \text{Ms}\) in Table 3 (page 461).

The methods of the superclass are copied into the method suite of the subclass. It is more memory efficient not to copy them and use a more sophisticated method dispatch instead that searches in super classes. Finally, since we work in an untyped setting, it is legitimate to initialize any field with zero. In a typed setting zero would be replaced by a default value for the type of the field. It is no contradiction to the untypedness of our semantics to have classes (class names) stored in objects as these classes represent runtime types needed for dynamic dispatch. There are no static types in use.

\(^2\) In Java this would be Object.
The semantics of these two classes according to the above are:

\[
\begin{align*}
\text{class } C & \text{ inherits } \text{Top} \{ \\
& \text{class } \text{Top} = \{ \\
& \quad f \\
& \quad m() = \text{this}.f.n() \\
& \quad n() = 0 \\
& \quad \text{Top}() = \text{this}.f = \text{this} \\
& \} \\
& \text{class } C \text{ inherits } \text{Top} \{ \\
& \quad n() = 1 \\
& \quad C() = \text{this} = \text{this} \quad \text{/* skip */} \\
& \} \\
\end{align*}
\]

The semantics of a module class \( C \ldots \) could be defined as

\[
\text{fix } \mu : \text{Mss}_\perp. \{ C = [\text{class } C \ldots ] \mu \}.
\]

Fixpoint induction would then provide the corresponding proof principle. But this definition does not give the desired result if classes depend on other classes. An example is used below to emphasize the problem:

\[
\begin{align*}
[x] \rho \mu \sigma & = \langle \rho(x), \sigma \rangle \\
[\text{this}] \rho \mu \sigma & = \langle \rho(\text{this}), \sigma \rangle \\
[\text{new } C()] \rho \mu \sigma & = \langle \ell, \mu.C.2(\ell, \sigma)[\ell.2 = C] \rangle \text{ where } \ell \not\in \text{dom } \sigma \\
[a.f] \rho \mu \sigma & = \text{let } \langle \ell, \sigma' \rangle = [a] \rho \mu \sigma \text{ in } \langle \sigma'.\ell.1.f, \sigma' \rangle \\
[a.f = b] \rho \mu \sigma & = \text{let } \langle \ell, \sigma' \rangle = [a] \rho \mu \sigma \text{ in } \langle b, \rho(\ell/x) \mu \sigma' \rangle \\
[a.m()] \rho \mu \sigma & = \text{let } \langle \ell, \sigma' \rangle = [a] \rho \mu \sigma \text{ in } \mu.\text{type}(\sigma').\ell.1.m(\ell, \sigma') \\
[\text{let } x = a \text{ in } b] \rho \mu \sigma & = \text{let } \langle \ell, \sigma' \rangle = [a] \rho \mu \sigma \text{ in } [b] \rho(\ell/x) \mu \sigma'.
\end{align*}
\]
If we join the two classes into one class descriptor \( \mu \) defined as follows

\[
\mu = \{ \text{Top = [class Top ...]} \} + \{ \text{C = [class C inherits Top ...]} \}
\]

and interpret the following expression \( t \)

```plaintext
let o = new Top()
in let _ = o.f = new C()
in o.m()
```

w.r.t. \( \mu \) we obtain:

\[
[\ell] \mu \sigma = \begin{cases}
\sigma_1 = \sigma[\ell := \langle \{ f = \ell \}, \text{Top} \}] & \text{in}
\sigma_2 = \mu.\text{Top.2}(\ell', \sigma_1)[\ell'.2 := \text{C}] & \text{in}
\sigma_3 = \sigma_2[\ell.f := \ell'] & \text{in}
\mu.\text{type}(\sigma_3.\ell).1.m(\ell, \sigma_3)
\end{cases}
\]

where \( \ell \) is fresh in \( \sigma \) and \( \ell' \) is fresh in \( \sigma_1 \). This simplifies to

```plaintext
let \sigma_4 = \sigma[\ell := \langle \{ f = \ell' \}, \text{Top} \}, \ell' := \langle \{ f := \ell' \}, \text{C} \}] \text{ in } \mu.\text{Top.1.m}(\ell, \sigma_4)
```

which by the semantics of class \text{Top} simplifies to

\[
\mu.\text{type}(\sigma_4.\ell.f).1.n(\sigma_4.\ell.f, \sigma_4)
\]

which in turn evaluates to

\[
\mu.\text{C.1.n}(\ell', \sigma_4)
\]

Obviously, the result depends on the semantics of the method \( n \) in \( \text{C} \), i.e. \( \mu.\text{C.1.n} \). But the fixpoint operation yields no defined result for \( \mu.\text{C} \), just one for \( \mu.\text{Top} \) and thus the result is undefined whereas it should be 1. The reason for this unexpected behaviour is that the fixpoint has closed the method suites available. It cannot be updated to contain the methods of other classes added later.

To avoid this problem we define the semantics of classes slightly differently, in a parameterized way.

4 A Modular Semantics for Classes

Modules depend on a context of class declarations unknown at the time of definition. Later they may be linked together with this context changing the semantics of the original modules. This effect will be modelled by (mutual) fixpoints. Recall that records defined by fixpoints can not get bigger (because of their invariance under extensions) but the values of their fields can become more defined. This is enough for our purposes, since in module or package definitions the number of methods and fields is fixed. Nevertheless, the record extension operator is needed to express a temporarily growing number of classes.
Definition 4. We define a polymorphic operator maprec that applies a function to all components of a record: maprec\(_A\): (A \rightarrow B) \rightarrow \text{Rec}_L(A) \rightarrow \text{Rec}_L(B) as follows:

\[ \text{maprec } f (\langle D, g \rangle) = \langle D, f \circ g \rangle \]

which is in analogy with map for lists.

A modular class definition has the following semantics taking into account further packaging with other classes:

Definition 5. The semantics \([\_]^m: \text{Mss} \rightarrow \text{Ms} \) is defined as

\[ [\text{class } C \ldots]^m = \lambda \mu : \text{Mss}. \text{fix } \tau : \text{Ms} \downarrow. [\text{class } C \ldots] \left(\{C = \tau\} + \mu \right) \]

where the parameter \(\mu\) represents the possible additional classes \(C\) may depend on.

Even if \(C\) is not intended to refer to any other class, in an open world it must reserve the right to refer to future subclasses of itself.

A “package” or “linkage” operator is needed that links a module with a given package, i.e., a list of linked (or packed) modules. Like modules packages have an additional parameter in order to be open to future extensions. In order to accomplish this, we define the type

\[ \text{CTs} = \text{Rec}_C(\text{Mss} \rightarrow \text{Ms}) \]

which represents collections of class transformers of type \(\text{Mss} \rightarrow \text{Ms}\) that describe the semantics of a class depending on a class descriptor for the context.

Definition 6. The package operator

\[ \text{pack}_C : (\text{Mss} \rightarrow \text{Ms}) \times \text{CTs} \rightarrow \text{CTs} \downarrow \]

is defined as follows:

\[ \text{pack}_C(M, P) = \text{fix } \delta : \text{CTs} \downarrow. \]

\[ \{C = \lambda \mu : \text{Mss}. M(\delta(\mu) \setminus C + \mu)\} + \]

\[ \text{maprec } (\lambda x : \text{Mss} \rightarrow \text{Ms}. \lambda \mu : \text{Mss}. x(\delta(\mu).C + \mu)) P. \]

Definition 7. The semantics of a package of class definitions can be defined as follows:

\[ [\text{class } C \ldots \text{cl}] = \text{pack}_C([\text{class } C \ldots]^m, [\text{cl}]) \]

and \([\epsilon] = \{\}\).

Lemma 1. Packaging with nothing does not have any effect, i.e.

\[ \text{pack}([\text{class } C = \ldots]^m, \emptyset) = \{C = [\text{class } C = \ldots]^m\} \]
Proof. Let \( M = \mathbf{class} C = \ldots \)^m and assume \( \delta = \{ C = M \} \) (1). If we can show that
\[
\{ C = \lambda \mu : \text{Mss.} \ [\mathbf{class} C = \ldots ]^m (\delta(\mu) \setminus C + \mu) \} = \{ C = M \}
\]
then by fixpoint induction we have shown
\[
\fix \delta : \text{CTs}_\bot. \{ C = \lambda \mu : \text{Mss.} \ [\mathbf{class} C = \ldots ]^m (\delta(\mu) \setminus C + \mu) \} = \{ C = M \}
\]
which implies the claim.
\[
\{ C = \lambda \mu : \text{Mss.} \ [\mathbf{class} C = \ldots ]^m (\delta(\mu) \setminus C + \mu) \} =_{(1)}
\]
\[
\{ C = \lambda \mu : \text{Mss.} \ [\mathbf{class} C = \ldots ]^m (\{ C = M(\mu) \} \setminus C + \mu) \} =
\]
\[
\{ C = \lambda \mu : \text{Mss.} \ [\mathbf{class} C = \ldots ]^m \mu \} =
\]
\[
\{ C = M \} .
\]

One can always “close” a package or module by just applying it to the empty environment.

4.1 Inner Classes

Inner classes are classes defined “on-the-fly” inside classes or methods. Simple class descriptors are not sufficient as argument to the interpretation function since packaging needs transformers. The type of the interpretation function thus changes to

\[
\rho : \text{CTs} \rightarrow \text{St} \rightarrow \text{Val} \times \text{St} .
\]

If “\( \text{let class} \ C \ldots \text{in} \ s \)” denotes the syntax for local inner class declarations the interpretation is as follows:

\[
[\text{let class} \ C \ldots \text{in} \ s] \rho \delta \sigma = [s] \rho \text{pack}_C([\mathbf{class} C \ldots ]^m, \delta) \sigma
\]

Whenever the interpretation really needs the method environment – ie. when evaluating a method call or creating an object – the environment can be closed for this moment of time to find the right closure:

\[
[\text{new} \ C()] \rho \delta \sigma = \langle \ell, (\delta.C \{\} \_2(\ell, \sigma)\_2(\ell, 2 : = C)\rangle
\]

where \( \ell \notin \text{dom} \sigma \)

\[
[a.m()] \rho \delta \sigma = \text{let} \ (\ell, \sigma') = [a] \rho \delta \sigma \text{ in } (\delta.\text{type}(\sigma', \ell) \{\} \_1.\text{m}(\ell, \sigma')
\]

The corresponding change to method store transformers means that we have to change the semantics of the class modules slightly:

\[
[\mathbf{class} C \ldots ]^m = \lambda \mu : \text{Mss.} \fix \tau : \text{Ms}_\bot. \ [\mathbf{class} C \ldots ]
\]
\[
\{ C = \lambda _\tau : \text{Mss.} \tau \} + \text{maprec} (\lambda x : \text{Ms.} \lambda _\tau : \text{Mss.} x) \mu
\]

5 Logics of Programs

Having fixed the denotational semantics of classes we can start reasoning about denotations. This is in analogy with the LCF (Logic of Computable Functions) project (see e.g. [10]) for the functional paradigm.
5.1 Specifications

First we have to define an appropriate notion of specification for classes following [6, 11, 14]. Compare also with [12].

As every class contains a number of methods, some common structures in specifications can be singled out: for every method there is a result specification and a transition specification.

\[ B_m \in \wp(Val \times St) \text{ result specification for } m \in M \]
\[ T_m \in \wp(Loc \times St \times Val \times St) \text{ transition specification for } m \in M \]

Taking into account that we are only interested in partial correctness (at least for the considerations of this paper) the meaning of the specifications can be described informally as follows:

\[ B_m : \text{"If method } m \text{ terminates its result fulfils the result specification."} \]
\[ T_m : \text{"If method } m \text{ terminates its effect fulfils the transition specification."} \]

A modular specification may depend on the specification of some other classes. The existence of other classes in the environment may indeed be part of the specification itself.

**Definition 8.** Specification building operators for methods (\(mth\)), classes (\(cls\)), and packages (\(pck\)) are defined below. For the sake of readability we abbreviate the type of result specifications \(RSpec = \wp(Val \times St)\) and transition specifications \(TSpec = \wp(Loc \times St \times Val \times St)\).

\[
mth : RSpec \times TSpec \rightarrow \wp(C \times Cl).
\]
\[
cls : \text{Rec}_M(RSpec \times TSpec) \rightarrow \wp(C \times Ms).
\]
\[
pck : \text{Rec}_C(\text{Rec}_M(RSpec \times TSpec)) \rightarrow \wp(Mss).
\]

\[
\langle C, f \rangle \in \text{mth}(B, T) \iff \forall \ell \in \text{Loc}. \forall v \in Val. \forall \sigma, \sigma' \in St.
\]
\[
\quad (\text{type}(\sigma.\ell) = C \land f(\ell, \sigma) = \langle v, \sigma' \rangle) \implies B(v, \sigma') \land T(\ell, \sigma, v, \sigma')
\]

\[
\langle C, \mu \rangle \in \text{cls}(R) \iff \forall m \in M. m \in \text{dom } R \Rightarrow m \in \text{dom } \mu \land \langle C, \mu, m \rangle \in \text{mth}(R, m)
\]

\[
\delta \in \text{pck}(R) \iff \forall C \in C. C \in \text{dom } R \Rightarrow C \in \text{dom } \delta \land \langle C, \delta, C \rangle \in \text{cls}(R, C)
\]

The following abbreviation will be used repeatedly:

\[
\gamma \in C \mapsto R \text{ iff } C \in \text{dom } \gamma \land \langle C, \gamma, C \rangle \in \text{cls}(R)
\]

**Proposition 1.** The specifications above are admissible predicates.

**Proof.** For example,

\[
\langle C, f \rangle \in \text{mth}(B, T) \iff \forall \ell \in \text{Loc}. \forall v \in Val. \forall \sigma, \sigma' \in St.
\]
\[
\quad (\text{type}(\sigma.\ell) \neq C \lor f(\ell, \sigma) \uparrow \lor (B(f(\ell, \sigma)) \land T(\ell, \sigma, f(\ell, \sigma)))
\]
As \( C \) is a flat predomain admissibility must only be shown with respect to the second component \( f \). Since admissible predicates are closed under universal quantification, disjunction, conjunction, and composition with continuous maps (and function application is continuous) it only remains to show that \( \uparrow, B, \) and \( T \) are admissible. The former is by definition, the latter are because they are predicates on a flat predomain. One can show similarly that the other predicates are admissible. Note that \( C \in \text{dom}\, \gamma \) is admissible in \( \gamma \) due to the non-extension order on records.

Specifications for class description transformers can also be defined in the logical relations style:

**Definition 9.** Given predicates \( P \in \wp(A) \) and \( Q \in \wp(B) \), a predicate \( P \to Q \in \wp(A \to B) \) is defined as follows:

\[
f \in P \to Q \iff \forall a \in A. a \in P \Rightarrow f(a) \in Q.
\]

**Corollary 1.** If \( Q \) is admissible so is \( P \to Q \).

### 5.2 Modular Proof Rules

As the semantics of classes and packages is defined via fixpoints it seems adequate to use fixpoint induction. Fortuitously, as shown above, the specifications in use are admissible such that fixpoint induction is applicable.

A spatial conjunction operator, \( \ast \), for records (of method suites) will be used below. It is defined in analogy with separation logic [5, 9] where the operator works on heaps. It is more appropriate than normal conjunction \( \land \) due to the modularity of class definitions.

**Definition 10.** Let \( P, Q \in \wp(\text{Mss}) \), then \( P * Q \in \wp(\text{Mss}) \) is defined as follows:

\[
\gamma \in P * Q \iff \exists \gamma_1, \gamma_2 : \text{Mss. dom}\, \gamma_1 \cap \text{dom}\, \gamma_2 = \emptyset \land \gamma = \gamma_1 + \gamma_2 \land \gamma_1 \in P \land \gamma_2 \in Q.
\]

We assume that \( * \) has higher precedence than \( \to \).

Admissibility of \( P * Q \) is a more delicate matter but fortunately it is not required for the rules below since \( * \) always appears on the left hand side of an implication \( \to \).

The rules below show which assumptions can be used when in order to prove packages which are broken down into classes. The correctness of classes under certain context assumptions can the be proved using standard techniques not discussed in this paper.

**Theorem 1.** The following proof rules are correct:

1. \[
\frac{\text{class } C \ldots }{\text{class } C \ldots} \text{m} \in \Gamma \to \text{cls}(R)
\]

2. \[
\frac{M \in \Gamma * \text{pck}(R') \to \text{cls}(R) \quad P \in (C \mapsto R) \ast \Gamma \to \text{pck}(R')}{\text{pack}_C(M, P) \in \Gamma \to \text{pck}(\{C = R\} + R')}
\]
Proof. Since the predicates in use are admissible one obtains correctness by fixpoint induction. For rule (1) assume (i) that \( \mu \in \Gamma \) and (induction hyp) that \( \tau \in \text{pck}(R) \). We have to show that \( \exists! R' \in \text{pck}(\{C = \tau\} + \mu) \in \text{pck}(R) \). But this holds by the premise of rule (1) since \( \{C = \tau\} + \mu \in C \mapsto R \ast \Gamma \) by (i) and (ind.hyp.). Rules (2) and (3) can be shown similarly with (3) using (2).

The following predicate states that a certain class in a class descriptor is a subclass of another. The predicate is well-defined since the subtype hierarchy is not circular.

Definition 11. Let \( C, D \in C \) and \( \gamma \in \text{Mss} \):

\[
D \leq_{\gamma} C \iff \gamma \cdot D.3 = C \lor (\gamma \cdot D.3 = D' \land D' \leq_{\gamma} C).
\]

5.3 Method Invocation and Inheritance

The semantics of method invocation depends on the (dynamic) class type of the callee object. The class is only known, however, for this, it is not known for arbitrary objects on the heap. To verify a property of the method it could be necessary to stipulate that the method behaves identically in all classes. This is unrealistic. It is more convenient to assume that it behaves identically for a certain branch of the class hierarchy. If it is known that the maximal class type of an object is \( C \), then the possible methods to be taken into account can be reduced to those in the hierarchy below \( C \). For all of those one could stipulate that they share some behaviour sufficient to prove the given specification. This can be done using the spatial implication operator \( \triangleright \) (see also [9]):

Definition 12. For any \( \gamma \in \text{Mss} \) and \( P, Q \in \wp(\text{Mss}) \) we have

\[
\gamma \in P \triangleright Q \iff \forall \beta : \text{Mss}. \beta \in P \Rightarrow \gamma + \beta \in Q.
\]

With the help of \( \triangleright \) one can now express that no extension of a context may contain a subclass of \( C \) that does not fulfill \( \phi \), briefly \( \text{Sub}(C, \phi) \triangleright \text{false} \) where \( \text{Sub}(C, \phi) \) is defined as shown below:

\[
\gamma \in \text{Sub}(C, \phi) \iff \exists D : C. D \in \text{dom} \gamma \land \gamma \cdot D \notin \phi \land D <_{\gamma} C.
\]

Thus, for a proof of \( \delta \in \Gamma \ast \text{Sub}(C, \phi) \triangleright \text{false} \rightarrow \Delta \) one can assume that the input \( \mu \in \Gamma \ast \text{Sub}(C, \phi) \triangleright \text{false} \) does not contain any subclasses of \( C \) that do not fulfill \( \phi \). In order to employ behavioural subtyping for \( C \) one chooses \( \phi \) to be the specification of \( C \).
6 Conclusion

We have presented a modular denotational semantics for a class-based language and sound proof rules for managing the modules. No recursive domains are necessary as long as persistent classes are not loaded at run time. To achieve the latter one could resort to the approach presented in [13] for object-based languages, paying however the price of a much more involved semantics and logics with more severe restrictions.

The advantage of the semantics presented in this paper is that fixpoint induction is sufficient to derive useful proof rules for modular specifications, the only restriction being that specifications are admissible. Furthermore, the denotational approach does not commit itself to a particular logic so the techniques are applicable to all kinds of languages.

Future work comprises the integration of data abstraction such that accessibility restrictions can be modelled. The applicability of the presented technique to separation logic needs to be investigated as much as the benefit of using separation logic for the records of class definitions.

Acknowledgements

The idea of this paper emerged in discussion with Peter O’Hearn and Uday Reddy about the alleged limitations of a simple denotational semantics for classes. Thanks to Thomas Streicher and Hubert Baumeister for useful remarks and pointers.

References


Validity of CTL Queries Revisited

Marko Samer and Helmut Veith
Institute of Information Systems
Database and Artificial Intelligence Group
Vienna University of Technology, Austria
\{samer, veith\}@dbai.tuwien.ac.at

Abstract. We systematically investigate temporal logic queries in model checking, adding to the seminal paper by William Chan at CAV 2000. Chan’s temporal logic queries are CTL specifications where one unspecified subformula is to be filled in by the model checker in such a way that the specification becomes true. Chan defined a fragment of CTL queries called CTL\(^v\) which guarantees the existence of a unique strongest solution. The starting point of our paper is a counterexample to this claim. We then show how the research agenda of Chan can be realized by modifying his fragment appropriately. To this aim, we investigate the criteria required by Chan, and define two new fragments CTL\(_{new}\) and CTL\(_d\) where the first is the one originally intended; the latter fragment also provides unique strongest solutions where possible but admits also cases where the set of solutions is empty.

1 Introduction

Temporal logic queries are a generalization of model checking [7, 6, 12] which allows system properties not only to be verified, but to be computed in a systematic manner. A temporal logic query is an incomplete CTL specification containing a special placeholder symbol “?”. Intuitively, the query asks for those system properties which yield a correct specification when inserted into the query. For example, the query AF AG ? asks for the invariants the system must eventually satisfy, the query AG (shutdown \(\rightarrow\) AG ?) asks for the invariants after system shutdown, and the query AG (? \(\rightarrow\) \(\neg\)AF ack) asks for properties which prevent that an acknowledgment is eventually sent. Query solving is the task of finding solutions, i.e., appropriate instantiations, to temporal logic queries in a given model. Query solving is more powerful than simple model checking because solving a query amounts to reasoning about a large number of specifications simultaneously. In particular, query solving can be used for diagnostic tasks and for understanding the behavior of an unknown system.

Temporal logic queries were first proposed by William Chan at CAV 2000 [4]. He realized that the number of solutions to CTL queries usually is very large for

* This work was supported by the European Community Research Training Network “Games and Automata for Synthesis and Validation” (GAMES) and by the Austrian Science Fund Project Z29-N04.
practically interesting systems. Thus, he focused on queries with unique strongest solutions – called exact solutions – from which all other solutions can be inferred. Unfortunately, not every CTL query has an exact solution, and determining whether a given query has an exact solution in every model is ExpTime-complete [4]. Consequently, Chan defined an expressive yet easily recognizable syntactic fragment CTL\textsuperscript{v}, and claimed that each query in CTL\textsuperscript{v} has an exact solution in every model. Moreover, Chan proposed an efficient symbolic query solving algorithm for CTL\textsuperscript{v} with the same complexity as CTL model checking.

Temporal logic queries have been well received in the model checking community. In the following, we shortly describe the most important developments initiated by Chan’s paper. Their main focus, however, is on generalizing Chan’s work, i.e., they investigate temporal logic queries in a broader framework and do not use CTL\textsuperscript{v} as basis for their work. As a result of these generalizations, their approaches have in general higher complexity, i.e., double exponential in the number of atomic propositions. However, they have the same cost as model checking when applied to queries with a single strongest solution.

Bruns and Godefroid [2] have investigated temporal logic queries where multiple strongest solutions are allowed. Based on an automata-theoretic approach, they have shown how to obtain a query solving algorithm for any logic having a translation to alternating automata. Gurfinkel, Devereux, and Chechik [9, 5], on the other hand, have proposed an extension of Chan’s approach to multiple placeholders in the framework of their multi-valued model checker XChek.

In the framework of CTL\textsuperscript{*}, Hornus and Schnoebelen [10] have shown that determining whether there exists an exact solution in a given model and computing this solution can be reduced to a linear number of model checking problems. Moreover, they have shown that if there exist multiple strongest solutions, the second one can be computed with a quadratic number of model checking calls, the third one with a cubic number of model checking calls, and so on. Their results can be seen as further evidence that it is important to have a good understanding of when there exists a unique strongest solution.

Current work announced by Ramakrishnan [13] in the area of logic programming based model checkers is investigating the connection between temporal logic queries and value-passing $\mu$-calculus with free variables.

Our contribution. In this paper, we present a CTL\textsuperscript{v} query along with a suitable Kripke structure, such that the query does not have an exact solution. This contradicts Chan’s main claim. The counterexample appeared when we attempted to prove formally the results in [4]. Since we consider Chan’s paper a highly interesting and important contribution, and the paper has been frequently cited, we believe that it is important to give a sound foundation to get a better understanding of temporal logic queries. Based on the insight obtained by the counterexample we define two new languages:

- $\text{CTL}^d$ generalizes Chan’s approach in that it guarantees an exact solution, but only in those cases where the set of solutions is not empty.
- $\text{CTL}^v_{\text{new}}$ is the subset of $\text{CTL}^d$ where all queries necessarily have a solution in every Kripke structure. Thus, $\text{CTL}^v_{\text{new}}$ fulfills Chan’s original criteria.
We will show that the queries in $\text{CTL}^d$ are distributive over conjunction, i.e., that any two solutions to a query can be conjoined (“collected”) into a stronger common solution, and that a strong compound solution can be broken down (“separated”) into individual solutions. It is then a natural step to restrict $\text{CTL}^d$ to $\text{CTL}_{\text{new}}^v$. We also discuss a syntactic characterization of validity and provide partial results towards this aim. All our results are accompanied by exact proofs. The full version of this paper is available as [14].

This paper is organized as follows. In Section 2, we introduce the formalisms used in our work. Section 3 summarizes Chan’s results on temporal logic queries and introduces $\text{CTL}^v$. After presenting the counterexample in Section 4, we investigate $\text{CTL}^d$ and $\text{CTL}_{\text{new}}^v$ in Sections 5 and 6 respectively. Section 7 provides partial results towards a syntactic characterization of validity. Finally, in Section 8, we summarize our results and indicate future research.

2 Preliminaries

We assume the reader is familiar with Kripke structures and the computation tree logic $\text{CTL}$ based on the temporal operators $X$ (“next”), $G$ (“global”), and $U$ (“until”). Let $K$ be a Kripke structure, $\pi = s_0, s_1, s_2, \ldots$ a path, and $\phi$ a $\text{CTL}$ formula. We write $\pi^n$ to denote the suffix of $\pi$ that begins at state $s_n$, and we write $\pi(n)$ to denote the state $s_n$ on $\pi$. If it is clear from the context, we write for simplicity $\pi^n$ also to denote the state $\pi(n)$. Furthermore, paths($s$) denotes the set of paths starting at state $s$. We write $K, s_0 \models \phi$ to denote $K, s_0 \models \phi$, where $s_0$ is the initial state in $K$. Note that $F \phi \equiv \text{true} U \phi$ and $\phi \rightarrow \psi \equiv \neg \phi \lor \psi$ are as usual. Since we will consider formulas in negation normal form only, all other operators are defined directly and not as abbreviations.

Following Chan [4], we use some additional $\text{CTL}$ operators. In particular, we use the weak until operator $\phi W \psi \equiv (G \phi) \lor (\phi U \psi)$. The other operators are variants of the strong until operator $U$ and the weak until operator $W$:

$$
\begin{align*}
\phi U \psi & \equiv \phi (\phi \land \psi) \\
\phi \bar{U} \psi & \equiv \phi (\neg \phi \land \psi) \\
\phi W \psi & \equiv \phi (\phi \land \psi) \\
\phi \bar{W} \psi & \equiv \phi (\neg \phi \land \psi)
\end{align*}
$$

Of course, the additional operators do not increase the expressive power of $\text{CTL}$, but they will give rise to stronger temporal logic query languages. Therefore, we will use the additional $\text{CTL}$ operators throughout this paper.

3 CTL Queries

In this section, we summarize Chan’s results on temporal logic queries [4]. Throughout the paper we will write $\gamma$, $\bar{\gamma}$, etc. to denote queries, and use all other Greek letters to denote formulas.

Definition 1 (CTL Query). A $\text{CTL}$ query $\gamma$ is a $\text{CTL}$ formula where exactly one subformula is replaced by the special symbol $?$ called placeholder. A query is positive, if the placeholder appears under an even number of negations; otherwise, it is negative.
Table 1. Dual Operators

<table>
<thead>
<tr>
<th></th>
<th>(\gamma) pos.</th>
<th>(\gamma) neg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\varphi \Rightarrow \gamma \psi)</td>
<td>(\varphi \Rightarrow \psi)</td>
<td>(\psi \Rightarrow \varphi)</td>
</tr>
<tr>
<td>(\varphi \land \gamma \psi)</td>
<td>(\varphi \land \psi)</td>
<td>(\varphi \lor \psi)</td>
</tr>
<tr>
<td>(\gamma[\top])</td>
<td>(\gamma[\text{true}])</td>
<td>(\gamma[\text{false}])</td>
</tr>
<tr>
<td>(\gamma[\bot])</td>
<td>(\gamma[\text{false}])</td>
<td>(\gamma[\text{true}])</td>
</tr>
</tbody>
</table>

For example, \(A(p \text{U AG}\ ?)\) is a positive CTL query, whereas the CTL query \(\text{AG}(\ ? \rightarrow \text{AF} p)\) is negative. To avoid tedious case distinctions depending on the polarity of CTL queries in the remainder of this paper, we will use the dual operators defined in Table 1.

Definition 2 (Solution). Let \(\gamma\) be a CTL query, \(K\) a Kripke structure, and \(\varphi\) a propositional formula. We write \(\gamma[\varphi]\) to denote the result of substituting \(\varphi\) for the placeholder in the query \(\gamma\). If \(K \models \gamma[\varphi]\), then we say that \(\varphi\) is a solution to \(\gamma\) in \(K\). We denote the set of all solutions to a query \(\gamma\) in a Kripke structure \(K\) by \(\text{sol}(K, \gamma) = \{\varphi \mid K \models \gamma[\varphi]\}\).

For example, consider the Kripke structure \(K\) shown in Fig. 1 and the CTL query \(\gamma = A((a \lor c) \text{U AG}\ ?)\). It is easy to see that \(b, d, a \land d\) are solutions to \(\gamma\) in \(K\). So we have \(K \models \gamma[b]\), \(K \models \gamma[d]\), and \(K \models \gamma[b \land d]\).

To obtain the maximum information a query provides, it is necessary to consider all solutions. Since the number of solutions is likely to be very large, it is desirable to have a strongest solution that subsumes all other solutions. In the remainder of this section we present Chan’s approach how to obtain such a strongest solution.

3.1 Monotone CTL Queries

The first important property we need to consider is monotonicity. A CTL query \(\gamma\) can be viewed as a function \(\gamma : \varphi \mapsto \gamma[\varphi]\) that maps propositional formulas to CTL formulas. Depending on the polarity of a query, the corresponding function is either monotonically increasing or monotonically decreasing.

Definition 3 (Monotonicity). A CTL query \(\gamma\) is monotone iff \(\varphi \Rightarrow \psi\) implies \(\gamma[\varphi] \Rightarrow \gamma[\psi]\) for all propositional formulas \(\varphi\) and \(\psi\).

It is easy to define a class of CTL queries that are guaranteed to be monotone.

Definition 4 (CTL\(^m\)). The language CTL\(^m\) is the largest set of CTL queries that do not contain a subquery of the form \(\gamma \text{U} \varphi\) or \(\gamma \text{W} \varphi\), where \(\gamma\) is a query.

For example, \(\text{EX A}(\varphi \text{U AG}\ ?)\) is in CTL\(^m\), whereas \(\text{EX A}((\varphi \lor ?) \text{U} \psi)\) is not in CTL\(^m\). The following lemma justifies our claim from above.
Lemma 1 ([4]). Each query $\gamma$ in $\text{CTL}^m$ is monotone.

Proof. Structural induction on $\gamma$. See [14] for details. $\square$

The following characterizes the cases when a monotone CTL query has a solution.

Lemma 2 ([4]). Let $\gamma$ be a monotone CTL query, and $K$ any Kripke structure.

1. $\gamma$ has a solution in $K$ iff $K \models \gamma[\top]$.
2. Every proposition is a solution to $\gamma$ in $K$ iff $K \models \gamma[\bot]$.

3.2 Exact Solutions and Valid Queries

Recall that our aim is to obtain a strongest solution that subsumes all other solutions to a given query. We call such a strongest solution an exact solution.

Definition 5 (Exact Solution). A solution $\xi$ to a CTL query $\gamma$ in a Kripke structure $K$ is exact iff for every solution $\varphi$, it holds that $\xi \Rightarrow \gamma \varphi$.

It is easy to see that $\xi$ is an exact solution to $\gamma$ in $K$ iff $\text{sol}(K, \gamma) \subseteq \{ \varphi \mid \xi \Rightarrow \gamma \varphi \}$. The following proposition shows that for monotone CTL queries also any implication of an exact solution is a solution. In other words, for an exact solution $\xi$ to a monotone CTL query $\gamma$ in $K$ we have $\text{sol}(K, \gamma) = \{ \varphi \mid \xi \Rightarrow \gamma \varphi \}$.

Proposition 1 ([4]). Let $\xi$ be an exact solution to a monotone CTL query $\gamma$ in a Kripke structure $K$. The propositional formula $\varphi$ is a solution to $\gamma$ in $K$ iff $\xi \Rightarrow \gamma \varphi$.

Note that not every monotone CTL query has an exact solution. Therefore, we are interested in queries that have an exact solution in every Kripke structure. For example, consider the Kripke structure $K$ shown in Fig. 1 and the query $\gamma = \text{AF} \exists$. It is easy to see that $a$ and $c$ are solutions to $\gamma$ in $K$, i.e., $K \models \gamma[a] \land \gamma[c]$. Now, assume that there exists an exact solution $\xi$ to $\gamma$ in $K$. Then, we know that $\xi \Rightarrow a$ and $\xi \Rightarrow c$, which is trivially equivalent to $\xi \Rightarrow a \land c$. Thus, by Proposition 1, we obtain that $a \land c$ must also be a solution to $\gamma$ in $K$. However, it is easy to see that $K \not\models \gamma[a \land c]$.

Definition 6 (Valid Query). A CTL query is valid iff it has an exact solution in every Kripke structure. We say a set of queries is valid if each query in this set is valid.

As we will see, validity of CTL queries is closely related to the following property.

Definition 7 (Distributivity over Conjunction). Let $\gamma$ be a CTL query. $\gamma$ is distributive over conjunction iff for all propositional formulas $\varphi$ and $\psi$, $\gamma[\varphi] \land \gamma[\psi] \Leftrightarrow \gamma[\varphi \land \gamma \psi]$.

The following lemma shows the relation between validity and distributivity.

Lemma 3 ([4]). A monotone CTL query is valid iff it has a solution in every Kripke structure and is distributive over conjunction.
3.3 Definition of CTL\textsuperscript{v}

We have seen in Section 3.2 that not all CTL queries are valid. Notwithstanding the ExpTime-completeness of deciding the validity of a query, Chan defined the following fragment CTL\textsuperscript{v} intended to comprise only valid queries.

**Definition 8 (CTL\textsuperscript{v}).** CTL\textsuperscript{v} is the smallest set of queries satisfying:

1. ? and \(\neg\) are CTL\textsuperscript{v} queries.
2. If \(\varphi\) is a CTL formula and \(\gamma\) is a CTL\textsuperscript{v} query, then \(\varphi \lor \gamma\), AX\(\gamma\), A(\(\gamma \overline{W}\varphi\)), and A(\(\varphi \overline{W}\gamma\)) are also CTL\textsuperscript{v} queries.
3. A persistence query (CTL\textsuperscript{p}) is also a CTL\textsuperscript{v} query.

The language CTL\textsuperscript{p} of persistence queries is the smallest set of queries satisfying:

4. If \(\gamma\) is a CTL\textsuperscript{v} query, then AG\(\gamma\) is a persistence query.
5. If \(\varphi\) is a CTL formula and \(\gamma\) is a persistence query, then \(\varphi \lor \gamma\), AX\(\gamma\), A(\(\varphi U \gamma\)), and A(\(\varphi W \gamma\)) are also persistence queries.

4 The Counterexample

Chan’s paper [4] did not contain a proof for its main claim, the validity of CTL\textsuperscript{v}. We present a counterexample to this claim.

**Proposition 2.** The language CTL\textsuperscript{v} contains invalid queries.

**Proof.** Consider the CTL query

\[\gamma = A(a U AX(c \lor AG ?))\]

and the Kripke structure \(\mathcal{K}\) shown in Fig. 2. It is easy to see that \(\gamma \in\) CTL\textsuperscript{v}, and that \(b\) and \(d\) are solutions to \(\gamma\) in \(\mathcal{K}\), i.e., \(\mathcal{K} \models \gamma[b] \land \gamma[d]\). However, \(b \land d\) is not a solution to \(\gamma\) in \(\mathcal{K}\), i.e., \(\mathcal{K} \not\models \gamma[b \land d]\). Thus, \(\gamma\) is not distributive over conjunction. Hence, by Lemma 3, \(\gamma\) is invalid. \(\square\)

In the remainder of this paper, we will construct a language of valid CTL queries following Chan’s approach. Instead of simply restricting CTL\textsuperscript{v} in such a way that the above counterexample is not possible, we take a more systematic approach. From Lemma 3 we know that validity means distributivity plus the existence of a solution in every Kripke structure. Among these properties, distributivity is both the much harder and the much more important one; we will therefore first concentrate on a possibly broad language for distributive queries, and then restrict it to valid queries.
5 \(\text{CTL}^d\) Queries

We will now present \(\text{CTL}^d\), a distributive language without the problem of \(\text{CTL}^v\). The intuition behind \(\text{CTL}^d\) is that disjunction is not allowed to appear in the scope of the next operator \(\text{AX}\) in the definition of the persistence queries, unless there appears the global operator \(\text{AG}\) between them. To incorporate this insight into the definition of \(\text{CTL}^d\), it is natural to split the language \(\text{CTL}^p\) of persistence queries into two sublanguages; the proofs to be shown below will provide further evidence for this choice.

**Definition 9 (\(\text{CTL}^d\)).** \(\text{CTL}^d\) is the smallest set of queries satisfying:

1. \(?\) and \(\neg ?\) are \(\text{CTL}^d\) queries.
2. If \(\phi\) is a \(\text{CTL}\) formula and \(\gamma\) is a \(\text{CTL}^d\) query, then \(\phi \land \gamma\), \(\phi \lor \gamma\), \(\text{AX} \gamma\), \(\text{A}(\gamma \cup \phi)\), \(\text{A}(\gamma \bar{U} \phi)\), \(\text{A}(\gamma \mathcal{W} \phi)\), and \(\text{A}(\phi \bar{W} \gamma)\) are also \(\text{CTL}^d\) queries.
3. An outer persistence query (\(\text{CTL}^p\)) is also a \(\text{CTL}^d\) query.

The language \(\text{CTL}^p\) of outer persistence queries is the smallest set of queries satisfying:

4. An inner persistence query (\(\text{CTL}^p\)) is also an outer persistence query.
5. If \(\phi\) is a \(\text{CTL}\) formula and \(\gamma\) is an outer persistence query, then \(\phi \lor \gamma\), \(\text{A}(\gamma \cup \phi)\), \(\text{A}(\phi \cup \gamma)\), \(\text{A}(\gamma \mathcal{W} \phi)\), and \(\text{A}(\phi \mathcal{W} \gamma)\) are also outer persistence queries.

The language \(\text{CTL}^p\) of inner persistence queries is the smallest set of queries satisfying:

6. If \(\gamma\) is a \(\text{CTL}^d\) query, then \(\text{AG} \gamma\) is an inner persistence query.
7. If \(\phi\) is a \(\text{CTL}\) formula and \(\gamma\) is an inner persistence query, then \(\phi \land \gamma\), \(\text{AX} \gamma\), \(\text{A}(\gamma \cup \phi)\), \(\text{A}(\phi \cup \gamma)\), \(\text{A}(\phi \bar{U} \gamma)\), \(\text{A}(\phi \bar{W} \gamma)\), \(\text{A}(\gamma \bar{W} \phi)\), \(\text{A}(\phi \bar{W} \gamma)\), \(\text{A}(\gamma \bar{U} \phi)\), and \(\text{A}(\phi \bar{U} \gamma)\) are also inner persistence queries.

It can be easily seen that \(\text{CTL}^d\) is a subset of \(\text{CTL}^m\).

5.1 Distributivity of \(\text{CTL}^d\)

This section is devoted to the nested inductive proof of our main result.

**Theorem 1.** Each query in \(\text{CTL}^d\) is distributive over conjunction.

According to Definition 7, we have to show that for each query \(\gamma \in \text{CTL}^d\) and all formulas \(\varphi\) and \(\psi\) it holds that \(\gamma[\varphi] \land \gamma[\psi] \iff \gamma[\varphi \land \gamma[\psi]]\). We split this equivalence into two implications:

**Definition 10 (Separating, Collecting).** Let \(\gamma\) be a \(\text{CTL}\) query. \(\gamma\) is separating iff it satisfies \(\gamma[\varphi] \land \gamma[\psi] \implies \gamma[\varphi \land \gamma[\psi]]\), and collecting iff it satisfies \(\gamma[\varphi] \land \gamma[\psi] \implies \gamma[\varphi \land \gamma[\psi]]\). \(\gamma\) is AG-collecting iff for each subquery of the form \(\text{AG} \bar{\gamma}\), \(\bar{\gamma}\) is collecting.

The separation property is shown very easily.
Lemma 4. Each query in \( \text{CTL}^d \) is separating.

Proof. Choose any query \( \gamma \in \text{CTL}^d \subset \text{CTL}^m \). Consider the two implications \( \varphi \land \gamma \psi \Rightarrow \gamma \varphi \) and \( \varphi \land \gamma \psi \Rightarrow \psi \), which are both tautologies. By Lemma 1, we obtain both \( \gamma[\varphi \land \gamma \psi] \Rightarrow \gamma[\varphi] \) and \( \gamma[\varphi \land \gamma \psi] \Rightarrow \gamma[\psi] \). This is trivially equivalent to \( \gamma[\varphi \land \gamma \psi] \Rightarrow \gamma[\varphi] \land \gamma[\psi] \).

Note that the separation property in fact holds for all monotone \( \text{CTL} \) queries. For the collection property, however, we have to do more work. By a series of lemmas we prove variations of the collection property for each language \( \text{CTL}^d \), \( \text{CTL}^{pi} \), and \( \text{CTL}^{po} \). These lemmas will then be combined in an inductive proof.

Definition 11 (Reachability). Let \( s_1 \) and \( s_2 \) be two states in a Kripke structure. We say that state \( s_2 \) is reachable from state \( s_1 \), in symbols \( s_1 \sim s_2 \), iff there exists a path \( \pi \in \text{paths}(s_1) \) and an \( n \in \mathbb{N} \) such that \( \pi^n \in \text{paths}(s_2) \).

The following lemma shows that solving collecting queries preceded by the \( \text{AG} \) operator is similar to finding invariants.

Lemma 5. Let \( \gamma \) be a collecting \( \text{CTL} \) query, and let \( s_1 \) and \( s_2 \) be two states such that \( s_1 \sim s_2 \). If \( s_1 \models \text{AG} \gamma[\varphi] \) and \( s_2 \models \text{AG} \gamma[\psi] \), then \( s_2 \models \text{AG} \gamma[\varphi \land \gamma \psi] \).

Proof. Since \( s_2 \models \text{AG} \gamma[\psi] \), we know that for each state \( s_3 \) with \( s_2 \sim s_3 \) it holds that \( s_3 \models \gamma[\psi] \). In addition, since \( s_1 \models \text{AG} \gamma[\varphi] \) and \( s_1 \sim s_2 \), we know that for each state \( s_3 \) with \( s_2 \sim s_3 \) it holds that \( s_3 \models \gamma[\varphi] \). Now, since \( \gamma \) is collecting, we obtain \( s_3 \models \gamma[\varphi \land \gamma \psi] \) for each state \( s_3 \) with \( s_2 \sim s_3 \). Hence, we have \( s_2 \models \text{AG} \gamma[\varphi \land \gamma \psi] \).

Note that each query in \( \text{CTL}^{pi} \) contains a subquery of the form \( \text{AG} \gamma \). Thus, we can use Lemma 5 as the induction start in Lemma 6.

Lemma 6. Let \( \gamma \in \text{CTL}^{pi} \) be \( \text{AG} \)-collecting, and let \( s_1 \) and \( s_2 \) be two states such that \( s_1 \sim s_2 \). If \( s_1 \models \gamma[\varphi] \) and \( s_2 \models \gamma[\psi] \), then \( s_2 \models \gamma[\varphi \land \gamma \psi] \).

Proof. Structural induction on \( \gamma \).

Induction start: If \( \gamma = \text{AG} \bar{\gamma} \), then, by Lemma 5, we obtain \( s_2 \models \gamma[\varphi \land \gamma \psi] \).

Induction step: Let \( \gamma = A(\theta U \bar{\gamma}) \) and assume \( s_1 \models \gamma[\varphi] \) as well as \( s_2 \models \gamma[\psi] \). W.l.o.g., we choose a path \( \pi \in \text{paths}(s_2) \), and a path \( \sigma \in \text{paths}(s_1) \) s.t. for some \( n \in \mathbb{N} \) we have \( \sigma^n = \pi \). Therefore, we know that there exists a smallest \( k \) and a smallest \( l \) such that \( \sigma^k \models \gamma[\varphi] \) and \( \sigma^{n+l} \models \gamma[\psi] \). Now, by induction hypothesis, we obtain \( \sigma^{\max(k, n+l)} \models \gamma[\varphi \land \gamma \psi] \), where \( \max(k, n + l) \geq n \). In addition, it is easy to see that \( \sigma^i \models \theta \) for all \( n \leq i < \max(k, n + l) \). Since \( \pi \) was chosen w.l.o.g., we obtain \( s_2 \models A(\theta U \bar{\gamma}[\varphi \land \gamma \psi]) \). Hence, we obtain \( s_2 \models \gamma[\varphi \land \gamma \psi] \).

See [14] for the remaining cases.

Note that in \( \text{CTL}^m \), all subqueries of a given query have the same polarity; the only exception is the subquery \( ? \) which has a different polarity than \( \neg \gamma \). Similarly as above, the case of inner persistence queries \( \text{CTL}^{pi} \) shown in Lemma 6 constitutes the induction start for the outer persistence queries in Lemma 7.
Lemma 7. Let $\gamma \in \text{CTL}^{p_o}$ be AG-collecting, $s_1, s_2$ be states and $\rho \in \text{paths}(s_1)$ be a path where $\rho^n \in \text{paths}(s_2)$ for some suitable $n \in \mathbb{N}$. If $s_1 \models \gamma[\varphi]$ and $s_2 \models \gamma[\psi]$, then there exists an $r \leq n$ such that $\rho^r \models \gamma[\varphi \land \gamma \psi]$.

Proof. Structural induction on $\gamma$.
Induction start: If $\gamma \in \text{CTL}^{p_1}$, then, by Lemma 6, we obtain $s_2 \models \gamma[\varphi \land \gamma \psi]$.
Induction step: Let $\gamma = \theta \lor \tilde{\gamma}$ and assume $s_1 \models \gamma[\varphi]$ as well as $s_2 \models \gamma[\psi]$. If $s_1 \not\models \theta$ and $s_2 \not\models \theta$, we know that $s_1 \models \tilde{\gamma}[\varphi]$ and $s_2 \models \tilde{\gamma}[\psi]$. Now, by induction hypothesis, we obtain that there exists $i_0 \leq n$ such that $\rho^{i_0} \models \tilde{\gamma}[\varphi \land \gamma \psi]$. Otherwise, if $s_1 \models \theta$ or $s_2 \models \theta$, we have $\rho^0 \models \theta$ or $\rho^n \models \theta$ respectively. Thus, in all cases there exists $r \leq n$, s.t. $\rho^r \models \theta \lor \tilde{\gamma}[\varphi \land \gamma \psi]$ which implies the assertion.

Let $\gamma = A(\theta \lor \tilde{\gamma})$ and assume $s_1 \models \gamma[\varphi]$ as well as $s_2 \models \gamma[\psi]$. We define $P = \{ \pi \in \text{paths}(s_1) \mid \forall i \leq n, \pi^{(i)} = \rho^{(i)} \}$ and choose w.l.o.g. any path $\sigma \in P$. Thus, we know that there exists a smallest $k$ and a smallest $l$ such that $\sigma^k \models \tilde{\gamma}[\varphi]$ and $\sigma^{n+l} \models \tilde{\gamma}[\psi]$. Now, by induction hypothesis, we obtain that there exists $k \leq i_0 \leq n + l$ such that $\sigma^{i_0} \models \tilde{\gamma}[\varphi \land \gamma \psi]$. We distinguish two cases: If $i_0 \leq n$, we trivially obtain $\rho^{i_0} \models A(\theta \lor \tilde{\gamma}[\varphi \land \gamma \psi])$. Note that in this case we do not have to consider the paths paths($\rho^{i_0}$) in detail, because the truth of the formula depends only on the first state $\rho^{(i_0)}$. Otherwise, if $i_0 > n$, we know for all $n \leq i < i_0$ that $\sigma^i \models \theta$ and $\sigma^{i_0} \models \tilde{\gamma}[\varphi \land \gamma \psi]$. Since $\sigma$ was chosen w.l.o.g., we obtain $\rho^n \models A(\theta \lor \tilde{\gamma}[\varphi \land \gamma \psi])$. Note that the paths in $P$ contain all elements of paths($s_2$) as suffixes.

See [14] for the remaining cases. □

Remark. The case of $\gamma = \theta \lor \tilde{\gamma}$ is important because it is this case which made us split up the persistence queries into inner and outer persistence queries. More precisely, the inductive proof for the more restrictive property of Lemma 6 does not go through for $\gamma = \theta \lor \tilde{\gamma}$. The chosen less restrictive property of Lemma 7 works for $\gamma = \theta \lor \tilde{\gamma}$, but its weakness in turn forces us to restrict the allowed operators in $\text{CTL}^{p_o}$. A posteriori, it is not surprising that disjunction plays a crucial role in the counterexample of Section 4.

The following property of $\text{CTL}^{p_o}$ queries is directly implied by the special case of $s_1 = s_2$ in the previous lemma.

Corollary 1. If $\gamma \in \text{CTL}^{p_o}$ is AG-collecting, then $\gamma$ is collecting.

Finally, Corollary 1 serves as the induction start for Lemma 8.

Lemma 8. If $\gamma \in \text{CTL}^d$ is AG-collecting, then $\gamma$ is collecting.

Proof. Structural induction on $\gamma$.
Induction start: If $\gamma$ is the (negated) placeholder\(^1\), then the assertion is trivially true, and if $\gamma \in \text{CTL}^{p_o}$, we obtain the assertion by Corollary 1.
Induction step: Let $\gamma = A(\gamma U \theta)$ and assume $s \models \gamma[\varphi] \land \gamma[\psi]$. W.l.o.g., we choose any path $\sigma \in \text{paths}(s)$. Therefore, we know that there exists a smallest $k$ such

\(^1\) This refers to the case where no subquery AG $\tilde{\gamma}$ exists.
that $\sigma^k \models \theta$. If $k > 0$, we know that $\sigma^{i_0} \models \bar{\gamma}[\varphi] \land \bar{\gamma}[\psi]$ for all $i_0 < k$. Now, by induction hypothesis, we obtain $\sigma^{i_0} \models \bar{\gamma}[\varphi \land \bar{\gamma}[\psi]]$ for all $i_0 < k$. The case $k = 0$ is trivial. Since $\sigma$ was chosen w.l.o.g., we obtain the assertion.

See [14] for the remaining cases.

The following lemma brings our auxiliary results together such that we obtain the collecting property for distributivity over conjunction.

**Lemma 9.** Each query in $\text{CTL}^d$ is collecting.

**Proof.** The proof is by induction over the number of subqueries of the form $\text{AG} \bar{\gamma}$ of $\gamma$. It is important to note that according to the definition of $\text{CTL}^d$, whenever we have two subqueries $\text{AG} \bar{\gamma}$ and $\text{AG} \bar{\gamma}$ of a query $\gamma$, then either $\text{AG} \bar{\gamma}$ is a subquery of $\text{AG} \bar{\gamma}$ or vice versa. Thus, all subqueries starting with $\text{AG}$ are linearly ordered in a natural way.

**Induction start:** If $\gamma$ contains no subquery of the form $\text{AG} \bar{\gamma}$, then the assumptions of Lemma 8 are trivially satisfied. Therefore, we obtain the assertion by Lemma 8. **Induction hypothesis:** Assume that the assertion holds for each query $\gamma$ that contains at most $n$ subqueries of the form $\text{AG} \bar{\gamma}$. **Induction step:** Assume $\gamma$ contains $n + 1$ subqueries of the form $\text{AG} \bar{\gamma}$. Thus, for each subquery $\text{AG} \bar{\gamma}$ of $\gamma$, we trivially know that $\bar{\gamma}$ contains at most $n$ subqueries of the form $\text{AG} \bar{\gamma}$. Hence, by induction hypothesis, we obtain that $\gamma$ is $\text{AG}$-collecting. Now, by Lemma 8, we obtain the assertion.

Now, Theorem 1 follows trivially by Lemma 4 and Lemma 9.

### 6 Valid Queries in $\text{CTL}^d$

If we are primarily concerned with finding an exact solution to CTL queries while accepting that there may not exist a solution, we can use queries in $\text{CTL}^d$. However, following the original motivation of Chan we are also interested in queries for which there certainly exists an exact solution in every Kripke structure. This leads us to the language $\text{CTL}_{\text{new}}^v$ of valid queries.

**Definition 12 (CTL$_{\text{new}}^v$).** $\text{CTL}_{\text{new}}^v$ is the smallest set of queries satisfying:

1. $\? \text{ and } \neg\? \text{ are } \text{CTL}_{\text{new}}^v \text{ queries.}$
2. If $\varphi$ is a CTL formula and $\gamma$ is a $\text{CTL}_{\text{new}}^v$ query, then $\varphi \lor \gamma$, $\text{AX} \gamma$, $\text{A}(\gamma \text{ W } \varphi)$, $\text{A}(\gamma \text{ W } \varphi)$, and $\text{A}(\varphi \text{ W } \gamma)$ are also $\text{CTL}_{\text{new}}^v$ queries.
3. A valid outer persistence query is also a $\text{CTL}_{\text{new}}^v$ query.

The language $\text{CTL}_{\text{po}}^v$ of valid outer persistence queries is the smallest set of queries satisfying:

4. A valid inner persistence query is also a valid outer persistence query.
5. If $\varphi$ is a CTL formula and $\gamma$ is a valid outer persistence query, then $\varphi \lor \gamma$, $\text{A}(\varphi \text{ U } \gamma)$, $\text{A}(\gamma \text{ W } \varphi)$, and $\text{A}(\varphi \text{ W } \gamma)$ are also valid outer persistence queries.
The language $\text{CTL}^v_{\text{new}}$ of valid inner persistence queries is the smallest set of queries satisfying:

6. If $\gamma$ is a $\text{CTL}^v_{\text{new}}$ query, then $\text{AG} \gamma$ is a valid inner persistence query.
7. If $\varphi$ is a $\text{CTL}$ formula and $\gamma$ is a valid inner persistence query, then $\text{AX} \gamma$, $\text{A}(\varphi \ U \gamma)$, $\text{A}(\varphi \ W \gamma)$, and $\text{A}(\varphi \bar{W} \gamma)$ are also valid inner persistence queries.

It can be easily seen that $\text{CTL}^v_{\text{new}}$ is a subset of $\text{CTL}^d$. For the validity of $\text{CTL}^v_{\text{new}}$ it only remains to show that each query in $\text{CTL}^v_{\text{new}}$ has a solution in every Kripke structure.

Lemma 10. If $\gamma \in \text{CTL}^v_{\text{new}}$, then $K \models [\gamma] \top$ for any Kripke structure $K$.

Proof. Structural induction on $\gamma$. See [14] for details.

Now, by Theorem 1, Lemma 10, Lemma 2, and Lemma 3, we have reached our final result in the following theorem.

Theorem 2. Each query in $\text{CTL}^v_{\text{new}}$ is valid.

7 Towards a Syntactic Characterization

In this section we will justify the restrictions on temporal operators in the definition of $\text{CTL}^d$ and $\text{CTL}^v_{\text{new}}$. In addition, we will shortly discuss a syntactic characterization of distributivity and validity.

In order to do this, we first show that releasing the constraints on the occurrences of temporal operators in $\text{CTL}^d$ necessarily violates distributivity over conjunction. Recall that the definition of $\text{CTL}^m$ excludes only the operators $\text{A}(\gamma \bar{U} \varphi)$ and $\text{A}(\gamma \bar{W} \varphi)$. Thus, these operators are the only ones that could violate monotonicity and hence separability. Counterexamples to monotonicity resp. separability can be easily found when allowing these operators in Definition 4 [14]. Moreover, counterexamples to the collection property concerning the $\text{CTL}^d$ operators when adding $\text{A}(\varphi \ U \gamma)$, $\text{A}(\varphi \bar{U} \gamma)$, $\text{A}(\varphi \ W \gamma)$, or $\text{A}(\varphi \bar{W} \gamma)$ to line 2 in Definition 9 can also be found very easily [14]. Now, it remains to consider the more complicated counterexamples to extensions of $\text{CTL}_{\text{pi}}$ and $\text{CTL}_{\text{po}}$ as follows in detail.

$\text{CTL}_{\text{pi}}$ Collection Property. Consider the Kripke structure shown in Fig. 3 together with the following queries:

- $\text{A}(a \ U \text{AX} \text{A}(\text{AG} \ ? \ U (c \lor (b \land d))))$
- $\text{A}(a \ U \text{AX} \text{A}(\text{AG} \ ? \ W (c \lor (b \land d))))$

If we added $\text{A}(\gamma \ U \varphi)$ or $\text{A}(\gamma \ W \varphi)$ to line 7 in Definition 9, then the corresponding query above would be in this extension of $\text{CTL}^d$. For both queries $\gamma$, however, we have $s_0 \models \gamma[b] \land \gamma[d]$, but $s_0 \not\models \gamma[b \land \gamma d]$. Hence, both operators must indeed be excluded from the definition of $\text{CTL}_{\text{pi}}$. 

Validity of CTL Queries Revisited 481

CTL<sub>Po</sub> Collection Property. Consider the Kripke structure shown in Fig. 4 together with the following queries:

- \( A(e \mathcal{U} (\neg c \land A(a \mathcal{U} (c \lor AG ?)))) \)
- \( A(e \mathcal{U} A(\neg c, A(a \mathcal{U} (c \lor AG ?)))) \)
- \( A(e \mathcal{U} A(A(a \mathcal{U} (c \lor AG ?)), \mathcal{W} \neg c)) \)
- \( A(e \mathcal{U} A((a \lor b), \mathcal{W} A(a \mathcal{U} (c \lor AG ?)))) \)

If we added \( \varphi \land \gamma, A(\gamma \mathcal{U} \varphi), A(\gamma \mathcal{W} \varphi), A(\varphi \mathcal{U} \gamma), A(\varphi \mathcal{W} \gamma), \) or \( A(\varphi \mathcal{W} \gamma) \) to line 5 in Definition 9, then the corresponding query above would be in this extension of CTL<sub>d</sub>. Again, for all of these queries \( \gamma \) above we have \( s_0 \models \gamma[b] \land \gamma[d] \), but \( s_0 \not\models \gamma[b \land \gamma d] \). Hence, these operators must indeed be excluded from the definition of CTL<sub>Po</sub>. The counterexample concerning the next operator has already been shown in Section 4. Hence, the operator AX \( \gamma \) also must be excluded from the definition of CTL<sub>Po</sub>.

Finally, the restrictions of CTL<sub>d</sub> to CTL<sub>new</sub> can also be justified by counterexamples to validity. Due to space restrictions, the reader is referred to [14] for these simple examples.

The above results demonstrate that CTL<sub>d</sub> and CTL<sub>new</sub> are maximal in the sense that the clauses in Definition 9 and Definition 12 cannot be extended by releasing the restrictions on the temporal operators.

Of course, it would be interesting and important to obtain an exact syntactic characterization of distributivity and validity. We believe, however, that this is a highly non-trivial question, and cannot be obtained by trivial modifications of CTL<sub>d</sub> and CTL<sub>new</sub>. As mentioned above, deciding both validity and distributivity is ExpTime-complete [4,14]. Therefore, it is not possible to define the natural syntactic fragment in terms of a simple grammar such as a context-free grammar (as the complexity of deciding membership in the language would be low). It may, however, in principle be possible to capture distributivity and validity by simple syntactic fragments such that every distributive resp. valid query is equivalent to a query in these fragments<sup>2</sup>. Alternatively, a characterization with respect to a special kind of grammar similar to [3] could also be of interest.

<sup>2</sup> Compare Maidl’s characterization of ACTL \( \cap \) LTL [11] which seemingly contradicts the PSpace-completeness of deciding whether a given ACTL formula is in LTL.
There are two kinds of queries which are not captured by our fragments:

1. Queries where the same subformula occurs several times at different places.
   For example, $A(\varphi \ U (AX \neg\varphi) \land ?)$ is distributive over conjunction but not in $\text{CTL}^d$, and $(AG \neg\varphi) \lor A(? \ U \varphi)$ is valid but not in $\text{CTL}^v_{\text{new}}$. It can be easily seen that our first example is very similar to the definition of the disjoint strong until operator $\overline{U}$, i.e., we could define a new operator equivalent to $A(\varphi \ U (AX \neg\varphi) \land ?)$ and extend our fragment in this way. However, there is an unlimited number of possible extensions in this way, e.g., consider the queries $A(\varphi \ U (AX AX \neg\varphi) \land ?)$, $A(\varphi \ U (AX AX AX \neg\varphi) \land ?)$, …

2. Queries where all subformulas are independent from each other. For example, $A(a \ U (d \land A((c \lor AG ?) \ U b)))$ is distributive over conjunction but not in $\text{CTL}^d$, and $A(a \ U A((c \lor AG ?) \ W b))$ is valid but not in $\text{CTL}^v_{\text{new}}$.

While we believe that the queries of type 2 can be dealt with by an extension of the approach in this paper, type 1 raises a much harder problem.

8 Conclusion and Future Work

In this paper we have systematically investigated Chan’s program on temporal logic query languages, and corrected some flaws as to put a thorough foundation for future work. Among these, we have defined the two fragments $\text{CTL}^d$ and $\text{CTL}^v_{\text{new}}$. $\text{CTL}^d$ has a more systematic appearance; in particular, the behavior of the variants of the until operator is completely analogous to the behavior of the corresponding variants of the weak until operator. This does obviously not hold true for $\text{CTL}^v_{\text{new}}$. The most challenging among the theoretical issues left is to capture the distributive and the valid queries by simple syntactic fragments as discussed in Section 7.

A natural different extension already mentioned in [4] is to consider LTL. While the work in [10] covers $\text{CTL}^*$, and hence LTL, it does not characterize the existence of exact solutions. We are currently working on a formal foundation of distributive resp. valid LTL queries.

Our original interest in temporal logic queries was triggered by practical applications on which we are currently working. The first one is invariant discovery [8], i.e., the automatic discovery of program invariants. The second application area is counterexample analysis. Recent work on software verification [1] is dealing with the question how to extract error causes from multiple counterexamples generated by a model checker. Indeed, temporal logic queries can be used in a natural way to determine error causes. Consider for example a safety property where the good states are described by a formula $\varphi$, and define the queries $\gamma_1 = AG(? \rightarrow AF \neg\varphi)$ and $\gamma_2 = AG(? \rightarrow EG \varphi)$. The solution to $\gamma_1$ describes the set of states which possibly violate $\varphi$, and the solution to $\gamma_2$ describes the set of states which are on a correct trace. From these sets, the error causes of [1] can be obtained easily.
References

Abstract. The notion of meta-variable plays a fundamental role when we define formal systems such as logical and computational calculi. Yet it has been usually understood only informally as is seen in most textbooks of logic. Based on our observations of the usages of meta-variables in textbooks, we propose two formal systems that have the notion of meta-variable.

In both calculi, each variable is given a level (non-negative integer), which classifies variables into object variables (level 0), meta-variables (level 1), metameta-variables (level 2) and so on. Then, simple arity systems are used to exclude meaningless terms like a meta-level function operating on the metameta-level. A main difference of the two calculi lies in the definitions of substitution. The first calculus uses textual substitution, which can often be found in definitions of quantified formulae: when a term is substituted for a meta-variable, free object-level variables in the term may be captured. The second calculus is based on the observation that predicates can be regarded as meta-level functions on object-level terms, hence uses capture-avoiding substitution.

We show both calculi enjoy a number of properties including Church-Rosser and Strong Normalization, which are indispensable when we use them as frameworks to define logical systems.

Keywords: Meta-variable, logical framework, context, \(\lambda\)-calculus

1 Introduction

The notion of meta-variable is a fundamental notion both in logic and computer science. It is because both logic and computer science mainly deal with linguistic objects such as formulas, proofs, programs etc., and whenever we make a general statement about these objects we use meta-variables to refer to these objects. For example, if we look at any book of logic, most variables we see are meta-variables. Meta-variables are also known as metamathematical variables [7] and syntactical variables [12]. However, it seems that, so far, only a very few attempts have been made to formalize the notion of meta-variable. One reason for this may be that we have to go to metameta-level to do so.
In this paper, we present two new formalizations of the concept of the meta-variable. These formalizations are based on our observations of the usages of meta-variables in text books and technical papers on logic.

The first observation is from Shoenfield [12] and Kleene [7]. In these books, we find the following sentence as one of the inductive clauses which define formulas. (We have slightly modified notations from the originals.)

If \( x \) is a variable and \( A \) is a formula, then \( \exists x A \) is a formula.

In the above sentence, both ‘\( x \)’ and ‘\( A \)’ are meta-variables, and when we use the sentence as a rule which is used to construct a concrete formula, we must instantiate these meta-variables by concrete linguistic objects of the object language. Thus, for example, we may instantiate \( x \) by a concrete variable \( x \) and \( A \) by a concrete formula \( x = x \). Then, by applying the instantiated rule, we have that \( \exists x x = x \) is a formula. Here, it is important to remark that the process of instantiation we just described is a form of substitution, but, unlike ordinary substitutions, the variables being substituted are meta-variables. There is another subtle point in this substitution process. To see this, we analyze the above instantiation process in two steps. Namely, we assume that the meta-variable ‘\( x \)’ is first instantiated and then, in the second step, the meta-variable ‘\( A \)’ is instantiated. Then, after the first step, we get the following sentence.

If \( x \) is a variable and \( A \) is a formula, then \( \exists x A \) is a formula.

In the second step, we substitute \( x = x \) for \( A \) in the above sentence, and we get the fully instantiated sentence:

If \( x \) is a variable and \( x = x \) is a formula, then \( \exists x x = x \) is a formula.

We note here that, unlike ordinary substitution, we have substituted \( x = x \) for \( A \) in \( \exists x A \) without following the usual convention of renaming the name of the binding variable (\( x \) in this case) to avoid the capture of free variables.

The second observation we now make is also a very common one. Often in the literature, notation like \( A(x) \) is used to indicate a formula where free occurrences of \( x \) in \( A(x) \) is implicitly understood. Thus, if \( t \) is a term, then \( A(t) \) stands for a formula which is obtained from \( A(x) \) by substituting \( t \) for \( x \) in \( A(x) \). In this usage, ‘\( x \)’, ‘\( t \)’ and ‘\( A \)’ are all meta-variables and the first two meta-variables range over variables and terms in the object language. As for the third meta-variable ‘\( A \)’ it is possible to interpret its range in two ways.

The first interpretation is to regard ‘\( A \)’ as ranging over functions which, when applied to terms, will yield formulas. In this interpretation \( A(t) \) denotes the result of applying the meta-level function \( A \) to a term \( t \). So, in this interpretation, \( A \) is a metameta-variable, since its denotation is not a linguistic object of the object language but it is a function in the meta language.

The second interpretation is to regard ‘\( A \)’ as ranging over abstracts of the object language which can be instantiated to formulas by supplying terms of the object language. This interpretation is possible only if the object language contains such abstracts as its formal entities. Higher-order abstract syntax employed by, e.g., Edinburgh LF [5] is based on this interpretation.
In this paper, we will introduce two typed calculi $\lambda M$ and $\lambda m$, which are respectively designed based on the above two observations. In $\lambda M$ and $\lambda m$, with each variable a non-negative integer, which we call the level of the variable, is associated. We consider level 0 as the object-level, level 1 as the meta-level, level 2 as the metameta-level and so on.

In these formalizations, we believe that we can take virtually any formal system as the object-level of our systems. However, for the sake of concrete presentation, we take as the object-level a system of symbolic expressions we introduced in Sato [10] which is simple but powerful enough to represent syntax of many of the commonly used formal systems such as the $\lambda$-calculus and predicate calculus. Both $\lambda M$ and $\lambda m$ will be constructed on top of this object-level system by adding higher-level structures, and we will show that these calculi enjoy nice properties such as confluence and strong normalizability. We will also show that the calculus $\lambda M$ can represent the notion of context naturally since a context, which is an expression containing some holes in it, is inherently a linguistic object in the meta-level and not in the object-level.

Due to lack of space, we have omitted some lemmas and details of proofs. A full version of this paper with proofs is accessible at http://www.sato.kuis.kyoto-u.ac.jp/~masahiko/index-e.html.

2 Informal Introduction to the Calculi

In this section we informally explain the two calculi $\lambda M$ and $\lambda m$ which we propose in this paper. We assume our object language contains constants, abstraction ($\langle x \rangle [M]$), and pair ($\langle M, N \rangle$). Abstraction and application in the meta-level are denoted by $\lambda X.M$ and $MN$. (We often use capital letters for meta-variables in examples, although both object- and meta-level variables belong to the same syntactic category in the formal definition.)

2.1 The Calculus $\lambda M$

The first calculus $\lambda M$ is based on the first observation in Section 1. Let us consider the first observation again and assume that we have just completed the first step. Then, we have the expression $\exists x A$. We can represent this expression by $\langle \exists', (x^0) [A^1] \rangle$ using a constant $'\exists'$. On the shoulder of a variable, we write a natural number to indicate its level, although we often omit the level if it is clear from the context. So, we simply write $x$ for $x^0$ and it corresponds to the concrete object-level variable $x$. In $\lambda M$, the instantiation process of the meta-variable $A$ by the object-level formula $x = x$ can be represented as the reduction process of the following $\beta$-redex:

$$(\lambda A. \langle \exists', (x) [A] \rangle)\langle '=?', \langle x, x \rangle \rangle.$$ 

In the reduction, as pointed out in Section 1 non-standard substitution is performed and we get:

$$(\exists', (x) [\langle '=?', \langle x, x \rangle \rangle]).$$
Calculi of Meta-variables 487

which represents the formula $\exists x\ x = x$ as expected. Note that the object-level variable $x$ is captured through the substitution.

The non-standard (textual) substitution we have just introduced gives rise to the following two technical problems.

The first one is the non-confluence of the calculus. As argued in the literature on context calculi [8,6,11], calculi that have textual substitution cannot be confluent unless we restrict the evaluation order. For instance, let $M$ be the term $(\lambda X^2. (\lambda x^1. X^2)y^0)x^1$. Depending on the evaluation-order, we will get different results $y^0$ and $x^1$. Our solution to this problem is (roughly) that, a redex may not be reduced if it contains variables of higher levels than the level of the redex. In this example, the inner redex has level-1, so its reduction is postponed until the variable $X^2$ disappears.

The second problem in formulating $\lambda M$ is that, since we restrict the evaluation-order, some reductions may get stuck. Consider the terms $(\lambda X^2. \lambda x^1. X^2)y^0z^0$, and $(\lambda x^1. \lambda X^2. X^2)y^0z^0$. The first term reduces to $y^0$, while the second term cannot be reduced. Since we do not consider terms like the second one meaningful, we introduce arities to rule out such terms. For instance, $\lambda x^1. X^2 : 0 \rightarrow 1$ 0 signifies that this term denotes a level-1 function from objects to objects. Similarly we have $\lambda X^2. \lambda x^1. X^2 : 0 \rightarrow 2 (0 \rightarrow 1 0)$. On the other hand, $\lambda x^1. \lambda X^2. X^2$ would have arity $0 \rightarrow 1 (0 \rightarrow 2 0)$, and it would denote a level-1 function which returns a level-2 function. We will exclude such a term by defining arity properly, and show that the evaluation in $\lambda M$ does not get stuck (Theorem 4).

Although $\lambda M$ has non-standard substitution, we need the standard capture-avoiding substitution as well, when the variable being substituted for and one being captured are of the same level. Let us see the following reduction:

$$(\lambda X^2. \lambda Y^2. \lambda z^1. X^2) (Y^2z^1) \rightarrow \lambda W^2. \lambda z^1. Y^2z^1$$

in which the variable $z^1$ is captured, while the variable $Y^2$ is not captured since its level is the same as that of the variable $X^2$.

2.2 The Calculus $\lambda m$

The second calculus $\lambda m$ formalizes the first interpretation of the second observation in Section 1.

The formula $A(t)$ can be represented as $A^2(t^1)$ using the level-2 variable $A^2$ of arity $0 \rightarrow 1$ 0, and the level-1 variable $t$ of arity 0 in $\lambda m$. The existential formula $\exists x\ A(x)$ is represented as:

$$\langle \exists', \langle x^0 \rangle [A^2(x^0)] \rangle,$$

and the substitution of $\lambda x^1 : 0. \langle =', \langle x^1, x^1 \rangle \rangle$ for $A^2$ is realized by $\beta$-reduction, but this time we do not use the non-standard substitution. Hence, the term:

1 The level of the redex $(\lambda X^i. M)N$ is $i$.
2 In this paper, we do not formalize the second interpretation in which $A$ in $A(t)$ ranges over abstracts of the object language. It should be a straightforward extension of this work, but details are left for future work.
\[(\lambda A^2 : 0 \rightarrow 1. \langle \exists', (x^0) [A^2(x^0)] \rangle)(\lambda x^1 : 0. \langle '='', (x^1, x^1) \rangle)\]

reduces (using \textit{standard} substitution) to

\[\langle '='', (x^0) [(='', (x^0, x^0))] \rangle\]

as expected.

3 The Calculus $\lambda M$

In this section, we give a formal definition of the first calculus $\lambda M$. The second calculus $\lambda m$ will be introduced in the next section.

3.1 Arities and Terms

We define \textit{arity} ($\alpha$) and its \textit{level} ($|\alpha|$) as follows:

1. 0 is an arity and $|0| = 0$.
2. If $\alpha$ and $\beta$ are arities, $0 < i$, $|\alpha| < i$, and $|\beta| \leq i$, then $\alpha \rightarrow_i \beta$ is an arity and $|\alpha \rightarrow_i \beta| = i$.

Note that the side condition of the second clause reflects the intended notion of level introduced in Section 1. Intuitively, the arity $\alpha \rightarrow_i \beta$ is for level-$i$ functions from terms of arity $\alpha$ to terms of arity $\beta$, thus $|\alpha| < i$ and $|\beta| < i$ must be satisfied. The restriction on $\beta$ is relaxed to $|\beta| \leq i$ to allow currying.

We assume that, for each natural number $i$ and arity $\alpha$, there are infinitely many variables, and the sets of variables of each level and arity are mutually disjoint. For a variable $x$ of level $i$ and arity $\alpha$, we sometimes write it as $x^i$. The set of all variables is denoted by $V$. A \textit{(variable) declaration} is an expression of the form $x^i : \alpha$, where $\alpha$ is an arity and either $|\alpha| < i$ or $|\alpha| = i = 0$. We say the level of this declaration is $i$. A \textit{hypothesis sequence} is a finite sequence of declarations. A \textit{judgment} is an expression of the form $\Gamma \vdash M : \alpha$ where $\Gamma$ is a hypothesis sequence and $\alpha$ is an arity.

We have the following rules that are used to derive judgments.

The first rule introduces variables for each $i$ and $\alpha$, where we assume $x$ is a variable of level $i$ and arity $\alpha$.

\[
\frac{x^i : \alpha \in \Gamma}{\Gamma \vdash x^i : \alpha} \quad \text{(var)}
\]

The next two rules introduce abstraction and application for level-$i$ ($i > 0$).

\[
\frac{\Gamma, x^i : \alpha \vdash M : \beta \quad |\beta| \leq i}{\Gamma \vdash \lambda x^i : \alpha. M : \alpha \rightarrow_i \beta} \quad \text{(abs)}
\]

\[
\frac{\Gamma \vdash M : \alpha \rightarrow_i \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta} \quad \text{(app)}
\]

Note that, in the rule (abs), the level of the variable $x^i$ and that of the arity $\alpha \rightarrow_i \beta$ should agree, and the side-condition $|\beta| \leq i$ is needed to form a (well-formed) arity $\alpha \rightarrow_i \beta$. Note also that we may not construct a term like $\lambda x^1 : 0. \lambda X^2 : 0. X^2$. 
The last group of rules are those for the level-0, the object language.

\[\frac{c \text{ is a constant}}{\Gamma \vdash c : 0} \quad (\text{const0})\]

\[\frac{\Gamma, x^0 : 0 \vdash M : 0}{\Gamma \vdash (x^0) \langle M \rangle : 0} \quad (\text{abs0})\]

\[\frac{\Gamma \vdash M : 0 \quad \Gamma \vdash N : 0}{\Gamma \vdash \langle M, N \rangle : 0} \quad (\text{pair0})\]

An expression \(M\) is said to be a term if a judgment of the form \(\Gamma \vdash M : \alpha\) is derivable for some \(\Gamma\) and \(\alpha\). We sometimes just write \(\lambda x^i. M\) for \(\lambda x^i : \alpha. M\) when the arity \(\alpha\) of \(x^i\) is irrelevant.

The scope of \(\lambda x.\) and free occurrences of variables in a term are defined as usual. For a term \(M\), the set of free variables in \(M\) is denoted by \(\text{FV}(M)\). The level of a term \(M\), denoted by \(|M|\), is the maximum level of variables in \(M\), or 0 if there is no variable in \(M\). Note that we take all variables (even variable occurrences in the scope of \(\lambda\)) into account—for instance, \(|\lambda x^2 : \alpha. x^2 y^1| = 2\). Note also that \(|M|\) is not necessarily equal to the level of its arity. The level of a hypothesis sequence \(\Gamma\), denoted by \(|\Gamma|\), is the maximum level of variables in \(\Gamma\), or 0 if \(\Gamma\) is the empty sequence.

### 3.2 \(\alpha\)-Equivalence

In the calculus \(\lambda M\), we need special care to define \(\alpha\)-equivalence: occurrences of a variable \(x\) in the scope of \(\lambda x.\) – usually called bound occurrences – may or may not be subject to renaming since textual substitution does not commute with naive variable renaming. For instance, we may identify \(\lambda x^1. y^2(\lambda z^1. x^i z^1)\) with \(\lambda x^1. y^2(\lambda u^1. x^1 u^1)\), but not with \(\lambda w^1. y^2(\lambda z^1. w^1 z^1)\). To see its reason, let us substitute \(x^1\) for \(y^2\) in these terms. Since the level of \(y^2\) is higher than \(x^1\) and \(w^1\), the textual substitution is used, and the first and the third terms become \(\lambda x^1. x^1(\lambda z^1. x^1 z^1)\), and \(\lambda w^1. x^1(\lambda z^1. w^1 z^1)\), resp., which do not have the same denotational meaning. Hence, we let an abstraction \(\lambda x^i. e\) be \(\alpha\)-convertible only when no variable at a level higher than \(i\) occurs in its scope.

We define the \(\alpha\)-equivalence after a few auxiliary definitions. A renaming of variables is a partial function from \(V\) to \(V\) which is injective and satisfies \(|f(x^i)| = i\) for all \(x^i\) in the domain of the partial function. For a partial function \(f\), its domain is denoted by \(\text{dom}(f)\). For a renaming \(f\) and a variable \(x^i\) (which may not be in \(\text{dom}(f)\)), \(f \downarrow x^i\) is a renaming of variables such that \(\text{dom}(f \downarrow x^i) = \text{dom}(f) - \{x^i\}\), and \(f \downarrow x^i\) agrees with \(f\) on its domain.

For a renaming \(f\) and terms \(M\) and \(N\) in \(\lambda M\), we derive a judgment of the form \(f \vdash M \simeq N\) by the following inference rules:

\[f(x^i) = y^i \quad f \vdash M \simeq M' \quad f \vdash N \simeq N' \quad f \vdash M \simeq M' \quad f \vdash MN \simeq M'N'\]

\[f \vdash \lambda x^i : \alpha. M \simeq \lambda x^i : \alpha. M' \quad (f(x^i) = x^i)\]

\[f \vdash M \simeq M' \quad g \vdash \lambda x^i : \alpha. M \simeq \lambda y^i : \alpha. M' \quad \left(\begin{array}{l} |M| \leq i, |M'| \leq i, \\ f(x^i) = y^i, g \downarrow x^i = f \downarrow x^i \end{array}\right)\]

\[f \vdash M \simeq M' \quad g \vdash \lambda x^i : \alpha. M \simeq \lambda y^i : \alpha. M' \quad \left(\begin{array}{l} |M| \leq i, |M'| \leq i, \\ f(x^i) = y^i, g \downarrow x^i = f \downarrow x^i \end{array}\right)\]
The last rule can be applied only when both $M$ and $M'$ are of level less than or equal to $i$. Otherwise, the term $\lambda x^i. M$ contains (not necessarily free) occurrences of meta-variables which have higher levels than $i$, and we cannot rename the bound variable $x^i$. In this case we can still apply the second last rule, since it does not rename the bound variable $x^i$. For brevity, we omit the inference rules for terms constructed by the rules ($\text{const}_0$), ($\text{abs}_0$), and ($\text{pair}_0$), which are similarly defined. For instance, a term $(x^0)[M]$ has the same rules as the term $\lambda x^i : \alpha. M$.

Let $\text{id}$ be the identity function on $V$. If $\text{id} \vdash M \simeq N$ is derived by the rules above, we say $M$ is $\alpha$-equivalent to $N$ (written by $M \equiv_\alpha N$). It is easy to show the relation $\equiv_\alpha$ is a congruence on terms.

### 3.3 Substitution and Reduction

The notion of reduction in the calculus $\lambda M$ is the union of those in the object language (which we do not specify) and the following $\beta$-reduction:

$$(\beta) \quad (\lambda x^i. M) N \rightarrow [x^i := N]M \quad \text{if} \quad |M| \leq i \quad \text{and} \quad |N| \leq i$$

in which $[x^i := N]M$ denotes the (non-standard) substitution defined below. We write $\xrightarrow{*}$ for the reflexive and transitive closure of $\rightarrow$.

For a level $i > 0$, a level-$i$ variable $x^i$, and terms $M$ and $N$ such that $|M| \leq i$ and $|N| \leq i$, we define $[x^i := N]M$ as follows:

1. $[x^i := N]x^i \triangleq N$
2. $[x^i := N]y^j \triangleq y^j$ \quad if $y^j \not\equiv x^i$
3. $[x^i := N](M_1 M_2) \triangleq ([x^i := N]M_1)([x^i := N]M_2)$
4. $[x^i := N](\lambda y^j. M) \triangleq \lambda y^j. [x^i := N]M$ \quad if $j < i$
5. $[x^i := N](\lambda y^j. M) \triangleq \lambda y^j. [x^i := N]M$ \quad if $y^j \not\in \text{FV}(N)$ and $x^i \not\equiv y^j$
6. $[x^i := N]c \triangleq c$
7. $[x^i := N](y^0)[M] \triangleq (y^0)[[x^i := N]M]$
8. $[x^i := N](M_1, M_2) \triangleq ([x^i := N]M_1, [x^i := N]M_2)$

The first three clauses are standard. For the fourth line, if the level of $y^j$ is strictly less than that of $x^i$, the substitution behaves like textual replacement, that is, free variables may get captured through this substitution. For the fifth line, the bound variable $y^j$ has the same level as $x^i$ that is being substituted, in which case the substitution is the standard capture-avoiding one, hence the side-conditions $y^j \not\in \text{FV}(N)$ and $x^i \not\equiv y^j$ must be satisfied. (The second side-condition $x \not\equiv y$ is not needed for the fourth clause because variables at different levels are assumed to be different.) This side-condition can be always satisfied by taking an $\alpha$-equivalent term. The last three clauses deal with the level 0 in a straightforward manner, except textual substitution.

For brevity, we identify $\alpha$-equivalent terms in the following. Under this convention, we simply have that $[x^i := N]M$ is defined if and only if $\max(|M|, |N|) \leq i$. 

The object-level may contain other redices than the β-redices in the above form, and to distinguish these two, we say that a redex \((\lambda x^i : \alpha. M)N\) is a meta-redex. A term is meta-normal if it does not have meta-redices and a reduction is called a meta-reduction if its redex is a meta-redex.

3.4 Replacing Level-0 Languages

Our level-0 language here is a simplest possible language, which has no notion of types or even computation. We have adopted such a language because it is simple but expressible enough to represent expressions that appear in typical logical systems. It would be possible, however, to adopt other languages such as untyped and simply typed \(\lambda\)-calculi as the level-0 language.

For example, for untyped \(\lambda\)-calculus, we could introduce another term constructor \(M_1 \cdot M_2\) by the rule
\[
\Gamma \vdash M : 0 \quad \Gamma \vdash N : 0 \\
\Gamma \vdash M \cdot N : 0 \quad \text{(app0)}
\]
and a reduction rule \((x^0) [M] \cdot N \rightarrow [x^0 := N]M\) (if \(|M| = |N| = 0\)) where the level-0 substitution would be defined as expected. It would be also straightforward to substitute the simply typed \(\lambda\)-calculus for the level-0 language, by extending the base arities from 0 to the set of simple types.

4 The Calculus \(\lambda m\)

As discussed in the previous sections, the second calculus \(\lambda m\) is based on the second observation discussed in the introduction. It is obtained from \(\lambda M\) by replacing the definition of substitution with the standard capture-avoiding one. Also, we need to use the standard definition of \(\alpha\)-equivalence to identify \(\lambda x^1.y^2(\lambda z^1.x^1 z^1)\) with \(\lambda w^1.y^2(\lambda z^1.w^1 z^1)\), which are not \(\alpha\)-equivalent in \(\lambda M\). Since the other definitions of arities, rules to derive judgments, \(\beta\)-reduction remain the same, we avoid repeating definitions and just show changes to be made.

For \(\alpha\)-equivalence, we replace the third and fourth rules to derive \(f \vdash M \simeq N\) with the following one:
\[
f \vdash M \simeq M' \quad g \vdash \lambda x^i : \alpha. M \simeq \lambda y^i : \alpha. M' \quad (f(x^i) = y^i, g \downarrow x^i = f \downarrow x^i)
\]
Notice that there is no restriction on the levels of the bodies \(M\) and \(M'\) of \(\lambda\)-abstraction. Similarly, the definition of substitution will be as follows.

For a level \(i > 0\), a level-\(i\) variable \(x^i\), and terms \(M\) and \(N\) such that \(|M| \leq i\) and \(|N| \leq i\), \([x^i := N]M\) is defined by:

1. \([x^i := N]x^i \triangleq N\)
2. \([x^i := N]y^j \triangleq y^j\) if \(y^j \neq x^i\)
3. \([x^i := N](M_1 M_2) \triangleq ([x^i := N]M_1)([x^i := N]M_2)\)
4. \([x^i := N](\lambda y^j. M) \triangleq \lambda y^j. [x^i := N]M\) if \(y^j \notin \text{FV}(N)\) and \(x^i \neq y^j\)
5. \([x^i := N]c \triangleq c\)
6. \([x^i := N] \langle y^0 \rangle [M] \triangleq \langle y^0 \rangle [x^i := N] M\) if \(y^0 \not\in \text{FV}(N)\)

7. \([x^i := N] \langle M_1, M_2 \rangle \triangleq \langle [x^i := N] M_1, [x^i := N] M_2 \rangle\)

Notice that the first three clauses are the same as before and the fourth clause
is now a familiar one that avoids variable capturing.

## 5 Examples

In this section, we show a few examples of \(\lambda M\) and \(\lambda m\).

### 5.1 Representing Formulas

In earlier sections, we informally explained how the formula \(x = x\) is substituted
for the meta-variable \(A\) in \(\exists x A\). Here we consider this process formally in \(\lambda M\).

The arity of the corresponding term is inferred in \(\lambda M\) as follows:

\[
\begin{array}{c}
\frac{x^0 : 0, A^1 : 0, x^0 : 0 \vdash A^1 : 0}{x^0 : 0, A^1 : 0 \vdash \langle \exists', (x^0) [A^1] \rangle : 0} \\
\vdots \\
\frac{x^0 : 0 \vdash \lambda A^1 : \exists', (x^0) [A^1] \colon 0 \rightarrow 1}{x^0 : 0 \vdash \langle \exists', (x^0, x^0) \rangle : 0}
\end{array}
\]

Note that, the variable \(x^0\) occurs free in the term of the conclusion, as indicated
by the hypothesis sequence.

We can compute this term as:

\[
\begin{array}{c}
(\lambda A^1 : 0. \langle \exists', (x^0) [A^1] \rangle)(\langle \exists' =', (x^0, x^0) \rangle) \\
\rightarrow [A^1 := \langle \exists' =', (x^0, x^0) \rangle][\exists', (x^0) [A^1]] \\
\equiv \langle \exists', [A^1 := \langle \exists' =', (x^0, x^0) \rangle] (x^0) [A^1] \rangle \\
\equiv \langle \exists', (x^0)[\langle \exists' =', (x^0, x^0) \rangle] \rangle
\end{array}
\]

Note that non-standard (textual) substitution is applied in the last step, since
the level of \(A^1\) is higher than that of \(x^0\). As a result, the free occurrences of \(x^0\)
get bound, and \(x^0\) does not occur free in the resulting term, which represents
the formula \(\exists x x = x\).

We also informally explained how the same example can be written in \(\lambda m\).
Its formal counterpart can be written as follows:\(^3\)

\[
\begin{array}{c}
\frac{\Gamma \vdash A^2 : 00}{\Gamma \vdash A^2 : 00} \\
\frac{\Gamma \vdash x^0 : 0}{\Gamma \vdash x^0 : 0} \\
\frac{A^2 : 00 \vdash \langle \exists', (x^0) [A^2(x^0)] \rangle : 0}{A^2 : 00 \vdash \langle \exists', (x^0) [A^2(x^0)] \rangle : 0} \\
\vdots \\
\frac{A^2 : 00 \vdash \langle \exists', (x^0) [A^2(x^0)] \rangle : 0}{y^1 : 0 \vdash \langle \exists' =', (y^1, y^1) \rangle : 0} \\
\vdots \\
\frac{\lambda A^2 : 00. \langle \exists', (x^0) [A^2(x^0)] \rangle : 0 \rightarrow 2}{\lambda y^1 : 0. \langle \exists' =', (y^1, y^1) \rangle : 0}
\end{array}
\]

\(^3\) We changed the level-1 variable \(x^i\) to \(y^1\) for readability. Formally this renaming is
justified by the \(\alpha\)-equivalence.
where we put an arity 00 = 0 → 1 0 and Γ = A^2 : 0 → 1 0, x^0 : 0 Note that we replaced the meta-variable A^1 in λM by an application term A(x), and to ensure this application is resolved in a meta-level, the arity of A should be 0 → 1 0, hence the level of the variable A must be 2 (or higher).

We omit the computation of the term here, since it is essentially the same as the standard β-reduction.

5.2 Representing Contexts

Contexts can be represented in λM using meta-variables naturally. Let M be a term, C be a context in the object language, i.e., a term with a hole [], and C[M] be the result of the hole-filling operation. In λM, the context C is represented as a term C ≡ (λX^1:0, C[X^1]), and C[M] as an application (CM), which reduces to C[M] by the textual substitution in λM.

Let us take an example from Hashimoto and Ohori’s context calculus [6]. Consider the context C ≡ (λu. (λx. [] u + y)3 and the term M ≡ (λz. C[x + z])x in lambda calculus. They can be written in λM (using our notation for object language) as:

C ≡ λX^1:0. (u^0) [(x^0) [X^1] · u^0] + y^0] · 3
M ≡ (z^0) [C (x^0 + z^0)] · x^0

We can reduce M as:

M → (z^0) [(X^1 := x^0 + z^0)((u^0) [(x^0) [X^1] · u^0] + y^0] · 3)] · x^0
≡ (z^0) [(u^0) [((x^0) [x^0 + z^0] · u^0] + y^0] · 3)] · x^0
→ * 3 + x^0 + y^0

Note that, by the side-condition of the β-reduction, we cannot reduce the outermost level-0 redex first. If it would be reduced first, we would get 3 + x^0 + x^0 as a result, so the Church-Rosser property would be broken. Hashimoto and Ohori’s context calculus has a similar restriction that the β-reduction is prohibited when the redex contains a hole.

A good point of our representation of contexts is that, since contexts are functions, they can be composed, in other words, we can fill a context in another context. As a simple example, let C and D be the contexts λx. [] and λy. [] in lambda calculus, and consider hole-filling of D in C, i.e. C[D]. The contexts C and D are represented in λM as C ≡ λX^1. (x^0) [X^1] and D ≡ λX^1. (y^0) [X^1], then we can compose them in the same way as composition of two functions:

C ◦ D ≡ λX^1. C (D X^1)

C ◦ D reduces to λX^1. (x^0) [(y^0) [X^1]] which represents the context λx. λy. []. It should be noted that, in several existing context calculi including Hashimoto-Ohori’s and our previous work [11], contexts cannot be composed, since these calculi keep track of possible bound variables in a hole.

6 Properties of λM and λm

We have a number of desirable properties for the calculi λM and λm, that is, they enjoy subject reduction, confluence, and strong normalization properties.
In the following, we focus on the properties of $\lambda M$, but the modification for $\lambda m$ is straightforward.

We can prove the subject reduction property in the standard way.

**Theorem 1 (Subject Reduction Property).** If $\Gamma \vdash M : \alpha$ is derived and $M \Rightarrow N$, then $\Gamma \vdash N : \alpha$ can be derived.

Note that even if $\Gamma \vdash (\lambda x^i. M).N : \alpha$ is derived, $[x^i := N]M$ is not necessarily derived. But we can prove from Theorem 2 and 3 that if $|\Gamma| \leq i$ then there exist terms $M'$ and $N'$ such that $M \Rightarrow M'$, $N \Rightarrow N'$, and $[x^i := N']M'$ is defined.

**Lemma 1.** Suppose $\Gamma \vdash M : \alpha$ is derived and the highest level of the redices of $M$ is $i$. Then, any reduction sequence that reduces only level-$i$ redices leads to a term that does not have redices of level-$i$ or higher.

We can prove this lemma by reducing it to the strong normalizability of the simply typed $\lambda$-calculus. We translate $\lambda M$ to the simply typed $\lambda$-calculus by mapping a level-$i$ abstraction $\lambda x^i. M$ to an abstract of the simply typed $\lambda$-calculus and a level-$j$ ($j < i$) abstraction $\lambda x^j. M$ to a pair of $x^j$ and $M$. Since we prove a stronger property in Theorem 6, we omit the details here.

Note that, we cannot simply map $\lambda$-abstractions of $\lambda M$ to those of the simply typed $\lambda$-calculus, since the textual substitution cannot be simulated by the capture-avoiding substitution in the standard calculi.

**Theorem 2.** If $\Gamma \vdash M : \alpha$ is derived, then $M$ has a meta-normal form.

**Proof.** By repeatedly using Lemma 1.

**Theorem 3.** If $\Gamma \vdash M : \alpha$ is derived and $M$ is meta-normal, then $|M| \leq \max(|\Gamma|, |\alpha|)$.

**Proof.** Suppose $M$ is a meta-normal term. We prove the theorem by induction on $M$.

If $M$ is $(\lambda x^i: \beta_1. N_0)N_1 \cdots N_n$ for some $i \geq 0$, $x^i, \beta_1, n \geq 0$, $N_0, \cdots, N_n$, then the arity of $N_0$ should be $\gamma \equiv \beta_2 \rightarrow j_2 \cdots \rightarrow j_n \beta_n \rightarrow_i \alpha$ where $\beta_k$ is the arity of $N_k$ for $k = 2, \cdots, n$. Since $N_0$ is meta-normal, we have $|N_0| \leq \max(|\Gamma|, i, |\gamma|)$ by induction hypothesis, and $|\gamma| \leq i$ by the side-condition of rule (abs). Therefore, we have $|\lambda x^i: \beta_1. N_0| = \max(i, |N_0|) \leq \max(i, |\Gamma|)$. Now, we have two cases.

1. $n = 0$. $|M| \leq \max(i, |\Gamma|) = \max(|\beta_1 \rightarrow_i \gamma|, |\Gamma|)$.
2. $n > 0$. We have, by induction hypothesis, $|N_k| \leq \max(|\Gamma|, |\beta_k|)$ for $1 \leq k \leq n$. Hence $|N_1| \leq \max(|\Gamma|, i)$. Since $M$ is meta-normal, $\max(|N_0|, |N_1|) > i$. Hence $|\Gamma| > i$. Since $|\gamma| \geq |\beta_k|$, we have $|N_k| \leq |\Gamma|$ for $1 \leq k \leq n$. Hence $|M| \leq |\Gamma|$.

We omit the proof of the case where $M$ is of the form $x^i N_1 \cdots N_n$ and we can easily prove the claim in other cases ($M$ is a constant $c$, $(x^0)[N_0]$, or $\langle N_0, N_1 \rangle$) by induction hypothesis.
The following theorem ensures that meta-level evaluation in $\lambda M$ does not get stuck.

**Theorem 4 (Normal Form Property).** Suppose $\Gamma \vdash M : \alpha$ is derived and $M$ is meta-normal. (1) If $|\Gamma| = |\alpha| = 0$, then $|M| = 0$. (2) If $|\Gamma| < |\alpha|$, then $M$ is a $\lambda$-abstraction.

**Proof.** (1) Clear from Theorem 3. (2) If $M$ is not a $\lambda$-abstraction, then it must be of the form $x^i N_1 N_2 \cdots N_n$ or $((\lambda x^i : \gamma . N_0) N_1 N_2 \cdots N_n)$ for $n \geq 0$. In the former case, $|\alpha| \leq |x^i| \leq |\Gamma|$. Contradiction. In the latter case, let $\beta$ be the arity of $N_0$. From Theorem 3 $|(\lambda x^i : \gamma . N_0) N_1| \leq |\beta|$. By the side-condition of rule $(\text{abs})$, $|\beta| \leq i$. Hence $|N_0| \leq i$ and $|N_1| \leq i$, which implies $(\lambda x^i : \gamma . N_0) N_1$ is a redex. Contradiction. $\Box$

We can prove the confluence and strong normalizability of $\lambda M$. The confluence can be proved using the standard technique (parallel reduction), but the strong normalizability is not trivial. Since space is limited, we just give the idea of the proof here. For the detailed proof, see the full version of this paper.

**Lemma 2 (Substitution Lemma).** Let $x^i$ and $y^i$ be distinct variables, $|M_0| \leq i$, $|M_1| \leq i$, and $|N| \leq i$. Then we have

$$[x^i := N][y^i := M_1]M_0 \equiv [y^i := [x^i := N]M_1][x^i := N]M_0$$

**Theorem 5 (Confluence).** The meta-reduction is confluent.

**Theorem 6 (Strong Normalizability).** If $\Gamma \vdash M : \alpha$ is derived, then $M$ is strongly normalizable with respect to the meta-reduction.

**Proof.** (idea) We can prove this theorem by using the reducibility method, but we cannot follow the standard way because substitution operations can be applied only when the level restriction is satisfied. In other words, we cannot express the lemma, which is usually claimed in proof by the reducibility method, that

If $\Gamma \vdash M : \alpha$ is derived, then a term that is obtained by substituting reducible terms for free variables in $M$ is reducible.

To deal with this difficulty, we extend $\lambda M$ so that it has ‘postponed substitutions’, that is, we define an extended judgment of the form $\Gamma_1, x^i := N : \alpha, \Gamma_2 \vdash M$ and give a reduction rule that substitutes $N$ for $x$ in $\Gamma_2$ or $M$ under some conditions. Then, we define reducibility sets that consist of extended judgments and prove lemmas similar to the ones in the case of simply typed lambda calculus. $\Box$

We remark that, for many object languages such as the simply typed lambda calculus, these theorems still hold if we replace the meta-reduction by the union of the meta-reduction and the reductions in the object language.
7 Related Work and Conclusion

We have proposed two formal systems $\lambda M$ and $\lambda m$ that formalize the notion of meta-variable, motivated by the observations how meta-variables are used in textbooks of logic.

Edinburgh Logical Framework (LF) [5] gives a typed framework by which we can define various logical systems as object calculi. Unlike LF and its descendants, we clearly distinguish the meta-levels from the object-level. The textual substitution in $\lambda M$ is another characteristic feature.

Geuvers and Jogjov [3] have introduced the notion of meta-variable into the proof assistant system, so that they can describe open proofs. Their motivation is similar to ours, so they have also encountered the problem that free variables are captured when a meta-variable is instantiated. Their solution to this problem is similar to the one in our work on the calculus of context [11].

Recently much effort has been devoted to formalize the notion of context in lambda calculi, and various calculi of context have been proposed. In the work of Talcott [13] and Mason [8], the notion of contexts are formalized outside of the object language, because contexts are meta-level objects in nature and should be characterized independently of the object language. In other work such as Dami [1], Hashimoto-Ohori [6], Sands [9], and Sato-Sakurai-Kameyama [11], contexts are built into the system, so that context manipulations and object-level calculations can be carried out in the same framework. By representing contexts in $\lambda M$, we have integrated the two approaches.

It turns out that $\lambda m$ is similar to the calculi for binding-time analysis in offline partial evaluation with multiple computation stages [12]. In those calculi, types are stratified like ours but, there, levels represent binding time—i.e., when certain expression can be computed. Reduction is similar to ours in that the order of computation is also determined by levels. Aside from the base language (for which they use typed or untyped $\lambda$-calculus), one subtle difference is that, in those calculi, variables can range over expressions of the same level, resulting in a relaxed condition on function types: $\alpha \to_i \beta$ is a type if $|\alpha| \leq i$ (not $|\alpha| < i$), and $|\beta| \leq i$.

Davies [2] also presents a reformulation $\lambda O$ of the calculus for binding-time analysis and shows that it corresponds to a proof system of linear-time temporal logic with a modal operator. Actually, our earlier attempt to formalize meta-variables [14] was done in a close style. In these calculi, terms are annotated with information that indicates levels of subexpressions, hence $\lambda m$ is a simpler calculus to formalize meta-variables.

As far as we know, the calculus $\lambda M$ is the first one which formalizes non-standard (textual) substitution and is confluent and strongly normalizing. We believe that our calculi can be a basis of formalizing logical and computation systems naturally and directly, but their further development is left for future work.
Acknowledgements

We thank anonymous referees for helpful comments. The first author thanks Henk Barendregt and Jan Willem Klop for having discussions on meta-variables with him. This work is supported in part by Grant-in-Aid for Scientific Research on Priority Areas Research No. 15017247 (Sato, Sakurai) and Grant-in-Aid for Young Scientists (B) No. 15700011 (Igarashi), from MEXT of Japan.

References

Henkin Models of the Partial $\lambda$-Calculus

Lutz Schröder*

BISS, Department of Computer Science, University of Bremen

Abstract. We define (set-theoretic) notions of intensional Henkin model and syntactic $\lambda$-algebra for Moggi’s partial $\lambda$-calculus. These models are shown to be equivalent to the originally described categorical models via the global element construction; the proof makes use of a previously introduced construction of classifying categories. The set-theoretic semantics thus obtained is the foundation of the higher order algebraic specification language HASCASL, which combines specification and functional programming.

Introduction

The partial $\lambda$-calculus has been introduced in [14,15,17] as a natural generalization of the simply typed $\lambda$-calculus that encompasses abstraction for partial functions. Partial functions generally serve to model both non-termination and irregular termination of programs. Consequently, they feature in several specification languages such as RSL [7], SPECTRUM [4], and CASL [2]. The language HASCASL [20], which offers a setting for both specification and implementation of higher order functional programs, is based on the partial $\lambda$-calculus. The work presented here forms the centerpiece of the semantical underpinnings of HASCASL.

The semantics of the partial $\lambda$-calculus as presented in [14] is given in terms of models in partial cartesian closed categories (pcccs). HASCASL is designed as an extension of the first order algebraic specification language CASL, which comes with a set-theoretic semantics; this necessitates the development of a set-theoretic semantics for the partial $\lambda$-calculus, in the spirit of Henkin’s semantics for classical higher order logic [8]. The notion of model chosen for HASCASL is that of intensional Henkin models, where partial function types are interpreted by sets that need not contain all set-theoretic partial functions and, moreover, may contain several elements representing the same set-theoretic function. This choice avoids problems such as incompleteness of deduction or non-existence of initial models, and moreover allows a more detailed analysis of the function space [20].

The central result presented here states that pccc models and intensional Henkin models are in a suitable sense equivalent, in accordance with promises made in [20]. This result generalizes (and elucidates) a corresponding statement made in [3] for the total $\lambda$-calculus. One direction of the equivalence works by

* Research supported by the DFG project HasCASL (KR 1191/7-1)

passing from a pccc model to the Henkin model determined by its global elements (i.e. the elements of a type $T$ in the Henkin model are the morphisms $1 \to T$ in the pccc model); it is non-trivial to prove that this process can indeed be reversed.

The required background w.r.t. the partial $\lambda$-calculus and pcccs is recalled in Sections 1 and 2. Section 2 also introduces the novel notion of pre-pccc. In Section 3, results on classifying categories proved in [19,18] are summarized and adapted to deal with pre-pcccs. The main novel results are presented in Sections 4 and 5. In Section 4, Henkin models are defined as first-order set-valued functors and shown to be equivalent to a notion of syntactic $\lambda$-algebra that generalizes the corresponding definition for the total case given in [3]. Moreover, we define internal languages of Henkin models, which play a crucial role in the proof of the equivalence with pccc-models carried out in Section 5. As general references for categorical concepts, we recommend [1,12].

1 The Partial $\lambda$-Calculus

The natural generalization of the simply typed $\lambda$-calculus to the setting of partial functions is the partial $\lambda$-calculus as introduced in [14,15,17]. The basic idea is that function types are replaced by partial function types, and $\lambda$-abstractions denote partial functions instead of total ones. Due to the fact that terms need not denote, equational reasoning requires some care. We will focus on existential equations here, to be read ‘both sides are defined and equal’; this will be discussed further below (Remark 8).

A signature consists of a set of sorts and a set of partial operators with given profiles (or arities) written $f : \bar{s} \to t$, where $t$ is a type and $\bar{s}$ is a multi-type, i.e. a (possibly empty) list of types. A type is either a sort or a partial function type $\bar{s} \to\!\to t$, with $\bar{s}$ and $t$ as above (one cannot resort to currying for multi-argument partial functions [14]). Following [14], we assume application operators in the signature, so that application does not require extra typing or deduction rules. For $\bar{t} = (t_1, \ldots, t_m)$, $\bar{s} \to\!\to \bar{t}$ denotes the multi-type $(\bar{s} \to\!\to t_1, \ldots, \bar{s} \to\!\to t_m)$, not to be confused with the (non-existent) ‘type’ $\bar{s} \to\!\to t_1 \times \cdots \times t_m$. A morphism between two signatures is a pair of maps between the corresponding sets of sorts and operators, respectively, that is compatible with operator profiles.

A signature gives rise to a notion of typed terms in context according to the typing rules given in Figure 1 where a context $\Gamma$ is a list $(x_1 : s_1, \ldots, x_n : s_n)$, shortly $(\bar{x} : \bar{s})$, of type assignments for distinct variables. More precisely, we speak simultaneously about terms and multi-terms, i.e. lists of terms. The judgement $\Gamma \vartriangleright \alpha : t$ reads ‘(multi-)term $\alpha$ has (multi-)type $t$ in context $\Gamma$’. The empty multi-term $()$, also denoted $*$, doubles as a term of ‘type’ $()$, also denoted 1. When convenient, we use a context to denote the associated multi-type, as e.g. in $\Gamma \to\!\to t$; moreover, we write $\lambda$-abstraction in the form $\lambda \Gamma. \alpha$ where suitable.
A partial λ-theory $T$ is a signature $Σ$ together with a set $A$ of axioms that take the form of existentially conditioned equations: an (existential) equation $α_1 \equiv α_2$ is read ‘$α_1$ and $α_2$ are defined and equal’. Equations $α \equiv α$ are abbreviated as $\text{def } α$ and called definedness judgements. An existentially conditioned equation (ECE) is a sentence of the form $\text{def } α \Rightarrow Γ φ$, where $α$ is a multi-term and $φ$ is an equation in context $Γ$, to be read ‘$φ$ holds on the domain of $α$’. By equations between multi-terms, we can express conjunction of equations (e.g. $\text{def}(α, β) \equiv \text{def } α \wedge \text{def } β$); $\text{true}$ will denote $\text{def } *$.

In Figure 2, we present a set of proof rules for existential equality in a partial λ-theory. The rules are parametrized over a fixed context $Γ$. We write $\text{def } α \vdash Γ φ$ if an equation $φ$ can be deduced from $\text{def } α$ in context $Γ$ by means of these rules; in this case, $\text{def } α \Rightarrow Γ φ$ is a theorem. The rules are essentially a version of the calculus presented in [14], adapted for existential (rather than strong) equations. Of course, there is no reflexive law, since $α \equiv α$ is false if $α$ is undefined. For conciseness, subderivations are denoted in the form $\text{def } α \vdash Δ φ$, where the context $Δ$ and the assumption $\text{def } α$ are to be understood as extending the ambient context and assumptions. Strong equations $Δ \triangleright α \equiv β$, or just $α \equiv β$, are abbreviations for ‘$\text{def } α \vdash Δ \text{def } β$ and $\text{def } β \vdash Δ \text{def } α$’; in particular, rule $(β)$ is really two rules. Rule $(ξ)$ implies that all λ-terms are defined.

The higher order rules $(ξ)$ and $(β)$ show a slight preference for strong equations. Note, however, that the usual form of the $η$-equation, $λ¯x. *: ¯t. α(¯x) = α$, is an ECE, not a strong equation.

A translation between partial λ-theories is a signature morphism which transforms axioms into theorems.

Remark 1. An essential feature of partial equational logic are conditioned terms $α \upharpoonright β$, which denote the restriction of a multi-term $α$ to the domain of a multi-term $β$ [5/15]. In our setting, conditioned terms can be coded using projection operators. Conditioned terms, in turn, provide a coding for λ-abstraction of multi-terms.

Remark 2. A notion of predicates is provided in the shape of terms $Γ \triangleright α : 1$, for which we write $α$ in place of $\text{def } α$. The sentence $\text{def } β$ can be coded as the predicate $(λx. *)(β)$.

The expressive power of ECEs is greatly increased in the presence of an equality predicate (see also [6/14]):
Definition 3. A partial λ-theory has **internal equality** if there exists, for each type $s$, a binary predicate (cf. Remark 2) $eq_s$ such that

$$eq_s(x, y) \Rightarrow x, y : s, x = y \quad \text{and} \quad \text{true} \Rightarrow x : s, eq_s(x, x)$$

Such a predicate allows coding conditional equations as ECEs. In combination with λ-abstraction, it gives rise to a full-fledged intuitionistic logic \[6,11,20\].

2 Partial Cartesian Closed Categories

As suggested in \[14\], the categorical correlate for partial λ-theories are partial cartesian closed categories (pcccs). The crucial feature of a pccc is that, in analogy to (total) morphisms in a cartesian closed category, partial morphisms from $A \times B$ to $C$ are in one-to-one correspondence to total morphisms from $A$ into a partial function space $B \to \varnothing C$. Here, a partial morphism is taken to be a span

$$\begin{array}{c}
\bullet \\
\downarrow f \\
\downarrow m \\
\downarrow B \\
\downarrow \text{A}
\end{array}$$

where $m$ is a monomorphism (of a restricted class) to be thought of as the domain of definition. Alternative approaches to the concept of ‘partial maps’ in categories regard partial maps as morphisms in their own right; see \[6,16\] for in-depth comparisons of the two concepts. The reason for choosing the former approach here is that the transition from pccc models to Henkin models via representable functors would not work in this straightforward manner with the latter approach.
Definition 4. Following [17], we call a class of monomorphisms in a category C a dominion if it contains all identities and is closed under composition and pullbacks, i.e. pullbacks of M-morphisms exist and are in M. A dominional category is a pair (C, M), where M is a dominion on C; an admissible subobject is an element of M. A functor F between dominional categories is called dominional if it preserves admissible subobjects and their pullbacks. F is a dominional equivalence if it is an equivalence of categories (which, of course, already implies pullback preservation) and reflects admissible subobjects.

A dominional category (A, N) is called a dominional subcategory of (C, M) if A is a subcategory of C and the inclusion A \hookrightarrow C is a dominional functor, i.e. if N \subseteq M, and A is closed under pullbacks of N-subobjects in C. (A, N) is called (dominionally) full if, moreover, A is full in C, and, for A ∈ Ob A, each M-subobject of A in C is in N.

A dominion M is closed under intersections. If m is a monomorphism and mg \in M, then g \in M. In particular, M contains all isomorphisms.

A partial morphism is a span (m, f) as above with m \in M. Partial morphisms (m_1, f_1) and (m_2, f_2) are regarded as equal if there exists an isomorphism h such that f_1h = f_2 and m_1h = m_2. The composite of (m, f) and a partial morphism (g, n) from B to C is defined as (mf^{-1}(n), gf^*), where

\[
\begin{array}{c}
\bullet \\
\downarrow f^{-1}(n) \\
\bullet \\
\downarrow f \\
B
\end{array}
\begin{array}{c}
\circledast \\
\downarrow f \\
\circledast
\end{array}
\]

is a pullback. The partial morphisms in (C, M) form a category P(C, M) which contains C as a (non-full) subcategory [16, 17]. Note that a dominional subcategory (A, N) of (C, M) is full iff P(A, N) is a full subcategory of P(C, M).

As usual, we call a category (functor, subcategory) cartesian if it has (preserves, is closed under) finite products; the terminal object is denoted by 1. In a cartesian dominional category (C, M), M is closed under products (but not under pairing). We say that (C, M) has internal equality if M contains all diagonals A \to A \times A. It then follows that C has equalizers (hence is finitely complete, shortly: left exact or lex), and that M contains all regular monomorphisms and is closed under pairing. Cartesian dominional categories are equivalent to first order partial equational theories [19].

Definition 5. A cartesian dominional category (C, M) is called a partial cartesian closed category (pccc) if the composite functor

\[
C \xleftarrow{\times A} C \xhookrightarrow{} P(C, M)
\]

is left adjoint for each object A in C

(This definition is slightly weaker than the one given in [14] in that we do not require finite completeness.) In a pccc we have, for each pair of objects A, B, a
partial function space $A \rightarrow B$ and a co-universal partial evaluation morphism $\text{ev}$ from $(A \rightarrow B) \times A$ to $B$. Explicitly, every partial morphism $C \times A \rightarrow B$ factors uniquely as $\text{ev} \circ (\hat{f} \times A)$ in $\mathbf{P}(\mathbf{C}, \mathbf{M})$ by a total morphism $\hat{f}$ called its abstraction. As a special case, we have a classifier $\top : 1 \rightarrow \Omega$ for $\mathcal{M}$-subobjects, where $\Omega = 1 \rightarrow 1$; i.e. $\top \in \mathcal{M}$, and each $\mathcal{M}$-subobject $m$ of $A$ has a unique characteristic map $\chi_m : A \rightarrow \Omega$ such that

$$
\begin{array}{ccc}
\bullet & \rightarrow & A \\
\downarrow & & \downarrow \chi_m \\
1 & \rightarrow & \Omega
\end{array}
$$

is a pullback. This implies that $\mathcal{M}$ consists of regular monomorphisms, so that $\mathcal{M} = \text{RegMono}(\mathbf{C})$ in case $(\mathbf{C}, \mathbf{M})$ has internal equality.

More generally, we will need to consider the case that partial function spaces exist only for certain objects, the relevant example being the syntactic category of a partial $\lambda$-theory without internal equality (cf. Section 3):

**Definition 6.** A pre-pccc is a cartesian dominional category $(\mathbf{C}, \mathbf{M})$, equipped with a class $\mathfrak{A} \subset \text{Ob} \mathbf{C}$ of objects called type objects such that $\mathfrak{A}$ contains 1 and is closed under partial function spaces in the sense that $(A_1 \times \cdots \times A_n \rightarrow B)$ exists and is in $\mathfrak{A}$ for all $A_1, \ldots, A_n, B \in \mathfrak{A}$, $n \geq 0$, and such that $\mathfrak{A}$ generates $(\mathbf{C}, \mathbf{M})$ as a cartesian dominional category. Pcccs are regarded as pre-pcccs with $\mathfrak{A} = \text{Ob} \mathbf{C}$. A cartesian dominional functor between two pre-pcccs is called partial cartesian closed (pcc) if it preserves type objects and their partial function spaces and evaluation morphisms.

(Compare this to the treatment of the simply typed $\lambda$-calculus in [10].) It will turn out that, for model theoretic purposes, the class of type objects is really a property of $(\mathbf{C}, \mathbf{M})$ rather than extra structure on it (cf. Remark 18). Note that in the total case, i.e. when $\mathcal{M} = \text{Iso}(\mathbf{C})$, every pre-pccc is already a pccc, i.e. in this case a cartesian closed category.

**Example 7.** Every (quasi-)topos is a pccc with internal equality (but not conversely). A typical example of a pccc without internal equality is the category of cpos and continuous functions with Scott open sets [23] as admissible subobjects.

**Remark 8.** In a higher-order setting, existential equations and strong equations are equivalent in expressiveness (e.g., the strong equation $\alpha = \beta$ is equivalent to the existential equation $\lambda x : 1. \alpha \equiv \lambda x : 1. \beta$). However, in the context of first order, which remains important even for higher-order languages as illustrated below, existential equations have stronger ties to the semantics (see also [19]). In particular, existential equations can, unlike strong equations, be interpreted as subobjects in any lex dominional category. Moreover, internal equality is necessarily existential. Finally, existential equations are suitable as premises in conditional equations, while implications with strong equations as premises fail to be in Horn form.
In [18], we show that partial λ-theories and pcccs are equivalent in the same sense as established for λ-theories and cartesian closed categories by Lambek and Scott [11]. Here, we need a variant of this result that makes use of pre-pcccs instead, the benefit being that the resulting classifying categories are more easily manageable. Note that throughout the exposition, the classifying pccc of a partial λ-theory \( T \) is denoted by \( \text{Cl}(T) \), and the classifying pre-pccc by \( \text{Sy}(T) \).

The correspondence between categories and theories works by assigning to each pre-pccc \((C, M, A)\) a partial λ-theory \( L(C, M, A) \), its internal language, and, conversely, to each partial λ-theory \( T \) a classifying pre-pccc \( \text{Sy}(T) \), also called the syntactic category. In [18], \( \text{Cl}(T) \) is constructed as the free pccc over \( \text{Sy}(T) \); moreover, it is shown that \( \text{Sy}(T) \) already is a pccc in case \( T \) has internal equality, and that the two constructions induce an equivalence of partial λ-theories and pcccs, respectively, with internal equality. The latter result is easily modified to obtain an equivalence between partial λ-theories and pre-pcccs. In particular, models of a partial λ-theory \( T \) in a pre-pccc \((C, M, A)\), i.e. translations \( T \rightarrow L(C, M, A) \), are in bijection with pcc functors \( \text{Sy}(T) \rightarrow (C, M, A) \); soundness of the deduction system of Figure 2 for such models is proved along the way. Thanks to the generatedness condition in the definition of pre-pccc, \( \text{Sy}(L(C, M, A)) \) is equivalent to \((C, M, A)\); we will thus liberally identify the two categories when reasoning about \((C, M, A)\). The relevant constructions are summarized below; the proofs are analogous to the ones in [18].

To begin, we define, for a given pre-pccc \((C, M, A)\), the signature \( \Sigma \) of \( L(C, M, A) \). The sorts in \( \Sigma \) are just the objects in \( A \). We recursively define an interpretation \([\_]\) in \( C \) for (multi-)types using the pre-pccc structure. The partial operators of profile \( \vec{s} \rightarrow t \) in \( \Sigma \) are the partial morphisms from \([\vec{s}]\) to \([t]\) in \((C, M)\), with evaluation morphisms (cf. Section 2) as application operators.

The interpretation \([\_]\) is then extended to contexts, terms, multi-terms, and definedness judgements: for a context \( \Gamma = (\vec{x} : \vec{s}) \), \([\Gamma] = [\vec{s}]\). Given a term or multi-term \( \Gamma \triangleright \alpha : \vec{t} \), we define a partial morphism

\[
[\Gamma. \text{def} \alpha] \xrightarrow{[\Gamma. \alpha]} [\vec{t}]
\]

by recursion over the term structure, using projections, partial composition, intersection/tupling, and abstractions, respectively. We will denote any existing domain-codomain restrictions of \([\Gamma. \alpha]\) by \([\Gamma. \alpha]\) as well.

This interpretation leads to a notion of satisfaction in \((C, M)\):

**Definition 9.** An ECE \( \phi \Rightarrow \Gamma \alpha = \beta \) in \( \Sigma \) holds in \((C, M)\) if the subobject \([\Gamma. \phi] \) is contained in \([\Gamma. \text{def}(\alpha, \beta)] \) and the morphisms \([\Gamma. \alpha]\) and \([\Gamma. \beta]\) coincide when restricted to \([\Gamma. \phi] \).

This allows us to complete the definition of \( L(C, M, A) \): the axioms of \( L(C, M, A) \) are the ECEs that hold in \((C, M)\).
Conversely, given a partial $\lambda$-theory $T$, the syntactic category $Sy(T)$ has objects $(\Gamma, \phi)$, where $\phi$ is a definedness judgement in context $\Gamma$ (with $(\Gamma, \text{true})$ abbreviated as $(\Gamma)$); morphisms $(\Gamma, \phi) \to (\Delta, \psi)$, where $\Gamma = (\bar{x} : \bar{s})$ and $\Delta = (\bar{y} : \bar{t})$, are multi-terms $\Gamma \triangleright \alpha : \bar{t}$ such that

$$\phi \vdash_{\Gamma} \psi[\bar{y}/\alpha] \land \text{def} \alpha,$$

taken modulo equality deducible from $\phi$. The identity on $(\Gamma, \phi)$ is represented by $\bar{x}$; composition is substitution. A subobject in $Sy(T)$ is admissible iff it has a representative of the form

$$\bar{x} : (\Gamma, \phi) \hookrightarrow (\Gamma, \psi).$$

Finally, the class $\mathfrak{A}$ of type objects in $Sy(T)$ consists of all objects of the form $(x : t)$. In the absence of internal equality, objects other than the type objects $(x : t)$ need not have partial function spaces, so that the pre-pccc $Sy(T)$ may fail to be a pccc.

4 Henkin Models and Their Internal Language

In [3], it is stated that intensional Henkin models of a (typed) total $\lambda$-theory are equivalent to models in cartesian closed categories; here, a Henkin model is a set-theoretic model where function types need not be interpreted by full function sets (this is the original idea of [8]), and the word intensional is meant to indicate that two inhabitants of a function type may be distinct although they produce the same output on all inputs. We now proceed to establish a corresponding result for the partial $\lambda$-calculus.

In the total case, the correspondence can be made precise as follows. Recall from [11] that a $\lambda$-theory $T$ generates a cartesian closed category (ccc) called its classifying category.

**Definition 10.** Given a ccc $C$, a *ccc model* of $C$ is a cartesian closed (cc) functor $F : C \to A$ into some ccc $A$. An (intensional) *Henkin model* of $C$, or of a (total) $\lambda$-theory $T$ of which $C$ is the classifying category, is a cartesian functor $C \to \text{Set}$. The use of the term ‘Henkin model’ is justified by the fact that such models are easily seen to be the same as Henkin models in the original sense (called *syntactic $\lambda$-algebras* in [3]). This is in tune with the intuition that Henkin models demote higher order to first order structure. Every ccc model gives rise to a Henkin model by composition with $\text{hom}(1, \_)$, since $\text{hom}(1, \_)$ is itself a cartesian functor:

$$C \xrightarrow{F} A \xrightarrow{\text{hom}(1, \_)} \text{Set}.$$

Conversely, every Henkin model $G : C \to \text{Set}$ arises in this way: one can construct from $G$ a ccc $A$ by taking the same objects as in $C$, and the elements
of $G(A \to B)$, where $A \to B$ denotes the function space, as morphisms from $A$ to $B$ in $A$. Moreover, one has a functor $G^* : C \to A$ that maps a morphism $f : A \to B$ to the element of $G(A \to B)$ determined by its name $1 \to (A \to B)$. The functor $G^*$ is a ccc model because the ccc structure is in fact equationally internalized in $C$. This model is determined uniquely in the sense that, for any cc model $F$ such that $\text{hom}(1, \_ \_ \_ \_) \circ F \cong G$, the codomain restriction of $F$ to the full subcategory spanned by its image is equivalent to $G^*$.

We have thus outlined a novel proof of the equivalence of ccc-models and Henkin models as stated in [3], emphasizing the fact that the correspondence is really via postcomposition with $\text{hom}(1, \_ \_ \_ \_)$. This will also be the program for the partial case. The analogy to the total case suggests the following definitions:

**Definition 11.** A pccc model of a pre-pccc $(C, M, A)$ is a pcc functor from $(C, M, A)$ into a pccc $(A, N)$. An (intensional) Henkin model of $(C, M, A)$ (or of a partial $\lambda$-theory $T$ with $\text{Sy}(T) = (C, M, A)$) is a cartesian dominional functor $(C, M) \to \text{Set}$.

(Note that the definition of Henkin model is independent of $A$! For pccc models, see Remark 18.)

**Remark 12.** In order to construct a pccc model, it suffices to provide a pcc functor into a pre-pccc. This follows from the fact that for every cartesian dominional category, the Yoneda embedding provides a full dominional cartesian extension to a pccc which preserves existing partial function spaces [17,18].

Henkin models may be described as syntactic $\lambda$-algebras modeled on the corresponding notion defined for the total $\lambda$-calculus in [3]:

**Definition 13.** A syntactic $\lambda$-algebra for a partial $\lambda$-theory $T$ is a family of sets $[s]$, indexed over all types of $T$, together with partial interpretation functions $[\Gamma. \alpha] : [\Gamma] \to [t]$ for each term $\Gamma \vdash \alpha : t$ in $T$, where $[\Gamma]$ denotes the extension of the interpretation to contexts via the cartesian product. This interpretation is subject to the following conditions:

(i) $[\Gamma. x_i]$, where $\Gamma = (\bar{x} : \bar{s})$, is the $i$-th projection;
(ii) $\bar{g} : t. \beta \circ [\Gamma. \beta] = [\Gamma. \gamma[\bar{g}/\beta]]$, where $\Gamma \vdash \beta : t$ is a multi-term, with the interpretation extended to multi-terms in the obvious way;
(iii) whenever $\phi \vdash r. \alpha \equiv \beta$ in $T$ and $[\Gamma. \phi](x)$ holds (i.e. is defined), then $[\Gamma. \alpha](x) = [\Gamma. \beta](x)$ are defined.

This is the definition of model used in the semantics of HasCasl given in [20].

**Remark 14.** It is implicit in [15] that syntactic $\lambda$-algebras are equivalent to the combinatorically defined $\lambda_p$-algebras considered there.

Both Henkin models and syntactic $\lambda$-algebras can be provided with the obvious notions of morphism. Using results of [19], one easily proves the following correspondence theorem.
Theorem 15. The category of Henkin models of a partial $\lambda$-theory $T$ is equivalent to the category of syntactic $\lambda$-algebras for $T$.

A pccc model $F : (\mathcal{C}, \mathcal{M}, \mathfrak{A}) \to (\mathcal{A}, \mathcal{N})$ gives rise to a Henkin model in the same manner as in the total case: since $\text{hom}(1, \_): (\mathcal{A}, \mathcal{N}) \to \mathbf{Set}$ is a cartesian dominiional functor, so is $\text{hom}(1, \_): (\mathcal{C}, \mathcal{M}) \to \mathbf{Set}$. (It is in particular at this point that the representation of partial morphisms as spans pays off.)

The central point in the proof that, conversely, Henkin models give rise to pccc models is the fact that a Henkin model $F$ has a theory $L(F)$, its internal language, which can be soundly interpreted in $F$. We shall use the following notational convention: a provably defined term $() \triangleright \alpha : A$ in $L(\mathcal{C}, \mathcal{M}, \mathfrak{A})$ determines a morphism $f : 1 \to A$; the map $Ff$ defines an element of $FA$ which we denote by $F(\alpha)$. Now $L(F)$ is constructed as follows:

The sorts of $L(F)$ are the elements of $\mathfrak{A}$; the operators of profile $\bar{A} \to B$ are the elements of $F(\bar{A} \to B)$, where we abuse multi-types $\bar{A}$ in $L(F)$ to denote their obvious interpretations in $(\mathcal{C}, \mathcal{M})$. The application operators are the elements $F(\lambda f : \bar{A} \to B, \bar{x} : \bar{A}, f(\bar{x}))$. Note that $\bar{A} \equiv F\alpha_1 \times \cdots \times F\alpha_n$.

We have the following structure on $F$, transferred from $\mathcal{C}$:

- For $\Gamma = (\bar{x} : \bar{A})$, we have an element $p_i = F(\lambda \Gamma. x_i) \in F(\Gamma \to A_i)$ representing the $i$-th projection for each $i$.
- For (multi-)types $\bar{A}, \bar{B}, C$, the morphism $\lambda \bar{x} : \bar{A}, f(\bar{g}(\bar{x})) : (f : \bar{B} \to C, \bar{g} : \bar{A} \to \bar{B}) \to (\bar{A} \to C)$ in $\mathcal{C}$ is mapped by $F$ to a multi-composition map $F(\bar{B} \to C) \times F(\bar{A} \to \bar{B}) \to F(\bar{A} \to C)$ written $(f, \bar{g}) \mapsto f \circ_p \bar{g}$.
- For (multi-)types $\bar{A}, \bar{B}, C$, the morphism $\lambda \bar{x} : \bar{A}, \lambda \bar{g} : \bar{B}, f(\bar{x}, \bar{y}) : (f : \bar{A}\bar{B} \to C) \to (\bar{A} \to \bar{B} \to C)$ induces an abstraction map $\Lambda : F(\bar{A}\bar{B} \to C) \to F(\bar{A} \to \bar{B} \to C)$.

A term $\Gamma \triangleright \alpha : \bar{C}$ in $L(F)$, where $\Gamma = (\bar{x} : \bar{A})$, can be given an interpretation $\langle \langle \Gamma. \alpha \rangle \rangle \in F(\Gamma \to \bar{C})$ in $F$, defined by recursion over the term structure using the operations introduced above:

\[
\begin{align*}
\langle \langle \Gamma. x_i \rangle \rangle &= p_i \\
\langle \langle \Gamma. (\alpha_1, \ldots, \alpha_n) \rangle \rangle &= (\langle \langle \Gamma. \alpha_1 \rangle \rangle, \ldots, \langle \langle \Gamma. \alpha_n \rangle \rangle) \\
\langle \langle \Gamma. f(\alpha) \rangle \rangle &= f \circ_p \langle \langle \Gamma. \alpha \rangle \rangle \\
\langle \langle \Gamma. \lambda \bar{g} : \bar{B}. \alpha \rangle \rangle &= \Lambda(\langle \langle \Gamma. \bar{g} : \bar{B}. \alpha \rangle \rangle)
\end{align*}
\]
We say that a sentence \( \phi \Rightarrow \Gamma \) holds in \( F \) if
\[
\llbracket \Gamma, \text{def} \alpha \land \phi \rrbracket = \llbracket \Gamma, \phi \rrbracket \quad \text{and} \quad \llbracket \Gamma, \alpha \restriction \phi \rrbracket = \llbracket \Gamma, \beta \restriction \phi \rrbracket.
\]
The axioms of \( L(F) \) are the sentences that hold in \( F \).

As announced, the interpretation of \( L(F) \) in \( F \) is sound:

**Theorem 16.** If \( \phi \vdash \Gamma \psi \), then \( \phi \Rightarrow \Gamma \psi \) holds in \( F \).

## 5 The Equivalence Theorem

We now proceed to prove that every Henkin model arises essentially uniquely from a pccc model by postcomposition with \( \text{hom}(1, \_\_\_) \). More formally:

**Theorem 17.** Let \( F : (C, M, A) \to \text{Set} \) be a Henkin model of a pre-pccc. Then there exists a pccc model \( F^* : (C, M, A) \to (A, N) \) such that
\[
\text{hom}(1, \_) \circ F^* \cong F.
\]
The codomain restriction of \( F^* \) to the full dominional subcategory of \( (A, N) \) spanned by its image is determined uniquely up to equivalence.

This includes the case that \( (C, M) \) is a pccc. Even then, the mentioned span \( (B, P) \) of the image of \( F^* \) in general need not be a pccc \([18]\). Of course, it has a pccc hull in \( (A, N) \), but in the light of an example given in \([18]\), it seems unlikely that this pccc is uniquely determined by \( F \). However, \( (B, P) \) is a pccc in two important special cases, namely if \( (C, M) \) has internal equality, or if \( C \) is a cartesian closed category, with \( M = \text{Iso}(C) \). The latter case is just the abovementioned result of \([3]\) for the total \( \lambda \)-calculus.

**Remark 18.** It is an immediate (and pleasing) consequence of Theorem 17 that the notion of pccc model of a pre-pccc \( (C, M, A) \) depends only on \( (C, M) \).

**Proof (Theorem 17).** The uniqueness statement follows from the fact that, for (multi-)types \( \overline{A}, \overline{B} \) in \( (C, M, A) \), the set \( \text{hom}(1, F(A \to B)) \) determines the partial morphisms from \( F^* \overline{A} \) to \( F^* \overline{B} \). This, in turn, determines the full dominional subcategory spanned by the image of \( A \) under \( F \), which is identical to the subcategory mentioned in the theorem by the generatedness condition for pre-pcccs.

The strategy for the existence proof is as follows: we shall define a translation
\[
\sigma : L(C, M, A) \to L(F)
\]
which gives rise to a functor \( (C, M, A) \cong \text{Sy}(L(C, M, A)) \to \text{Sy}(L(F)) \) that preserves the pccc structure, since the latter is given syntactically. \( \text{Sy}(L(F)) \) is a pre-pccc, which by Remark 12 has a full dominional extension to a pccc \( (A, N) \), so that we obtain a pccc model \( F^* : (C, M) \to \text{Sy}(L(F)) \hookrightarrow (A, N) \). The proof is finished by establishing that \( \text{hom}(1, F^* A) \cong FA \).
The translation $\sigma$ is the identity on sorts (i.e. type objects). An operator $f : \bar{A} \to B$ in $L(C, M, \mathfrak{A})$ is mapped to $F(\lambda \bar{x} : \bar{A}. f(\bar{x})) \in F(\bar{A} \to \to B)$. Thus defined, $\sigma$ preserves application operators. We have to show that $\sigma$ maps axioms of $L(C, M, \mathfrak{A})$ to theorems (in fact, axioms) in $L(F)$. This is easily proved using the following lemma.

**Lemma 19.** For a term $\Gamma \triangleright \alpha : A$ in $L(C, M, \mathfrak{A})$, $$\langle \langle \Gamma. \sigma(\alpha) \rangle \rangle = F(\lambda \Gamma. \alpha) \in F(\Gamma \to \to A).$$

It remains to be shown that hom$(1, _) \circ F^*$ is isomorphic to $F$. We first treat the case that $(C, M)$ is a pccc. In this case, we can define a functor $\hat{F} : C \to \text{Set}$ by $$\hat{F}A = \{ x \in F(1 \to \to A) \mid \text{true } \Rightarrow \text{def } x(*) \text{ holds in } F \}.$$

**Lemma 20.** We have natural isomorphisms $$\text{hom}(1, _) \circ F^* \cong \hat{F} \cong F.$$ This lemma completes the proof of Theorem 17 for the case that $(C, M)$ is a pccc. In the general case, one shows that the restrictions of hom$(1, _) \circ F^*$ and $F$, respectively, to $\mathfrak{A}$ regarded as a full subcategory of $C$ are isomorphic via the same intermediate step as above. This isomorphism can then be extended to all of $C$ thanks to the observation that $\mathfrak{A}$ contains the classifier and hence in fact generates $C$ via finite products and pullbacks. \(\square\)

**Remark 21.** Since $\text{Cl}(T)$ is the free pccc over $\text{Sy}(T)$, pccc models of $\text{Cl}(T)$ are equivalent to pccc models of $\text{Sy}(T)$. This means that every Henkin model of $\text{Sy}(T)$ extends to a Henkin model of $\text{Cl}(T)$; whether or not this extension is unique is the subject of further research.

**Remark 22.** Some care is required concerning the notion of satisfaction in Henkin models/syntactic $\lambda$-algebras. The set of sentences $\phi \Rightarrow_{\Gamma} \alpha \overset{\xi}{=} \beta$ such that $\llbracket \Gamma. \phi \rrbracket(x)$ entails $\llbracket \Gamma. \alpha \rrbracket(x) \overset{\xi}{=} \llbracket \Gamma. \beta \rrbracket(x)$ in a given syntactic $\lambda$-algebra $\mathcal{A}$ fails to be closed under $(\xi)$ (see also [3,13]), so that this notion of satisfaction would not coincide with the one given by the associated pccc model. We remedy this by using instead the notion of satisfaction in Henkin models as defined in Section 4. Then an ECE holds in $\mathcal{A}$ iff it holds in the associated pccc model; in particular, the rules of Figure 2 are sound and complete for Henkin models.

### 6 HasCasl

**HasCasl** is a higher order extension of the first order specification language Cascal based on the partial $\lambda$-calculus and Haskell-style type-class polymorphism. It has been introduced in [20] and used in order to formalize monad-independent computation logics in [22,21], as well as in a case study concerned with higher order Petri nets [9].

Basic HasCasl does not have internal equality. However, internal equality, the internal logic defined on top of it, and HOLCF-style general recursion are
added as optional features in a bootstrap fashion and are provided with built-in syntactical sugar. This allows the combination of programming and specification within a single language. A simple example is the implementation of a function that iterates a partial transition function until a given predicate fails, subject to the requirement that transitions are possible while the predicate holds:

\[
\text{spec \ Repeat = Recursion and Bool then}
\]

\[
\text{type State}
\]

\[
\text{ops step : \ State \rightarrow \ ? State}
\]

\[
\text{cont : \ State \rightarrow \ Bool}
\]

\[
\text{repeat : \ State \rightarrow \ ? State}
\]

\[
\text{internal \ \{ \forall s : \ State \cdot \ cont s \implies \ def \ step s \}}
\]

\[
\text{program \ repeat s = if \ cont s \ then \ repeat (step s) \ else \ s}
\]

Here, the keywords \textit{internal} and \textit{program} invoke syntactical sugar for the internal logic and general recursion, respectively. Note that, thanks to the axiom, undefinedness of \textit{repeat} occurs only for states that admit infinitely many steps.

The benefit of the results above w.r.t. the semantics of HASCASL is that the latter now profits from the best of both worlds. As stated above, the semantics of HASCASL is defined in terms of syntactic \(\lambda\)-algebras, or, equivalently, Henkin models. This makes for a tight connection to the set-theoretic semantics of first-order CASL. For instance, thanks to the fact that both CASL models and HASCASL models can be understood as set-valued cartesian dominional functors, one rather easily proves

\begin{theorem}
Given a CASL signature \(\Sigma\), every model of \(\Sigma\) embeds universally into a Henkin-model of \(\Sigma\), where \(\Sigma\) is reinterpreted as a HASCASL signature.
\end{theorem}

On the other hand, arguments concerning the existence of models for certain specifications are often easier when the notion of pccc models is used. For example, the models of the HASCASL type definition

\[
\text{type } L ::= \text{abs}(\text{rep} : L \rightarrow \ ? L)
\]

that axiomatizes the untyped partial \(\lambda\)-calculus are essentially the solutions of the corresponding domain equation in arbitrary pcccs, e.g. in categories of domains. (More precisely, the specification only requires that the partial function type, denoted \(L \rightarrow \ ? L\) in HASCASL, is a retraction of \(L\).) Pcccs where such a type exists necessarily fail to have internal equality, since the arising internal logic would lead to a Russell-type paradox. Consider, however, the HASCASL specification

\[
\text{INTERNALLogic then type } L ::= \text{abs}(\text{rep} : L \rightarrow L)
\]

that axiomatizes the untyped total \(\lambda\)-calculus (\(L \rightarrow L\) denotes the total function type) in the presence of internal equality, introduced by referencing a named specification of the internal logic as laid out in \[20\]. This specification is consistent: take a cartesian closed category \(\mathbf{C}\) with reflexive object \(U\). The functor
category $\text{Set}^{\text{C}^{\text{op}}}$ is a topos and as such gives rise to a Henkin model with internal equality. Moreover, the Yoneda embedding $Y : \text{C} \rightarrow \text{Set}^{\text{C}^{\text{op}}}$ preserves the cartesian closed structure [23], so that $Y(U)$ is a reflexive object and hence provides us with a model of the above type.

7 Conclusion and Future Work

We have established equivalence results for notions of model for the partial $\lambda$-calculus, to wit, syntactic $\lambda$-algebras in the spirit of [3], first order (i.e. cartesian dominional) functors from the classifying category into $\text{Set}$, and higher order models in arbitrary partial cartesian closed categories (pccc models), thus completing the semantical foundations of $\text{HasCasl}$. These results generalize known results for the total $\lambda$-calculus [3], and illuminate them by demonstrating that the correspondence works simply by postcomposing pccc models with the global element functor $\text{hom}(1, \_ \_ )$. We have emphasized some of the immediate benefits of this result for the model theory of $\text{HasCasl}$. More generally, the equivalence result reconciles the mostly set-theoretic viewpoint of mainstream logic (or algebraic specification) and the concepts of categorical logic and semantics. The connection to Henkin’s seminal work, which concerned models of higher order logic, is tightened by the fact that the partial $\lambda$-calculus with internal equality in fact incorporates full (intuitionistic) higher order logic [20].

One of the upshots of this is that the known deduction system for pccc models is sound and complete for Henkin models. A pressing task in the further development of $\text{HasCasl}$ is to provide proof support using this deduction system via an encoding into an existing theorem prover.

It is hard to resist the temptation to generalize the correspondence between first order and higher order models to base categories other than $\text{Set}$. The proof of the conjecture that higher order models in $\mathcal{V}$-enriched categories are equivalent to first order models in $\mathcal{V}$ is the subject of further research.

Acknowledgements

The author wishes to thank Till Mossakowski, Christian Maeder, and Kathrin Hoffmann for collaboration on $\text{HasCasl}$, and Christoph Lüth for useful comments and discussions.

References


18. L. Schröder, *Classifying categories for partial higher order logic*, available as http://www.tzi.de/~lschrode/hascasl/hoclasscat.ps


Nominal Unification

Christian Urban, Andrew Pitts, and Murdoch Gabbay

University of Cambridge, Cambridge, UK

Abstract. We present a generalisation of first-order unification to the practically
important case of equations between terms involving binding operations. A sub-
stitution of terms for variables solves such an equation if it makes the equated
terms $\alpha$-equivalent, i.e. equal up to renaming bound names. For the applications
we have in mind, we must consider the simple, textual form of substitution in
which names occurring in terms may be captured within the scope of binders
upon substitution. We are able to take a ‘nominal’ approach to binding in which
bound entities are explicitly named (rather than using nameless, de Bruijn-style
representations) and yet get a version of this form of substitution that respects $\alpha$-
equivalence and possesses good algorithmic properties. We achieve this by adapt-
ing an existing idea and introducing a key new idea. The existing idea is terms
involving explicit substitutions of names for names, except that here we only use
explicit permutations (bijective substitutions). The key new idea is that the uni-
fication algorithm should solve not only equational problems, but also problems
about the freshness of names for terms. There is a simple generalisation of the
classical first-order unification algorithm to this setting which retains the latter’s
pleasant properties: unification problems involving $\alpha$-equivalence and freshness
are decidable; and solvable problems possess most general solutions.

1 Introduction

Decidability of unification for equations between first-order terms and algorithms for
computing most general unifiers form a fundamental tool of computational logic with
many applications to programming languages and computer-aided reasoning. However,
very many potential applications fall outside the scope of first-order unification, because
they involve term languages with binding operations where at the very least we do not
wish to distinguish terms differing up to the renaming of bound names.

There is a large body of work studying languages with binders through the use of
various $\lambda$-calculi as term representation languages, leading to higher-order unification
algorithms for solving equations between $\lambda$-terms modulo $\alpha\beta\eta$-equivalence. However,
higher-order unification is technically complicated without being completely satisfac-
tory from a pragmatic point of view. The reason lies in the difference between substi-
tution for first-order terms and for $\lambda$-terms. The former is a simple operation of tex-
tual replacement (sometimes called grafting [7], or context substitution [12, Sect. 2.1]),
whereas the latter also involves renamings to avoid capture. Capture-avoidance ensures
that substitution respects $\alpha$-equivalence, but it complicates higher-order unification al-
gorithms. Furthermore it is the simple textual form of substitution rather than the more
complicated capture-avoiding form which occurs in many informal applications of ‘uni-
fication modulo $\alpha$-equivalence’. For example, consider the following schematic rule

© Springer-Verlag Berlin Heidelberg 2003
which might form part of the inductive definition of a binary evaluation relation \( \downarrow \) for the expressions of an imaginary functional programming language:

\[
\frac{\text{app}(\text{fn } a.Y, X) \downarrow V}{\text{let } a = X \text{ in } Y \downarrow V}.
\]  

(1)

Here \( X, Y \) and \( V \) are metavariables standing for unknown programming language expressions. The binders \( \text{fn } a.(-) \) and \( \text{let } a = X \text{ in } (-) \) may very well capture free occurrences of \( a \) when we instantiate the schematic rule by replacing \( Y \) with an expression. For instance, using the rule scheme in a bottom-up search for a proof of

\[
\frac{\text{let } a = 1 \text{ in } a \downarrow 1}{\text{let } a = X \text{ in } Y \downarrow V}.
\]  

(2)

we would use a substitution that involves capture, namely \([X := 1, Y := a, V := 1]\), to unify the goal with the conclusion of (1)—generating the new goal \( \text{app}(\text{fn } a.a, 1) \downarrow 1 \) from the hypothesis of the rule. The problem with this is that in informal practice we usually identify terms up to \( \alpha \)-equivalence, whereas textual substitution does not respect \( \alpha \)-equivalence. For example, up to \( \alpha \)-equivalence, the goal

\[
\frac{\text{let } b = 1 \text{ in } b \downarrow 1}{\text{let } X \text{ in } Y \downarrow V}
\]

(3)

is the same as (2). We might think (erroneously!) that the conclusion of (1) is the same as \( \text{let } a = X \text{ in } Y \downarrow V \) without changing the rule’s hypothesis—after all, if we are trying to make \( \alpha \)-equivalence disappear into the infrastructure, then we must be able to replace any part of what we have with an equivalent part. So we might be tempted to unify the conclusion with (3) via the textual substitution \([X := 1, Y := b, V := 1]\), and then apply this substitution to the hypothesis to obtain a wrong goal, \( \text{app}(\text{fn } a.b, 1) \downarrow 1 \). Using \( \lambda \)-calculus and higher-order unification saves us from such sloppy thinking, but at the expense of having to make explicit the dependence of metavariables on bindable names via the use of function variables and application. For example, (1) would be replaced by

\[
\frac{\text{app}(\text{fn } \lambda a.F a) X \downarrow V}{\text{let } X (\lambda a.F a) \downarrow V} \quad \text{or, modulo } \eta \text{-equivalence,} \quad \frac{\text{app}(\text{fn } F) X \downarrow V}{\text{let } X F \downarrow V}.
\]  

(4)

Now goal (3) becomes \( \text{let } 1 \lambda b.b \downarrow 1 \) and there is no problem unifying it with the conclusion of (4) via a capture-avoiding substitution of 1 for \( X \), \( \lambda c.c \) for \( F \) and 1 for \( V \).

This is all very fine, but the situation is not as pleasant as for first-order terms: higher-order unification problems can be undecidable, decidable but lack most general unifiers, or have such unifiers only by imposing some restrictions [20]; see [6] for a survey of higher-order unification. We started out wanting to compute with binders modulo \( \alpha \)-equivalence, and somehow the process of making possibly-capturing substitution respectable has led to function variables, application, capture-avoiding substitution and \( \beta \eta \)-equivalence. Does it have to be so? No!

For one thing, several authors have already noted that one can make sense of possibly-capturing substitution modulo \( \alpha \)-equivalence by using explicit substitutions in the term representation language: see [7, 13, 15, 26]. Compared with those works, we make a number of simplifications. First, we find that we do not need to use function variables, application or \( \beta \eta \)-equivalence in our representation language—leaving just binders and \( \alpha \)-equivalence. Secondly, instead of using explicit substitutions of names for names, we use explicit permutations of names. The idea of using name-permutations, and in particular name-swappings, when dealing with \( \alpha \)-conversion dates back to [10] and there is
growing evidence of its usefulness (see [3, 4], for example). When a name substitution is actually a permutation, the function it induces from terms to terms is a bijection; this bijectivity gives the operation of permuting names very good logical properties compared with name substitution. Consider for example the $\alpha$-equivalent terms $fn\ a.\ b$ and $fn\ c.\ b$, where $a$, $b$ and $c$ are distinct. If we apply the substitution $[b\mapsto a]$ (renaming all free occurrences of $b$ to be $a$) to them we get $fn\ a.\ a$ and $fn\ c.\ a$, which are no longer $\alpha$-equivalent.

Thus renaming substitutions do not respect $\alpha$-equivalence in general, and any unification algorithm using them needs to take extra precautions to not inadvertently change the intended meaning of terms. The traditional solution for this problem is to introduce a more complicated form of renaming substitution that avoids capture of names by binders. In contrast, the simple operation of name-permutation respects $\alpha$-equivalence; for example, applying the name-permutation $(ab)$ that swaps all occurrences of $a$ and $b$ (be they free, bound or binding) to the terms above gives $fn\ b.\ a$ and $fn\ c.\ a$, which are still $\alpha$-equivalent. We exploit such good properties of name-permutations to give a conceptually simple unification algorithm.

In addition to the use of explicit name-permutations, we also compute symbolically with predicates expressing freshness of names for terms. This seems to be the key novelty of our approach. Although it arises naturally from the work reported in [11, 23], it is easy to see directly why there is a need for computing with freshness, given that we take a ‘nominal’ approach to binders. (In other words we stick with concrete versions of binding and $\alpha$-equivalence in which the bound entity is named explicitly, rather than using de Bruijn-style representations, as for example in [7, 26].) A basic instance of our generalised form of $\alpha$-equivalence identifies $fn\ a.\ X$ with $fn\ b.\ (ab)\cdot X$ provided $b$ is fresh for $X$, where the subterm $(ab)\cdot X$ indicates an explicit permutation—namely the swapping of $a$ and $b$—waiting to be applied to $X$. We write ‘$b$ is fresh for $X$’ symbolically as $b\#X$; the intended meaning of this relation is that $b$ does not occur free in any (ground) term that may be substituted for $X$. If we know more about $X$ we may be able to eliminate the explicit permutation in $(ab)\cdot X$; for example, if we knew that $a\#X$ holds as well as $b\#X$, then $(ab)\cdot X$ can be replaced by $X$. It should already be clear from these simple examples that in our setting the appropriate notion of term-equality is not a bare equation, $t \approx t'$, but rather a hypothetical judgement of the form

$$\nabla \vdash t \approx t'$$

(5)

where $\nabla$ is a freshness environment—a finite set $\{a_1\#X_1, \ldots, a_n\#X_n\}$ of freshness assumptions. For example $\{a\#X, b\#X\} \vdash fn\ a.\ X \approx fn\ b.\ X$ is a valid judgement of our nominal equational logic. Similarly, judgements about freshness itself will take the form

$$\nabla \vdash a \# t.$$  

(6)

**To summarise:** We will represent languages involving binders using the usual notion of first-order terms over a many-sorted signature. These give us terms with: distinguished constants naming bindable entities, that we call atoms; terms $a.t$ expressing a generic form of binding of an atom $a$ in a term $t$; and terms $\pi\cdot X$ representing an explicit permutation of atoms $\pi$ waiting to be applied to whatever term is substituted for the variable $X$. Section 2 presents this term-language together with a syntax-directed inductive definition of the provable judgements of the form (5) and (6) which for ground terms (i.e. ones with no variables) agrees with the usual notions of $\alpha$-equivalence and ‘not a
free variable of’. However, on open terms our judgements differ from these standard notions and appear to be an extension that has not yet been studied in the literature (including [11, 24, 23]). Section 3 considers unification in this setting. Solving equalities between abstractions \( a.t \approx? a'.t' \) entails solving both equalities \( t \approx? (a.a') \cdot t' \) and freshness problems \( a \#? t' \). Therefore our general form of nominal unification problem is a finite collection of individual equality and freshness problems. Such a problem \( P \) is solved by providing not only a substitution \( \sigma \) (of terms for variables), but also a freshness environment \( \nabla \) (as above), which together have the property that \( \nabla \vdash \sigma(t) \approx \sigma(t') \) and \( \nabla \vdash a \#? \sigma(t'') \) hold for each individual equality \( t \approx? t' \) and freshness \( a \#? t'' \) in the problem \( P \). Our main result with respect to unification is that solvability is decidable and that solvable problems possess most general solutions (for a reasonably obvious notion of ‘most general’). The proof is via a unification algorithm that is very similar to the first-order algorithm given in the now-common transformational style [18]. (See [17, Sect. 2.6] or [1, Sect. 4.6] for expositions of this.) Section 4 considers the relationship of our version of ‘unification modulo \( \alpha \)-equivalence’ to existing approaches. Section 5 assesses what has been achieved and the prospects for applications. To appreciate the kind of problem that nominal unification solves, you might like to try the following quiz about the \( \lambda \)-calculus [2] before we apply our algorithm to solve it at the end of Section 3.

**Quiz** Assuming that \( a \) and \( b \) are distinct variables, is it possible to find \( \lambda \)-terms \( M_1, \ldots, M_7 \) that make the following pairs of terms \( \alpha \)-equivalent?

1. \( \lambda a. \lambda b. (M_1 b) \) and \( \lambda b. \lambda a. (a M_1) \)
2. \( \lambda a. \lambda b. (M_2 b) \) and \( \lambda b. \lambda a. (a M_2) \)
3. \( \lambda a. \lambda b. (b M_3) \) and \( \lambda b. \lambda a. (a M_3) \)
4. \( \lambda a. \lambda b. (b M_4) \) and \( \lambda b. \lambda a. (a M_4) \)

If it is possible to find a solution for any of these four problems, can you describe what all possible solutions for that problem are like?

**Answers:** see Example 2.

---

### 2 Nominal Equational Logic

We take a concrete approach to the syntax of binders in which bound entities are explicitly named. Furthermore we separate the names of bound entities from the names of variables, which is inspired for example by the \( \pi \)-calculus [21], in which the restriction operator binds channel names and these are quite different from names of unknown processes. Names of bound entities will be called atoms. This is partly for historical reasons (stemming from the work by the second two authors [11]) and partly to indicate that the internal structure of such names is irrelevant to us: all we care about is their identity (i.e. whether or not one atom is the same as another) and that the supply of atoms is inexhaustible.

Although there are several general frameworks in the literature for specifying languages with binders, not all of them meet the requirements mentioned in the previous paragraph. Use of the simply typed \( \lambda \)-calculus for this purpose is common; but as discussed in the Introduction, it leads to a problematic unification theory. Among first-order frameworks, Plotkin’s notion of binding signature [25, 9], being unsorted, equates names used in binding with names of variables standing for unknown terms; so it is not sufficiently general for us. A first-order framework that does meet our requirements is the notion of nominal algebras in [16]. The nominal signatures that we use in this paper
are a mild (but practically useful) generalisation of nominal algebras in which name-abstraction and pairing can be mixed freely in arities (rather than insisting as in [16] that the argument sort of a function symbol be normalised to a tuple of abstractions).

**Definition 1.** A nominal signature is specified by: a set of sorts of atoms (typical symbol $\nu$); a disjoint set of sorts of data (typical symbol $\delta$); and a set of function symbols (typical symbol $f$), each of which has an arity of the form $\tau \to \delta$. Here $\tau$ ranges over (compound) sorts given by the grammar $\tau ::= \nu | \delta | 1 | \tau \times \tau | \langle \nu \rangle \tau$. Terms of sort $\langle \nu \rangle \tau$ are binding abstractions of atoms of sort $\nu$ over terms of sort $\tau$. We will explain the syntax and properties of such terms in a moment.

**Example 1.** Here is a nominal signature for expressions in a small fragment of ML:

- sort of atoms: $vid$
- sort of data: $exp$
- function symbols: $\mathbf{vr} : vid \to exp$
- $\mathbf{app} : exp \times exp \to exp$
- $\mathbf{fn} : \langle vid \rangle exp \to exp$
- $\mathbf{1v} : exp \times \langle vid \rangle exp \to exp$
- $\mathbf{1f} : \langle vid \rangle ((\langle vid \rangle exp \times exp) \to exp$.

The function symbol $\mathbf{vr}$ constructs terms of sort $exp$ representing value identifiers (named by atoms of sort $vid$); $\mathbf{app}$ constructs application expressions from pairs of expressions; $\mathbf{fn}$, $\mathbf{1v}$ and $\mathbf{1f}$ construct terms representing respectively function abstractions ($\mathbf{fn} x \Rightarrow e$), local value declarations ($\mathbf{let~val~x~=} e_1 \mathbf{in~e_2~end}$) and local recursive function declarations ($\mathbf{let~fun~f~x~=} e_1 \mathbf{in~e_2~end}$). The arities of the function symbols specify which are binders and in which way their arguments are bound. This kind of specification of binding scopes is of course a feature of higher-order abstract syntax [22], using function types $\nu \to \tau$ in simply typed $\lambda$-calculus where we use abstraction sorts $\langle \nu \rangle \tau$. We shall see that the latter have much more elementary (indeed, first-order) properties compared with the former. To make this point clear we deliberately use a first-order syntax for terms, and not higher-order abstract syntax, although we often refer to abstractions, binders and free atoms by analogy with the $\lambda$-calculus.

**Definition 2.** Given a nominal signature, we assume that there are countably infinite and pairwise disjoint sets of atoms (typical symbol $a$) for each sort of atoms $\nu$, and variables (typical symbol $X$) for each sort $\tau$. The terms over a nominal signature and their sorts are inductively defined as follows, where we write $t : \tau$ to indicate that a term $t$ has sort $\tau$.

- **Unit value** $\langle \rangle : 1$.
- **Pairs** $\langle t_1, t_2 \rangle : \tau_1 \times \tau_2$, if $t_1 : \tau_1$ and $t_2 : \tau_2$.
- **Data** $f t : \delta$, if $f$ is a function symbol of arity $\tau \to \delta$ and $t : \tau$.
- **Atoms** $a : \nu$, if $a$ is an atom of sort $\nu$.
- **Atom-abstraction** $a.t : \langle \nu \rangle \tau$, if $a$ is an atom of sort $\nu$ and $t : \tau$.
- **Suspension** $\pi \cdot X : \tau$, if $\pi = (a_1 b_1)(a_2 b_2) \cdots (a_n b_n)$ is a finite list whose elements $(a_i b_i)$ are pairs of atoms, with $a_i$ and $b_i$ of the same sort, and $X$ is a variable of sort $\tau$. In the case that $\pi$ is the empty list $[]$, we just write $X$ for $\pi \cdot X$.

Recall that every finite permutation can be expressed as a composition of swappings $(a_i b_i)$; the list $\pi$ of pairs of atoms occurring in a suspension term $\pi \cdot X$ specifies a finite permutation of atoms waiting to be applied once we know more about the variable.
We wish to give a definition of \( \alpha \)-equivalence for terms over a nominal signature that is respected by substitution of terms for variables, even though the latter may involve capture of atoms by binders. To do so we will need to make use of an auxiliary relation of freshness between atoms and terms, whose intended meaning is that the atom does not occur free in any substitution instance of the term. As discussed in the Introduction, our judgements about term equivalence \( t \approx t' \) need to contain hypotheses about the freshness of atoms with respect to variables \( a \# X \); and the same goes for our judgements about freshness itself \( a \# t \). Figure 2 gives a syntax-directed inductive definition of equivalence and freshness using judgements of the form

\[
\nabla \vdash t \approx t' \quad \text{and} \quad \nabla \vdash a \# t
\]

where \( t \) and \( t' \) are terms of the same sort over a given nominal signature, \( a \) is an atom, and the freshness environment \( \nabla \) is a finite set of freshness constraints \( a \# X \),
each specified by an atom and a variable. Rules (≈-abstraction-2), (≈-suspension) and (#-suspension) in Fig. 2 make use of the following definitions.

**Definition 3.** Recall from Def. 2 that we specify **finite permutations of atoms** by finite lists \((a_1 b_1)(a_2 b_2) \cdots (a_n b_n)\) representing the composition of finitely many swappings \((a_i b_i)\) with \(a_i\) and \(b_i\) of the same sort. Since we will apply permutations to terms on the left, the order of the composition is from right to left. So with this representation, the composition of a permutation \(\pi\) followed by a swap \((a b)\) is given by list-cons, written \((a b): : \pi\); the composition of \(\pi\) followed by another permutation \(\pi'\) is given by list-concatenation, written as \(\pi' @ \pi\); the **identity permutation** is given by the empty list \([\]\); and the **inverse** of a permutation is given by list reversal, written as \(\pi^{-1}\). The **permutation action**, \(\pi \cdot t\), of a finite permutation of atoms \(\pi\) on a term \(t\) is defined as in Fig. 1; it pushes the list \(\pi\) into the structure of the term \(t\) until it ‘piles up’ in front of suspensions (applying the actual permutation that \(\pi\) represents to atoms that it meets on the way). The **disagreement set** of two permutations \(\pi\) and \(\pi'\) (used in rule (≈-suspension) in Fig. 2) is defined by

\[
ds(\pi, \pi') \overset{\text{def}}{=} \{ a \mid \pi \cdot a \neq \pi' \cdot a \}.
\]

Note that every disagreement set of the lists \(\pi\) and \(\pi'\) is a subset of the **finite** set of atoms occurring in either of the lists, because if \(a\) does not occur in those lists, then from Fig. 1 we get \(\pi \cdot a = a = \pi' \cdot a\). To illustrate the use of disagreement sets, consider

\[
\{ a \# X, c \# X \} \vdash (a c)(a b) \cdot X \approx (b c) \cdot X
\]

which holds by (≈-suspension), since the disagreement set of \((a c)(a b)\) and \((b c)\) is \(\{a, c\}\).

**Lemma 1.** \(\nabla \vdash - \approx -\) is an equivalence relation; it is preserved by all of the term-forming operations in Def. 2; and it respects the freshness relation (i.e. if \(\nabla \vdash a \# t\) and \(\nabla \vdash t \approx t'\), then \(\nabla \vdash a \# t'\)). Both \(\approx\) and \(#\) are preserved by the permutation action given in Fig. 1 in the following sense: if \(\nabla \vdash t \approx t'\), then \(\nabla \vdash \pi \cdot t \approx \pi \cdot t'\); and if \(\nabla \vdash a \# t\), then \(\nabla \vdash \pi \cdot a \# \pi \cdot t\).

**Proof.** Although reasoning about \(\approx\) and \(#\) is rather pleasant once the above facts are proved, establishing them first is rather tricky—mainly because of the large number of cases, but also because the facts in the lemma are inter-dependent; in addition some further properties of the permutation action and disagreement sets need to be established first (statements omitted)\(^1\).

The main reason for using suspensions in the syntax of terms is to enable a definition of substitution of terms for variables which allows capture of free atoms by atom-abstractions while still respecting \(\alpha\)-equivalence. The following lemma establishes this. First we give some terminology and notation for term-substitution.

**Definition 4.** A **substitution** \(\sigma\) is a sort-respecting function from variables to terms with the property that \(\sigma(X) = X\) for all but finitely many variables \(X\). We shall write

\(^1\) A machine-checked proof of all the results using the theorem prover Isabelle can be found at http://www.cl.cam.ac.uk/~cu200/Unification.
dom(σ) for the finite set of variables X satisfying σ(X) ≠ X. If dom(σ) consists of variables X₁, ..., Xₙ and σ(Xᵢ) = tᵢ for i = 1..n, we shall sometimes write σ as

\[ \sigma = [X₁ := t₁, ..., Xₙ := tₙ]. \]  

We write σ(t) for the result of applying a substitution σ to a term t; this is the term obtained from t by replacing each suspension π·X in t (as X ranges over dom(σ)) by the term π·σ(X) got by letting π act on the term σ(X) using the definition in Fig. 1. For example, if σ = [X := (b,Y)] and t = a.(a b)·X, then σ(t) = a.(a, (a b)·Y). Given substitutions σ and σ’, and freshness environments ∇ and ∇’, we write

\[ (a) \ ∇' \vdash σ(∇) \quad \text{and} \quad (b) \ ∇ \vdash σ \approx σ' \]  

(9)
to mean that (for a) ∇’ ⊨ a # σ(X) holds for each (a # X) ∈ ∇ and (for b) ∇ ⊨ σ(X) ≈ σ’(X) holds for all X ∈ dom(σ) ∪ dom(σ’).

Lemma 2 (Substitution). Substitution commutes with the permutation action: σ(π·t) = π·σ(t). Substitution preserves ≈ and # in the following sense:

- if ∇’ ⊨ σ(∇) and ∇ ⊨ t ≈ t’, then ∇’ ⊨ σ(t) ≈ σ(t’);
- if ∇’ ⊨ σ(∇) and ∇ ⊨ a # t, then ∇’ ⊨ a # σ(t).

Proof. The first sentence follows by induction on the structure of t. The second follows by induction on the proofs of ∇ ⊨ t ≈ t’ and ∇ ⊨ a # t from the rules in Fig. 2, using the first sentence and the (proof of) Lemma 1. \(\square\)

We claim that the relation ≈ defined in Fig. 2 gives the correct notion of α-equivalence for terms over a nominal signature. This is reasonable, given Lemma 1 and the fact that, by definition, it satisfies rules (≈-abstraction-1) and (≈-abstraction-2). Further evidence is provided by the following theorem, which shows that for ground terms ≈ agrees with the following more traditional definition of α-equivalence.

Definition 5 (Naïve α-equivalence). Define the binary relation t =ₐ t’ between terms over a nominal signature to be the least sort-respecting congruence relation satisfying a.t =ₐ b.[a→b]t whenever b is an atom (of the same sort as a) not occurring at all in t. Here [a→b]t indicates the result of replacing all free occurrences of a with b in t.

Theorem 1 (Adequacy). If t and t’ are ground terms (i.e. terms with no variables and hence no suspensions) over a nominal signature, then the relation t =ₐ t’ of Def. 5 holds if and only if ∅ ⊨ t ≈ t’ is provable from the rules in Fig. 2. Furthermore, ∅ ⊨ a # t is provable if and only if a is not in the set FA(t) of free atoms of t.

Proof. The proof is similar to the proof of [11, Proposition 2.2]. \(\square\)

For non-ground terms, the relations =ₐ and ≈ differ. For example a.X =ₐ b.X always holds, whereas ∅ ⊨ a.X ≈ b.X is not provable unless a = b. This disagreement is to be expected, since we noted in the Introduction that =ₐ is not preserved by substitution, whereas from Lemma 2 we know that ≈ is.

Remark 1. Further evidence for the status of ≈ and # is provided by a natural interpretation of judgements provable from the rules in Fig. 2 in the universe of FM-sets [11]. The details will appear in the full version of this paper.
3 Unification

Given terms $t$ and $t'$ of the same sort over a nominal signature, can we decide whether or not there is a substitution of terms for the variables in $t$ and $t'$ that makes them equal in the sense of the relation $\approx$ introduced in the previous section? Since instances of $\approx$ in general are established modulo freshness constraints, it makes more sense to ask whether or not there is both a substitution $\sigma$ and a freshness environment $\nabla$ for which $\nabla \vdash \sigma(t) \approx \sigma(t')$ holds. As for ordinary first-order unification, solving such an equational problem may throw up several equational subproblems; but an added complication here is that because of rule ($\approx$-abstraction-2) in Fig. 2, equational problems may generate freshness problems, i.e. ones involving the relation $\#$.

We are thus led to the following definition of unification problems for nominal equational logic.

**Definition 6.** A unification problem $P$ over a nominal signature is a finite set of atomic problems, each of which is either an **equational problem** $t \approx t'$ where $t$ and $t'$ are terms of the same sort over the signature, or a **freshness problem** $a \# t$ where $a$ is an atom and $t$ a term over the signature. A solution for $P$ consists of a pair $(\nabla, \sigma)$ where $\nabla$ is a freshness environment and $\sigma$ is a substitution satisfying

- $\nabla \vdash a \# \sigma(t)$ for each $(a \#? t) \in P$ and
- $\nabla \vdash \sigma(t) \approx \sigma(t')$ for each $(t \approx? t') \in P$.

Such a pair is a **most general solution** for $P$ if given any other solution $(\nabla', \sigma')$, then there is a substitution $\sigma''$ satisfying $\nabla' \vdash \sigma''(\nabla)$ and $\nabla' \vdash \sigma'' \circ \sigma \approx \sigma'$. (Here we have used the notation of (9); and $\sigma'' \circ \sigma$ denotes the substitution composition of $\sigma$ followed by $\sigma''$, given by $(\sigma'' \circ \sigma)(X) \overset{\text{def}}{=} \sigma''(\sigma(X))$.)

**Theorem 2 (Nominal unification).** There is an algorithm which, given any nominal unification problem, decides whether or not it has a solution and if it does, returns a most general solution.

**Proof.** We describe an algorithm using labelled transformations directly generalising the presentation of first-order unification in [17, Sect. 2.6], which in turn is based upon the approach in [18]. (See also [1, Sect. 4.6] for a detailed exposition, but not using labels.) We use two types of labelled transformation between unification problems, namely

$P \overset{\sigma}{\rightarrow} P'$ and $P \overset{\nabla}{\rightarrow} P'$

where the substitution $\sigma$ is either the identity $\varepsilon$, or a single replacement $[X := t]$; and where the freshness environment $\nabla$ is either empty $\emptyset$, or a singleton $\{a \# X\}$. The legal transformations are given in Fig. 3. This figure uses the notation $P \cup P'$ to indicate disjoint union of problem sets; and the notation $\sigma P$ to indicate the problem resulting from applying the substitution $\sigma$ to all the terms occurring in the problem $P$.

Given a unification problem $P$, the algorithm proceeds in two phases. In the first phase it applies as many $\overset{\sigma}{\rightarrow}$ transformations as possible (non-deterministically). If this results in a problem containing no equational subproblems then it proceeds to the second phase; otherwise it halts with failure. In the second phase it applies as many $\overset{\nabla}{\rightarrow}$ transformations as possible (non-deterministically). If this does not result in the empty
Then failure happens if there are certain subproblems manifestly having no solutions for a problem. From these and the fact that \( P \) (where \( P \) is a sequence of the form \( P(a.t) \)) shows that each phase of the algorithm must terminate. So one just has to show that if the algorithm succeeds on \( P(a) \) if the algorithm fails on \( P(b) \) if the algorithm succeeds on \( P \). Therefore part (a) is a consequence of the following two properties of transformations, where we write \( \mathbb{U}(P) \) for the set of all solutions for a problem \( P \):

\[
\begin{align*}
\text{if } (\nabla', \sigma') &\in \mathbb{U}(P) \text{ and } P \xrightarrow{\sigma_1} P', \text{ then } (\nabla', \sigma') \in \mathbb{U}(P') \text{ and } \nabla' \vdash \sigma' \circ \sigma \approx \sigma' &\quad (11) \\
\text{if } (\nabla', \sigma') &\in \mathbb{U}(P) \text{ and } P \xrightarrow{} P', \text{ then } (\nabla', \sigma') \in \mathbb{U}(P') \text{ and } \nabla' \vdash \sigma'(\nabla). &\quad (12)
\end{align*}
\]

For part (b), one first shows

\[
\begin{align*}
\text{if } (\nabla', \sigma') &\in \mathbb{U}(P') \text{ and } P \xrightarrow{\sigma} P', \text{ then } (\nabla', \sigma' \circ \sigma) \in \mathbb{U}(P) &\quad (13) \\
\text{if } (\nabla', \sigma') &\in \mathbb{U}(P'), \text{ then } P \xrightarrow{\sigma} P' \text{ and } P' \xrightarrow{\sigma'} P' \text{ and } \nabla'' \vdash \sigma'(\nabla), \text{ then } (\nabla' \cup \nabla'', \sigma') \in \mathbb{U}(P). &\quad (14)
\end{align*}
\]

From these and the fact that \((\emptyset, \varepsilon) \in \mathbb{U}(\emptyset)\), one gets that if a sequence like (10) exists, then \((\nabla, \sigma) \overset{\text{def}}{=} (\nabla_1 \cup \cdots \cup \nabla_m, \sigma_n \circ \cdots \circ \sigma_1)\) is in \( \mathbb{U}(P) \). Furthermore from (11)}
and (12), we get that any other solution $(\nabla', \sigma') \in \mathcal{U}(P)$ satisfies $\nabla' \vdash \sigma'(\nabla)$ and $\nabla' \vdash \sigma' \circ \sigma \approx \sigma'$, so that $(\nabla, \sigma)$ is indeed a most general solution. □

Example 2. Using the first three function symbols of the nominal signature of Example 1 to represent $\emptyset$ necessarily preserve the validity of the judgements in Fig. 2. For example, Nominal unification unifies variables, but it does not preserve the validity of the judgements in Fig. 2.

Applying the nominal unification algorithm described above, we find that

- $P_1$ has no solution;
- $P_2$ has a most general solution given by $\nabla_2 = \emptyset$ and $\sigma_2 = [X_2 := \forall r b, X_3 := \forall r a]$;
- $P_3$ has a most general solution given by $\nabla_3 = \emptyset$ and $\sigma_3 = [X_4 := (a b) \cdot X_5]$;
- $P_4$ has a most general solution given by $\nabla_4 = \{b \# X_7\}$ and $\sigma_3 = [X_6 := (b a) \cdot X_7]$.

Derivations for $P_1$ and $P_3$ are sketched in Fig. 4. Using the Adequacy Theorem 1, one can interpret these solutions as the following statements about the $\lambda$-terms from the quiz.

**Quiz answers**

1. There is no $\lambda$-term $M_1$ making the first pair of terms $\alpha$-equivalent.
2. The only solution for the second problem is to take $M_2 = b$ and $M_3 = a$.
3. For the third problem we can take $M_5$ to be any $\lambda$-term, so long as we take $M_4$ to be the result of swapping all occurrences of $a$ and $b$ throughout $M_4$.
4. For the last problem, we can take $M_7$ to be any $\lambda$-term that does not contain free occurrences of $b$, so long as we take $M_6$ to be the result of swapping all occurrences of $b$ and $a$ throughout $M_7$, or equivalently (since $b$ is not free in $M_7$), taking $M_6$ to be the result of replacing all free occurrences of $a$ in $M_7$ with $b$.

**Remark 2 (Atoms are not variables).** Nominal unification unifies variables, but it does not unify atoms. Indeed the operation of identifying two atoms by renaming does not necessarily preserve the validity of the judgements in Fig. 2. For example, $\emptyset \vdash a.b \approx c.b$ holds if $b \neq a, c$; but renaming $b$ to be $a$ in this judgement we get $\emptyset \vdash a.a \approx c.a$, which does not hold so long as $a \neq c$. Referring to Def. 2, you will see that we do allow variables ranging over sorts of atoms; and such variables can be unified like any other variables. However, if $A$ is such a variable, then it cannot appear in abstraction position, i.e. as $A.t$. This is because we specifically restricted abstraction to range over atoms, rather than over arbitrary terms of atom sort. Such a restriction seems necessary to obtain single, most general, solutions to nominal unification problems. For without such a restriction, because of rule $(\approx\text{-abstraction-2})$ we would also have to allow variables to appear on the left-hand side of freshness relations and in suspended permutations. So then we would get unification problems like $\{(A B) \cdot C \approx? C\}$, where $A$, $B$ and $C$ are variables of atom sort; this has two incomparable solutions, namely $(\emptyset, [A := B])$ and $(\{A \# C, B \# C\}, \varepsilon)$. 

Nominal Unification 523
that work is Miller’s valence by capture-avoiding substitution of terms for function variables. Notable among
\( \lambda \)-higher-order unification, i.e. solving equations between

4 Related Work

Most previous work on unification for languages with binders is based on forms of
higher-order unification, i.e. solving equations between \( \lambda \)-terms modulo \( \alpha \beta \eta \)-equivalence by capture-avoiding substitution of terms for function variables. Notable among
that work is Miller’s higher-order pattern unification used in his \( L_\lambda \) logic programming language [20]. This kind of unification retains the good properties of first-order unification:
a linear-time decision procedure and existence of most general unifiers. However it imposes a restriction on the form of \( \lambda \)-terms to be unified; namely that function variables may only be applied to distinct bound variables. An empirical study by Michaylov and Pfenning [19] suggests that most unifications arising dynamically in higher-order logic programming satisfy Miller’s restriction, but that it rules out some useful programming idioms. For us, the main disadvantage of \( L_\lambda \) is one common to most approaches
based on higher-order abstract syntax: one cannot directly express the common idiom
of possibly-capturing substitution of terms for metavariables. Instead one has to replace metavariables, \( X \), with function variables applied to distinct lists of (bound) variables, \( F x_1 \ldots x_n \), and use capture-avoiding substitution.

Dowek et al [8] present a version of higher-order pattern unification for \( \lambda \sigma \) (a \( \lambda \)-calculus with de-Bruijn indices and explicit substitutions) in which unification problems are solved, like in nominal unification, by textual replacements of terms for variables; however a ‘pre-cooking’ operation ensures that the textual replacements can be (faithfully) related to capture-avoiding substitutions. It seems that nominal unification problems can be encoded into higher-order pattern unification problems using a non-trivial translation (the details of this encoding still remain to be investigated). But even if it turns out that it is possible to simulate nominal unification by higher-order pattern unification, the calculations involved in translating our terms into higher-order pattern

\[ P_1 \xrightarrow{\tau} \{ fn \ b \ . \ app \ X_1 \ , \ vr \ b \} \cong \ ? \ fn \ b \ . \ app \ ( vr \ b \ , \ ( a \ b ) \cdot X_1 ) \cdot \ a \ \# \ ? \ fn \ a \ . \ app \ ( vr \ a \ , \ X_1 ) \} \}
\]

\( \xrightarrow{\tau} \{ app \ X_1 \ , \ vr \ b \} \cong \ ? \ app \ ( vr \ b \ , \ ( a \ b ) \cdot X_1 ) \cdot \ a \ \# \ ? \ fn \ a \ . \ app \ ( vr \ a \ , \ X_1 ) \} \)

\( \ldots \)

\( \xrightarrow{\tau} \{ X_1 \cong \ ? \ vr \ b \ \\
\text{vr} \ b \cong \ ? \ ( a \ b ) \cdot X_1 \cdot \ a \ \# \ ? \ fn \ a \ . \ app \ ( vr \ a \ , \ X_1 ) \} \)

\( \xrightarrow{\tau} \{ vr \ b \cong \ ? vr \ a \cdot a \ \# \ ? \ fn \ a \ . \ app \ ( vr \ a \ , \ vr \ b ) \} \text{ with } \sigma = \{ X_1 := vr \ b \} \)

\( \xrightarrow{\tau} b \cong \ ? a \cdot a \ \# \ ? \ fn \ a \ . \ app \ ( vr \ a \ , vr \ b ) \}

FAIL

\[ P_4 \xrightarrow{\tau} \{ fn \ b . \ app \ ( vr \ b \ , \ X_6 ) \cong \ ? \ fn \ a \ . \ app \ ( vr \ a \ , \ X_7 ) \} \}
\]

\( \xrightarrow{\tau} \{ app \ ( vr \ b \ , \ X_6 ) \cong \ ? \ app \ ( vr \ b \ , \ ( a \ b )\cdot X_7 ) \cdot \ b \ \# \ ? \ app \ ( vr \ a \ , \ X_7 ) \} \}
\]

\( \ldots \)

\( \xrightarrow{\tau} \{ b \cong \ ? b \cdot X_6 \cong \ ? ( a \ b )\cdot X_7 \cdot \ b \ \# \ ? \ app \ ( vr \ a \ , \ X_7 ) \} \}
\]

\( \sigma \xrightarrow{\tau} \{ b \cong \ ? app \ ( vr \ a \ , \ X_7 ) \} \text{ with } \sigma = \{ X_6 := ( a \ b )\cdot X_7 \} \)

\( \xrightarrow{\tau} b \cong \ ? ( vr \ a \ , \ X_7 ) \}
\]

\( \sigma \xrightarrow{\tau} \{ b \cong \ ? ( vr \ a \ , \ X_7 ) \}
\]

\( \sigma \xrightarrow{\tau} \{ b \cong \ ? X_7 \}
\]

\( \nabla \xrightarrow{\sigma} \emptyset \text{ with } \nabla = \{ b \ # \ X_7 \}
\)

Fig. 4. Example derivations.
terms and then using higher-order pattern unification are far more intricate than our simple algorithm that solves nominal unification problems directly.

Hamana [13, 14] manages to add possibly-capturing substitution to a language like Miller’s $L_\lambda$. This is achieved by adding syntax for explicit renaming operations and by recording implicit dependencies of variables upon bindable names in a typing context. The mathematical foundation for Hamana’s system is the model of binding syntax of Fiore et al [9]. The mathematical foundation for our work appeared concurrently [10] and is in a sense complementary. For in Hamana’s system the typing context restricts which terms may be substituted for a variable by giving a finite set of names that must contain the free names of such a term; whereas we give a finite set of names which the term’s free variables must avoid. Since $\alpha$-conversion is phrased in terms of avoidance, i.e. freshness of names, our approach seems more natural if one wants to compute $\alpha$-equivalences concretely. On top of that, our use of name permutations, rather than arbitrary renaming functions, leads to technical simplifications. In any case, the bottom line is that Hamana’s system seems more complicated than the one presented here and does not possess most general unifiers.

5 Conclusion

In this paper we have proposed a solution to the problem of finding possibly-capturing substitutions that unify terms involving binders up to $\alpha$-conversion. To do so we considered a many-sorted first-order term language with distinguished collections of constants called atoms and with atom-abstraction operations for binding atoms in terms. This provides a simple, but flexible, framework for specifying binding operations and their scopes, in which the bound entities are explicitly named. By using variables prefixed with suspended permutations, one can have substitution of terms for variables both allow capture of atoms by binders and respect $\alpha$-equivalence (renaming of bound atoms). The definition of $\alpha$-equivalence for the term language makes use of an auxiliary freshness relation between atoms and terms which generalises the ‘not a free atom of’ relation from ground terms to terms with variables; furthermore, because variables stand for unknown terms, hence with unknown free atoms, it is necessary to make hypotheses about the freshness of atoms for variables in judgements about term equivalence and freshness. This reliance on ‘freshness’ is the main novelty—it arises from the work reported in [11, 23]. It leads to a new notion of unification problem in which instances of both equivalence and freshness have to be solved by giving term-substitutions and (possibly) freshness conditions on variables in the solution. We showed that this unification problem is decidable and unitary.

Cheney and Urban [5] are investigating the extent to which nominal unification can be used in resolution-based proof search for a form of first-order logic programming for languages with binders (with a view to providing better machine-assistance for structural operational semantics). Such a logic programming language should permit a concrete, ‘nominal’ approach to bound entities in programs while ensuring that computation (which in this case is the computation of answers to queries) respects $\alpha$-equivalence between terms. This is illustrated with the Prolog-like program in Fig. 5, which implements a simple typing algorithm for $\lambda$-terms. Interesting is the third clause. First, note the term $(\lambda x. M)$, which unifies with any $\lambda$-abstraction. The binder $x,$
roughly speaking, has in the ‘nominal’ approach a value which can be used in the body of the clause, for example for adding \((\text{pair } x \ A)\) to the context \(\Gamma\). Second, the freshness constraint \(x\#\Gamma\) ensures that \(\Gamma\) cannot be replaced by a term that contains \(x\) freely. Since this clause is intended to implement the usual rule for typing \(\lambda\)-abstractions

\[
\Gamma \vdash \lambda x. M : A \subseteq B
\]

its operational behaviour is given by: choose fresh names for \(\Gamma\), \(x\), \(M\), \(A\) and \(B\) (this is standard in Prolog-like languages), unify the head of the clause with the goal formula, apply the resulting unifier to the body of the clause and make sure that \(\Gamma\) is not replaced by a term that contains freely the fresh name we have chosen for \(x\).

Similar facilities for \textit{functional programming} already exist in the FreshML language, built upon the same foundations: see [27] and www.freshml.org. We are also interested in the special case of ‘nominal matching’ and its application to term-rewriting modulo \(\alpha\)-equivalence.

If these applications show that nominal unification is practically useful, then it becomes important to study its complexity. The presentations of the term language in Section 2 and of the algorithm in Section 3 were chosen for clarity and to make the proof of correctness easier\(^2\) rather than for efficiency. In any case, it remains to be investigated whether the swapping and freshness computations that we have added to ordinary, first-order unification result in greater than linear-time complexity.

\textbf{Acknowledgements:} We thank Gilles Dowek, Roy Dyckhoff, Dale Miller, Francois Pottier and Helmut Schwichtenberg for comments on this work. This research was supported by UK EPSRC grants GR/R29697 (Urban) and GR/R07615 (Pitts and Gabbay).

\section*{References}


\(^2\) See \url{http://www.cl.cam.ac.uk/~cu200/Unification} for the Isabelle proof scripts.


Abstract. In contrast to hardware which is finite-state and based on relatively few ample principles, software systems generally give rise to infinite state spaces, and are described in terms of programming languages involving rich semantical concepts. The challenges of software verification can be addressed only by a combined effort of different communities including, most notably, model checking, theorem proving, symbolic computation, static analysis, compilers, and abstract interpretation. We focus on a recent family of tools which use predicate abstraction and theorem proving to extract a finite state system amenable to model checking.

The last fifteen years have seen model checking mature from a purely academic pursuit to an important tool for hardware developers in industry which complements more traditional techniques such as simulation and testing. The main practical problem in model checking is the state explosion problem, i.e., the enormous size of the finite-state system. In addition to information-preserving symbolic methods using BDDs or SAT procedures, abstraction is the most important method to combat state explosion. The critical problem is how to establish an appropriate level of abstraction of the system to make the analysis tractable: Too fine a level of granularity leads to a large state space, while a coarse abstraction may result in false negatives. As manual abstraction is tedious and error-prone, counterexample-guided abstraction methods are used to automatically improve an initial abstraction mapping until a good abstraction is found.

Simultaneously, (semi)manual proof assistants have also been used successfully for hardware verification tasks involving intricate mathematical reasoning, for example, to assert the correctness of floating point arithmetic.

As hardware verification has matured, and the number of critical software applications is constantly increasing, software verification has attracted renewed interest during the last years. Software however abounds in features not present in hardware, and programs do in general not yield finite state models. Specific

* This research was supported by the Austrian Science Fund Project N Z29-N04, the EU Research and Training Network GAMES, and the Office of Naval Research (ONR). The views and conclusions contained in this document are those of the author.
concepts distinguishing hardware from software include complex base types such as strings and high precision floating point variables, dynamic data structures, object orientation, user-defined types and classes, procedure calls, recursion, function variables, dynamic method lookup, overloading, templates, include files, exceptions, callbacks, use of files and databases, absent source code for libraries and system calls, as well as embedded, mobile and self-modifying code. It is thus evident that software verification is a highly interdisciplinary enterprise which has to draw on the experience and methods from diverse fields including theorem proving, symbolic computation, compilers, static analysis, and abstract interpretation. Despite differences in tradition, mathematical schools, and engineering aptitude, the collaboration between these fields is steadily increasing.

Predicate abstraction [5] is a paradigmatic approach to combine theorem proving, abstraction, and model checking. In predicate abstraction, the abstract model is described in terms of logical formulas (“predicates”) ranging over the concrete model; the relationships between the predicates are computed using a theorem prover, and the abstract model is then verified by a model checker. Several model checkers for C based on predicate abstraction have been developed, in particular SLAM [1,2] which is targeted at device drivers, BLAST [7] for verifying embedded systems using a lazy abstraction technique, and MAGIC [3] whose focus is on modular verification and security applications. The verification tools use automated theorem provers, efficient decision procedures, and static analysis to extract an abstract model, and employ counterexample-guided refinement techniques for improving the abstraction.

References

More Computation Power for a Denotational Semantics for First Order Logic

Kess F.M. Vermeulen
CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
kees.vermeulen@cwi.nl

Abstract. This paper starts from a denotational semantics for first order logic proposed by Apt. He interprets first order logic as a programming language within the declarative paradigm, but different from the usual approach of logic programming. For his proposal Apt obtains soundness with respect to the standard interpretation of first order logic. Here we consider the expressivity (or: computation power) of the system proposed, i.e., the programs for which no error messages are produced. To gain computational realism a decrease in expressivity compared to first order logic is inevitable. In [1] the expressivity was compared with logic programming with both positive and negative results. Here we consider three ways of improving the situation: replacing the original interpretation of conjunction-as-sequential-composition by a symmetric interpretation of conjunction; extending the framework with the option of recursive definitions by Clark completions; replacing the interpretation of disjunction-as-choice by an interpretation which is in the style of ‘backtracking’: only produce an error if all options lead to error. For each improvement we obtain a soundness result. The three improvements are developed independently in a way that makes sure that also their combination preserves soundness. The combination gives a considerable improvement of the expressivity of the denotational semantics that Apt proposed, both in the comparison with first order logic and in the comparison with logic programming.

Keywords: declarative programming, denotational semantics, expressivity of programming languages

1 Introduction

The declarative programming paradigm has a lot to offer. It treats simple declarative statements as instructions to investigate how to make the statement true: programs make you dreams come true. Logic programming (LP) has shown that such promises are not always beyond the scope of reason. Still, as was pointed out in Apt’s paper [1], there are drawbacks inherent in logic programming. In particular, the Horn clause fragment, the fragment initially considered in Kowalski [7], is too small for serious programming purposes. Extensions of the fragment to arbitrary first order formulas — see, for example [8] and [2] — turned out to complicate matters considerably and the attempts did not converge to a commonly accepted solution. Second, in logic programming computation relies heavily on
the term model. Or, to put it differently, it is confined to the Herbrand universe. This makes it hard for programmers to benefit from specific properties of the model for which their programs are written. The enhancement of logic programming to constraint programming (CLP) improves the situation\textsuperscript{1}, but it does so by introducing an oracle: the constraint solver.

In Apt\textsuperscript{[1]} these points are the motivation for a new attempt for the interpretation of first order logic as a programming language. These theoretical developments go hand in hand with the development of Alma\textsubscript{0}, as described in \textsuperscript{[3]}. In fact, \textsuperscript{[1]} tries to give a denotational semantics for the declarative fragment of Alma\textsubscript{0}.

In the proposed set up ordinary terms are replaced by \( J \)-terms. The second important ingredient of \textsuperscript{[1]} is its treatment of equations: they sometimes work as an instruction for unification, sometimes as a test for ‘real’ equality. This way the spirit of computation of logic programming is preserved and at the same time the gap between the level of terms and the intended model is narrowed. The interpretation of equations is then extended in a natural way to the full first order language, where computational realism is ensured by the introduction of a special value for \textit{error}. The result is a natural interpretation of first order logic as a programming language which is sound with respect to the standard interpretation. But, as was already pointed out in \textsuperscript{[1]}, the comparison with LP is not decisive: there are clear examples where Apt’s proposal works better, but also examples where LP outperforms the denotational semantics.

Hence it is clearly desirable to further extend the computation power of the proposal, i.e., the set of programs that does not give rise to an error statement. The expressivity question was taken up by Smaus\textsuperscript{[10]} for equations. Here we discuss three other ways to improve the expressivity of the interpretation. First we reconsider the interpretation of conjunction. Apt interprets \( \land \) as sequential composition. This introduces an order sensitivity into the system that seriously restricts its expressivity. Hence we propose an alternative interpretation for \( \land \) which is symmetric, i.e., not sensitive to order. Next we include the option of definition by recursion for predicates. We use ideas from \textsuperscript{[4]} and use Clark completions of logic programs. Finally, we consider the interpretation of \( \lor \). Apt\textsuperscript{[1]} interprets \( \lor \) as choice. Hence, if one of the disjuncts produces an error message the overall system reproduces this message for the disjunction as a whole. Here we prevent this propagation of \textit{errors} and replace the treatment of \( \lor \) with an interpretation reminiscent of backtracking: \textit{error} is reported only if none of the choices gives a proper outcome.

This gives us three ways of improving the system \textit{qua} expressivity. They are considered independently and each improvement is shown to respect \textit{soundness} with respect to first order logic. It will be clear that they can be combined to gain a considerable overall improvement of expressivity in a sound way.

In what follows we start by summarizing the proposal by Apt (section\textsuperscript{[2]}). Then, the three improvements are presented (in sections\textsuperscript{[3], [4] and [5]}). Finally, we sum up the results in the conclusion (section\textsuperscript{[6]} where we also present additional

\textsuperscript{1} Consult \textsuperscript{[9]} and \textsuperscript{[6]} for more information.
discussion and options for further work. The overall conclusion will be that by
the proposed improvements we obtain a considerable gain in expressivity for the
approach of Apt [1] and hence achieve an improvement in both the comparison
with LP and the standard interpretation of the first order language. Thereby
the denotational semantics becomes an even more attractive way of working out
the idea of declarative programming and starts to look like a serious alternative
to LP.

2 The Semantic Toolkit

We start by recalling the semantics provided by Apt in [1]. The idea of this
semantics is to provide a uniform computational meaning for the first-order for-

mulas independent of the underlying interpretation and without a constraint
store. This yields a limited way of processing formulas in the sense that occasion-
ally an error may arise. Let us recall the relevant definitions:

**Definition 1.** Consider a language of terms L and an algebra J for it.

- Consider a term of L in which we replace some of the variables by the el-
ements of the domain of J. We call the resulting object a generalized term.
- Given a generalized term t we define its J-evaluation as follows. Each ground
term of L evaluates to a unique value in J. In a generalized term t we re-
place each maximal ground subterm of t by its value in J (maximal partial
evaluation). The resulting generalized term is called a J-term and denoted
\[ [t]_J \].
- A J-substitution is a finite mapping from variables to J-terms that assigns
to each variable x in its domain a J-term different from x. The notation is:
\[ \{x_1/h_1, \ldots, x_n/h_n\} \]. We write \[ \epsilon \] for the J-substitution with the empty
domain. Application of a J-substitution \[ \theta \] to a generalized term \[ t \] is defined
in the standard way and denoted by \[ t\theta \]. The set of J-substitutions is called
Subs.
- The composition of two J-substitutions \[ \theta \] and \[ \eta \], is written as \[ \theta \eta \] and is
defined as the unique J-substitution \[ \gamma \] such that for a variable \[ x \]
\[ x\gamma = [[(x\theta)\eta]]_J. \]

J-substitutions generalize both the usual notion of substitution and the no-
tion of a valuation, i.e., the assignment of domain values to variables. Now we
use J-substitutions to interpret equations as follows: (Here and elsewhere we do
not indicate the dependency of the semantics on the underlying interpretation
or algebra.)

\[ [s = t](\theta) := \begin{cases}
\{\theta\{s\theta/[t\theta]_J\}\} & \text{if } s\theta \text{ is a variable that does not occur in } t\theta, \\
\{\theta\{t\theta/[s\theta]_J\}\} & \text{if } t\theta \text{ is a variable that does not occur in } s\theta \\
\{\theta\} & \text{if } [s\theta]_J \text{ and } [t\theta]_J \text{ are identical,} \\
\{\text{error}\} & \text{if } s\theta \text{ and } t\theta \text{ are ground and } [s\theta]_J \neq [t\theta]_J, \\
\emptyset & \text{otherwise.}
\end{cases} \]
Now consider an interpretation $I$ based on the algebra $J$. Given an atomic formula $A(t_1, \ldots, t_n)$ and a $J$-substitution $\theta$ we denote the interpretation of $A$ in $I$ by $A_I$. We say that:

- $A(t_1, \ldots, t_n)\theta$ is \textit{true} if $A(t_1, \ldots, t_n)$ is ground and $(\llbracket t_1\theta \rrbracket_J, \ldots, \llbracket t_n\theta \rrbracket_J) \in A_I$,

- $A(t_1, \ldots, t_n)\theta$ is \textit{false} if $A(t_1, \ldots, t_n)$ is ground and $(\llbracket t_1\theta \rrbracket_J, \ldots, \llbracket t_n\theta \rrbracket_J) \notin A_I$.

Note that this way the interpretation of $A$ is fixed by $I$. In LP terminology: it is a \textquote{built-in} predicate. We do not (yet) have the option of defining predicates by recursion. To deal with existential quantification we use the $\text{DROP}_x$ operation which is defined as follows:

\[
\begin{align*}
x\text{DROP}_x(\theta) & := x \\
y\text{DROP}_x(\theta) & := y\theta \text{ for } y \neq x
\end{align*}
\]

$\text{DROP}_x$ is extended pointwise to subsets of $\text{Subs} \cup \{\text{error}\}$. This way we add \textit{error} as a kind of \textquote{special substitution}.

For \textit{error} we set: $[\phi](\text{error}) := \{\text{error}\}$. The following example clarifies the way we interpret atoms and conjunction.

\textbf{Example 1.} Consider the standard algebra for the language of arithmetic with the set of integers as domain. We denote its elements by $\ldots, -2, -1, 0, 1, 2, \ldots$. Each constant $i$ evaluates to the element $i$. Then we have:

1. $[y = z - 1 \land z = x + 2](\{x/1\}) = [z = x + 2](\{x/1, y/z - 1\}) = \{x/1, y/2, z/3\}$,

2. $[y = 1 \land z = 1 \land y - 1 = z - 1](\epsilon) = \{y/1, z/1\}$,

3. $[y = 1 \land z = 2 \land y < z](\epsilon) = \{y/1, z/2\}$,

4. $[x = 0 \land \neg(x = 1)](\epsilon) = \{x/0\}$,

5. $[y - 1 = z - 1 \land y = 1 \land z = 1](\epsilon) = \{\text{error}\}$,

6. $[y < z \land y = 1 \land z = 2](\epsilon) = \{\text{error}\}$,

7. $\neg(x = 1) \land x = 0](\epsilon) = \{\text{error}\}$.

\footnote{Here and elsewhere we write $t$ for a sequence $t_1, \ldots, t_n$.}
In this semantics conjunction is not commutative: it is sequential composition. Consequently it is important in which order the formulas are processed. Also, all error assignments are propagated, without exception.

In [1] (and also in [3]) the semantics is shown to be sound w.r.t. first order logic, i.e.:
- \( \eta \in \llbracket \phi \rrbracket (\theta) \Rightarrow \models_I \phi \eta \)
- \( \llbracket \phi \rrbracket (\theta) = \emptyset \Rightarrow \models_I \neg \phi \theta \)

This is a crucial result: it shows that the answers that the programs produce are reliable, as long as there are no error messages. But it does not show how often we get an error message. In this paper we take up this question and investigate how to get more successful runs for more programs.

### 3 Symmetric Conjunction

We refer to the full first order language as \( L_\land \). We compare it to \( L_\overline{\land} \), the language in which the binary connective \( \overline{\land} \) replaces \( \land \). \( \overline{\land} \) is a symmetric conjunction and is interpreted as follows:

\[
\llbracket \phi_1 \overline{\land} \phi_2 \rrbracket (\theta) = \llbracket \phi_1 \rrbracket (\llbracket \phi_2 \rrbracket (\theta)) \cup \llbracket \phi_2 \rrbracket (\llbracket \phi_1 \rrbracket (\theta))
\]

Hence \( \phi_1 \overline{\land} \phi_2 \) tells us to compute the \( \phi_i \) (\( i = 1, 2 \)) in arbitrary order. This is one way to get out of the problem noted in the example: we have simple local instructions to consider both orders for the execution of the \( \phi_i \).

Note that \( L_\overline{\land} \) inherits the soundness of \( L_\land \): \( \phi_1 \overline{\land} \phi_2 \in L_\overline{\land} \) is interpreted as the union of the interpretations of the formulas \( \phi_1 \land \phi_2 \) and \( \phi_2 \land \phi_1 \in L_\land \), both of which have the same truth value in first order logic. As the semantics of \( \land \) respects this truth value in a sound way, also the semantics of \( \overline{\land} \) has to be sound. More formally we note that:

\[
\begin{align*}
\eta \in \llbracket \phi_1 \overline{\land} \phi_2 \rrbracket (\theta) & \Rightarrow \\
\eta \in \llbracket \phi_1 \land \phi_2 \rrbracket (\theta) \text{ or } \eta \in \llbracket \phi_2 \land \phi_1 \rrbracket (\theta) & \Rightarrow \text{ (by soundness of } L_\land \text{)} \\
\models_I \phi \eta \text{ for both } i = 1, 2 & \Rightarrow \\
\models_I (\phi_1 \overline{\land} \phi_2) \eta &
\end{align*}
\]

and a similar argument works for the case where \( \llbracket \phi_1 \overline{\land} \phi_2 \rrbracket (\theta) = \emptyset \).

We want to compare executions of one formula \( \phi \in L_\overline{\land} \) with computations of all the formulas in \( L_\land \) that can be obtained by taking permutations of the conjuncts that occur in \( \phi \). For this purpose we introduce the following translation:

**Definition 2.** The permutation translation \( \pi(\cdot) : L_\overline{\land} \rightarrow L_\land \) is given by the following table:

<table>
<thead>
<tr>
<th>( \phi \in L_\overline{\land} )</th>
<th>( \pi(\phi) \in L_\land )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s = t )</td>
<td>( s = t )</td>
</tr>
<tr>
<td>( A(t_1, \ldots, t_n) )</td>
<td>( A(t_1, \ldots, t_n) )</td>
</tr>
<tr>
<td>( \phi_1 \lor \phi_2 )</td>
<td>( \pi(\phi_1) \lor \pi(\phi_2) )</td>
</tr>
<tr>
<td>( \neg \phi )</td>
<td>( \neg \pi(\phi) )</td>
</tr>
<tr>
<td>( \exists x \phi )</td>
<td>( \exists x \pi(\phi) )</td>
</tr>
<tr>
<td>( \phi_1 \overline{\land} \phi_2 )</td>
<td>( (\pi(\phi_1) \land \pi(\phi_2)) \lor (\pi(\phi_2) \land \pi(\phi_1)) )</td>
</tr>
</tbody>
</table>
Proposition 1. For every $\phi$ and $\psi$, the cost of including $\phi \land \psi$ as soon as we consider conjunctions of three formulas. Consider, for given $\pi \in \mathcal{L}_\land$ the way down’ in the subformulas $\phi_i$ ($i = 1, 2$). This way of translating has its limits: it respects the bracket structure of larger conjunctions. This shows up as soon as we consider conjunctions of three formulas. Consider, for given $\phi_1$, $\phi_2$ and $\phi_3$: $\psi_1 = \phi_1 \land (\phi_2 \land \phi_3)$ and $\psi_2 = (\phi_1 \land \phi_2) \land \phi_3$. Then we find:

\[
\begin{align*}
o \pi(\psi_1) &= (\phi_1 \land (\phi_2 \land \phi_3)) \lor (\phi_1 \land (\phi_3 \land \phi_2)) \lor ((\phi_2 \land \phi_3) \land \phi_1) \lor ((\phi_3 \land \phi_2) \land \phi_1) \\
o \pi(\psi_2) &= ((\phi_1 \land \phi_2) \land \phi_3) \lor ((\phi_2 \land \phi_1) \land \phi_3) \lor (\phi_3 \land (\phi_1 \land \phi_2)) \lor (\phi_3 \land (\phi_2 \land \phi_1))
\end{align*}
\]

Both translations generate four orders of evaluation for the three $\phi_i$, where there are, of course, six orders of three formulas to consider. A binary connective will always generate a bracketing of the three formulas and then it cannot generate more than four orders of computation in the semantics. This is a limit on what one can do with one binary connective in the current set up.

The translation provides a good way to test the behavior of $\land$. We can check whether the permutation translation is correct: does the systematic replacement of $\land$ by $\land$ in a formula $\phi$, lead to the evaluation of all the orders that turn up in $\pi(\phi)$? In other words, does the following property hold for all $\theta$:

\[
\boxed{[\phi]_\land = [\pi(\phi)]_\land}
\]

\textbf{Proof:} The proof is by induction on the complexity of $\phi \in \mathcal{L}_\land$. We only give the crucial case for $\land$: $[\phi_1 \land \phi_2]_\land(\theta) = [\phi_1]_\land([\phi_2]_\land(\theta)) \cup [\phi_2]_\land([\phi_1]_\land(\theta)) = [(\pi(\phi_1) \land \pi(\phi_2))]_\land(\theta) = [\pi(\phi_1 \land \phi_2)]_\land(\theta)$ \hfill $\square$

To conclude this section, we present an interesting trick: we show how, at the cost of more error messages, order of computation can be regulated in $\mathcal{L}_\land$. Hence the trick shows that replacing $\land$ by $\land$ does not lead to a significant loss of control over the order of computation in programs.

In items 2 and 5 of the list on page 533 we saw that:

\[
\begin{align*}
[(y = 1 \land z = 1) \land y - 1 = z - 1](\epsilon) &= \{\{y/1, z/1\}\} \\
[(y - 1 = z - 1 \land (y = 1 \land z = 1)](\epsilon) &= \{\text{error}\}
\end{align*}
\]

because $[y - 1 = z - 1](\epsilon) = \{\text{error}\}$. Now:

\[
\begin{align*}
[(y - 1 = z - 1 \land (y = 1 \land z = 1)](\epsilon) &= \{\{y/1, z/1\}, \text{error}\}
\end{align*}
\]

This shows that $\land$ does improve the situation of item 5: we find the obvious outcome $\{y/1, z/1\}$. But it also shows that it does not improve the situation for item 2: $\land$ automatically considers both orders, so error gets included.

This observation generalizes to a trick for fixing the order of computation at the cost of including error messages.
Example 2. Consider six distinct variables $x, y, u, v, p, q$ and set:

- $\rho_1 = x = y$
- $\rho_2 = (x + 1 = y + 1 \land u = v)$
- $\rho_3 = (x + 1 = y + 1 \land u + 1 = v + 1) \land p = q$

Call $r_i = [\rho_i]$ (for $i = 1, 2, 3$). It is easy to check that, starting from the empty state $\epsilon$, $r_1; r_2; r_3$ is the only successful order of execution of the $r_i$. It gives: $\langle \epsilon, \{x/y, u/v, p/q\}\rangle$.

The trick shows that the semantics of equations allows control over the order of execution of a program, simply by adding equations of the type $\rho_i$ at the appropriate points in the program. In particular, this allows us to simulate sequential $\land$ with symmetric $\land$. For example:

$A(t) \land B(s)$ compares to $(A(t) \land \rho_1) \land (B(s) \land \rho_2)$. The example can easily be extended into a kind of 'backwards translation,' but there is one proviso: the control of order obtained by adding equations of variables generates extra error messages in many cases! Setting this proviso aside, from the point of view of expressivity it does not really matter whether we add $\land$ to the language with $\land$ or whether we replace $\land$ by $\land$ (as long as enough variables are available).

By combination of the symmetric conjunction with the interpretation of disjunction discussed in section we can also remove the extra error messages, to get an even more elegant way of regulating the order of execution of programs, in a setting where conjunction is symmetric.

4 Recursive Procedures

Apt and Smaus discuss the incorporation of recursive procedures in the framework. They show how this makes the connection with logic programs precise. However, they give an operational rather then a denotational semantics for first order logic. Here we present the appropriate denotational analogue. Then we compare the way the two conjunctions work out in the extension with recursive procedures.

First we introduce new atomic predicates $p_i(x^i_1, \ldots, x^i_{m_i})$ ($1 \leq i \leq n$) for the recursive procedures. For each of these predicates a definition: $\psi_i(x^i_1, \ldots, x^i_{m_i})$ is available. Such a definition is a positive formula in which the new predicates can occur. Let’s agree to refer to the positive fragment of a language $L$ by adding the superscript $+: L^+$. The free variables of the definition of a procedure are always a subset of the variables of the procedure itself. Definitions like this are collected in sets for which we use notation $\mathcal{D}$.

3 Note that the successful order of execution is generated by the formula $\rho_1 \land (\rho_2 \land \rho_3)$, but not by the formula $(\rho_1 \land \rho_2) \land \rho_3$. Hence $\land$ is not associative. To put this in a more computational perspective: there are limits to the options for delay of computation in a setting with just one binary connective $\land$.

4 This means that $\neg$ does not occur in $\psi_i$. As we regard $\phi \rightarrow \psi$ as shorthand for $(\neg(\phi \land \neg\psi))$ and $\forall x \psi$ as shorthand for $\neg\exists x \neg\phi$, implications and universal quantifications do not occur in positive formulas either.
Apt and Smaus [4] have shown that the definition sets $D$ correspond to Clark completions of logic programs. Because of this correspondence it makes sense to think of $p_i(x_1^i, \ldots, x_m^i)/t_1, \ldots, x_m^i/t_m$ as a procedure call. Below we write $x^i$ for the sequence $(x_1^i, \ldots, x_m^i)$ and $p_i(t^i)$ for a procedure call. The extension of $L_\Lambda$ with the procedures defined by $D$ is called $L_{\Lambda,D}$. We can do exactly the same thing for $L_\pi$, of course, to obtain $L_{\pi,D}$.

We do not just want to consider definitions by recursion using the sequential conjunction, but also for the symmetric conjunction. Then, we will also investigate the correctness of the permutation translation for the extended languages. In order to do this, we first extend the definition of the permutation translation to recursive procedures by setting: $\pi(p_i(t^i)) = p_i(t^i)$. The second thing we have to take care of is the proper transfer of definitions from $L_{\Lambda,D}$ to $L_{\pi,D}$. For a sensible comparison it is necessary that we translate the definitions. As notation we use: $\chi_i(x^i) = \pi(\psi_i(x^i))$. This is how the translation is extended. Next we extend the semantics to include interpretations of procedure calls by using approximating formulas for the definitions in $D$.

**Definition 3.** Let a definition set $D = \{\psi_1(x^1), \ldots, \psi_n(x^n)\}$ be given for procedures $p_1(x^1), \ldots, p_n(x^n)$. Then we set for each $1 \leq i \leq n$ and each natural number $\alpha \geq 0$:

- $\psi_i^0 = \bot$
- $\psi_i^{\alpha+1} = \psi_i\{p_1/\psi_1^\alpha, \ldots, p_n/\psi_n^\alpha\}$

A similar definition works for the translated definitions, $\chi_i$ in $L_\Lambda$.

Let’s consider a simple example to see how it works. In the example we see first the definitions of two recursive procedures $p_i(x,y)$ ($i = 1, 2$). Then we see the approximating formulas for the definitions.

**Example:**

- $\psi_1(x,y) = r(x,y) \lor \exists z \ (r(x,z) \land p_2(z,y))$
- $\psi_2(x,y) = s(x,y) \lor \exists z \ (s(x,z) \land p_1(z,y))$

- $\psi_1(x,y)^0 = \bot$
- $\psi_2(x,y)^0 = \bot$
- $\psi_1(x,y)^1 = r(x,y)$
- $\psi_2(x,y)^1 = s(x,y)$
- $\psi_1(x,y)^2 = r(x,y) \lor \exists z \ (r(x,z) \land s(z,y))$
- $\psi_2(x,y)^2 = s(x,y) \lor \exists z \ (s(x,z) \land r(z,y))$
- $\psi_1(x,y)^3 = r(x,y) \lor \exists z \ (r(x,z) \land (s(z,y) \lor \exists u \ (s(z,u) \land r(u,y))))$
- $\psi_2(x,y)^3 = s(x,y) \lor \exists z \ (s(x,z) \land (r(z,y) \lor \exists u \ (r(z,u) \land s(u,y))))$
- $\ldots$

---

5 Here we do not mention the variables to improve readability. $\bot$ is used as a shorthand for some convenient contradiction.
Note that formulas never contain procedure calls. So, we already have interpretations $[[\psi_i(t^i)^\alpha]]_{\pi}(\theta)$ and $[[\chi_i(t^i)^\alpha]]_{\wedge}(\theta)$ (for $\alpha \geq 0$). Hence the following definition makes sense:

**Definition 4.** We define for each $\theta \in \text{Subs}_J \cup \{\text{error}\}$ and $1 \leq i \leq n$:

\[
\begin{align*}
\circ [p_i(t^i)]_{\pi}(\theta) &= \bigcup \{[[\psi_i(t^i)^\alpha]]_{\pi}(\theta) : \alpha \geq 0\} \\
\circ [p_i(t^i)]_{\wedge}(\theta) &= \bigcup \{[[\chi_i(t^i)^\alpha]]_{\wedge}(\theta) : \alpha \geq 0\}
\end{align*}
\]

This gives a perfectly consistent definition of a denotational semantics for the procedure calls, but is it the one we want? After all, the idea was that the $p_i(x^i)$ would correspond to recursively defined procedures. Hence we are after a semantics in terms of least fixed points of appropriate operators. So, we have to find the appropriate operators and check that the approximating formulas indeed give us the least fixed point of these operators.

Here it is convenient to regard $[[\phi]]$ as a binary relation on $\text{Subs}_J \cup \{\text{error}\}$. This works by considering $\eta[[\phi]]_{\theta}$ instead of $\eta \in [[\phi]](\theta)$. We can read $D = \{\psi_1(x^1), \ldots, \psi_n(x^n)\}$ as a set of formula schemes in $L_{\wedge}^+$ with parameters $p_i(x^i)$ ($1 \leq i \leq n$) and each $\psi_i(x^i)$ can be seen as an $n$-ary operator that assigns to denotations: $[[\phi_1]], \ldots, [[\phi_n]]$, the value: $[\psi_i\{p_1/\phi_1, \ldots, p_n/\phi_n\}]$.

The operator $[[\psi_i(x^i)]]$ is continuous, because $\psi_i(x^i)$ only contains the connectives: $\vee$, $\wedge$, $\exists x$. And the denotational semantics of these connectives only uses the operations in $\{\cup, ;, \text{DROP}_{ru}\}$, each of which is continuous. Hence the definitions $\psi_i(x^i)$ generate continuous operators on binary relations and the theory of fixed points for continuous operators applies: the operators have a least fixed point and we can approximate these fixed points from below, in the standard way.

The exact same argument works for the $\chi_i(x^i)$ in the language with the sequential conjunction. This gives the following result:

**Proposition 2.** For each $1 \leq i \leq n$:

\[
\begin{align*}
\circ [p_i(t^i)]_{\pi} &= \text{lfp}([\psi_i(t^i)]_{\pi}) \\
\circ [p_i(t^i)]_{\wedge} &= \text{lfp}([\chi_i(t^i)]_{\wedge})
\end{align*}
\]

At this point we have a sensible way of extending $L_{\pi}$ and $L_{\wedge}$ with recursive procedures and we have a proper denotational semantics for the extended languages. This is the second significant improvement of the expressivity of the denotational format proposed by Apt in [1].

Now all that remains to be done is to check that the two ways of extending expressivity can be combined. We do this by checking the correctness of the translation $\pi$ with the added clause for procedure calls. Fortunately this is easy to check, as the following lemma holds:

---

6 Note that in the argument details about substitutions (of terms $t^i$ for variables $x^i$ and of formulas $\phi_i$ for parameters $p_i$) have been omitted.
Lemma 1. \([p_i(t^i)]_{\pi} = [p_i(t^i)]_{\lambda}\).

Proof: 
\[
[p_i(t^i)]_{\pi} = (by \text{ the extended definition of } [\cdot]_{\pi})
\]
\[
\bigcup\{[\psi_i(t^i)^{\alpha}]_{\pi} : \alpha \geq 0\} = (\psi_i(t^i)^{\alpha} \text{ does not contain parameters } p_j, \text{ so it enjoys } (\sharp))
\]
\[
\bigcup\{[\pi(\psi_i(t^i)^{\alpha})]_{\lambda} : \alpha \geq 0\} = (\text{re-ordering disjuncts gives the } \chi_i^{\alpha} \text{ from the permutations of } \psi_i^{\alpha} \text{ and vice versa})
\]
\[
\bigcup\{[\chi_i(t^i)^{\alpha}]_{\lambda} : \alpha \geq 0\} = (by \text{ the extended definition of } [\cdot]_{\lambda})
\]
\[
[p_i(t^i)]_{\lambda}
\]
\[\square\]

As we have set \(\pi(p_i(t^i)) = p_i(t^i)\), this indeed gives the correctness of \(\pi(\cdot)\) for the language extended with recursive procedures.

Proposition 3. The extension of the permutation translation \(\pi(\cdot)\) to \(L_{\pi D}\) preserves correctness: for each \(\phi \in L_{\pi D}: [\phi]_{\pi} = [\pi(\phi)]_{\lambda}.\)

By now we have considered two ways of extending the expressivity of the proposal in [1] and we have shown that they can be combined. In the next step we reconsider the semantics of \(\lor\), in particular the amount of error messages that \(\lor\) produces.

5 Choice without error

In order to produce even more sound computations without error messages, we alter the semantics of disjunction. The inspiration for this alternative notion is backtracking in LP: search along alternative branches of the computation tree and only report error if no successful computation arises.

The altered semantics is presented with a slightly modified notion of computation state: instead of working with \(\text{pow}(\text{Subs} \cup \{\text{error}\})\), we now work with \(\text{pow}(\text{Subs}) \cup \{\text{error}\}\). This means, informally, that we now consider error as an additional computation state, where before we considered error as an additional substitution. We do this because we do not want computation states in which both error and real substitutions are present, i.e., we want to avoid the situation \(\{\text{error}, \theta, \ldots\}\). Instead of checking all the time that we succeed in doing so, we prefer to exclude such combinations from the outset, by working in \(\text{pow}(\text{Subs}) \cup \{\text{error}\}\).

As an example where we now get an improved result we consider \(\left\langle (y = x + 1 \land x = 0) \lor (x = 0 \land y = x + 1) \right\rangle(\epsilon).\) In the original set up this is interpreted as an instruction to either compute \(\left\langle (y = x + 1 \land x = 0) \right\rangle(\epsilon)\) or \(\left\langle (x = 0 \land y = x + 1) \right\rangle(\epsilon).\) (So, it corresponds to the symmetric conjunction \(y = x + 1 \wedge x = 0\) introduced in section 3.) The result is \(\{\text{error}, \{x/0, y/1\}\}.\) In the semantics proposed in this section we only produce \(\{x/0, y/1\}\). This way only the ‘sensible options’ that the original semantics produces, remain. error messages only arise if there are no sensible solutions.

On \(\text{pow}(\text{Subs}) \cup \{\text{error}\}\) we consider an ordering \(\sqsubseteq\) instead of \(\subseteq\). This is necessary, because we now have a special computation state that is not a subset.
of \( Subs \). So, we have to explain how it compares to subsets of \( Subs \). The definition is as follows (\( \Theta, \Theta' \subseteq Subs \)):

\[
\Theta \subseteq \Theta' \text{ iff } \Theta \subseteq \Theta' \text{ and } \\
\emptyset \subseteq \text{error} \subseteq \Theta \text{ (for } \Theta \neq \emptyset) \]

In a picture:

![Diagram](image-url)

It is easy to check that this is a lattice. We write \( \sqcup \) (and \( \sqcap \)) for the join (and meet, resp.) of the lattice. Hence, the truth table for \( \sqcup \) is as follows:

<table>
<thead>
<tr>
<th>( \sqcup )</th>
<th>\emptyset</th>
<th>\text{error}</th>
<th>\Theta</th>
</tr>
</thead>
<tbody>
<tr>
<td>\emptyset</td>
<td>\emptyset</td>
<td>\text{error}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>\text{error}</td>
<td>\text{error}</td>
<td>\text{error}</td>
<td>\ Theta</td>
</tr>
<tr>
<td>\Theta'</td>
<td>\Theta'</td>
<td>\Theta'</td>
<td>\Theta'</td>
</tr>
<tr>
<td>\Theta'</td>
<td>\Theta'</td>
<td>\Theta'</td>
<td>\Theta'</td>
</tr>
</tbody>
</table>

Here and later on \( \Theta \) and \( \Theta' \) are subsets of \( Subs \). Note that the new ingredient of the semantics, \( \sqcup \), avoids the co-occurrence of \text{error} with real substitutions: nowhere in the table a co-occurrence of substitutions with \text{error} messages turns up. This is crucial for the correctness of the definition of the semantics below.

We also have to define \( DROP_x \) for the new \text{error} state. It will be no surprise that we choose:

\[
DROP_x(\text{error}) := \text{error}.
\]

The semantics basically remains ‘as before’, i.e., as in section \[2\] and \[1\]. But the definitions now have to be read as definitions in the new semantic universe. So,

\[
[\phi_1 \lor \phi_2](\theta) = [\phi_1](\theta) \sqcup [\phi_2](\theta)
\]

and \text{error} replaces \{\text{error}\} in the other clauses of the definition. For example,

\[
[\phi](\text{error}) = \text{error}.
\]
We obtain a soundness result for the new set up of the semantics by comparing it to the semantics from \( [\cdot] \), for which we write \( [\cdot]_o \): the old semantics. The proposition below makes explicit what we promised above: in the new version of the semantics we have simply removed \textit{error} messages from the old version. We give the proposition as a result about \( [\phi](\theta) \) (for \( \theta \in \text{Subs} \)). Then it is obvious how to generalize it to arbitrary computation states.

**Proposition 4.**

1. \( \theta \in [\phi](\eta) \Rightarrow \theta \in [\phi]_o(\eta) \)
2. \( [\phi](\eta) = \emptyset \Rightarrow [\phi]_o(\eta) = \emptyset \)
3. \( [\phi]_o(\theta) = \{\text{error}\} \Rightarrow [\phi](\theta) = \text{error} \)

**Proof:**

The third clause follows from the first two. The proof of the first two cases is a simultaneous induction on the complexity of \( \phi \). For the atomic cases, we just note that the two versions of the semantics co-incide. Now the induction proceeds without problems. We just present two cases. Of course we should give the crucial case: \( \lor \). And for purpose of illustration we also include another case: \( \land \).

\[
\phi_1 \land \phi_2: \quad 1. \theta \in [\phi_1 \land \phi_2](\eta) \Rightarrow \\
\exists \zeta: \zeta \in [\phi_1](\eta) \& \theta \in [\phi_2](\zeta) \Rightarrow \text{(by induction hypothesis)} \\
\exists \zeta: \zeta \in [\phi_1]_o(\eta) \& \theta \in [\phi_2]_o(\zeta) \Rightarrow \\
\theta \in [\phi_1 \land \phi_2]_o(\eta)
\]

2. \( [\phi_1 \land \phi_2](\eta) = \emptyset \Rightarrow \)

\( \) either: \( [\phi_1](\eta) = \emptyset \). Now the induction hypothesis (for 2) gives \( [\phi_1]_o(\eta) = \emptyset \), from which \( [\phi_1 \land \phi_2]_o(\eta) = \emptyset \) follows.

else: \( [\phi_1](\eta) \neq \emptyset \). But in this case we know that \textit{error} has not turned up in \( [\phi_1](\eta) \), as it does not appear in \( [\phi_2]([\phi_1](\eta)) \). So we know, by induction hypothesis (for 1), that for each of the \( \zeta \in [\phi_1](\eta) \), also \( \zeta \in [\phi_1]_o(\eta) \), and that for each of these \( \zeta \): \( [\phi_2](\zeta) = \emptyset \). Now we apply the induction hypothesis (for 2) to each \( \zeta \in [\phi_1](\eta) \).

\[
\phi_1 \lor \phi_2: \quad 1. \theta \in [\phi_1 \lor \phi_2](\eta) \Rightarrow \\
\theta \in [\phi_1](\eta) \text{ for some } i \in \{1, 2\} \Rightarrow \text{(by induction hypothesis)} \\
\theta \in [\phi_1]_o(\eta) \text{ for some } i \in \{1, 2\} \Rightarrow \\
\theta \in [\phi_1 \lor \phi_2]_o(\eta)
\]

2. \( [\phi_1 \lor \phi_2](\eta) = \emptyset \Rightarrow \)

\( [\phi_1](\eta) = \emptyset \) for both \( i \in \{1, 2\} \Rightarrow \text{(by induction hypothesis)} \\
[\phi_1]_o(\eta) = \emptyset \) for both \( i \in \{1, 2\} \Rightarrow \\
[\phi_1 \lor \phi_2]_o(\eta) = \emptyset \)

From this proposition the soundness of the semantics follows as a corollary from the soundness of the old semantics, \( [\cdot]_o \).

**Corollary 1.** The semantics is sound with respect to first order logic, i.e.:

- For each \( J \)-substitution \( \eta \in [\phi](\theta) \):
  \( J \models \phi \eta \)
- If \( [\phi](\theta) = \emptyset \), then:
  \( J \models \neg \phi \theta \)
So, we have succeeded in yet another extension of the expressivity of the denotational semantics in [1]: disjunction-as-backtracking. For the combination of this extension with the extensions of the previous two sections, simply note that the definitions we gave there still make perfect sense with the new notion of computation state and disjunction: also when we treat \textit{error} as a special computation state rather than a special substitution, we can define a symmetric conjunction and use Clark completions for the incorporation of recursion. Then the relevant propositions and their proofs only require minimal alterations.

### 6 Conclusion and Prospects

In this paper we have considered ways of extending the expressivity of the denotational semantics for first order logic proposed by Apt in [1]. Such extensions of expressivity correspond to extensions in computation power: we can write more ‘\textit{error}-free’ programs.

The extensions of expressivity considered are obtained by: diminishing order sensitivity of the semantics; adding options for definition by recursion; interpreting choice in the style of backtracking.

We were able to show that the proposed extensions are \textit{sound}: the extensions do not give rise to a conflict with respect to the standard semantics of the first order language. Also, we have seen that the three ways of improving expressivity can be combined and that their combination also preserves soundness. Hence, we now have an interpretation of first order logic as a programming language with a computation power that is significantly larger than in the original proposal in [1].

The increase of expressivity is also relevant for the comparison with LP. In LP conjunction is symmetric, as witnessed by the strong completeness result for LP. So, by removing the order sensitivity of conjunction already one important disadvantage is removed from the approach of [1]. Recursion is at the heart of LP. Hence adding the option of definition by recursion here clearly is another crucial way of regaining ground on LP. And also the removal of \textit{error} messages in the interpretation of \( \lor \) is relevant: it reminds us of backtracking in LP.

So, we conclude that the view of first order logic as a programming language proposed in [1] can be made to work for a considerable fragment of the first order language. Furthermore, this fragment compares favorably with the expressive power of LP.

As further topics of immediate interest we note the combination of the semantics of this paper with tools from \textit{constraint programming}. LP has been able to gain computational attractiveness by combinations with such tools, resulting in \textit{constraint logic programming} (CLP). For the semantics of [1] this combination has been achieved in [3].
References


Effective Model Completeness of the Theory of Restricted Pfaffian Functions

(Abstract)

Nicolai Vorobjov

Dept. of Computer Science
University of Bath

Pfaffian functions, introduced by Khovanskii in late 70s, are analytic functions satisfying triangular systems of first order partial differential equations with polynomial coefficients. They include for instance algebraic and elementary transcendental functions in the appropriate domains, iterated exponentials, and fewnomials. A simple example, due to Osgood, shows that the first order theory of reals expanded by restricted Pfaffian functions does not admit quantifier elimination. On the other hand, Gabrielov and Wilkie proved (non-constructively) that this theory is model complete, i.e., one type of quantifiers can be eliminated. The talk will explain some ideas behind recent algorithms for this quantifier simplification which are based on effective cylindrical cell decompositions of sub-Pfaffian sets. Complexities of these algorithms are essentially the same as the ones which existed for a particular case of semialgebraic sets.
Effective Quantifier Elimination
over Real Closed Fields

(Abstract)

Nicolai Vorobjov
Dept. of Computer Science
University of Bath

In early 30s A. Tarski, motivated by a problem of automatic theorem proving in elementary algebra and geometry, suggested an algorithm for quantifier elimination in the first order theory of the reals. The complexity of Tarski’s algorithm is a non-elementary function of the format of the input formula. In mid-70s a group of algorithms appeared based on the idea of a cylindrical cell decomposition and having an elementary albeit doubly-exponential complexity, even for deciding closed existential formulae. The tutorial will explain some ideas behind a new generation of algorithms which were designed during 80s and 90s and have, in a certain sense, optimal (singly-exponential) complexity. In a useful particular case of closed existential formulae (i.e., deciding feasibility of systems of polynomial equations and inequalities) these new algorithms are theoretically superior to procedures known before in numerical analysis and computer algebra.
Fast Infinite-State Model Checking
in Integer-Based Systems

Tatiana Rybina and Andrei Voronkov

University of Manchester
{rybina, voronkov}@cs.man.ac.uk

Abstract. In this paper we discuss the use of logic for reachability analysis for
infinite-state systems. Infinite-state systems are formalized using transition sys-
tems over a first-order structure. We establish a common ground relating a large
class of algorithms by analyzing the connections between the symbolic represen-
tation of transition systems and formulas used in various reachability algorithms.
We consider in detail the so-called guarded assignment systems and local reach-
ability algorithms. We show how an implementation of local reachability algo-
rithms and a new incremental algorithm for finding Hilbert’s base in the system
BRAIN resulted in much faster reachability checking than in systems using con-
straint libraries and decision procedures for Presburger’s arithmetic. Experimental
results demonstrate that problems in protocol verification which are beyond
the reach of other existing systems can be solved completely automatically.

1 Introduction

Reachability properties arise in many applications of verification. In this paper we dis-
cuss reachability algorithms in infinite-state systems. Infinite-state systems are formal-
ized using transition systems over a first-order structure. We analyze the connections
between the symbolic representation of transition systems and formulas which are used
in various reachability algorithms. Our main results are related to the so-called guarded
assignment systems.

This paper serves two main purposes. First, it formalizes infinite-state systems using
model-theoretic notions and discusses reachability algorithms based on this formaliza-
tion. Though many results and observations of this paper form part of folklore circulat-
ing in the infinite-state model checking community, we believe that our formalization
is useful since it gives a common model-theoretic approach to otherwise quite diverse
formalizations. Second, we observe that for a large class of systems, called guarded
assignment systems (GAS), the reachability analysis is simpler since only formulas of
a special form are used for satisfiability- and entailment-checking. Many known for-
malizations of broadcast, cache coherence, and other protocols belong to the simplest
kind of GAS, called simple GAS. It follows from our results that the so-called local
backward reachability algorithms can be used for simple GAS over structures in which
satisfiability-checking is decidable for conjunctions of atomic formulas. This, for ex-
ample, allows one to extend the existing reachability algorithms to some theories of
queues.

© Springer-Verlag Berlin Heidelberg 2003
We also discuss how this general framework applies to a special case of integer-based transition systems. We show how the main operations required for reachability analysis in such systems can be implemented using an incremental algorithm for building Hilbert’s bases of systems of linear equations and inequations over integers.

This paper is organized as follows. In Section 2 we briefly overview relevant parts of model theory and define several classes of first-order formulas. In Section 3 we give a model theory-based formalization of transition systems and their symbolic representations and introduce basic forward and backward reachability algorithms. We also discuss requirements on the underlying first-order structures. In Section 5 we define guarded assignment systems and reachability algorithms for them. In Section 6 we introduce local reachability algorithms for guarded assignment systems. Finally, in Section 10 we discuss some other issues related to our formalization of infinite-state systems and reachability.

2 Preliminaries

In this section we define notation and several classes of formulas which will be used in the rest of the paper. We assume knowledge of standard model-theoretic definitions, such as first-order formulas, structure and truth, which can be found in any standard textbook on logic or model theory.

Unless stated otherwise, we will deal with a fixed first-order structure $\mathbb{M}$ with a domain $\mathcal{D}$ and assume that all formulas are formulas of the signature of this structure. A valuation for a set of variables $V$ in $\mathbb{M}$ is any mapping $s : V \to \mathcal{D}$. We will use the standard model-theoretical notation $\mathbb{M}, s \models A$ to denote that the formula $A$ is true in the structure $\mathbb{M}$ under a valuation $s$. When we use this notation, we assume that $s$ is defined on all free variables of $A$.

A formula $A$ with free variables $V$ is said to be satisfiable (respectively, valid) in $\mathbb{M}$ if there exists a valuation $s$ for $V$ in $\mathbb{M}$ such that $\mathbb{M}, s \models A$ (respectively, for every valuation $s$ we have $\mathbb{M}, s \models A$).

A formula $A$ is called

- quantifier-free if $A$ contains no quantifiers.
- positive existential if $A$ can be built from atomic formulas using just $\exists$, $\land$, $\lor$;
- a conjunctive constraint if $A$ can be built from atomic formulas using just $\exists$, $\land$;
- a simple constraint if $A$ is a conjunction of atomic formulas.

3 Transition Systems and Reachability: A Formalization in Logic

In this section we introduce a logical formalization of transition systems.

3.1 Infinite-State Transition Systems and Their Symbolic Representation

Our formalization of transition systems is as follows. A transition system has a finite number of variables with values in a possibly infinite domain $\mathcal{D}$. A state is a mapping
from variables to values. Transitions may change values of variables. A symbolic representation of such a system uses first-order formulas interpreted in a structure with the domain \( D \).

**DEFINITION 1 (Transition System)** A *transition system* is a tuple \( S = (\mathcal{V}, \mathcal{D}, T) \), where

1. \( \mathcal{V} \) is a finite set of *state variables*.
2. \( \mathcal{D} \) is a non-empty set, called the *domain*. Elements of \( \mathcal{D} \) are called *values*.

A *state* of the transition system \( S \) is a function \( s : \mathcal{V} \rightarrow \mathcal{D} \).

3. \( T \) is a set of pairs of states, called the *transition relation* of \( S \).

A transition system \( S \) is *finite-state* if \( \mathcal{D} \) is finite, and *infinite-state* otherwise.

We call any set of pairs of states a *transition*. Transition systems arising in practical applications often have variables ranging over different domains. For example, some state variables may range over the natural numbers, while others over the boolean values or other finite domains. We introduce a single domain for simplicity. It is not hard to generalize our formalization to several domains using many-sorted first order logic. In the sequel we assume a fixed transition system \( S = (\mathcal{V}, \mathcal{D}, T) \). Suppose that \( \mathcal{M} \) is a first-order structure whose domain is \( \mathcal{D} \). For example, if \( \mathcal{D} \) is the set of natural numbers, then the structure \( \mathcal{M} \) can be the set of natural numbers together with the order \(<\), operation \(+\) and constants \(0\), \(1\). In addition to the set of state variables \( \mathcal{V} \), we also introduce a set \( \mathcal{V}' \) of *next state variables* of the same cardinality as \( \mathcal{V} \). We fix a bijection \( \cdot' : \mathcal{V} \rightarrow \mathcal{V}' \) such that \( v \in \mathcal{V} \) we have \( v' \in \mathcal{V}' \).

We can treat the variables in \( \mathcal{V} \cup \mathcal{V}' \) also as logical variables. Then any mapping \( s : \mathcal{V} \rightarrow \mathcal{D} \) can be considered as both a state of the transition system \( S \) and a valuation for \( \mathcal{V} \) in the structure \( \mathcal{M} \), and similarly for \( s' : \mathcal{V}' \rightarrow \mathcal{D} \).

**DEFINITION 2 (Symbolic Representation)** Let \( S \) be a set of states and \( A \) be a formula with free variables in \( \mathcal{V} \). We say that \( A \) *symbolically represents* \( S \) in \( \mathcal{M} \), or simply *represents* \( S \) if for every valuation \( s \) for \( \mathcal{V} \) in \( \mathcal{M} \) we have

\[
S \ni s \iff \mathcal{M}, s \models A.
\]

Likewise, we say that a formula \( B \) with free variables in \( \mathcal{V} \cup \mathcal{V}' \) (symbolically) *represents a transition* \( T \) in \( \mathcal{M} \) if for every pair of valuations \( s, s' \) in \( \mathcal{M} \) for \( \mathcal{V} \) and \( \mathcal{V}' \) respectively we have

\[
(s, s') \in T \iff \mathcal{M}, s, s' \models B.
\]

If a formula \( Tr \) symbolically represents the transition relation \( T \) of a transition system \( S \), we simply say that \( Tr \) symbolically represents \( S \).

In the sequel we will follow the following convention. We will often identify a symbolic representation of a transition with the transition itself. For example, when \( T \) is a formula with free variables in \( \mathcal{V} \cup \mathcal{V}' \) we can refer to \( T \) as a transition.
DEFINITION 3 We say that a state $s_n$ is forward reachable from a state $s_0$ w.r.t. $T$ if there exists a sequence of states $s_1, \ldots, s_{n-1}$ such that for all $i \in \{0, \ldots, n-1\}$ we have $(s_i, s_{i+1}) \in T$. In this case we also say that $s_n$ is reachable from $s_0$ in $n$ steps and that $s_0$ is backward reachable from $s_n$ in $n$ steps.

Instead of “forward reachable” we will say “reachable”. When we speak about reachability with respect to $T$, and $T$ is clear from the context, we will simply say “reachable”.

The reachability problem can now be defined as follows.

DEFINITION 4 (Reachability Problem) The reachability problem for $M$ is the following decision problem. Given formulas $In$, $Fin$, and $Tr$ such that

1. $In$ represents a set of states $I$, called the set of initial states;
2. $Fin$ represents a set of states $F$, called the set of final states;
3. $Tr$ represents the transition relation of a transition system $S$,

do there exist states $s_1 \in I$, $s_2 \in F$ such that $s_2$ is reachable from $s_1$ w.r.t. $Tr$? 

In fact, reachability is a family of decision problems parametrized by the structure $M$.

When we discuss instances of the reachability problem, we will call the formulas $In$ and $Fin$ the initial and final conditions, respectively, and $Tr$ the transition formula.

The reachability problem for infinite-state systems is, in general, undecidable. Various results on reachability are discussed in many papers, including [13, 1, 20, 2].

Let us give an example instance of the reachability problem. Before defining it formally, we will introduce a convenient syntax for representing transitions. Suppose that $T = T_1 \cup \ldots \cup T_n$ is a transition such that every $T_i$ can be symbolically represented by $P_i(V) \land Q_i(V, V')$ such that the free variables of $P_i(V)$ are in $V$. Then we will write the transition $T$ as

$$ P_1(V) \Rightarrow Q_1(V, V'), \ldots, P_n(V) \Rightarrow Q_n(V, V'). \quad (1) $$

We will call the formula $P_i(V)$ the guard or the precondition, and the formula $Q_i(V')$ the postcondition of this transition. Note that the notions of pre- and post-conditions refer to a particular representation of the transition, since the same transition may have representations using different pre- and post-conditions. Guarded representation (1) thus denotes the transition

$$ (P_1(V) \land Q_1(V, V')) \lor \ldots \lor (P_n(V) \land Q_n(V, V')), $$

that is, the union of all transitions $P_i(V) \Rightarrow Q_i(V, V')$.

An example of a transition system presented using the guarded representation is given in Figure 1.

3.2 Algorithms for Checking Reachability

The algorithms for checking reachability are based on the idea of building a symbolic representation of the set of reachable states. There are two main kinds of reachability
drinks > 0 ∧ customers > 0 ⇒ drinks' = drinks − 1 (*dispense-drink*)
true ⇒ drinks' = drinks + 64 (*recharge*)
true ⇒ customers' = customers + 1 (*customer-coming*)
customers > 0 ⇒ customers' = customers − 1 (*customer-going*)

Fig. 1. Guarded representation of transitions

Before defining these algorithms, let us discuss symbolic representations of forward and backward reachable states. We assume fixed initial and final conditions $\text{In}(V)$, $\text{Fin}(V)$ and the transition formula $\text{Tr}(V, V')$.

Let $A(V)$ be a formula which represents a set of states $S$. It is not hard to argue that the set of states reachable in one step from a state in $S$ can be represented by the formula

$$\exists V_1 (A(V_1) ∧ Tr(V_1, V)).$$

Likewise, the set of states backward reachable in one step from a state in $S$ is represented by the formula

$$\exists V_1 (A(V_1) ∧ Tr(V, V_1)).$$

This observation implies the following facts about forward and backward reachability.

**LEMMA 5 (Forward Reachability)** Consider the sequence of formulas $FR_i$ defined as follows:

$$FR_0(V) = \text{In}(V); FR_{i+1}(V) = FR_i(V) ∨ \exists V_1 (FR_i(V_1) ∧ Tr(V_1, V)).$$

Then each $FR_i$ symbolically represents the set of states reachable from $I$ in at most $i$ steps.

**LEMMA 6 (Backward Reachability)** Consider the sequence of formulas $BR_i$ defined as follows:

$$BR_0(V) = \text{Fin}(V); BR_{i+1}(V) = BR_i(V) ∨ \exists V_1 (BR_i(V_1) ∧ Tr(V, V_1)).$$

Then each $BR_i$ symbolically represents the set of states backward reachable from $F$ in at most $i$ steps.

Using these lemmas, one can prove two theorems which form the basis of reachability algorithms.

**THEOREM 7 (Reachability)** (i) There exists a final state reachable from an initial state if and only if there exists a number $i ≥ 0$ such that $M \models \exists V (FR_i(V) ∧ Fin(V))$.

(ii) If there exists a number $i ≥ 0$ such that $M \not\models \exists V (FR_i(V) ∧ Fin(V))$ and $M \models ∀ V (FR_{i+1}(V) → FR_i(V))$, then there exists no final state reachable from an initial state. The same statements hold if one replaces $FR$ by $BR$ and $Fin$ by $In$. 
This theorem reduces, in some sense, reachability-checking to checking whether formulas of a special form are true in the structure $M$. The form of these formulas depends on the symbolic representations $In$, $Fin$, $Tr$ of the sets of initial and finite states and the transition relation.

The basic reachability algorithms are based on explicit computation of the formulas $FR_i$ and $BR_i$ and the use of Theorem 7.

**DEFINITION 8** The forward reachability algorithm $Forward$ and the backward reachability algorithm $Backward$ are shown in Figures 2 and 3. They are parametrized by a function $simp$, called a simplification function. 

Usually, this formula transforms formulas into equivalent ones, i.e., $M \models \forall V (A \leftrightarrow simp(A))$ for all formulas $A$.

In different reachability algorithms, the simplification function $simp$ may make a simple transformation, for example into a normal form, but may also perform more complex ones, such as quantifier elimination. We will sometimes need the simplification function $simp$ to preserve some class of formulas. To this end, we introduce the following definition. We call a function $simp$ on formulas stable w.r.t. a class of formulas $C$, if for every formula $A \in C$ we have $simp(A) \in C$.

### 3.3 First-Order Theories of Infinite Structures

In this section we analyze first-order structures which are suitable for reachability algorithms. In both forward and backward reachability algorithms one has to solve repeatedly the following problems:

1. **Satisfiability**: whether the formula $current(V) \land Fin(V)$ or $current(V) \land In(V)$ is satisfiable in $M$.
2. **Entailment**: whether the formula $next(V) \rightarrow current(V)$ or $prev(V) \rightarrow current(V)$ is valid in $M$. 

**procedure** Forward

**input**: formulas $In$, $Fin$, $Tr$

**output**: “reachable” or “unreachable”

**begin**

```plaintext
Procedure Forward

Input: Formulas In, Fin, Tr
Output: "reachable" or "unreachable"

Begin

current(V) := In(V)

While M \not\models \exists V (current(V) \land Fin(V))

next(V) := simp(current(V) \lor 

\exists V_1 (current(V_1) \land Tr(V_1, V))

If M \not\models \forall V (next(V) \rightarrow current(V))

Then return "unreachable"

Current(V) := next(V)

End

End
```

**Fig. 2.** Forward reachability algorithm

**procedure** Backward

**input**: Formulas $In$, $Fin$, $Tr$

**output**: “reachable” or “unreachable”

**begin**

```plaintext
Procedure Backward

Input: Formulas In, Fin, Tr
Output: "reachable" or "unreachable"

Begin

Current(V) := Fin(V)

While M \not\models \exists V (current(V) \land In(V))

Prev(V) := simp(current(V) \lor 

\exists V_1 (current(V_1) \land Tr(V, V_1))

If M \models \forall V (prev(V) \rightarrow current(V))

Then return "unreachable"

Current(V) := prev(V)

End

End
```

**Fig. 3.** Backward reachability algorithm
Of course the reachability algorithms of Figures 2 and 3 can only be considered as algorithms modulo the assumption that satisfiability- and entailment-checking are decidable.

It is not hard to argue that the formulas current(\textit{V}) at the \textit{i}th iteration of the while-loop are equivalent to the formulas \textit{FR}_i for forward reachability and \textit{BR}_i for backward reachability, provided that \textit{simp} is equivalence-preserving. If \textit{simp}(A) = A for every formula A, then the formulas current(\textit{V}) are exactly the formulas \textit{FR}_i for the forward reachability algorithm and \textit{BR}_i for the backward reachability algorithm. This implies that the reachability algorithms have the following properties.

\textsc{Theorem 9} (Soundness and Semi-Completeness)
The algorithms \textit{Forward} and \textit{Backward} have the following properties:

1. there is a final state reachable from an initial state if and only if the algorithm returns “reachable”;
2. if the algorithm returns “unreachable”, then there is no final state reachable from an initial state. \hfill \qed

On some inputs the algorithms do not terminate. In this case we say that the reachability analysis diverges.

Complexity of the satisfiability and entailment problems depends on the structure \textit{M} and the form of the formulas current used in the algorithm.

If the first-order theory of \textit{M} is decidable, one can implement satisfiability- and entailment-checking using a decision procedure for the first-order theory of \textit{M}. In some cases, one can even use off-the-shelf decision procedures or libraries. For example, [8] use the Omega Library [25] for deciding Presburger arithmetic. The use of general-purpose algorithms to decide specific classes of formulas may be inefficient. Then specialized tools may be needed to decide these classes of formulas more efficiently.

It is not hard to see that in many important special cases the formulas \textit{FR}_i and \textit{BR}_i have a special form.

\textsc{Lemma 10} Let \textit{In}, \textit{Fin}, and \textit{Tr} be positive existential formulas and \textit{simp} be stable w.r.t. positive existential formulas. Then the formulas current in the algorithms \textit{Forward} and \textit{Backward} are positive existential too. \hfill \qed

It follows from this lemma that it is enough to solve satisfiability and entailment for positive existential formulas only, as soon as the initial and final conditions and the transition formula are positive existential formulas, which they often are. As we shall see below, for some important special cases even quantifier-free formulas suffice.

If the first-order theory of \textit{M} admits quantifier elimination, i.e., every formula is effectively equivalent to a quantifier-free formula, then it is enough to check satisfiability and entailment of quantifier-free formulas only. This can be achieved by applying quantifier elimination, i.e., replacing quantified formulas by equivalent quantifier-free formulas, whenever quantified formulas appear, for example, when transitions are applied. However, quantifier elimination may be expensive.
4 Integer-Based Systems

Denote by $\mathbb{I}$ the set of integers. Consider the structure

$$\mathbb{I} = (\mathbb{I}, >, <, \geq, \leq, +, -, 0, 1, 2, \ldots),$$

where all the function and predicate symbols (for example $>$) have their standard interpretation over integers. The first-order theory of $\mathbb{I}$ is decidable, which means that one can use the reachability algorithms Forward and Backward even when the initial and final conditions and the transition formula are arbitrary first-order formulas. In addition, infinite-state transition systems (of a very simple form) over the domain of integers can be used for specifying safety properties of a large classes of protocols. The first-order theory of $\mathbb{I}$ (essentially, the Presburger arithmetic) admits quantifier elimination, if we extend the signature of $\mathbb{I}$ by the predicates expressing $m \mid (x - n)$ for concrete natural numbers $m, n$.

4.1 Integer-Based Transition Systems

We will call the transition systems over $\mathbb{I}$ the integer transition systems and the reachability problem over $\mathbb{I}$ the integer reachability problem.

We will often use the following simple property of $\mathbb{I}$.

**Lemma 11** In $\mathbb{I}$ every quantifier-free formula $A$ is equivalent to a disjunction of simple constraints. □

Off-the-shelf tools, such as Omega Library [25] or the Composite Library [32], are available to decide the Presburger arithmetic or its fragments.

The fact that satisfiability- and entailment-checking are decidable does not necessarily imply that the reachability problem is decidable. Neither does it imply that termination of the reachability algorithms is decidable. In this section we note the following undecidability result which will carry over to all classes of transition systems and reachability algorithms studied in this paper.

**Theorem 12** Consider the instances of the integer reachability problem with three state variables $c_1, c_2, s$ whose transition relation is a disjunction of conjunctions of the following formulas: $c_i > 0$ or $s = n$, where $n$ is a natural number, $c'_i = c_i + 1$, $c'_i = c_i - 1$, $c'_i = c_i$, or $s' = m$, where $m$ is a natural number. There is exactly one initial and one final state. Then

1. the reachability problem for this class is undecidable.
2. termination of the algorithm Forward is undecidable.
3. termination of the algorithm Backward is undecidable. □

One can easily prove this theorem by encoding two-counter machines by transition systems of this form. The systems used in the theorem are simple guarded assignment systems (the definition is given below).

Some decidability results for integer-based systems are given in e.g., [13, 1]. For example, safety properties of broadcast protocols can be represented as instances of the integer reachability problem, and the backward reachability algorithm terminates on these instances [13].
4.2 Types of Protocols

A large number of protocols can be easily modelled by integer transition systems. Examples are Petri nets, parametric and broadcast protocols. A typical way to change parametric systems into integer transition systems is as follows. Suppose that there is a (parametric) number of processes represented as variables $v_1, v_2, \ldots$ ranging over a finite domain $\{d_1, \ldots, d_n\}$. Then, instead of using formulas $v_i = d_j$ to denote that $v_i$ has the value $d_j$ we introduce new variables $n_j$ to denote the number of variables $v_i$ having the value $d_j$. The transitions of the original system can then be reformulated using new variables. For example, the transition $v_i = d_1 \land v'_i = d_2$ becomes $n_1 > 0 \land n'_1 = n_1 - 1 \land n'_2 = n_2 + 1$. Most of the benchmarks described in Section 9 are obtained by such a transformation.

5 Guarded Assignment Transition Systems

In this section we consider an important special case of transition systems in which quantifier-free formulas can be used for backward reachability, even without the use of expensive quantifier elimination algorithms. Guarded assignment systems are in the core of the model-checking system BRAIN [27]

5.1 Guarded Assignments

As usual, we assume that $\mathcal{V}$ the set of state variables of the transition system.

**Definition 13 (Guarded Assignment)** A guarded assignment is any formula (or transition) of the form

$$P \land v'_1 = t_1 \land \ldots \land v'_n = t_n \land \bigwedge_{v \in \mathcal{V} - \{v_1, \ldots, v_n\}} v' = v,$$

where $P$ is a formula with free variables $\mathcal{V}, \{v_1, \ldots, v_n\} \subseteq \mathcal{V}$, and $t_1, \ldots, t_n$ are terms with variables in $\mathcal{V}$. We will write guarded assignments as

$$P \Rightarrow v_1 := t_1, \ldots, v_n := t_n. \quad (2)$$

The formula $P$ is called the guard of this guarded assignment.

A guarded assignment is quantifier-free (respectively, positive existential), if so is its guard. A guarded assignment is called simple existential if its guard is a conjunctive constraint, and simple if its guard is a simple constraint.

Formula (2) represents a transition which applies to states satisfying $P$ and changes the values of variables $v_i$ to the values of the terms $t_i$. Note that a guarded assignment $T$ is a deterministic transition: for every state $s$ there exists at most one state $s'$ such that $(s, s') \in T$. Moreover, such a state $s'$ exists if and only if the guard of this guarded assignment is true at $s$, i.e., $\mathcal{M}_s, s \models P$. 
**DEFINITION 14 (Guarded Assignment System)** A transition system is called a guarded assignment system, or simply GAS, if its transition relation is a union of a finite number of guarded assignments. A GAS is called quantifier-free (respectively, positive existential, simple existential, or simple) if every guarded assignment in it is also quantifier-free (respectively positive existential, simple existential, or simple).

An example integer guarded assignment system for representing a drink dispenser is given in Figure 4. This system is a union of transitions shown in this figure. Since all guards are simple constraints, the system is simple.

Note that every guarded assignment represents a deterministic transition, but a guarded assignment system may represent a non-deterministic transition system because several guards may be true in the same state. Not every transition system is a guarded assignment system. Indeed, in every guarded assignment system with a transition relation \( T \) for every state \( s \) there exists a finite number of states \( s' \) such that \((s, s') \in T\). The greatest transition \( \text{true} \) over any infinite structure \( \mathcal{M} \) does not have this property.

**THEOREM 15** Every integer quantifier-free GAS is also a simple GAS.

**Proof.** Since every quantifier-free formula is equivalent to a formula in disjunctive normal form, we can assume that the guards are in disjunctive normal form. Since in \( \mathbb{I} \) the negation of an atomic formula is equivalent to a disjunction of atomic formulas (for example \( \neg x = y \) is equivalent to \( x > y \lor y > x \) and \( \neg x > y \) is equivalent to \( y \geq x \)), we can assume that every guard is a disjunction \( C_1 \lor \ldots \lor C_n \) of conjunctions of atomic formulas. But every guarded assignment

\[
C_1 \lor \ldots \lor C_n \Rightarrow A
\]

is equivalent to the union of the simple guarded assignments \( C_1 \Rightarrow A, \ldots, C_n \Rightarrow A \),

\[
C_1 \Rightarrow A, \ldots, C_n \Rightarrow A,
\]

so we can equivalently replace it by \( n \) simple guarded assignments.

One can generalize this theorem to structures different from \( \mathbb{I} \). Indeed, the only property of \( \mathbb{I} \) used in this proof is that the negation of an atomic formula is equivalent to a positive quantifier-free formula.

The notion of a guarded assignment system is not very restrictive. Indeed, broadcast protocols and Petri nets can be represented as integer simple guarded assignment systems. All transition systems for cache coherence protocols described in [10] are integer simple GAS.
Let us note one interesting property of GAS related to properties other than reachability. Let us call a state \( s \) a **deadlock state** if there is no state \( s' \) such that \((s, s') \in T\), where \( T \) is the transition relation.

**Theorem 16**  
Let the transition relation \( T \) be a union of guarded assignments \((P_1 \Rightarrow A_1), \ldots, (P_n \Rightarrow A_n)\). Then the set of all deadlock states is represented by the formula \( \neg(P_1 \lor \ldots \lor P_n) \).  

This theorem shows that checking **deadlock-freedom** (i.e., non-reachability of a deadlock state) of GAS may be as easy as checking reachability properties, for example, if the GAS is quantifier-free, then the set of deadlock states can also be represented by a quantifier-free formula. Theorem 16 is also used in the system **BRAIN** to generate the deadlock-freedom conditions automatically.

### 5.2 Reachability Algorithms for Guarded Assignment Systems

We introduced simple guarded assignment systems because they have a convenient property related to backward reachability algorithms. Essentially, one can use backward reachability algorithms for these systems when the underlying structure has a much weaker property than the decidability of the first-order theory.

Let us see how the formulas \( FR_i \) and \( BR_i \) look like in the case of simple guarded assignment systems. Let \( u \) be a guarded assignment of the form \( P(v_1, \ldots, v_n) \Rightarrow v_1 := t_1, \ldots, v_n := t_n \).

For simplicity we assume that \( V = \{v_1, \ldots, v_n\} \). This can be achieved by adding “dummy” assignments \( v := v \) for every variable \( v \in V - \{v_1, \ldots, v_n\} \). Let also \( A(v_1, \ldots, v_n) \) be a formula whose free variables are in \( V \). For every term \( t \) denote by \( t' \) the term obtained from \( t \) by replacing every occurrence of every state variable \( v_i \) by \( v'_i \).

Define the following formulas:

\[
A^u(v_1, \ldots, v_n) \overset{\text{def}}{=} \exists V' (A(v'_1, \ldots, v'_n) \land P(v'_1, \ldots, v'_n) \land v_1 = t'_1 \land \ldots \land v_n = t'_n); \\
A^{-u}(v_1, \ldots, v_n) \overset{\text{def}}{=} P(v_1, \ldots, v_n) \land A(t_1, \ldots, t_n).
\]

**Lemma 17**  
Let \( A(v_1, \ldots, v_n) \) represent a set of states \( S \). Then

1. the formula \( A^u(v_1, \ldots, v_n) \) represents the set of states reachable in one step from \( S \) using \( u \);
2. the formula \( A^{-u}(v_1, \ldots, v_n) \) represents the set of states backward reachable in one step from \( S \) using \( u \).

**Proof.** For forward reachability the statement is straightforward, so let us prove the backward reachability part. The set of states backward-reachable in one step from \( S \) is represented by the formula

\[
\exists V' (P(v_1, \ldots, v_n) \land A(v'_1, \ldots, v'_n) \land v'_1 = t'_1 \land \ldots \land v'_n = t'_n). \tag{4}
\]

Using the fact that in first-order logic the formula \( \exists x (x = t \land B(x)) \), where \( t \) does not occur in \( x \), is equivalent to \( B(t) \), it is not hard to argue that formula (4) this formula is equivalent to \( A^{-u}(v_1, \ldots, v_n) \).
**procedure** GASForward

**input:** formulas $I_n$, $F_n$, finite set of guarded assignments $U$

**output:** “reachable” or “unreachable”

**begin**

$current := I_n$

**while** $M \not\models \exists V (current \land F_n)$

$next := current \lor \lor_{u \in U} current^u$

**if** $M \models \forall V (next \rightarrow current)$

**then return** “unreachable”

$current := next$

**return** “reachable”

**end**

**Fig. 5.** Forward reachability algorithm for GAS

**procedure** GASBackward

**input:** formulas $I_n$, $F_n$, finite set of guarded assignments $U$

**output:** “reachable” or “unreachable”

**begin**

$current := I_n$

**while** $M \not\models \exists V (current \land I_n)$

$prev := current \lor \lor_{u \in U} current^{-u}$

**if** $M \models \forall V (prev \rightarrow current)$

**then return** “unreachable”

$current := prev$

**return** “reachable”

**end**

**Fig. 6.** Backward reachability algorithm for GAS

One can deduce from this lemma that the formulas $A^u$ and $A^{-u}$ have a special form, as expressed by the following lemma.

**Lemma 18** Let $u$ be a guarded assignment. Then the function $A \mapsto A^u$ is stable w.r.t. positive existential formulas and conjunctive constraints. The function $A \mapsto A^{-u}$ is stable w.r.t. positive existential formulas, conjunctive constraints, quantifier-free formulas, and simple constraints. $\square$

So far forward and backward reachability were treated symmetrically. Lemma 18 shows an asymmetry between forward and backward reachability for simple GAS: the predicate transformer corresponding to backward reachability yields simpler formulas. Using Lemma 17, one can modify forward and backward reachability algorithms of Figures 2 and 3 for guarded assignment transition systems.

**Definition 19** The forward reachability algorithm GASForward and the backward reachability algorithm GASBackward for GAS are shown in Figures 5 and 6. When we check reachability for a GAS whose transition relation is a union of guarded assignments $u_1, \ldots, u_n$, we let $U = \{u_1, \ldots, u_n\}$ in the input of the algorithm. $\square$

**Theorem 20** (Soundness and Semi-Completeness) The algorithms GASForward and GASBackward have the following properties:

1. there is a final state reachable from an initial state if and only if the algorithm returns “reachable”;
2. if the algorithm returns “unreachable”, then there is no final state reachable from an initial state.

Moreover, GASForward (respectively, GASBackward) terminates if and only if Forward (respectively Backward) terminates. $\square$
One can prove soundness and semi-completeness of the algorithms GASForward and GASBackward similar to that of Theorem 9.

**THEOREM 21** If the input GAS and the formulas $In$, $Fin$ are quantifier-free, then all of the formulas current in the algorithm GASBackward are quantifier-free too. If, in addition, the input GAS and $In$, $Fin$ are negation-free, then current is also negation-free.

This theorem shows that for quantifier-free GAS satisfiability and entailment in the algorithm GASBackward should only be checked for quantifier-free formulas. It does not hold for the forward reachability algorithm GASForward. Checking satisfiability and validity of quantifier-free formulas is possible if the existential theory (i.e., the set of all sentences of the form $\exists X A$, where $A$ is quantifier-free) of the structure $\mathbb{M}$ is decidable. There exist theories whose full first-order theory is undecidable but existential theory is decidable. A simple example is Presburger arithmetic extended by the divisibility predicate $[4, 23, 22]$, though it is not clear how useful is this theory for applications. A more practical example are some theories of queues considered in $[6, 5]$. This shows that there are infinite domains for which backward reachability may be easier to organize than forward reachability. Of course one can omit the entailment checks from the algorithms, but then they will only be applicable for checking reachability, but not for checking non-reachability.

### 6 Local Algorithms

In this section we study so-called *local* algorithms, which are different from the algorithms discussed above. They are simpler since checking for satisfiability and entailment for simple GAS is only performed for conjunctive constraints, but not so powerful since they may diverge for instances of reachability for which the previously described algorithms terminate. The idea of these algorithms is to get rid of disjunctions which appear in the formulas current even when the input contains no disjunctions at all. Instead of a disjunction $C_1 \lor \ldots \lor C_n$, one deals with the set of formulas $\{C_1, \ldots, C_n\}$. The entailment check is not performed on the disjunction, but separately on each member $C_i$ of the disjunction, and is therefore called a *local entailment check*.

**DEFINITION 22** (Local Reachability Algorithms) The *local forward reachability algorithm* LocalForward and *local backward reachability algorithm* LocalBackward are given in Figures 7 and 8. They are parametrized by a function *select* which selects a formula in a set of formulas. The input set of guarded assignments $U$ is defined as in the algorithms GASForward and GASBackward. The input sets of formulas $IS$ and $FS$ are any sets of formulas such that $In = \bigvee_{A \in IS} A$ and $Fin = \bigvee_{A \in FS} A$.

Local algorithms can be used not only for guarded assignment systems, but for every system in which the transition relation is represented by a finite union of transitions.

**THEOREM 23** (Soundness and Semi-Completeness) The algorithms LocalForward and LocalBackward have the following properties:
procedure LocalForward
input: sets of formulas IS, FS,
finite set of guarded assignments U
output: “reachable” or “unreachable”
begin
if there exist I ∈ IS, F ∈ FS such that
M ⊨ ∃V(I ∧ F) then
return “reachable”
unused := IS
used := ∅
while unused ≠ ∅
S := select(unused)
used := used ∪ {S}
unused := unused − {S}
forall u ∈ U
N := Su
if there exists F ∈ FS such that
M ⊨ ∃V(N ∧ F) then
return “reachable”
if for all C ∈ used ∪ unused
M ̸⊨ ∀V(N → C) then
unused = unused ∪ {N}
forall C′ ∈ used ∪ unused
if M ⊨ ∀V(C′ → N) then
remove C′ from used or unused
return “unreachable”
end

Fig. 7. Local forward reachability algorithm

procedure LocalBackward
input: sets of formulas IS, FS,
finite set of guarded assignments U
output: “reachable” or “unreachable”
begin
if there exist I ∈ IS, F ∈ FS such that
M ⊨ ∃V(I ∧ F) then
return “reachable”
unused := FS
used := ∅
while unused ≠ ∅
S := select(unused)
used := used ∪ {S}
unused := unused − {S}
forall u ∈ U
N := S−u
if there exists I ∈ IS such that
M ⊨ ∃V(N ∧ I) then
return “reachable”
if for all C ∈ used ∪ unused
M ̸⊨ ∀V(N → C) then
unused = unused ∪ {N}
forall C′ ∈ used ∪ unused
if M ⊨ ∀V(C′ → N) then
remove C′ from used or unused
return “unreachable”
end

Fig. 8. Local backward reachability algorithm

1. there is a final state reachable from an initial state if and only if the algorithm returns “reachable”;
2. if the algorithm returns “unreachable”, then there is no final state reachable from an initial state.

One can prove soundness and semi-completeness of the algorithms LocalForward and LocalBackward similar to that of Theorem 9. However, we cannot guarantee that the algorithms terminate if and only if their non-local counterparts terminate.

Example 1. Consider an example. Consider an example integer GAS with one variable v whose transition relation is represented by a single guarded assignment

\[ \text{true} \rightarrow v := v - 1. \]

Take 0 > 0 as the initial condition and v ≠ 0 as the final condition. The non-local backward reachability algorithm for GAS at the second iteration will generate the formula v ≠ 0 ∨ v ≠ 1, and at the third iteration v ≠ 0 ∨ v ≠ 1 ∨ v ≠ 2. Since these formulas
entail each other, the algorithm terminates. However, the local backward reachability algorithm for GAS generates formulas $v \neq 0$, $v \neq 1$, $v \neq 2$ which do not entail one another, and hence diverges. One can give a similar example for forward reachability.

It is not difficult to prove the following result.

**Theorem 24** If LocalForward (respectively, LocalBackward) terminates, then so does GASForward (respectively, GASBackward).

Note that termination of the local algorithms may depend on the selection function $select$. Let us call the selection function *fair* if no formula remains in *unused* forever.

**Theorem 25** If the local forward (respectively backward) algorithm terminates for some selection function, then it terminates for every fair selection function.

It is difficult to say if there are problems coming from real applications for which a non-local algorithm terminates but its local counterpart does not. One can prove, for example, that for broadcast protocols LocalBackward always terminates. In fact, this algorithm is implicitly used in the proof of decidability of broadcast protocols in [13].

The main property of the local algorithms is that subsumption and entailment-checking is only performed on simpler classes of formulas.

**Theorem 26** If the input GAS is conjunctive and the sets $IS$, $FS$ are sets of conjunctive constraints, then all of the formulas current in both algorithms LocalForward and LocalBackward are conjunctive constraints. If the input GAS is simple and the sets $IS$, $FS$ are simple constraints then all of the formulas current in LocalBackward are simple constraints too.

The last property of local reachability algorithms is heavily used in the system BRAIN. For some structures (for example the reals or the integers) the algorithms for satisfiability-checking of simple constraints are a heavily investigated subject. If the structure is $\mathbb{R}$, then a simple constraint is essentially a system of linear equations and inequations. Checking satisfiability of simple constraints means solving such a system. For solving systems of linear equations and inequations over integers or reals several off-the-shelf tools are available. For example, [11] implement satisfiability- and entailment-checking for conjunctive constraints over real numbers using the corresponding built-in functions of the SICStus Prolog constraint library.

When a backward reachability algorithm is performed on GAS, we get a quantifier elimination effect due to the special form of formulas $A^{-u}$. Theorems 21 and 26 show that in many cases entailment can be checked only between quantifier-free formulas or even simple constraints. Let us show that in some cases entailment-checking can be performed using satisfiability-checking. It is easy to see that a formula $\forall V (A \rightarrow B)$ is valid if and only if the formula $\exists V (A \land \neg B)$ is unsatisfiable. Therefore, for quantifier-free GAS and backward reachability algorithms one can use satisfiability-checking for quantifier-free formulas also for entailment-checking.

In the case of simple GAS the situation is not much more complicated. Suppose that we would like to check an entailment problem of the form $\forall V (A_1 \land \ldots \land A_n \rightarrow$
\( B_1 \land \ldots \land B_m \). This problem is equivalent to unsatisfiability of the formula \( \exists V (A_1 \land \ldots \land A_n \land (\neg B_1 \lor \ldots \lor \neg B_m)) \). This formula is equivalent to

\[
\exists V (A_1 \land \ldots \land A_n \land \neg B_1) \lor \\
\ldots \\
\exists V (A_1 \land \ldots \land A_n \land \neg B_m).
\]

Therefore, to check the original entailment problem, one has to check \( m \) satisfiability problems for the formulas \( A_1 \land \ldots \land A_n \land \neg B_i \). These formulas are not simple constraints any more, but can be made into simple constraints or disjunctions of simple constraints if the first-order theory of the structure \( \mathbb{M} \) has the following property: the negation of any atomic formula is effectively equivalent to a disjunction of atomic formulas. The first-order theory of \( \mathbb{I} \) has this property, for example the formula \( \neg x = y \) is equivalent to \( x > y \lor y > x \).

This observation shows that in many situations it is enough to implement satisfiability-checking for simple constraints in order to implement backward reachability. For example, for \( \mathbb{I} \) it is enough to implement algorithms for solving systems of linear equations and inequations.

One general obstacle for efficiency of reachability algorithms is the problem of accumulated variables, i.e., existentially quantified variables introduced by applications of the corresponding predicate transformers. We have shown that for quantifier-free GAS and backward reachability algorithms no new variables are accumulated. For forward reachability algorithms, GAS still have an advantage of (normally) accumulating only a small number of extra variables. Indeed, consider a guarded assignment \( u \) of the form

\[
P(v_1, \ldots, v_k) \Rightarrow v_1 := t_1, \ldots, v_n := t_n
\]

and assume that \( V = \{v_1, \ldots, v_n, v_{n+1}, \ldots, v_k\} \). Denote by \( t'_1, \ldots, t'_n \) the terms obtained from \( t_1, \ldots, t_n \) by replacing each variable \( v_i \) for \( i \in \{1, \ldots, n\} \) by \( v'_i \) (note that the variables \( v_{n+1}, \ldots, v_k \) are not replaced). It is not hard to argue that \( A^u(v_1, \ldots, v_k) \) is equivalent to

\[
\exists v'_1 \ldots \exists v'_n (A(v'_1, \ldots, v'_n, v_{n+1}, \ldots, v_k) \land P(v'_1, \ldots, v'_n, v_{n+1}, \ldots, v_k) \land \\
v_1 = t'_1 \land \ldots \land v_n = t'_n).
\]

In this formula the new quantifiers bind the variables \( v'_1, \ldots, v'_n \), i.e., only those variables whose values are changed by the transition. In transition systems formalizing protocols (e.g., from [10]) the number of variables whose values are changed by the transition is usually small compared to the overall number of state variables.

In some cases accumulated existentially quantified variables are not the only source of complexity. For example, for finite domains the number of variables is usually large, and repeated substitutions of terms \( t_i \) for \( v_i \) make the formula current grow exponentially. One way to make current shorter is to introduce extra existentially quantified variables to name common subexpressions, in which case the formulas current will not be quantifier-free even for simple GAS.
7 Integer-Based Systems and Hilbert’s Basis

To implement the LocalBackward one has to implement procedures for the following problems:

1. Backward application of guarded assignments: given a simple constraint $S$ and guarded assignment $u$, compute $S^{-u}$.
2. Satisfiability of simple constraints: given a simple constraint $C$, is $C$ satisfiable in $I$?
3. Entailment of simple constraints: given simple constraints $N$ and $C$, is the formula $N \rightarrow C$ valid in $I$?

To implement the reachability algorithm efficiently, one has to implement efficiently these three procedures. As our experimental data show, for hard problems the number of entailment-checks is considerably larger than the number of transition applications and satisfiability-checks. Therefore, entailment-checking should be implemented especially efficiently.

A simple constraint over $I$ can be considered as a system of linear equations and inequations over integers with variables in $V$. Since every equation $uV + l = 0$ can be equivalently replaced by two inequations $uV + l \leq 0$ and $-uV - l \leq 0$, in the sequel we will only discuss inequations. Satisfiability of simple constraints is known to be NP-complete and entailment coNP-complete. If one uses relaxation (see [10]) to use real numbers instead of integers, then both problems can be solved in polynomial time.

In this section we present some properties of the set of non-negative solutions to a simple constraint. Proofs can be found in, e.g., Schrijver [28]. In this section we consider $V$ as a vector of variables rather than a set and restrict ourselves to simple constraints with the variables $V$. Denote by $0$ a vector of 0’s. Every simple constraint $C$ with variables in $V$ can be written as a system of linear inequations with integer coefficients:

$$LV + l \leq 0.$$ (5)

where $L$ is a matrix with integer coefficients and $l$ is an integer vector. We call a solution to such a system any vector $V$ of non-negative integers which satisfies all inequations in the system. Let us emphasize that in this section we will only consider non-negative solutions. We will show below in Section 8 how to treat arbitrary integer solutions. For every system $C$ of the form (5) denote by $C_{hom}$ the corresponding system of homogeneous linear Diophantine inequations

$$LV \leq 0.$$ (6)

We call a solution $v$ to (5) non-decomposable if it cannot be represented in the form $v_1 + v_2$, where $v_1$ is a solution to (5) and $v_2$ is a non-zero solution to (6). Likewise, we call a solution to (6) non-decomposable if and only if it cannot be represented as a sum of two non-zero solutions to (6). It is easy to see that for systems of homogeneous linear Diophantine equations a solution is non-decomposable if and only if it is minimal w.r.t. the point-wise partial order on vectors, see e.g., Contejean and Devie [9].
**DEFINITION 27 (Basis)** A pair of sets of vectors \((N, H)\) is called a *basis* for a simple constraint \(C\) if the following conditions hold.

1. Every vector in \(N\) is a non-decomposable solution to \(C\).
2. Every vector in \(H\) is a non-zero non-decomposable solution to \(C_{\text{hom}}\).
3. Every solution \(v\) to \(C\) can be represented as a sum \(v = w + \sum_{i=1}^{k} m_i w_i\), where \(w \in N\), \(k \geq 0\) and for all \(i = 1 \ldots k\) \(m_i\) is a non-negative integer and \(w_i \in H\).
4. Every solution \(v\) to \(C_{\text{hom}}\) can be represented as a sum \(v = \sum_{i=1}^{k} m_i w_i\), where \(k \geq 0\) and for all \(i = 1 \ldots k\) \(m_i\) is a non-negative integer and \(w_i \in H\).

This definition is a modification of the standard definition of Hilbert’s basis [17] for the case of systems of linear inequations.

**THEOREM 28** Every simple constraint has a basis, and this basis is unique.

Algorithms for finding the basis of systems of linear Diophantine inequations are described in, e.g., Contejean and Devie [9], Ajili and Contejean [3], and Tomas and Filgueiras [29]. BRAIN uses a novel algorithm [30]. This algorithm, as well as other algorithms for funding Hilbert’s basis, is too complex to be described here. In general, it is more difficult to find the basis of a simple constraint than to check its solvability\(^1\). The solvability problem is NP-complete, but the number of vectors in the basis can be exponential in the size of the system. Nevertheless, we will show that the construction of the basis may speed up reachability-checking.

BRAIN uses an *incremental algorithm* for building the basis. We call an *incremental basis-finding function* any function \(ibff\) of two arguments, such that for every pair of simple constraints \((C_1, C_2)\), if \(B\) is the basis for \(C_1\), then \(ibff(B, C_2)\) is the basis of \(C_1 \land C_2\). Essentially, an incremental basis-funding function uses a basis computed previously for \(C_1\) to find a basis for \(C_1 \land C_2\).

### 8 BRAIN

In this section we explain how BRAIN implements the three important procedures used in the local backward reachability algorithm: backward application of guarded assignments, satisfiability and entailment. All three algorithms are implemented using repeated calls to an incremental basis-finding function \(ibff\). In order to use the basis incrementally, BRAIN stores the basis together with every computed simple constraint. This technique is similar to a technique used by Halbwachs, Proy, and Roumanoff [14] for real-valued systems. We assume that all variables range over non-negative integers and show how to handle arbitrary integers later. We call an *augmented constraint* a pair \((C, B)\) consisting of a simple constraint \(C\) and its basis \(B\).

\(^1\) Strangely enough, our experiments have shown that the existing algorithms for building the basis often outperform some well-known algorithms for checking solvability taken from integer programming packages. This could probably be explained by the fact that these packages are mostly intended for optimization and do not cope well with systems having several unbounded variables.
Entailment-checking. The algorithm for entailment-checking in BRAIN is based on the following theorem.

**THEOREM 29** Let \((C_1, (N_1, H_1))\) be an augmented constraint and \(C_2\) be a simple constraint. Then \(\emptyset \models \forall V (C_1 \rightarrow C_2)\) if and only if the following two conditions hold: (i) every vector \(v \in N_1\) is a solution to \(C_2\); (ii) every vector \(w \in H_1\) is a solution to \(C_2^{\text{hom}}\).

This theorem gives us an algorithm for entailment-checking: to check the entailment problem for augmented constraints, one has to check that the vectors of the basis of \(C_1\) are solutions to \(C_2\) or to the corresponding homogeneous system \(C_2^{\text{hom}}\). Checking that a particular vector is a solution to a system can obviously be solved in time polynomial in the size of the vector and the system. As a consequence, we obtain the following theorem.

**THEOREM 30** Entailment of augmented constraints can be solved in polynomial time.

The algorithm implicit in Theorem 29 is used in BRAIN to check the entailment. To check entailment in polynomial time one can use instead of the basis any pair of sets of vectors \((N, H)\) satisfying conditions (3) and (4) of the definition of basis, that is non-decomposability is not necessary. However, it is easy to prove that every pair of vectors \((N, H)\) with these properties contains the basis, and thus using only non-decomposable vectors saves both space and time.

Satisfiability-checking. Evidently, a simple constraint \(C\) is satisfiable if for its basis \((N, H)\) we have \(N \neq \emptyset\). So satisfiability-checking for augmented constraints is trivial. Note that the reachability algorithm makes two kinds of satisfiability-checks:

1. checking whether the new formula \(N\) (i.e., \(S^-u\)) is satisfiable;
2. intersection-checks, when we check satisfiability of the formula \(N \land I\).

The latter kind of satisfiability-checking can be performed by any satisfiability-checking procedure. But the first kind of satisfiability checks in BRAIN is combined with the backward applications of transitions for the reasons mentioned below.

Backward application of transitions. Repeated backward applications of transitions in the reachability algorithm may create too large constraints. To explain this, let us consider the formula for computing the set of states backward reachable from the set states presented by a simple constraint \(C_1(v_1, \ldots, v_n)\). If the guarded assignment \(u\) has the form \(C_2 \Rightarrow v_1 := t_1, \ldots, v_n := t_n\), then by Lemma 17 the formula \(C_1^{-u}\) is \(C_2 \land C_1(t_1, \ldots, t_n)\). The number of atomic formulas in this simple constraint is the number of atomic formulas in \(C_1\) plus the number of atomic formulas in \(C_2\). Every iteration of the reachability algorithm yields longer constraints in which, for hard examples described below in Section 9, the number of atoms may be over a hundred. It is often the case that a large number of these atoms are a consequence of the remaining atoms in the constraint and can be safely removed (in our hardest examples the number of
non-redundant atoms usually does not exceed ten). Redundant atoms in constraints do not change the basis, but they slow down entailment, since our algorithm for checking validity of \((C_1 \rightarrow C_2)\) is, roughly speaking, linear in the number of atoms in \(C_2\).

We can get rid of redundant constraints in \(C_2 \land C_1(t_1, \ldots, t_n)\) together with checking satisfiability of this constraint and building a basis for it using an incremental basis-finding function. The procedure for this is given in Figure 9. The input to this procedure is the sequence of atoms in \(C_2 \land C_1(t_1, \ldots, t_n)\) in any order. When a new atom \(A_i\) should be added to the constraint, it is first checked whether the addition of this atom changes the basis. If it does not, then the atom is redundant.

The current version of BRAIN only works with variables ranging over non-negative integers. Integers can be implemented using the same technology as follows. If an integer-valued variable is restricted by \(v \leq n\) (or respectively by \(n \leq v\)), then it can be replaced by a variable \(w = n - v\) (or respectively by \(w = v - n\)) ranging over non-negative integers. For every unrestricted integer-valued variable \(v\) one can introduce two variables \(w_1, w_2\) ranging over non-negative integers and replace all occurrences of \(v\) by \(w_1 - w_2\).

9 Experiments

In this section we present the results of experiments carried out on a number of benchmarks taken from several Web pages. The examples can be found on the Web page \(www.cs.man.ac.uk/~voronkov/BRAIN/\). We compare the performance of our system BRAIN with that of the following systems: HyTech (Henzinger, Ho, and Wong-Toi [15]), Action Language Verifier (Bultan [7]), and DMC (Delzanno and Podelski [11]).

All benchmarks were carried out on the same computers (Sparc 300 with 2G of RAM memory). These computers are slow (about 8–10 times slower than the modern PCs), but we did not have access to a network of PCs with large RAM. The systems HyTech, Action Language, and BRAIN are implemented in C++ or C, and were compiled using the same version 2.92 of the GNU C/C++ compiler. DMC is implemented in Sicstus Prolog. In several cases we had to interrupt the systems because they consumed...
over 2G of memory. DMC never consumed more than 14M of memory, but was inter-
rupted after several weeks of running. We were interested in hard benchmarks, but oc-
casionally, for the sake of comparison, included figures for relatively easy benchmarks,
because only HyTech and BRAIN could solve some of the hard ones. All runtimes are
given in seconds.

Note that HyTech and DMC use relaxation, i.e., they solve real reachability prob-
lems instead of integer reachability problems. Therefore, they are correct only when
they report non-reachability. Among the systems compared with BRAIN only Action
Language Verifier checks for integer reachability.

We took most of the benchmarks presented in this paper from Giorgio Delzanno’s
Web page www.disi.unige.it/person/DelzannoG/. The problems specified in these benchmarks were used to verify cache coherence protocols, properties of Java programs, and some other applications. The results are presented in Table 1. For
each we present the runtimes and memory consumption (in megabytes). We write −
when the compared system could not solve the problem because of the time or memory
limit.

The table shows that BRAIN is normally faster than HyTech, and sometimes con-
siderably faster. It also consumes less memory than HyTech. There are three problems
(with the suffix -inv in the name) on which HyTech was faster (denoted by negative
numbers in the speedup column). We will comment on these problems below. Consid-
ering that HyTech’s implementation uses a polyhedra library based on [14] we cannot
explain a considerable difference in the memory consumption between BRAIN and
HyTech.

For non-trivial problems BRAIN is normally several hundred times faster than
DMC, except for problems with invariants, where the difference is not so high. On non-
trivial problems BRAIN without invariants is also normally at least 500 times faster
than Action Language Verifier, on problems with invariants the difference is not so
high. BRAIN also uses less memory than Action Language Verifier.

The problems with invariants were obtained from the original problems by adding
invariants: some simple properties obtained by forward reachability analysis. A typical
invariant has the form \( v_1 + \ldots + v_k = m \), where \( m \) is a natural number. In fact, it
bounds the variables \( v_1, \ldots, v_k \) to a finite region. Such invariants cause a problem to
BRAIN, because the basis for problems with such an invariant usually contains all, or
a large portion, of the points in this region explicitly. We believe that this problem is
not essential for the approach, but is rather particular to the current implementation of
BRAIN in which the basis is stored explicitly, point-wise. A symbolic representation
of this finite region, or the use of suitable datastructures for presenting finite-domain
variables should solve this problem.

There are several problems which could only be solved by BRAIN, but not by any
of the other systems. However, we would like to note that all of these systems are
on some benchmarks more powerful than BRAIN since they can use techniques such
as widening or transitive closure which the current version of BRAIN does not have.
Examples are some versions of the ticket protocol.

To give the reader an idea of the complexity of the problems solved by BRAIN, we
present statistics about the number of operations such an entailment-checks performed
## Table 1. Statistics

<table>
<thead>
<tr>
<th>problem</th>
<th>variables</th>
<th>intersections</th>
<th>entailment</th>
<th>transitions</th>
<th>time</th>
<th>memory</th>
<th>time</th>
<th>memory</th>
<th>speedup</th>
<th>time</th>
<th>memory</th>
<th>speedup</th>
<th>time</th>
<th>memory</th>
<th>speedup</th>
<th>time</th>
<th>memory</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BRAIN</td>
<td>HyTech</td>
<td>ALV</td>
<td>DMC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csm5</td>
<td>13</td>
<td>3,313</td>
<td>790,119</td>
<td>3,576</td>
<td>9.26s</td>
<td>2</td>
<td>40.32s</td>
<td>18</td>
<td>4.35</td>
<td>87m32s</td>
<td>297</td>
<td>567</td>
<td>13h15m</td>
<td>11</td>
<td>5165</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csm10</td>
<td>13</td>
<td>27,308</td>
<td>60,803,697</td>
<td>29,736</td>
<td>457s</td>
<td>11</td>
<td>868s</td>
<td>140</td>
<td>1.9</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>csm15</td>
<td>13</td>
<td>107,503</td>
<td>990,874,884</td>
<td>117,496</td>
<td>119m13s</td>
<td>60</td>
<td>154m32s</td>
<td>509</td>
<td>1.3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>csm5-inv</td>
<td>13</td>
<td>106</td>
<td>835</td>
<td>152</td>
<td>0.37s</td>
<td>1</td>
<td>0.96s</td>
<td>2</td>
<td>2.6</td>
<td>2.87s</td>
<td>28</td>
<td>7.77</td>
<td>3.2s</td>
<td>10</td>
<td>8.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csm10-inv</td>
<td>13</td>
<td>106</td>
<td>835</td>
<td>152</td>
<td>0.89s</td>
<td>1</td>
<td>0.96s</td>
<td>2</td>
<td>1.07</td>
<td>2.87s</td>
<td>28</td>
<td>3.22</td>
<td>3.2s</td>
<td>10</td>
<td>3.59</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>csm15-inv</td>
<td>13</td>
<td>106</td>
<td>835</td>
<td>152</td>
<td>1.88s</td>
<td>1</td>
<td>0.96s</td>
<td>2</td>
<td>-1.95</td>
<td>2.87s</td>
<td>28</td>
<td>1.52</td>
<td>3.2s</td>
<td>10</td>
<td>1.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consistencyprot</td>
<td>12</td>
<td>813</td>
<td>30,557</td>
<td>880</td>
<td>0.96s</td>
<td>1</td>
<td>17.66s</td>
<td>10</td>
<td>18.4</td>
<td>17.66s</td>
<td>10</td>
<td>18.4</td>
<td>888s</td>
<td>10</td>
<td>925</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consistencyprot-inv</td>
<td>12</td>
<td>813</td>
<td>29,803</td>
<td>880</td>
<td>7.25s</td>
<td>1</td>
<td>0.13s</td>
<td>2</td>
<td>-55</td>
<td>164.8s</td>
<td>73</td>
<td>22.7</td>
<td>19m52s</td>
<td>11</td>
<td>164.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consprod</td>
<td>18</td>
<td>162,817</td>
<td>698,478,060</td>
<td>181,650</td>
<td>173m31s</td>
<td>22</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>consprod-inv</td>
<td>18</td>
<td>187</td>
<td>5,126</td>
<td>742</td>
<td>1.19s</td>
<td>1</td>
<td>1.2s</td>
<td>4</td>
<td>1.01</td>
<td>25.87s</td>
<td>68</td>
<td>21.7</td>
<td>4.9</td>
<td>10</td>
<td>4.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>incdec</td>
<td>32</td>
<td>41,971</td>
<td>12,762,257</td>
<td>42,252</td>
<td>170.3s</td>
<td>10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>incdec-inv</td>
<td>32</td>
<td>873</td>
<td>54,824</td>
<td>4,004</td>
<td>22.9s</td>
<td>1</td>
<td>120.9s</td>
<td>80</td>
<td>5.3</td>
<td>10h20m</td>
<td>846</td>
<td>1625</td>
<td>96.3s</td>
<td>10</td>
<td>4.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bigjava</td>
<td>44</td>
<td>122,516</td>
<td>93,410,447</td>
<td>134,828</td>
<td>171m31s</td>
<td>25</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bigjava1</td>
<td>44</td>
<td>127,185</td>
<td>95,800,396</td>
<td>139,688</td>
<td>189m16s</td>
<td>25</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bigjava-inv</td>
<td>44</td>
<td>7,581</td>
<td>3,378,979</td>
<td>45,103</td>
<td>40m2s</td>
<td>11</td>
<td>32m5s</td>
<td>849</td>
<td>-1.25</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bigjava1-inv</td>
<td>44</td>
<td>49,531</td>
<td>48,384,856</td>
<td>104,538</td>
<td>744m14s</td>
<td>19</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
by BRAIN during each run. This statistics shows why DMC is hopelessly slow on some of these problems: for example, in the case of \texttt{cs1m15} one can hardly check almost $10^9$ entailment problems in reasonable time using general-purpose constraint-solving tools. BRAIN solves them in less than 2 hours (on a fast PC with Intel this time would be less than 15 minutes). The table shows that, for most of the benchmarks, entailment seems to be the most important operation. It also demonstrates slowdown of BRAIN on the problems with invariants: indeed the number of operations per second in these problems is much smaller than that for their original formulations without the invariants.

10 Other Issues

In this section we discuss some other issues related to our formalization of transition systems. There are still many topics not covered here, such as fairness constraints, or concurrent applications of transitions.

10.1 Termination of the Local Algorithms

Simple constraints are usually easier to handle, especially in the case of constraints over the reals or integers. To deal only with simple constraints, one should use local backward reachability algorithms (see Theorem 26). As Example 1 shows, the use of local algorithms can make backward reachability analysis not terminate when it terminates for non-local algorithms.

10.2 Widening

When the reachability algorithms diverge, one can apply technique known as \textit{approximation} or \textit{widening}. This technique can be formalized as follows. Instead of using an equivalence-preserving simplification function \texttt{simp} we use a \textit{widening function} \texttt{wide}, such that $\mathbb{M} \models \forall V (A \rightarrow \text{wide}(A))$. As a result, the set of states computed by the reachability algorithms in the limit, is a superset of the set of reachable states. The algorithms using widening are not semi-decision procedures and correct only if they return “unreachable”. Widening can be used to make diverging algorithms converge in some cases, but also to make the formulas current simpler.

10.3 Transitive Closure

\textit{Transitive closure} is a technique used by [8] to make the reachability analysis converge. The idea is to try to use a transitive closures of some transitions instead of the transitions themselves. For example, the transition

$$v_1 > 0 \Rightarrow v_1 := v_1 - 1, v_2 := v_2 + 1$$

over $\mathbb{I}$ can be replaced by its transitive closure

$$v_1 > 0 \land v'_1 < v_1 \land v'_1 + v'_2 = v_1 + v_2.$$
The use of transitive closure can make otherwise non-terminating reachability analysis terminate. Unlike widening, it does not change the set of computed reachable states, so it is sound and semi-complete in the same sense as the other algorithms presented here. However, transitive closure applied to a GAS gives a system which is not a GAS, and hence our results on GAS are not applicable any more.

10.4 Relaxation

The system DMC [11] uses the technique known in integer programming as relaxation. Instead of solving the reachability problem over the integers, one solves “the same” reachability problem over the reals. Similarly, [10] analyzes integer reachability problems using the Hy-Tech model checker in which the state variables are real-valued [15]. Relaxation is attractive since solving satisfiability and entailment problems over the reals is easier than solving these problems over the integers. For example, satisfiability and entailment of simple constraints can be solved in polynomial time over the reals but are respectively NP-complete and coNP-complete over the integers. In this section we formalize relaxation in a general setting.

Let $M_1$ and $M_2$ be two structures of the same signature $\Sigma$ over the domains $D_1$ and $D_2$ respectively, and $\epsilon$ be an isomorphic embedding of $M_1$ into $M_2$, i.e., a function from $D_1$ to $D_2$ with the following properties

1. For every predicate symbol $P$ of arity $n$ of the signature $\Sigma$ and valuation $s$ for a set of variables $\{x_1, \ldots, x_n\}$ in $M_1$ we have $M_1, s \models P(x_1, \ldots, x_n)$ if and only if $M_2, \epsilon \circ s \models P(x_1, \ldots, x_n)$.
2. For every function symbol $P$ of arity $n$ of the signature $\Sigma$ and valuation $s$ for a set of variables $\{x, x_1, \ldots, x_n\}$ in $M_1$ we have $M_1, s \models x = f(x_1, \ldots, x_n)$ if and only if $M_2, \epsilon \circ s \models x = f(x_1, \ldots, x_n)$.

These properties guarantee that for any quantifier-free formula $A$ and valuation $s$ for its variables in $M_1$ we have $M_1, s \models A$ if and only if $M_2, s \models A$.

**Definition 31 (Relaxation)** Let $R = (Tr, In, Fin)$ be an instance of the reachability problem over a structure $M_1$ and there exists an isomorphic embedding of $M_1$ into a structure $M_2$. The instance $(Tr, In, Fin)$ of the reachability problem over $M_2$ is called the relaxation of $R$.

That is, the original problem and its relaxation use the same formulas but are considered over different structures.

**Theorem 32** Let $R_1 = (Tr, In, Fin)$ be an instance of the reachability problem over $M_1$ and $R_2$ be its relaxation over $M_2$. Suppose that all formulas in $(Tr, In, Fin)$ are equivalent to existential formulas in $M_1$. If $R_2$ does not hold, then $R_1$ does not hold too.

This theorem means that one can solve relaxation instead of the original problem to show non-reachability as soon as the initial and final conditions and the transition formula are existential or equivalent to existential formulas in $M_1$. All special classes
of transition systems (for example, positive existential) introduced in this paper have this property. One should be careful about using relaxation when properties other than reachability are checked. For example, if guards in a GAS are existentially quantified formulas, than one cannot in general check deadlock-freedom using relaxation. As stated in Theorem 16, the deadlock formula for GAS is \( \neg (P_1 \lor \ldots \lor P_n) \), which may not be equivalent to an existential formula, as soon as at least one of the \( P_i \) is existentially quantified.

### 10.5 Other Temporal Formulas

As observed in many papers, a possibility to build fixpoints implies that properties more complex than reachability could be verified. It is not hard to argue that the following property, related to the fixpoint construction, is true for all of the algorithms of this paper.

**Theorem 33** If the forward reachability algorithm terminates with the answer “unreachable” then current symbolically represents the set of all states reachable from initial states. If the backward reachability algorithm terminates with the answer “unreachable” then current symbolically represents the set of all states backward reachable from the final states.

This property allows one to extend the reachability algorithms and some results to more general classes of temporal formulas than those expressing reachability, but the analysis of such extended algorithms is beyond the scope of this paper.

### 11 Related Work

Podelski [24] and Delzanno and Podelski [11] formalize model checking procedures as constraint solving using a rather general framework. In fact, what they call constraints are first-order formulas over structures. They treat a class of properties more general than reachability and use GAS and local algorithms. Our formalization has much in common with their formalization but our results are, in a way, orthogonal, because they do not study special classes of formulas. Among all results proved in this paper, only soundness and semi-completeness are similar to those studied by Delzanno and Podelski. All other results of this paper are new. In addition, Delzanno and Podelski do not discuss selection functions, so their algorithms are less general than ours. The class of systems for which Delzanno and Podelski’s algorithms can be applied is also less general than the one studied here. Indeed, they require a constraint solver also to implement variable elimination. If a theory has such a solver, then this theory also has quantifier-elimination (every formula is equivalent to a boolean combination of constraints). The solver is also able to decide satisfiability of constraints, so in fact they deal with theories which are both decidable and have quantifier elimination. They do not notice that in many special cases (as studied in this paper) quantifier elimination is not required. Thus, our results encompass some important theories (such as those of queues in Bjørner [5]) not covered by the results of Delzanno and Podelski.
An approach to formalizing reachability for hybrid systems using first-order logic is presented by Lafferriere, Pappas, and Yovine [21]. They also note that one can use arbitrary first-order formulas (over reals) due to quantifier elimination. Henzinger and Majumdar [16] present a classification of state transition systems, but over the reals.

Lemma 17 which observes that for GAS backward reachability algorithms do not introduce extra variables is similar to the axiom of assignment for the weakest preconditions. This property is heavily used in BRAIN [27]. It is possible that it has been exploited in other backward reachability symbolic search tools but we could not find papers which observe this property.

Non-local reachability procedures for model checking were already formulated in Emerson and Clarke [12], Queille and Sifakis [26]. Delzanno and Podelski [11] studied algorithms based on local entailment in the framework of symbolic model checking. They do not consider the behaviour of local algorithms for different selection functions. Local algorithms could be traced to early works in automated reasoning and later works on logic programming with tabulation (e.g., Warren [31]), constraint logic programming, and constraint databases (e.g., Kanellakis, Kuper, and Revesz [18]).

Kesten, Maler, Marcus, Pnueli, and Shahar [19] present a general approach to symbolic model checking of infinite-state systems, but based on a single language rather than different languages for different structures.

Guarded assignment systems (under various names) are studied in many other papers, too numerous to be mentioned here. There are also many papers on (un)decidability results for infinite-state systems, both over integers and other structures, not mentioned here.

Our technique of using Hilbert’s basis is similar to a technique used to deal with real-valued systems described in Halbwachs, Proy and Roumanoff [14]. They represent convex polyhedra using systems of generators, i.e., two finite sets of vectors (called vertices and rays). This representation allows one to perform efficient entailment checks using a property similar to that of Theorem 29.

One can implement satisfiability- and entailment-checking using the decision procedure for the first order theory of \( \mathbb{I} \). In some cases, one can even use off-the-shelf decision procedures or libraries. For example, Bultan, Gerber, and Pugh [8] use the Omega Library [25] for deciding Presburger arithmetic, Bultan [7] uses the Composite Library (Yavuz-Kahveci, Tuncer and Bultan [32]), Delzanno and Podelski [11] use the CLP(\( \mathbb{R} \)) library of Sicstus Prolog. The use of decision procedures for Presburger arithmetic has several advantages, since formulas more general than simple constraints can be handled. As a consequence, one can use non-local reachability algorithms, forward reachability, and apply techniques such as widening and transitive closure which cannot be handled by the current version of BRAIN. However, the use of general-purpose algorithms to decide specific classes of formulas may be inefficient, which is confirmed by our experiments.

In the future we are going to develop BRAIN into an advanced infinite-state model-checker based on the implementation method proposed here, which will be both faster and more flexible. In particular, we will include in BRAIN other reachability algorithms and techniques such as widening. This will, however, require an implementation of
quantifier elimination and an extension of our method, and especially Theorem 29, to constraints with existentially quantified variables and divisibility constraints.

To cope with the problem of large finite regions, one has to introduce their convenient symbolic representation, which may require reworking of all algorithms. It would also be interesting to apply our method to real-valued systems, or systems with both integer- and real-valued variables.

Acknowledgments

We thank Howard Barringer, Giorgio Delzanno, and Andreas Podelski for their comments on earlier versions of this paper.

References

A system consists of a process, of an environment and of possible ways of interaction between them. The synthesis problem is: given a specification $\phi$ find a program $P$ for the process such that the overall behaviour of the system satisfies no matter what the behaviour of the environment is.

One of the motivations for the development of automata theory was the interest in verifying and synthesizing switching circuits. A circuit can be modeled as a function transforming an infinite sequence of inputs into an infinite sequence of outputs. Church in 1963 has posed the problem of synthesizing such a function for a specification given in S1S, the monadic second order logic of one successor. The solution to this problem was given by Buchi and Ladweber in 1969. Nowadays we understand that this problem, and many other variations of the synthesis problem, reduce to finding a winning strategy in a two player game with perfect information and regular winning conditions.

A distributed system consists of some number of processes, each interacting in some way with the environment and other processes. The distributed synthesis problem is to find a program for each of the processes so that the global behaviour satisfies the given specification no matter what the behaviour of the environment is.

There are numerous ways of describing a distributed system and hence there are numerous ways of formalizing the distributed synthesis problem. One way, suggested by Pnueli and Rosner, is to consider a system as an architecture: a graph of communication channels connecting boxes into which we need to put I/O programs. The other way, introduced by Rudie and Wonham, is to specify the sets of controllable and observable actions for each of the components of the system. For these and other settings there are algorithms solving the distributed synthesis problem for some cases.

In the distributed synthesis we do not understand very well what makes the problem decidable or undecidable. Each decidability/undecidability proof depends very strongly on the formalism being used. One of the reasons for this is that, unlike for simple synthesis, we cannot reduce distributed synthesis problems to the problem of solving a two player game for a simple reason that in the distributed case there are more than two parties.

In this talk we will summarize the existing approaches to synthesis and distributed synthesis problems. We will propose a notion of distributed game that can serve as a common framework for distributed synthesis problems. In such a game there is one environment player and there is a coalition of process players, each having only a partial information about the play. We will state a couple of results on solvability of such games and indicate why they imply most of the known results for distributed synthesis problems.
Logical Relations for Dynamic Name Creation

Yu Zhang and David Nowak
LSV, CNRS UMR 8643, ENS de Cachan
94235 Cachan Cedex, France
{zhang,nowak}@lsv.ens-cachan.fr

Abstract. Pitts and Stark’s nu-calculus is a typed lambda-calculus which forms a basis for the study of interaction between higher-order functions and dynamically created names. A similar approach has received renewed attention recently through Sumii and Pierce’s cryptographic lambda-calculus, which deals with security protocols. Logical relations are a powerful tool to prove properties of such a calculus, notably observational equivalence. While Pitts and Stark construct a logical relation for the nu-calculus, it rests heavily on operational aspects of the calculus and is hard to be extended. We propose an alternative Kripke logical relation for the nu-calculus, which is derived naturally from the categorical model of the nu-calculus and the general notion of Kripke logical relation. This is also related to the Kripke logical relation for the name creation monad by Goubault-Larrecq et al. (CSL’2002), which the authors claimed had similarities with Pitts and Stark’s logical relation. We show that their Kripke logical relation for names is strictly weaker than Pitts and Stark’s. We also show that our Kripke logical relation, which extends the definition of Goubault-Larrecq et al., is equivalent to Pitts and Stark’s up to first-order types; our definition rests on purely semantic constituents, and dispenses with the detours through operational semantics that Pitts and Stark use.

Keywords: Kripke logical relation, name creation, nu-calculus, categorical models of lambda calculi

1 Introduction

The nu-calculus of Pitts and Stark is a typed lambda-calculus, extended with state in the form of dynamically created names. Through the interaction between names and functions, the language can capture notions of scope, visibility and sharing. It has also connections to local declarations in general; to the mobile processes of the π-calculus; to security protocols in the spi-calculus; and particularly, to the cryptographic lambda-calculus of Sumii and Pierce [15]. When we regard dynamically created names as dynamically created keys, the nu-calculus is indeed a syntactical subset of the cryptographic lambda-calculus. It contains a primitive for key generation but lacks those for encryption and decryption.

* Partially supported by the RNTL project EVA, the ACI jeunes chercheurs “Sécurité informatique, protocoles cryptographiques et détection d’intrusions” and the ACI Cryptologie “PSI-Robuste”.

© Springer-Verlag Berlin Heidelberg 2003
Logical relations are a powerful technique to prove properties of typed lambda calculi such like the nu-calculus and the cryptographic lambda-calculus. In [11,12], Pitts and Stark define an operational logical relation to establish the observational equivalence between nu-calculus expressions. They also prove that this logical relation is complete for first-order types. But the operational logical relation is a quite syntactic one and it is defined based on some operational semantics of the nu-calculus, which makes it hard to be extended.

We propose another logical relation for the nu-calculus in a natural way. This logical relation follows the general notion of Kripke logical relations [5] and is derived from a denotational model of the nu-calculus, which is a functor category with a strong monad. It is also related to the Kripke logical relation for the name creation monad by Goubault-Larrecq, Lasota and Nowak [3]. The latter claims to have similarities with Pitts and Stark’s operational logical relation, while we shall show that it is indeed strictly weaker than Pitts and Stark’s. We extend Goubault-Larrecq, Lasota and Nowak’s definition so as to derive naturally our Kripke logical relation in a categorical style. We also show that, this logical relation is equivalent to Pitts and Stark’s operational logical relation up to first-order types.

As preliminaries, Section 2 gives a brief review of the nu-calculus and the operational logical relation, more details of which can be found in [11,12,14]. Section 3 introduces a categorical model $\text{Set}^Z$ for the nu-calculus [12,13] to provide a basis for our Kripke logical relation. We give in Section 4 the definition of the Kripke logical relation, while before that, we derive another one from Goubault-Larrecq, Lasota and Nowak’s definition for name creation monad in [3] and show that this one is strictly weaker than Pitts and Stark’s operational logical relation. At the end of Section 4, we show that ours is equivalent to Pitts and Stark’s up to first-order types.

2 The nu-Calculus and the Operational Logical Relation

2.1 The nu-Calculus

The nu-calculus is a small language designed to show the interaction between dynamically created names and higher-order functions [11,12]. It is a typed call-by-value lambda-calculus extended with the notion of a name: names have their own type $\nu$, they can be created fresh, passed around and tested for equality. Higher-order functions and booleans are also available, but to ensure termination functions cannot be defined recursively. New names are created in expressions of the form $\nu n. M$, which binds a fresh name to the identifier $n$ and then evaluates $M$. The use of a call-by-value semantics means that although only the expression $M$ can refer to $n$ explicitly, the new name itself may escape from this scope. For example, $(\lambda x^\nu.x = x)(\nu n. n)$ evaluates to $true$, with $x$ bound to a name rather than to $\nu n. n$ itself.

The syntax of the nu-calculus is based on the simply-typed lambda-calculus. Types take the form

$$\sigma ::= o \mid \nu \mid \sigma \to \sigma$$
where \( o \) and \( \nu \) are two ground types denoting *booleans* and *names* respectively. Terms take the form

\[
M ::= x \quad \text{variable} \\
    | \ n \quad \text{name} \\
    | \ \text{true} \ | \ \text{false} \quad \text{boolean values} \\
    | \ \text{if} \ M \ \text{then} \ M \ \text{else} \ M \quad \text{conditional} \\
    | \ M = M \quad \text{equality of names} \\
    | \ \nu n. M \quad \text{local name declaration} \\
    | \ \lambda x^\sigma. M \quad \text{function abstraction} \\
    | \ MM \quad \text{function application}
\]

where \( x \in \text{Var} \), an infinite set whose elements are called *variables*, and \( n \in \text{Nme} \), an infinite set (disjoint from \( \text{Var} \)) whose elements are called *names*. Function abstraction is a variable-binding construct (occurrences of \( x \) in \( M \) are bound in \( \lambda x^\sigma. M \)), whereas local name declaration is a name-binding construct (occurrences of \( n \) in \( M \) are bound in \( \nu n. M \)). We implicitly identify expressions which only differ in their choice of bound variables and names (\( \alpha \)-conversion).

An expression is *closed* if it has no free variables; a closed expression may still have free names. We denote by \( M[M'/x] \) (respectively \( M[M'/n] \)) the result of substituting the expression \( M' \) for free occurrences of the variable \( x \) (respectively, the name \( n \)) in the expression \( M \). The substitution is *capture avoiding*. This can always be arranged by \( \alpha \)-converting \( M \).

Expressions will be assigned types via typing assertions of the form

\[
s, \Gamma \vdash M : \sigma
\]

where \( s \) is a finite subset of \( \text{Nme} \), \( \Gamma \) is a finite set of typed variables, \( \sigma \) is a type, and \( M \) is an expression (more precisely, an \( \alpha \)-equivalence class of expressions) with free names in \( s \) and free variables in \( \Gamma \).

An expression is in *canonical form* if it is either a name, a variable, one of the boolean constants \( \text{true} \) or \( \text{false} \), or a function abstraction. These are the *values* of the \( \nu \)-calculus. As what Pitts and Stark do in \[11,12\], we define the sets

\[
\begin{align*}
\text{Exp}_\sigma&(s, \Gamma) = \{ M \mid s, \Gamma \vdash M : \sigma \} \\
\text{Can}_\sigma&(s, \Gamma) = \{ C \mid C \in \text{Exp}_\sigma(s, \Gamma), C \ \text{canonical} \} \\
\text{Exp}_\sigma&(s) = \text{Exp}_\sigma(s, \emptyset) \\
\text{Can}_\sigma&(s) = \text{Can}_\sigma(s, \emptyset)
\end{align*}
\]

of expressions and canonical expressions at any type \( \sigma \) and for any finite sets \( s, \Gamma \) of names and typed variables.

The operational semantics of the \( \nu \)-calculus is specified in \[11,12\] by an inductively defined evaluation relation. Elements of the relation take the form

\[
s \vdash M \Downarrow^s_\sigma C
\]
where \( s \) and \( s' \) are disjoint, \( M \in \text{Exp}_\sigma(s) \) and \( C \in \text{Can}_\sigma(s \oplus s') \). This is intended to mean that in the presence of the names \( s \), expression \( M \) of type \( \sigma \) evaluates to canonical form \( C \) while creating the fresh names in \( s' \) in the process. The sets \( s \) and \( s \oplus s' \) can be seen as initial and final states of the computation. See [11,12] for details of the evaluation relation.

### 2.2 The Operational Logical Relation

An operational logical relation between nu-calculus expressions is given by Pitts and Stark in [11,12]. It rests on the operational semantics of the nu-calculus and a certain kind of relation between sets of names.

**Definition 1 (Spans).** If \( s_1 \) and \( s_2 \) are sets of names, then a span \( R : s_1 \rightleftharpoons s_2 \) is an injective partial map from \( s_1 \) to \( s_2 \). That is, the graph \( R \subseteq s_1 \times s_2 \) satisfies

\[
(n_1 \; R 
\; n_2) \; \& \; (n'_1 \; R 
\; n'_2) \implies (n_1 = n'_1 \iff n_2 = n'_2).
\]

A span can also be represented as a pair of injections \( s_1 \hookrightarrow R 
\twoheadrightarrow s_2 \).

If \( R' : s'_1 \rightleftharpoons s'_2 \) is another span, with \( s'_1 \) and \( s'_2 \) disjoint from \( s_1 \) and \( s_2 \) respectively, then the disjoint union of \( R \) and \( R' \) is also a span:

\[
R \oplus R' : s_1 \oplus s'_1 \rightleftharpoons s_2 \oplus s'_2.
\]

The domain and co-domain of definition for \( R : s_1 \rightleftharpoons s_2 \) are defined by:

\[
\text{dom}(R) = \{ n_1 \in s_1 \mid \exists n_2 \in s_2 \; (n_1 \; R 
\; n_2) \}
\]

\[
\text{cod}(R) = \{ n_2 \in s_2 \mid \exists n_1 \in s_1 \; (n_1 \; R 
\; n_2) \}
\]

The intuition behind spans is that they should capture the use that nu-calculus expressions make of public and private names. So if \( R : s_1 \rightleftharpoons s_2 \), then the bijection between \( \text{dom}(R) \subseteq s_1 \) and \( \text{cod}(R) \subseteq s_2 \) represents matching use of “visible” names, while the remaining elements not in the graph of \( R \) are “unseen” names.

**Definition 2 (Operational logical relation).** If \( R : s_1 \rightleftharpoons s_2 \) is a span then the relations

\[
R^\text{can}_\sigma \subseteq \text{Can}_\sigma(s_1) \times \text{Can}_\sigma(s_2)
\]

\[
R^\text{exp}_\sigma \subseteq \text{Exp}_\sigma(s_1) \times \text{Exp}_\sigma(s_2)
\]

are defined by induction over the structure of the type \( \sigma \), according to:

\[
\lambda x^\sigma.M_1 \stackrel{R^\text{can}_{\sigma \to \sigma'}}{\longrightarrow} (\lambda x^\sigma.M_2) \iff \\
\forall R' : s'_1 \rightleftharpoons s'_2, C_1 \in \text{Can}_\sigma(s_1 \oplus s'_1), C_2 \in \text{Can}_\sigma(s_2 \oplus s'_2)
\]
Logical Relations for Dynamic Name Creation

$$(C_1 (R \oplus R')^\text{can}_\sigma C_2 \implies M_1[C_1/x] (R \oplus R')^\text{exp}_\sigma M_2[C_2/x])$$

$M_1 R^\text{exp}_\sigma M_2 \iff$

$$\exists R': s_1' \Leftrightarrow s_2', C_1 \in \text{Can}_\sigma(s_1 \oplus s_1'), C_2 \in \text{Can}_\sigma(s_2 \oplus s_2')$$

$$(s_1 \vdash M_1 \Downarrow s_1' \sigma \downarrow C_1 \& s_2 \vdash M_2 \Downarrow s_2' \sigma \downarrow C_2 \& (R \oplus R')^\text{can}_\sigma C_2)$$

The relation $R^\text{can}_\sigma$ and $R^\text{exp}_\sigma$ coincide on canonical forms, and we may write them as $R_\sigma$ indiscriminately.

This operational logical relation can be used to establish observational equivalences in the nu-calculus. Consider the two nu-calculus expressions of type $\nu \rightarrow o$: $\nu n.\lambda x.((x = n))$ and $\lambda x.\nu.f.\false$. According to the evaluation relation in [11,12],

$$s, \Gamma \vdash \nu n.\lambda x.((x = n)) \Downarrow \{n\} \lambda x.((x = n))$$

and

$$s, \Gamma \vdash \lambda x.\nu.f.\false \Downarrow \emptyset \lambda x.\nu.f.\false.$$  

The first term creates a new name, which is definitely private to the function, and then compare it with the argument. Since no external context can supply this name, the function always returns the value $\false$, so it is observationally equivalent to the second term. The only possible span for new names is the empty relation $R': \{n\} \Rightarrow \emptyset$. Whatever the span $R: s_1 \Rightarrow s_2$ is, we have

$$\lambda x.((x = n)) (R \oplus R')^\text{can}_\nu \rightarrow o \lambda x.\nu.f.\false$$

and consequently, the above two expressions are related.

3 A Categorical Model for the nu-Calculus

As the definition of $R^\text{exp}_\sigma$ shows, the operational logical relation relies heavily on the operational semantics of the nu-calculus. Moreover this definition silently assumes that each term has a canonical form — it is indeed the case in nu-calculus. This prevents one from extending this relation to non-terminating calculi with names. This section introduces the categorical model $\text{Set}_I$ defined by Stark in [12,13] to give the denotational semantics for the nu-calculus. This model is a functor category equipped with a strong monad and it provides us not only a denotational model for the behavior of nu-calculus expressions, but also a sound basis for a logical relation to be defined next.

3.1 Categorical Model

Let $\mathcal{I}$ be the category of finite sets and injective functions. Let $\mathcal{C}$ be the category $\text{Set}_\mathcal{I}$ of functors from $\mathcal{I}$ to $\text{Set}$ and natural transformations. For short, we write $TAs$ for $T(A)(s)$ and similarly for other notations. Let $+$ denote disjoint union in $\mathcal{I}$.

We consider the strong monad $(T, \eta, \mu, t)$ on $\mathcal{C}$ defined in [12,13,13,3]:

1. Note that $+$ is not a coproduct in $\mathcal{I}$ which in fact does not have a coproduct. However $+$ is functorial in both components, associative, and has a neutral element.
\(- TA = \text{colim}_{s'} A(- + s') : I \to \text{Set}.\) Explicitly:

- \(TA\)s is the set of equivalence classes of pairs \((s', a)\), where \(s' \in I\) and \(a \in A(s + s')\), modulo the smallest equivalence relation \(\equiv_C\) such that \((s', a) \equiv_C (s'', A(\text{id}_s + i)a)\) for each morphism \(i : s' \to s''\) in \(I\). We write \([s, a]\) for the equivalence class of \((s, a)\).

- On morphism \(i, TA\) maps the equivalence class of \((s', a)\), to the equivalence class of \((s', a')\).

- For any \(F : F \to G\) in \(\mathcal{C}\), \(\mathcal{T}F s : \mathcal{T}F s \to \mathcal{T}G s\) is defined by \(\mathcal{T}F s[s', a] = [s', f(s + s')a]\). This is compatible with \(\equiv_C\) because \(f\) is natural.

- \(\eta_{As} : As \to TA\)s is defined by \(\eta_{Asa} = [\emptyset, a]\).

- \(\mu_{As} : T^2 A \to TA\)s is defined by \(\mu_{As}[s', [s'', a]] = [s' + s'', a]\)

- \(t_{A_1, A_2} s : A_1 s \times TA_2 s \to T(A_1 \times A_2)s\) is defined by \(t_{A_1, A_2} s(a, [s', b]) = [s', (A_1 i_s, s'a, b)]\) where \(i_s, s' : s \to s + s'\) is the canonical injection.

It is standard that this category is cartesian closed. If \(F, G : I \to \text{Set}\) are functors then their exponent is defined by the standard construction in covariant presheaves [4]:

\[
G^F s = \text{Set}^I(I(s, -) \times F, G)
\]

\[
G^F ifs''(j, a'') = f s''(j \circ i, a''),
\]

where \(s, s', s'', i : s \to s', j : s' \to s'' \in I\) and \(f \in G^F s, a'' \in Fs''\).

### 3.2 The Interpretation of the nu-Calculus

The interpretation of the nu-calculus is based on a computational metalanguage [12][13], which stems from Moggi's computational lambda calculus [8][9][10]. In this particular metalanguage, besides the usual boolean type, name type and function types, we have a unary type constructor \(T\): if \(A\) is a type, then elements of \(TA\) are computations of type \(A\). The difference between a value and a computation is that a computation may generate new names before returning a value. A nu-calculus expression is regarded as a computation in the metalanguage which generates a set of names and then returns a value, and an expression in canonical form simply returns a value without generating any name.

The category \(\text{Set}^I\) is qualified to be a categorical model of this metalanguage [12][13]. In this category, objects of \(I\) represent stages of computation, i.e., what names have been created. For a functor \(A : I \to \text{Set}\), the set \(As\) is composed of values defined over the names in \(s\). Morphisms in \(I\) and their images in \(\text{Set}\) correspond to name substitutions.

We begin by defining two functors \(B\) and \(N\) from \(I\) to \(\text{Set}\). Functor \(B\) is defined by:

\[
\forall s \in I, Bs = \text{Bool} \quad \forall i : s \to s' \in I, Bi = \text{id}_{\text{Bool}},
\]

and functor \(N\):

\[
\forall s \in I, Ns = s \quad \forall i : s \to s' \in I, Ni = i,
\]
where $\text{Bool}$ is a set containing only two elements — $tt$ and $ff$. Types in the nu-calculus are interpreted as objects of $\text{Set}^I$:

$$
\llbracket o \rrbracket = B \\
\llbracket \nu \rrbracket = N \\
\llbracket \sigma \rightarrow \sigma' \rrbracket = (T[\sigma'])^{[\sigma]}.
$$

According to what Stark does in [12,13], for each valid type assertion $s$, $\Gamma \vdash M : \sigma$ in the nu-calculus, the semantics of $M$ in the categorical model is given with an index $s$:

$$
\llbracket M \rrbracket \Gamma, s \in (T[\sigma])^{[\Gamma]}_s \quad \text{where } [\Gamma] = \prod_{x : \sigma \in \Gamma} [\sigma].
$$

If $M$ is a closed term, i.e., $M \in \text{Exp}_\sigma(s)$, then $\llbracket M \rrbracket_s \in T[\sigma]_s$, identifying $A^1$ with $A$ where $1$ is the terminal object in $\text{Set}^I$ and $1_s$ is a singleton for all $s$. The semantics of a closed canonical expression $\llbracket C \rrbracket_s$ is always of the form $[\emptyset, |C|_s]$, an element of the set $T[\sigma]_s$. Here $|C|_s \in [\sigma]_s$ and $|-|_s$ is the canonical interpretation defined for each closed canonical form. If $s \vdash M \Downarrow \sigma' \ C$, where $M \in \text{Exp}_\sigma(s)$ and $C \in \text{Can}_\sigma(s)$, the semantics of the expression $M$ can also be given by the canonical interpretation of the corresponding canonical form $C$:

$$
\llbracket M \rrbracket_s = [\nu s'. C]_s = [s', C|_{s+s'}] \in T[\sigma]_s.
$$

A definability result for the model $\text{Set}^I$ is also given in [12].

Lemma 1 (Definability). Suppose that $\sigma$ is a ground or first-order type of the nu-calculus and $s$ is some set of names. If $a \in [\sigma]_s$ and $e \in T[\sigma]_s$, then there are expressions $C \in \text{Can}_\sigma(s)$ and $M \in \text{Exp}_\sigma(s)$ such that $a = |C|_s$ and $e = \llbracket M \rrbracket_s$. \hfill $\square$

4 The Kripke Logical Relation

4.1 A Weak Logical Relation

We try to define a logical relation on the categorical model $\text{Set}^I$ so that it does not depend on any syntactical aspect of the nu-calculus. The general Kripke logical relation [5] gives a natural way to construct such a logical relation. A Kripke logical relation is based on a Kripke lambda models [5,6] which includes a partially-ordered set of possible worlds and have a set of elements of each type at each possible world. Informally, the relationship between elements of type $\sigma$ at different worlds is that every element of $\sigma$ at $w$ will continue to be an element of $\sigma$ at every possible $w' \geq w$.

Strictly speaking, a Kripke logical relation is indexed by elements of a poset. It has a relation for each type and a possible world instead of just having a relation for each type [5]. This can be naturally extended to be indexed by objects of a category as argued in [1]. The category $\text{Set}^I$ in the last section just
defines a Kripke model. We take the sets in \( \mathcal{I} \) as the possible worlds and set inclusion as the partial order between them.

In [7], a categorical definition of logical relations based on subsconing was proposed. This definition was extended in [3] in order to define a notion of logical relation for monadic types. According to this extended logical relation, we define a Kripke logical relation \( R_A^s \subseteq A_s \times A_s \) in the category \( \text{Set}^\mathcal{I} \) by induction over the structure of the corresponding type of \( A \) in a metalanguage for \( \text{Set}^\mathcal{I} \) [8,9,10] (here we do not distinguish on notation between an object of a category and the corresponding type in the metalanguage). Notably, we specify the types corresponding to the objects \( B \) and \( N \) as two constant types:

\[
\begin{align*}
&b_1 R_B^s b_2 \iff b_1 = b_2, \\
n_1 R_N^s n_2 \iff n_1 = n_2, \\
f_1 R_{GF}^s f_2 \iff \forall s', i : s \to s' \in \mathcal{I}, a_1, a_2 \in Fs' \quad (a_1 R_{F'}^s a_2 \implies f_1 s' (i, a_1) R_{G'}^s f_2 s' (i, a_2)), \\
[b_1, a_1] R_{TA}^s [b_2, a_2] \iff \exists s_0, i_1 : s_1 \to s_0, i_2 : s_2 \to s_0 \in \mathcal{I} \quad A(id_s + i_1) a_1 R_{A}^{s+s_0} A(id_s + i_2) a_2.
\end{align*}
\]

To define the above logical relation, we do not need any operational semantics for the nu-calculus such as Pitts and Stark define in [11,12]. And since it is defined on the pure categorical model \( \text{Set}^\mathcal{I} \), it can be easily generalized to study other typed lambda-calculi which \( \text{Set}^\mathcal{I} \) is suitable to model.

However, there is a small flaw in this logical relation. With the \( \text{span} \) in Pitts and Stark’s operational logical relation, we can distinguish between public and private names while we cannot in the above Kripke logical relation. In fact, the above logical relation is strictly weaker than the operational one. Consider the two expressions which we have already shown to be related by the operational logical relation:

\[ \nu n. \lambda x^n. (x = n) R_{\nu \to \emptyset}^{\exp} \lambda x^n. \text{false}. \]

By (1), their semantics are

\[
\begin{align*}
[[\nu n. \lambda x^n. (x = n)] s] &= \{n\}, [[\lambda x^n. (x = n)] s] \in T((TB)^N) s, \\
[[\lambda x^n. \text{false}] s] &= [\emptyset], [[\lambda x^n. \text{false}] s] \in T((TB)^N) s.
\end{align*}
\]

If these two terms can be related by the above Kripke logical relation, then by the definition of \( R_{TA}^s \) with \( A = (TB)^N \) we get

\[
(TB)^N(id_s + i_1) |\lambda x^n. (x = n)|_{s+s_0} R_{N \to TB}^{s+s_0} (TB)^N(id_s + i_2) |\lambda x^n. \text{false}|_{s+s_0}
\]

for some \( s_0 \in \mathcal{I} \) and \( i_1 : \{n\} \to s_0, i_2 : \emptyset \to s_0 \in \mathcal{I} \). Without losing the universality, we just let \( s_0 = \{n\}, i_1 = \text{id}_{\{n\}} \), and then get

\[
|\lambda x^n. (x = n)|_{s+\{n\}} R_{N \to TB}^{s+\{n\}} |\lambda x^n. \text{false}|_{s+\{n\}}.
\]

Now applying \( n \) to both functions, we get consequently \( tt R_B^{s+\{n\}} ff \), which is a contradiction to the definition of the relation for boolean type.
A direct way to represent this identification of public names is to provide the set of public names in the logical relation. We modify the structure of possible worlds by adding a name set \( w \) so as to make the pair \( \langle w, s \rangle \) instead of \( s \) as the possible world. Intuitively, the new set \( w \) stands for the set of public names of two nu-calculus expressions, so we require \( w \subseteq s \). The partial order on the modified possible worlds is given by

\[
\langle w_1, s_1 \rangle \leq \langle w_2, s_2 \rangle \iff w_1 \subseteq w_2 \& s_1 \subseteq s_2,
\]

and we change the relation for name type:

\[
n_1 \mathbb{R}^{\langle w, s \rangle}_N n_2 \iff n_1 = n_2 \& n_1, n_2 \in w.
\]

The intuition behind this relation is that two names are related if and only if they are equal and they are both public names.

### 4.2 Observation Category

Whereas in \([3]\) the chosen observation category was the categorical model \( \text{Set}^\mathcal{I} \) itself, here we refine the observation category in order to be able to derive a stronger logical relation.

Consider the comma category \( \mathcal{I} \to \), whose objects are tuples \( \langle w, i, s \rangle \) with \( i : w \to s \) in \( \mathcal{I} \), and whose morphisms are pairs \( \langle j, k \rangle : \langle w, i, s \rangle \to \langle w', i', s' \rangle \) where \( j : w \to w' \) in \( \mathcal{I} \) and \( k : s \to s' \) in \( \mathcal{I} \) such that the following diagram commutes:

\[
\begin{array}{ccc}
w & \xrightarrow{i} & s \\
\downarrow{j} & & \downarrow{k} \\
w' & \xrightarrow{i'} & s'
\end{array}
\]

We write \( i \) for \( \langle w, i, s \rangle \) when the domain \( w \) and the codomain \( s \) of \( i \) are clear from context.

Let \( \mathbb{C} \) be the category \( \text{Set}^{\mathcal{I} \to} \) of functors from \( \mathcal{I} \to \) to \( \text{Set} \) and natural transformations. We define a strong monad \(( T, \eta, \mu, t )\) on \( \mathbb{C} \) by:

- \( TA = \text{colim}_{i'} A(\cdot + i') : \mathcal{I} \to \text{Set} \).
  
  - \( TA_i \) is the set of equivalence classes of pairs \( (i', a) \), where \( i' : w' \to s' \) in \( \mathcal{I} \) and \( a \in A(i + i') \), modulo the smallest equivalence relation \( \equiv_C \) such that
    
    \( (i', a) \equiv_C (i'', A(\text{id}_i + \langle j, k \rangle)a) \) for each morphism \( \langle j, k \rangle : \langle w', i', s' \rangle \to \langle w'', i'', s'' \rangle \) in \( \mathcal{I} \to \). We write \([i, a]\) for the equivalence class of \((i, a)\).
  
  - On morphism \( \langle j, k \rangle, TA(j, k) \) maps the equivalence class of \((i', a)\) to the equivalence class of \((i', A(\langle j, k \rangle + \text{id}_{i'}a))\).

- For any \( f : A \to B \) in \( \mathbb{C} \), \( Tfi : TAi \to TBi \) is defined by \( Tfi[i', a] = [i', f(i' + i')a] \). This is compatible with \( \equiv_C \) because \( f \) is natural.

- \( \eta_{Ai} : Ai \to TAi \) is defined by \( \eta_{Ai}a = [\emptyset, a] \).

- \( \mu_{Ai} : T^2Ai \to TAi \) is defined by \( \mu_{Ai}[i', [i'', a]] = [i' + i'', a] \)

- \( t_{A,B} : Ai \times TBi \to T(A \times B)i \) is defined by \( t_{A,B}(i, [i', b]) = [i', (Ai_i,v,a,b)] \)

where \( i_i : i \to i + i' \) is the canonical injection.
4.3 Natural Derivation of the Logical Relation

Let \( U : \mathcal{I} \to \mathcal{I} \) be the forgetful functor which maps an object \( \langle w, i, s \rangle \) to \( s \) and a morphism \( \langle j, k \rangle \) to \( k \). Let \( |\cdot| : \mathcal{C} \to \mathcal{C} \) be the functor \( \text{id}_{\text{Set}^\mathcal{I}} \) (where \( \mathcal{C} = \text{Set}^\mathcal{I} \) and \( \mathcal{C} = \text{Set}^{\mathcal{I}^\to} \)). On an object \( A, |A| \) is equal to \( A \circ U \). On a morphism \( f \) (which is a natural transformation), for each \( \langle w, i, s \rangle \) in \( \mathcal{I}^\to \), the component \( |f|_{\langle w, i, s \rangle} \) is equal to \( (f \circ U)_{\langle w, i, s \rangle} \), that is to say \( f_s \). Let \( \sigma : T|_\cdot | \to |T_\cdot | \) be the monad morphism defined by

\[
\sigma_A i[\langle w', i', s' \rangle, a] = [s', a]
\]

This is well defined as \( |A|(i + i') = A(s + s') \). We can then define \( \sigma_{\langle A_1, A_2 \rangle} : TA_1 \times A_2 \to |TA_1| \times |TA_2| \) by

\[
\sigma_{\langle A_1, A_2 \rangle} i = (\sigma_{A_1} i \circ T\pi_1, \sigma_{A_2} i \circ T\pi_2)
\]

That is to say

\[
\sigma_{\langle A_1, A_2 \rangle} i[\langle w', i', s' \rangle, (a_1, a_2)] = ([s', a_1], [s', a_2])
\]

\( \mathcal{C} \) has a mono factorization system consisting of point-wise surjections and point-wise injections. And it is clear that \( T \) and finite product preserve point-wise surjections. The conditions required in \(^2\) are therefore satisfied. We thus define the monadic type component of our Kripke logical relation. Each injection \( f : S\langle w, i, s \rangle \leftrightarrow |A_1 \times A_2| \langle w, i, s \rangle = A_1 s \times A_2 s \) is a representation of a relation \( S\langle w, i, s \rangle \subseteq A_1 s \times A_2 s \) up to isomorphism. The logical relation, denoted \( \tilde{S} \), is given by mono factorization of the composition of \( Tf \) with \( \sigma_{\langle A_1, A_2 \rangle} \):

\[
\begin{array}{c}
TS \xrightarrow{Tf} TA_1 \times A_2 \\
\Downarrow \sigma_{\langle A_1, A_2 \rangle} \\
\tilde{S} \subseteq |TA_1| \times |TA_2|
\end{array}
\]

Explicitly, \( \tilde{S} i \leftrightarrow |TA_1| i \times |TA_2| i \) is given by:

\[
[i_1, a_1] \quad \tilde{S} i \quad [i_2, a_2] \quad \iff \exists i_0 \in \mathcal{I}^\to \ \exists \langle j_1, k_1 \rangle : i_1 \to i_0 \in \mathcal{I}^\to \ \exists \langle j_2, k_2 \rangle : i_2 \to i_0 \in \mathcal{I}^\to \\
|A_1|(id_i + \langle j_1, k_1 \rangle) \quad S(i + i_0) \quad |A_2|(id_i + \langle j_2, k_2 \rangle)
\]

It can be simplified in

\[
[i_1, a_1] \quad \tilde{S} i \quad [i_2, a_2] \quad \iff \exists i_0 \in \mathcal{I}^\to \ \exists \langle j_1, k_1 \rangle : i_1 \to i_0 \in \mathcal{I}^\to \ \exists \langle j_2, k_2 \rangle : i_2 \to i_0 \in \mathcal{I}^\to \\
A_1(id_s + k_1) \quad S(i + i_0) \quad A_2(id_s + k_2)
\]

where \( s \) is the codomain of \( i \).

\(^2\) More precisely, the pair \( (|\cdot|, \sigma) \) is the monad morphism from \( T \) to \( T \) where \( \sigma \) is a natural transformation. \( \sigma \) was named ‘distributivity law’ in \(^3\). It turned out to be a bad choice as it could be confused with ‘distributive law’ which denotes a close but different concept by Beck \(^2\). Also, while ‘distributive laws’ tend to be rare, monad morphisms abound.
Lemma 2. The full subcategory $\mathcal{I}^\subseteq$ of $\mathcal{I}^\rightarrow$ consisting only of inclusions is equivalent to the whole category $\mathcal{I}^\rightarrow$.

Proof. Let $F : \mathcal{I}^\subseteq \to \mathcal{I}^\rightarrow$ be the inclusion functor. Let $G : \mathcal{I}^\rightarrow \to \mathcal{I}^\subseteq$ be the functor which maps $\langle w, i, s \rangle$ to $\langle i(w), s \rangle$, and $\langle j, k \rangle : \langle w, i, s \rangle \to \langle w', i', s' \rangle$ to $i' \circ j \circ i^{-1} : \langle i(w'), s' \rangle \to \langle i'(w'), s' \rangle$. Then clearly, $G \circ F$ is identity, and $F \circ G$ maps $\langle w, i, s \rangle$ to $\langle i(w), s \rangle$ which are isomorphic through $\langle i, id_s \rangle$ and $\langle i^{-1} \mid i(w), id_s \rangle$. So $(F,G)$ is an equivalence of categories.

Thus the definition of $\hat{S}i$ is equivalent to

$$[(w_1,s_1),a_1] \quad \hat{S}(w,s) \quad [(w_2,s_2),a_2] \quad \iff \exists s_0 \in \mathcal{I} \cdot \exists w_0 \subseteq s_0 \cdot \exists k_1 : s_1 \to s_0 \in \mathcal{I} \cdot \exists k_2 : s_2 \to s_0 \in \mathcal{I}.$$  

$A_1(id_s + k_1) \quad S(w + w_0 \subseteq s + s_0) \quad A_2(id_s + k_2).$

Note that a Kripke logical relation indexed by elements of a poset can be naturally extended to be indexed by objects of a category (in this case $\mathcal{I}^\rightarrow$) as argued in [1]. We therefore arrive at:

Definition 3 (Kripke logical relation). The relations $\mathbb{R}_A^{(w,s)} \subseteq As \times As$, where $w \subseteq s$ and $A \in \text{Set}^\mathcal{I}$, are defined by induction over the structure of the corresponding type in the metalanguage, according to:

$$b_1 \mathbb{R}_B^{(w,s)} b_2 \iff b_1 = b_2$$

$$n_1 \mathbb{R}_N^{(w,s)} n_2 \iff n_1 = n_2 \land n_1, n_2 \in w$$

$$f_1 \mathbb{R}_F^{(w,s)} f_2 \iff \forall s', \forall k : s \to s' \in \mathcal{I} \cdot \forall w' \mid k(w) \subseteq w' \subseteq s' \cdot \forall a_1, a_2 \in Fs', \quad (a_1 \mathbb{R}_F^{(w',s')} a_2 \iff f_1 s'(k, a_1) \mathbb{R}_G^{(w',s')} f_2 s'(k, a_2)).$$

$$[s_1,a_1] \mathbb{R}_T^{(w,s)} [s_2,a_2] \iff \exists s_0, w_0 \subseteq s_0, i_1 : s_1 \to s_0, i_2 : s_2 \to s_0 \in \mathcal{I}, \quad A(id_s + i_1)a_1 \mathbb{R}_A^{(w+w_0,s+s_0)} A(id_s + i_2)a_2. \quad \Box$$

Lemma 3. For all canonical forms $C_1, C_2 \in \text{Can}_\sigma(s)$,

$$[C_1]_s \mathbb{R}_{\mathcal{T}[\sigma]}^{(w,s)} [C_2]_s \iff [C_1]_s \mathbb{R}_{\mathcal{T}[\sigma]}^{(w,s)} [C_2]_s$$

Proof. The interpretation of nu-calculus canonical forms into category $\text{Set}^\mathcal{I}$ provides us $[C_k]_s = [\emptyset, [C_k]_s]$ $(k = 1, 2)$. According to the definition of the Kripke logical relation,

$$[\emptyset, [C_k]_s] \mathbb{R}_{\mathcal{T}[\sigma]}^{(w,s)} [\emptyset, [C_2]_s] \iff [\sigma](id_s + i_1) [C_k]_s \mathbb{R}_{\mathcal{T}[\sigma]}^{(w+w_0,s+s_0)} [\sigma](id_s + i_2) [C_2]_s$$

for some $s_0, w_0 \subseteq s_0$ and $i_1, i_2 : \emptyset \to s_0 \in \mathcal{I}$. Since whatever $s_0$ and $w_0$ are, $i_1$ and $i_2$ are always the empty injection, we simply let $s_0, w_0$ be the empty set and $i_1, i_2$ be $id_\emptyset$. Then we get $[\sigma](id_s + i_k) [C_k]_s = [C_k]_s$ $(k = 1, 2)$, and consequently,

$$[C_1]_s \mathbb{R}_{\mathcal{T}[\sigma]}^{(w,s)} [C_2]_s \iff [C_1]_s \mathbb{R}_{\mathcal{T}[\sigma]}^{(w,s)} [C_2]_s. \quad \Box$$
4.4 Equivalence between Operational and Kripke Logical Relations

Through this Kripke logical relation, we can relate the two nu-calculus expressions in Section 4.1 again. To prove that, for $R_{TA}^{(w,s)}$ with $A = (TB)^N$, we let $w_0 = \emptyset$ and by the definition of $R_{TA}^{(w,s)}$, we need to prove

$$\lambda x^\nu.(x = n)|_{s + \{n\}} \rangle_{(TB)^N} \lambda x^\nu.\text{false}|_{s + \{n\}}.$$

By the definition of $R_N^{(w,s + \{n\})}$, related names must be in set $w$ and are definitely different from $n$, so applying any related name to the function $\lambda x^\nu.(x = n)|_{s + \{n\}}$ must return the value $tt$ (by (1)), which is same as the result of applying it to the second function. Then, by the definition of $R_N^{(w,s + \{n\})}$ and $R_B^{(w,s + \{n\})}$, the above two functions can be related.

**Theorem 1.** If $R : s_1 \Rightarrow s_2$ is a span and $s_1 \vdash M_1 : \sigma$, $s_2 \vdash M_2 : \sigma$ where $\sigma$ is a ground type or first-order type, then

$$M_1 \ R_\sigma \ M_2 \iff \exists s, w \subseteq s, i_1 : s_1 \rightarrow s, i_2 : s_2 \rightarrow s \in \mathcal{I}, \quad T[\sigma]i_1([M_1]_{s_1}) \ R_{TA}^{(w,s)} T[\sigma]i_2([M_2]_{s_2}).$$

**Proof.** If $M_1$ and $M_2$ are two expressions in canonical form, by the definition of the category $\text{Set}^\mathbb{Z}$ in Section 3.1,

$$T[\sigma]i_k[M_1]_{s_k} = T[\sigma]i_k[\emptyset, [M_1]_{s_k}] = [\emptyset, [\sigma]i_k[M_1]_{s_k}] \quad (k = 1, 2).$$

Then by Lemma 3, the theorem is equivalent to

$$M_1 \ R_{\sigma}^{\text{can}} \ M_2 \iff [\sigma]i_1([M_1]_{s_1}) \ R_{TA}^{(w,s)} [\sigma]i_2([M_2]_{s_2})$$

for some $w \subseteq s$. So we can make the proof by induction on the structure of types in the metalanguage.

- The case for boolean type is obvious.
- For name type, we should define a proper $w$ and the injections $i_1, i_2$ to represent the span $R$ in the operational logical relation:

$$\forall n_1 \in s_1, n_2 \in s_2 \ (i_1(n_1) = i_2(n_2) \iff n_1 \ R n_2)$$

and $w$ is the image of the domain of $R$ under $i_1$, equivalently the image of the codomain of $R$ under $i_2$. Because $Ni_k = i_k$, we have

$$n_1 \ R_{\nu}^{\text{can}} \ n_2 \iff Ni_1(n_1) = Ni_2(n_2) \iff Ni_1(n_1) \ R_N^{(w,s)} Ni_2(n_2).$$

- For first-order function types in the metalanguage ($\sigma \equiv \tau \rightarrow \tau'$), the only possibility is the nu-calculus canonical expressions of the form $\lambda x^\tau^\nu. M$. Suppose the type of $M$ is $\tau'$, then the canonical interpretation $[\lambda x^\tau^\nu. M]$ is of type $[\tau] \rightarrow T[\tau']$. The definability result of the model $\text{Set}^\mathbb{Z}$ (Lemma 1) ensure that for each $f \in (T[\tau'])^{[\tau']^1}s$, we can find a canonical function expression
\[ M \to \mathcal{V}\] where \( \mathcal{V} \) is type of \( \tau '\), such that |\( \mathcal{V} \cdot M'\)| = \( f \). Then by the induction hypothesis and the definitions of \( R^\text{can}_{\tau \to \tau'} \) and \( \mathcal{R}^{(w,s)}_{(T[\tau'])[\tau]} \), we get

\[
\lambda x^{\tau}.M_1^\text{can}_{\tau \to \tau'} \lambda x^{\tau}.M_2^\text{can}_{\tau \to \tau'} \iff (T[\tau'])^{[\tau]}i_1(\lambda x^{\tau}.M_1'|s) \mathcal{R}^{(w,s)}_{(T[\tau'])[\tau]} (T[\tau'])^{[\tau]}i_2(\lambda x^{\tau}.M_2'|s)
\]

for some \( w \subseteq s \) and \( i_1 : s_1 \to s, i_2 : s_2 \to s \in \mathcal{I} \).

- The case for the monadic types in the metalanguage corresponds to \( R^\text{ex}_{\sigma} \) in the operational logical relation. If \( s_1 \vdash M_1 \downarrow^s_\sigma C_1 \) and \( s_2 \vdash M_2 \downarrow^s_\sigma C_2 \), then by the definition of \( R^\text{ex}_{\sigma} \) we can find some \( R' : s'_1 \vdash s'_2 \) such that \( C_1 (R \oplus R')^\text{can}_{\sigma} C_2 \), and consequently by the induction hypothesis, we get

\[
[s][\tau]j_1(\lambda x_{s_1}^{s'_1}|s) \mathcal{R}^{(w',s')}_{T[\tau]}(s)[\tau]j_2(\lambda x_{s_2}^{s_2}|s_2)
\]

for some \( w' \subseteq s' \) and \( j_1 : s_1 + s'_1 \to s', j_2 : s_2 + s'_2 \to s' \in \mathcal{I} \). On the other hand, by (1) and the definition of \( \mathcal{Set}^{\mathcal{I}} \), we have

\[
T[s][\tau]i_k([M_k]_{s_k}) = [s_k'][\sigma](i_k + \text{id}_{s_k'})|C_k|_{s_k + s_k'} \quad (k = 1, 2),
\]

and according to the definition of \( \mathcal{R}^{(w,s)}_{T[\sigma]} \),

\[
T[s][\tau]i_1([M_1]_{s_1}) \mathcal{R}^{(w,s)}_{T[\sigma]} T[s][\tau]i_2([M_2]_{s_2}) \iff \exists s_0, w_0 \subseteq s_0, i'_1 : s'_1 \to s_0, i'_2 : s'_2 \to s_0 \in \mathcal{I},
\]

\[
[s][\tau](i_1 + i'_1)|C_1|_{s_1 + s'_1} \mathcal{R}^{(w + w_0, s + s_0)}_{T[\sigma]} [s][\tau](i_2 + i'_2)|C_2|_{s_2 + s'_2}.
\]

Let \( i_k, i'_k \) \((k = 1, 2)\) be defined as

\[
\forall n \in s_k, i_k(n) = j_k(n) \quad \text{and} \quad \forall n' \in s'_k, i'_k(n') = j_k(n'),
\]

and let \( s + s_0 = s' \), \( w + w_0 = w' \), then we get

\[
M_1 \mathcal{R}^{\text{ex}}_{\sigma} M_2 \iff T[s][\tau]i_1([M_1]_{s_1}) \mathcal{R}^{(w,s)}_{T[\sigma]} T[s][\tau]i_2([M_2]_{s_2}). \quad \square
\]

We only claim the equivalence between these two logical relations for ground types and first-order function types. The failure of the equivalence for high-order function types is due to the non-fully abstract categorical model, while a similar Kripke logical relation defined directly on the computational metalanguage has been shown to be equivalent to the operational logical relation for any type \( \mathcal{I} \).

## 5 Conclusion

We construct a Kripke logical relation for Pitts and Stark’s nu-calculus in this paper and we show that it is equivalent to their operational logical relation up to first-order types. Because our Kripke logical relation is defined on a categorical model, it does not rely on any operational aspect of the nu-calculus, hence it is easier to be generalized. Our future work is to use a similar method to derive logical relations for the cryptographic lambda calculus, which is syntactically an extension of the nu-calculus (Sumii and Pierce define a logical relation for this language in \[15\], but it is still very operational).
Acknowledgements

We are grateful to Jean Goubault-Larrecq, John Power and Ian Stark for useful discussions and to anonymous reviewers for their comments.

References

“Computational mathematics” (algorithmic mathematics) is the part of mathematics that strives at the solution of mathematical problems by algorithms. In a superficial view, some people might believe that computational mathematics is the easy part of mathematics in which trivial mathematics is made useful by repeating trivial steps sufficiently many times on current powerful machines in order to achieve marginally interesting results. The opposite is true: Many times, computational mathematics needs and stimulates deeper mathematics, i.e. deeper mathematical theorems with more difficult proofs than “pure” mathematics. This is so because, in order to establish an algorithmic method for a given mathematical problem, i.e. in order to reduce the solution of a given problem to the few operations that can be executed on machines, deeper insight on the given problem domain is necessary than the insight necessary for establishing the reduction of the given problem to powerful nonalgorithmic abstract mathematical operations as, for example, choice functions and the basic quantifiers of set theory.

Computational mathematics comes in two flavors: “numerical mathematics”, in which the original problems are replaced by approximate versions and one is satisfied with approximate solutions to the approximate problems, and “exact algorithmic mathematics” in which the original problems are solved by algorithms in the original domains or isomorphic representations of these domains. Exact algorithmic mathematics can be divided into “discrete mathematics”, in which the objects in the underlying mathematical domains are finitary, and “computer algebra”, in which the objects in the underlying mathematical domains according to their original definition are infinite and the possibility of an isomorphic finitary representation in itself is a non-trivial mathematical question.

For many mathematical problems it can be mathematically proved that exact algorithmic solutions are not possible or are possible only by algorithms with a certain complexity. Even in these cases, algorithmic mathematics can and should go on by considering either approximate versions or special cases of the problem.

Mathematical logic is the mathematical meta-theory of mathematics. The characteristic feature of mathematics is its method of gaining knowledge from given knowledge by reasoning. Hence, the meta-theory of mathematics is essentially the theory of reasoning. As any other mathematical theory, one can and should ask the question of how much of reasoning can be made algorithmic. The
part of mathematical logic that deals with algorithmic methods for reasoning is called “computational logic”. Although it is well known that the algorithmization of mathematical reasoning in its most general form in a certain sense is not possible, the algorithmization of reasoning under certain restrictions or for certain limited - but still extremely broad - areas of mathematics is possible and, in fact, is one of the most challenging mathematical endeavors with enormous practical significance.

In fact, as a result of analyzing mathematical invention in the various areas of mathematics, it turns out that the transition from the object level to the meta-level is not limited to mathematical logic but is one of the main - but mostly hidden - instruments of mathematical progress in every field of mathematics and at the core of mathematical intelligence and invention. We therefore advocate that future mathematical systems must provide a frame for considering both the object and the meta-level of mathematical theories and must provide a means for the transition from the object level to the meta-level. In fact, “symbolic computation” is a term that more and more is used as a common term for both computer algebra flavored computational mathematics on the object level and computational logic on the meta-level. In other words, “symbolic computation” grows into the most general frame for all aspects of algorithmization. More concretely, in the recent research efforts of the symbolic computation community, the interaction of computer algebra and computational logic and the applications of the results of this interaction for the future automation of “mathematical knowledge management” moves into the center of interest.

In the talk, we will illustrate the above general outline by examples of symbolic computation algorithms and its underlying theories and by some demos in the Theorema software system.
Simple Stochastic Parity Games

Krishnendu Chatterjee, Marcin Jurdziński, and Thomas A. Henzinger

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, USA

Abstract. Many verification, planning, and control problems can be modeled as games played on state-transition graphs by one or two players whose conflicting goals are to form a path in the graph. The focus here is on simple stochastic parity games, that is, two-player games with turn-based probabilistic transitions and \( \omega \)-regular objectives formalized as parity (Rabin chain) winning conditions. An efficient translation from simple stochastic parity games to nonstochastic parity games is given. As many algorithms are known for solving the latter, the translation yields efficient algorithms for computing the states of a simple stochastic parity game from which a player can win with probability 1.

An important special case of simple stochastic parity games are the Markov decision processes with Büchi objectives. For this special case a first provably subquadratic algorithm is given for computing the states from which the single player has a strategy to achieve a Büchi objective with probability 1. For game graphs with \( m \) edges the algorithm works in time \( O(m\sqrt{m}) \). Interestingly, a similar technique sheds light on the question of the computational complexity of solving simple Büchi games and yields the first provably subquadratic algorithm, with a running time of \( O(n^2/\log n) \) for game graphs with \( n \) vertices and \( O(n) \) edges.

1 Introduction

Many verification, AI planning, and control problems can be formalized as state-transition graphs, and solved by finding paths in these graphs that meet certain criteria. Uncertainty about a process evolution is often modeled by probabilistic transitions, and then instead of searching for paths we are interested in measuring the probability that a path satisfies a given criterion, or finding controllers that maximize this probability. For decades there have been several separated communities studying such problems in the context of stochastic games [13], Markov decision processes (MDP’s) [9], AI planning, and model checking. Only recently some unification has been attempted. MDP’s can be naturally viewed as 1-player stochastic games and the book of Filar and Vrieze [8] provides a unified rigorous treatment of the theories of MDP’s and stochastic games. They coin the term Competitive MDP’s to encompass both 1- and 2-player stochastic games.

* This research was supported in part by the DARPA grant F33615-C-98-3614, the ONR grant N00014-02-1-0671, the NSF grants CCR-9988172 and CCR-0225610, and the Polish KBN grant 7-T11C-027-20.
games. We also suggest to cast various games based on state-transition models into a unified framework. For this purpose we use the following parameters.

- Number of players: \(1/2\): Markov chains; 1: nondeterministic state-transition systems; \(1/2\): MDP’s; 2: game graphs; \(1/2\): stochastic games.
- The players’ knowledge about the course of the game: simple (or turn-based) games: the state determines who plays next; concurrent games: the players choose moves simultaneously and independently, without knowing each other’s choices \([13,1,5]\).
- Winning objectives: qualitative (\(\omega\)-regular) objectives \([14]\): finite objectives (reachability and safety), or infinite objectives (liveness, such as Büchi or general parity conditions), quantitative (reward) objectives \([8]\): discounted reward, or limiting average reward, or total reward.
- Winning criteria: qualitative criteria \([5]\): sure winning, almost-sure winning (with probability 1), or limit-sure winning (with probability arbitrarily close to 1), quantitative criteria: exact probability of winning, or expected reward.

We mention a few notable examples of models and problems studied in various communities that fit into the above categorization.

- Summable MDP’s \([9,8]\): simple \(1/2\)-player games with maximum expected discounted reward.
- Mean-payoff games \([17]\): simple 2-player games with maximum limiting average reward.
- Parity games \([12,14]\): simple 2-player games with parity objectives.
- Quantitative simple stochastic games \([2]\): simple \(2\)-player games with reachability objectives and exact probability of winning.
- Qualitative concurrent \(\omega\)-regular games \([4]\): concurrent \(1/2\)-player games with parity objectives and various qualitative winning criteria.
- Quantitative concurrent \(\omega\)-regular games \([6]\): concurrent \(1/2\)-player games with parity objectives and exact probability of winning.

In earlier work \([11]\) we studied the complexity of algorithms for solving concurrent parity games. In particular, we have given efficient reductions from the problem of solving concurrent Büchi and co-Büchi games (under the almost-sure winning criterion) to the extensively studied problem of solving simple (nonconcurrent) parity games \([15,10,16]\). In this paper we focus on the following three types of games:

- **Qualitative simple stochastic parity games**: simple \(2\)-player games with parity objectives and almost-sure winning criterion.
- **Qualitative Büchi MDP’s**: simple \(1\)-player games with Büchi winning objectives and almost-sure winning criterion.
- **Simple Büchi games**: simple 2-player games with Büchi winning objectives.

We use \(n\) to denote the number of vertices and \(m\) to denote the number of edges of a game graph. Our main results can be summarized as follows.
Theorem 1. Every qualitative simple stochastic parity game with priorities in the set \(\{0, 1, 2, \ldots, d-1\}\) can be translated to a simple parity game with the same set of priorities, with \(O(d \cdot n)\) vertices and \(O(d \cdot (m+n))\) edges, and hence it can be solved in time \(O(d \cdot (m+n) \cdot (nd)^{[d/2]})\).

Theorem 2 (Pure memoryless determinacy). From every vertex of a simple stochastic parity game either one player has a pure memoryless strategy to win with probability 1, or there is a \(\delta > 0\) such that the other player has a pure memoryless strategy to win with probability at least \(\delta\). Hence the almost-sure and limit-sure winning criteria coincide.

Theorem 3. Qualitative Büchi MDPs can be solved in time \(O(m\sqrt{m})\).

This implies also a complexity improvement for solving MDP’s with reachability objectives under the almost-sure winning criterion, for which the best algorithm so far had \(O(mn)\) running time [3]. Interestingly, the novel technique we use for Büchi MDP’s allows us to shed some light on the important problem of finding subquadratic algorithms for simple Büchi games.

Theorem 4. Simple Büchi games with \(O(n)\) edges can be solved in time \(O(n^2 / \log n)\).

This result and reductions in [11] prove that concurrent games with a constant number of actions and with reachability and Büchi objectives under the almost-sure and limit-sure winning criteria can also be solved in subquadratic time (the best algorithms so far had \(O(n^2)\) running time [4]).

2 Simple Stochastic Parity Games

Given \(n \in \mathbb{N}\), we write \([n]\) for the set \(\{0, 1, 2, \ldots, n\}\) and \([n]_+\) for the set \(\{1, 2, \ldots, n\}\). A 2\(\frac{1}{2}\)-player game (or simple stochastic game, or SSG) \(G = (V, E, (V_2, V_3, V_\lozenge))\) consists of a directed graph \((V, E)\) and a partition \((V_2, V_3, V_\lozenge)\) of the vertex set \(V\). For technical convenience we assume that every vertex has at least one outgoing edge. For simplicity we only consider the case when \(G\) is binary. An infinite path in \(G\) is a infinite sequence \(\langle v_0, v_1, v_2, \ldots \rangle\) of vertices such that \((v_k, v_{k+1}) \in E\) for all \(k \in \mathbb{N}\). We write \(\Omega\) for the set of all infinite paths. The game is played with three players that move a token from vertex to vertex so that an infinite path is formed: from vertices in \(V_2\), player Even (\(\Box\)) moves the token along an outgoing edge to a successor vertex; from vertices in \(V_3\), player Random (\(\lozenge\)) moves the token. If there are two outgoing edges, then player Random always moves the token to one of the two successor vertices with probability \(1/2\). Since player Random does not have a proper choice of moves, as the other two players do, we use the \(\frac{1}{2}\)-player terminology for player Random. The 2-player games are the special case of the 2\(\frac{1}{2}\)-player games with \(V_\lozenge = \emptyset\). The 1\(\frac{1}{2}\)-player games (or MDP’s) are the special case of the 2\(\frac{1}{2}\)-player games with \(V_\lozenge = \emptyset\). In other
words, in 2-player games, the only players are Even and Odd; and in $1^{1/2}$-player games, the only players are Even and Random.

**Strategies.** For a finite set $A$, a probability distribution on $A$ is a function $f : A \rightarrow [0, 1]$ such that $\sum_{a \in A} f(a) = 1$. We denote the set of probability distributions on $A$ by $\mathcal{D}(A)$. A \textit{(mixed) strategy} for player Even is a function $\sigma : V^* \cdot V_\square \rightarrow \mathcal{D}(V)$ such that for every finite and nonempty sequence $\bar{v} \in V^* \cdot V_\square$ of vertices, which represents the history of the play so far, $\sigma(\bar{v})$ is the next move to be chosen by player Even. A strategy must prescribe only available moves, i.e., if $(v, u) \not\in E$, then $\sigma(\bar{v} \cdot v)(u) = 0$. The strategy $\sigma$ is pure if for all $\bar{w} \in V^*$ and $v \in V_\square$, there is a vertex $u$ such that $\sigma(\bar{w} \cdot v)(u) = 1$. The strategies for player Odd are defined analogously. We write $\Sigma$ and $\Pi$ for the sets of all strategies for players Even and Odd, respectively. A \textit{memoryless strategy} is a strategy which does not depend on the history of the play but only on the current vertex. A pure memoryless strategy for player Even can be represented as a function $\sigma : V_\square \rightarrow V$ such that $(v, \sigma(v)) \in E$ for all $v \in V_\square$.

For an initial vertex $v$, and two strategies $\sigma \in \Sigma$ and $\pi \in \Pi$ for players Even and Odd, respectively, we define $\text{Outcome}(v, \sigma, \pi) \subseteq \Omega$ to be the set of paths that can be followed when a play starts from vertex $v$ and the players use the strategies $\sigma$ and $\pi$. Formally, $\langle v_0, v_1, v_2, \ldots \rangle \in \text{Outcome}(v, \sigma, \pi)$ if $v_0 = v$, and for all $k \geq 0$, we have that $v_k \in V_\bigcirc$ implies $(v_k, v_{k+1}) \in E$, $v_k \in V_\square$ implies $\sigma(v_0, v_1, \ldots, v_k)(v_{k+1}) > 0$, and $v_k \in V_\bigcirc$ implies $\pi(v_0, v_1, \ldots, v_k)(v_{k+1}) > 0$. Once a starting vertex $v$ and strategies $\sigma \in \Sigma$ and $\pi \in \Pi$ for the two players have been chosen, the probabilities of events are uniquely defined, where an \textit{event} $\mathcal{A} \subseteq \Omega$ is a measurable set of paths. For a vertex $v$ and an event $\mathcal{A} \subseteq \Omega$, we write $\Pr^v_{\sigma, \pi}[\mathcal{A}]$ for the probability that a path belongs to $\mathcal{A}$ if the game starts from $v$ and the players use the strategies $\sigma$ and $\pi$.

**Winning objectives.** A \textit{winning objective} for a SSG $G$ is a set $W \subseteq \Omega$ of infinite paths. We consider the following winning objectives.

- \textit{Büchi objective.} For a set $T \subseteq V$ of target vertices, the Büchi objective is defined as $\text{Büchi}(T) = \{(v_0, v_1, v_2, \ldots) \in \Omega : v_k \in T \text{ for infinitely many } k \geq 0\}$.
- \textit{Parity objective.} Let $p : V \rightarrow [d]$ be a function that assigns a \textit{priority} $p(v)$ to every vertex $v \in V$, where $d \in \mathbb{N}$. For an infinite path $\bar{v} = \langle v_0, v_1, \ldots \rangle \in \Omega$, we define $\text{Inf}(\bar{v}) = \{ i \in [d] : p(v_i) = i \text{ for infinitely many } k \geq 0 \}$. The \textit{Even parity objective} is defined as $\text{Parity}(p) = \{ \bar{v} \in \Omega : \min(\text{Inf}(\bar{v})) \text{ is even} \}$, and the \textit{Odd parity objective} as $\text{co-Parity}(p) = \{ \bar{v} \in \Omega : \min(\text{Inf}(\bar{v})) \text{ is odd} \}$.

Note that for a priority function $p : V \rightarrow [1]$ with only two priorities (0 and 1), an even parity objective $\text{Parity}(p)$ is equivalent to the Büchi objective $\text{Büchi}(p^{-1}(0))$, i.e., the target set consists of the vertices with priority 0. A $2^{1/2}$-player \textit{parity game} (or parity SSG) is a pair $(G, p)$, where $G$ is a $2^{1/2}$-player game and $p$ is a priority function. If $G$ is a 2-player (resp. $1^{1/2}$-player) game, then $(G, p)$ is a 2-player parity game (resp. $1^{1/2}$-player parity game, or parity MDP). A 2-player Büchi game is a pair $(G, T)$, where $G$ is a 2-player game and $T$ is a set of target vertices. If $G$ is a $1^{1/2}$-player game, then $(G, T)$ is a $1^{1/2}$-player Büchi game (or Büchi MDP).
Winning criteria. Consider an SSG $G$ with winning objective $W$. We say that a strategy $\sigma \in \Sigma$ for player Even is

- **sure winning** from vertex $v$ if for all strategies $\pi \in \Pi$ of player Odd, we have $\text{Outcome}(v, \sigma, \pi) \subseteq W$;
- **almost-sure winning** from vertex $v$ if for all strategies $\pi \in \Pi$ of player Odd, we have $\Pr_{v,\sigma,\pi}[W] = 1$;
- **positive-probability winning** from vertex $v$ if there is a $\delta > 0$, such that for all strategies $\pi \in \Pi$ of player Odd, we have $\Pr_{v,\sigma,\pi}[W] \geq \delta$.

The definitions for player Odd are similar. We shall see that player Even has an almost-sure winning strategy for $W$ from $v$ if and only if player Odd does not have a positive-probability winning strategy for $\Omega \setminus W$ from $v$. For 2-player games all three of the above winning criteria coincide, i.e., the existence of a positive-probability winning strategy for a player implies the existence of a sure winning strategy. In this paper we consider the dual criteria of almost-sure winning (i.e., winning with probability 1) and positive-probability winning, for $2^{1/2}$- and $1^{1/2}$-player games, and the criterion of sure winning for 2-player games:

- The problem of solving a $2^{1/2}$-player (resp. $1^{1/2}$-player) parity game $(G, p)$ is to compute the set of vertices of $G$ from which the player Even has an almost-sure winning strategy for the objective $W = \text{Parity}(p)$.
- The problem of solving a 2-player parity game $(G, p)$ is to compute the set of vertices of $G$ from which the player Even has a sure winning strategy for the objective $W = \text{Parity}(p)$.

3 Solving $2^{1/2}$-Player Parity Games

The main result of this section is an algorithm for solving $2^{1/2}$-player parity games, which is obtained by an efficient reduction to 2-player parity games, i.e., a proof of Theorem 1. As in our earlier work [11], the key technical tool for the correctness proof of the reduction is the notion of ranking functions, which witness the existence of winning strategies for the players. Our ranking functions are closely related to the semantics of the $\mu$-calculus formulas that express the winning sets of concurrent stochastic parity games [4], but due to the lack of concurrency in our games, the defining conditions for our ranking functions are considerably simpler. Two corollaries of our proof are of independent interest. First, we establish the existence of pure memoryless winning strategies for both players in simple stochastic parity games (Theorem 2). This is in contrast to concurrent games, where players need mixed strategies with infinite memory [4]. Second, in simple stochastic parity games the almost-sure and limit-sure winning criteria coincide (Theorem 2), which is not the case for concurrent games [4].

3.1 Ranking Functions

In this subsection we provide a characterization of the “universal” parity MDP problem. We define certain sufficient conditions for establishing that for all
strategies \( \pi \), the Markov chain \( M_\pi \) satisfies the parity condition with probability 1, or with probability at least \( \delta > 0 \). These sufficient conditions are then used in the next subsection to prove correctness of our solution for \( 2^{1/2} \)-player parity games: a strategy \( \sigma \) is winning for a player if and only if the parity MDP \( G_\sigma \) is a solution to the universal parity MDP problem.

Consider a parity MDP \( M = (V,E,(V_\text{O}, V_\text{O}), p : V \to [d]) \). Without loss of generality assume that \( d \) is even. A ranking function for player Even labels vertices with \((d/2)\)-tuples of natural numbers: \( \varphi = (\varphi^1, \varphi^3, \ldots, \varphi^{d-1}) : V \to [n]^{d/2} \cup \{\infty\}, \) for some \( n \in \mathbb{N} \). For succinctness, for all odd \( k \in [d] \) we write \( \overrightarrow{\varphi}^k(v) \) to denote the tuple \((\varphi^1(v), \varphi^3(v), \ldots, \varphi^{k}(v)) \). We often call \( \varphi(v) \) the rank of vertex \( v \), and we call \( \overrightarrow{\varphi}^k(v) \) the \( k \)-th rank of vertex \( v \). A ranking function for player Odd is a function \( \psi = (\psi^0, \psi^2, \ldots, \psi^d) : V \to [n]^{d/2+1} \cup \{\infty\}; \) we use similar notational conventions as with ranking functions for player Even. For all \( v \in V_\text{O} \), we write \( \Pr_v[\overrightarrow{\varphi}^k_\leq] \) (resp. \( \Pr_v[\overrightarrow{\varphi}^k_<] \)) for the one-step probability of reaching from vertex \( v \) a successor \( u \) of \( v \) such that \( \overrightarrow{\varphi}^k(u) \leq_{\text{lex}} \overrightarrow{\varphi}^k(v) \) (resp. \( \overrightarrow{\varphi}^k(u) <_{\text{lex}} \overrightarrow{\varphi}^k(v) \)). In other words, \( \Pr_v[\overrightarrow{\varphi}^k_<] \) is the probability in vertex \( v \) of strictly decreasing the \( k \)-th rank in one step, and \( \Pr_v[\overrightarrow{\varphi}^k_\leq] \) is the probability of not increasing the \( k \)-th rank. Moreover, we write \( \Pr_v[\varphi < \infty] \) (or for notational convenience \( \Pr_v[\overrightarrow{\varphi}^1_\leq] \)) for the one-step probability of reaching from \( v \) a successor \( u \) of \( v \) such that \( \varphi(u) \neq \infty \). We always use these notations in the context of expressions such as \( \Pr_v[\overrightarrow{\varphi}^k_<] = 1 \) or \( \Pr_v[\overrightarrow{\varphi}^k_\leq] \geq \varepsilon \). By slight abuse of notation, for vertices \( v \in V_\text{O} \) we also write \( \Pr_v[\overrightarrow{\varphi}^k_<] \) and \( \Pr_v[\overrightarrow{\varphi}^k_\leq] \) in such expressions, and then we mean those expressions to hold if and only if they hold for all mixed one-step strategies in vertex \( v \). It is easy to verify that if either of the two expressions above holds for all pure one-step strategies in \( v \), then it also holds for all mixed one-step strategies.

**Definition 1 (Almost-sure ranking).** A ranking function \( \varphi : V \to [n]^{d/2} \cup \{\infty\} \) for player Even is an almost-sure ranking if there is an \( \varepsilon \geq 0 \) such that for every vertex \( v \) with \( \varphi(v) \neq \infty \), the following condition \( C_v \) holds:

\[
\begin{align*}
- \ p(v) \ even: & v_{\text{odd}} \ i \in [p(v)] (\Pr_v[\overrightarrow{\varphi}^{i-1}_\leq] = 1 \land \Pr_v[\overrightarrow{\varphi}^i_\leq] \geq \varepsilon) \lor (\Pr_v[\overrightarrow{\varphi}^{p(v)-1}_\leq] = 1), \\
- \ p(v) \ odd: & v_{\text{odd}} \ i \in [p(v)] (\Pr_v[\overrightarrow{\varphi}^{i-1}_\leq] = 1 \land \Pr_v[\overrightarrow{\varphi}^i_\leq] \geq \varepsilon).
\end{align*}
\]

**Proposition 1.** Let \( k \in [d] \) be an odd priority. Then for every vertex \( v \) with \( \varphi(v) \neq \infty \) the following conditions hold.

(a) If \( p(v) = k \), then in one step from vertex \( v \) the \( k \)-th rank decreases with probability at least \( \varepsilon \).

(b) If \( p(v) > k \), then in one step from vertex \( v \) either the \( k \)-th rank decreases with probability at least \( \varepsilon \), or the \( k \)-th rank does not increase (with probability 1).

**Lemma 1.** Let \( \varphi \) be an almost-sure ranking for a parity MDP. Then for every (mixed) strategy of player Odd, the Even parity objective is satisfied with probability 1 from every vertex \( v \) with \( \varphi(v) \neq \infty \).

**Proof.** Once the strategy for player Odd is fixed, a play in the parity MDP is an infinite random walk. We argue that this random walk satisfies the Even
In order to prove that with probability 1, the lowest priority occurring infinitely often is even, it suffices to show that for every odd priority \( k \in [d] \), if vertices of priority \( k \) keep occurring in the random walk, then with probability 1 eventually a vertex of lower priority occurs. First note that from the definition of an almost-sure ranking, it follows that all successors of a vertex with finite rank have finite rank: one of the conditions \( \Pr_{v}[\phi^{k}] = 1 \) must hold so the \( k \)-th rank cannot increase in any step and thus the rank stays finite. Let \( k \in [d] \) be odd. For the sake of contradiction assume that from some point on vertices of priority \( k \) keep occurring, but no vertex of a lower priority ever occurs. In this case Proposition 1 implies that in every step either the \( k \)-th rank does not increase with probability 1, or it decreases with probability at least \( \varepsilon \), and moreover, in every step from a vertex of priority \( k \) the \( k \)-th rank decreases with probability at least \( \varepsilon \). As there are \( N = (n+1)(k+1)/2 \) different values of a \( k \)-th rank, within at most \( N \) visits to a vertex of priority \( k \) the \( k \)-th rank must decrease to \((0, \ldots, 0)\) with probability at least \( \varepsilon^{N} \). Thus with probability 1 a vertex with \( k \)-th rank \((0, \ldots, 0)\) is eventually reached. This contradicts the assumption that priority \( k \) occurs infinitely often: no vertex of priority \( k \) can have its \( k \)-th rank equal to \((0, \ldots, 0)\), because by Proposition 1(a) a step from such a vertex has to decrease the \( k \)-th rank with positive probability and \((0, \ldots, 0)\) is the smallest rank.

**Definition 2 (Positive-probability ranking).** A ranking function \( \psi : V \rightarrow [n]^{d/2+1} \cup \{\infty\} \) for player Odd is a positive-probability ranking if there is an \( \varepsilon \geq 0 \) such that for every vertex \( v \) with \( \psi(v) \neq \infty \), the following condition \( D_v \) holds:

- \( p(v) \) even: \( \Pr_{v}[\phi_{0}] \geq \varepsilon \) \( \vee \) \( \Pr_{v}[\phi_{i-2}] = 1 \wedge \Pr_{v}[\phi_{i}] \geq \varepsilon \),
- \( p(v) \) odd: \( \Pr_{v}[\phi_{0}] \geq \varepsilon \) \( \vee \) \( \Pr_{v}[\phi_{i-2}] = 1 \wedge \Pr_{v}[\phi_{i}] \geq \varepsilon \) \( \vee \) \( \Pr_{v}[\phi_{p(v)}] = 1 \).

A proof similar to that of Lemma 1 can be used to prove the following lemma.

**Lemma 2.** Let \( \psi \) be a positive-probability ranking for a parity MDP. Then there is a \( \delta > 0 \) such that for every (mixed) strategy of player Even, the Odd parity objective is satisfied with probability at least \( \delta \) from every vertex \( v \) with \( \psi(v) \neq \infty \).

### 3.2 The Reduction

Given a 2\( 1/2 \)-player parity game \( G = (V, E, (V_\square, V_\vee, V_\circ), p : V \rightarrow [d]) \), we construct a 2-player parity game \( \overline{G} \) with the same set \([d]\) of priorities. For every vertex \( v \in V_\square \cup V_\vee \), there is a vertex \( \overline{v} \in \overline{V} \) with “the same” outgoing edges, i.e., \( (v, u) \in E \) if and only if \( (\overline{v}, \overline{u}) \in \overline{E} \). Each random vertex \( v \in V_\circ \) is substituted by the gadget presented in Figure 1. More formally, the players play the following
Finally, in a vertex \((v, k)\) of priority \(p(v)\). First, in vertex \(\bar{v}\) player Odd chooses a successor \((\hat{v}, k)\), each of priority \(p(v)\), where \(k \in [p(v) + 1]\) is even. Then in vertex \((\hat{v}, k)\) player Even chooses from at most two successors: vertex \((\hat{v}, k - 1)\) of priority \(k - 1\) if \(k > 0\), or vertex \((\hat{v}, k)\) of priority \(k\) if \(k \leq p(v)\). Finally, in a vertex \((\hat{v}, k)\) the choice is between all vertices \(\bar{u}\) such that \((v, u) \in E\), and it belongs to player Even if \(k\) is odd, and to player Odd if \(k\) is even.

**Lemma 3.** For every vertex \(v\) in \(G\), if player Even (resp. Odd) has a sure winning strategy from vertex \(v\) in \(G\), then player Even (resp. Odd) has an almost-sure (resp. positive-probability) winning strategy from \(v\).

**Proof.** We prove the claim for player Even. The case of player Odd is similar and is omitted here. If \(\overline{V}\) is the set of vertices from which player Even has a winning strategy in \(G\), then there is a ranking function (also called a progress measure [10]) \(\varphi: V \to [n]^{d/2} \cup \{\infty\}\) (where \(n \leq |V|\)) such that \(\varphi(w) \neq \infty\) for all \(w \in \overline{V}\). This ranking function induces a memoryless winning strategy \(\overline{\sigma}\) for player Even in \(G\) [10]. We define a ranking function \(\varphi\) and a memoryless strategy for player Even \(\sigma\) for \(G\) by setting \(\varphi(v) = \overline{\varphi}(\overline{v})\) and \(\sigma(v) = \overline{\sigma}(\overline{v})\) for every \(v \in V\).

Taking the strategy subgraph of \(\sigma\) in \(G\) we obtain a parity MDP \(M\). In order to prove the claim, by Lemma 1 it suffices to argue that \(\varphi\) is an almost-sure ranking for \(M\). It is easy to verify that the ranking condition \(C_v\) holds for all vertices \(v \in V_0\) with \(\varphi(v) \neq \infty\). We prove that the ranking condition \(C_v\) holds for all vertices \(v \in V_0\) with \(\varphi(v) \neq \infty\). Let \(k \in [p(v)]_+\). Since vertex \(\overline{v}\) belongs to player Odd, the edge leading to vertex \((\hat{v}, k)\) must be in \(M\). Vertex \((\hat{v}, k)\) belongs to player Even, so either the edge leading to vertex \((\hat{v}, k - 1)\) or the one leading to vertex \((\hat{v}, k)\) belongs to \(M\). In the former case, by analyzing the inequalities between \(\varphi\)-ranks of vertices on the path from \(\overline{v}\) to the successor of \((\hat{v}, k - 1)\) in \(M\) which hold by the definition of a ranking function [10], we can deduce that \(\Pr_v[\varphi^{k-1}] \geq 1/2\) holds. In the latter case, we get that \(\Pr_v[\varphi^{k} \leq 1] = 1\) holds. Considering all edges that lead out of vertex \(\overline{v}\) in \(M\), we conclude that the following condition holds: \(\Pr_v[\varphi^{\infty} = 1] \wedge \bigwedge_{i \in [p(v)]} \Pr_v[\varphi^i \leq 1] = 1 \lor \Pr_v[\varphi^i \leq 1] = 1/2\). This condition can be shown to imply the ranking condition \(C_v\) using the two simple properties that if \(i, j \in [p(v)]\) are odd and \(i < j\), then \(\Pr_v[\varphi^i \leq 1] = 1\) implies \(\Pr_v[\varphi^j \leq 1] = 1\), and \(\Pr_v[\varphi^i \leq 1] \geq \varepsilon\) implies \(\Pr_v[\varphi^j \leq 1] \geq \varepsilon\). 

![Fig. 1. Gadget for reducing 2^{1/2}-player parity games to 2-player parity games.](image-url)
Algorithm 1 Classical algorithm for Büchi MDP’s

Input: $1^1/2$-player Büchi game $(G, T)$. Output: $W_\bigcirc$ and $W_\Box = V \setminus W_\bigcirc$.
1. $G_0 := G$; 2. $W_0 := \emptyset$; 3. $i := 0$
4. repeat
   4.1 $W_{i+1} := \text{One-Iteration-Of-The-Classical-Algorithm}(V_i)$
   4.2 $V_{i+1} := V_i \setminus W_{i+1}$; $i := i + 1$
   until $W_i = \emptyset$
5. $W_\bigcirc := \bigcup_{k=1}^i W_k$

Procedure One-Iteration-Of-The-Classical-Algorithm

Input: set $V_i \subseteq V$. Output: set $W_{i+1} \subseteq V_i$.
1. $R_i := \text{Reach}(T \cap V_i, V_i)$; 2. $T_{i} := V_i \setminus R_{i}$; 3. $W_{i+1} := \text{Attr}_\bigcirc(T_{i}, V_i)$

4 An $O(m\sqrt{m})$ Algorithm for $1^1/2$-Player Büchi Games

In this section we consider $1^1/2$-player games with Büchi winning objectives, i.e., Büchi MDP’s. There are two players, Even and Random. We write $T$ for the set of target vertices, which player Even attempts to visit infinitely often. By $W_\Box$ we denote the set of vertices from which player Even has an almost-sure winning strategy, and by $W_\bigcirc$ the set from which the Büchi objective is violated with positive probability for all strategies of player Even. We call these sets the winning sets for player Even and Random, respectively. The main result of this section is an $O(n\sqrt{n})$ algorithm for computing $W_\Box$ and $W_\bigcirc$ for a $1^1/2$-player Büchi game with $n$ vertices and $O(n)$ edges. This proves Theorem 3, because a game graph with $m$ edges can be easily converted in $O(m + n)$ time to an equivalent game graph with $O(m)$ vertices and $O(m)$ edges.

In the rest of the paper we use the following notations for a graph $G = (V, E)$ and a set $S \subseteq V$ of vertices. We write $\text{succ}(v, G) = \{ u \in V : (v, u) \in E \}$ for the set of immediate successors of vertex $v$. We define $\text{In}(S, G) = \{ (v, u) \in E : v \notin S \text{ and } u \in S \}$ to be the set of edges that enter set $S$, and $\text{Source}(S, G) = \{ v \in V : (v, u) \in \text{In}(S, G) \text{ for some } u \}$ is the set of sources of edges that enter $S$. We write $\text{Reach}(S, G)$ for the set of vertices from which there is a path in graph $G$ to a vertex in $S$. Let $(V_\Box, V_\bigcirc)$ be a partition of the set $V$ of vertices (player $\Box$ moves from the vertices in $V_\Box$ and player $\bigcirc$ moves from the vertices in $V_\bigcirc$) where $\bigcirc \in \{\Diamond, \bigcirc\}$). We inductively define the set $\text{Attr}_\bigcirc(S, G)$ of vertices from which player $\Box$ has a strategy to reach the set $S$ in the following way. Set $R_0 = S$, and for $k \geq 0$, set $R_{k+1} = R_k \cup \{ v \in V_\Box : (v, u) \in E \text{ for some } u \in R_k \}$ $\cup \{ v \in V_\bigcirc : u \in R_k \text{ for all } (v, u) \in E \}$. Let $\text{Attr}_\bigcirc(S, G) = \bigcup_k R_k$. We define the set $\text{Attr}_\bigcirc(S, G)$ in a similar way. We fix a graph $G$ until the end of the paper and by a slight abuse of notation instead of putting graphs as the second parameters of all the above definitions, we will write a subset $S$ of the vertices to stand for the subgraph of $G$ that is induced by the set $S$ of vertices.

The classical algorithm (Algorithm 1) works as follows. First it finds the set of vertices $R$ from which the target set $T$ is reachable. The rest of the vertices $Tr = V \setminus R$ (a “trap” for player Even) are identified as winning for
player Random. Then the set of vertices $W$, from which player Random has a strategy to reach its winning set $Tr$, is computed. Set $W$ is identified as a subset of the winning set for player Random and it is removed from the vertex set. The algorithm then iterates on the reduced game graph. In every iteration it performs a backward search from the current target set to find vertices which can reach the target set. Each iteration takes $O(n)$ time.

Our improved algorithm (Algorithm 2) differs from the classical algorithm by selectively performing a backward search as the classical algorithm does, or a cheap forward exploration of edges from vertices that are good candidates to be included in the winning set of player Random. In Step 4.1 of the improved algorithm, if the number of edges entering the set of vertices included in the previous iteration into the winning set of player Random is at least as big as a constant $k$, then we run an iteration of the classical algorithm. Otherwise, i.e., if this number is smaller than $k$, then let $S$ be the set of sources of edges that enter the set of vertices winning for player Random in the previous iteration. The vertices in $S$ are considered as candidates to be included into the winning set of player Random in the current iteration. In Step 4.2.2.1 (procedure Dovetail-Explore) a dovetailing exploration of edges is performed from all vertices in $S$. From all vertices $v \in S$, up to $\ell$ edges are explored in a round-robin fashion. If the forward exploration of edges from a vertex $v \in S$ terminates before $\ell$ edges are explored, and none of the explored vertices is in the target set, then the explored subgraph is included in the winning set of player Random. If Step 4.2.2.1 fails to identify a winning set of player Random, then in Step 4.2.4 an iteration of the classical algorithm is executed. The winning set discovered by the iteration of the classical algorithm in Step 4.2.4 must contain at least $\ell$ edges as otherwise it would have been discovered in Step 4.2.2.1.

**Lemma 4.** Algorithm 2 correctly computes the sets $W^\circ$ and $W^\square$.

**Proof.** We prove by induction that $W_i$ computed in any iteration of the improved algorithm satisfies $W_i \subseteq W^\circ$. Base case: $W_0 = \emptyset \subseteq W^\circ$.

Inductive case: we argue that $W_i \subseteq W^\circ$ implies $W_{i+1} \subseteq W^\circ$.

1. If Step 4.1 is executed, then $W_{i+1} \subseteq W^\circ$ by correctness of the classical algorithm.

2. If Step 4.2.2 is executed, then a nonempty set $R_v$ is included in $Tr_{i+1}$ in Step 3.3 of procedure Dovetail-Explore. By the condition in Step 3.3 of Dovetail-Explore, no vertex in $R_v$ can reach a vertex outside of $R_v$, and since $R_v \cap T \cap V_i = \emptyset$, we conclude that $T_i$ cannot be reached from any vertex in $R_v$. Therefore $R_v \subseteq W^\circ$ and $Tr_{i+1} \subseteq W^\circ$. Hence $U \subseteq W^\circ$ and $W_{i+1} \subseteq W^\circ$.

3. If Steps 4.2.2 and 4.2.4 are executed, then $U \subseteq W^\circ$ and the correctness of an iteration of the classical algorithm imply $W_{i+1} \subseteq W^\circ$.

The other inclusion $W^\circ \subseteq W^\square$ follows from the correctness of the classical algorithm, because the termination condition of the loop in Step 4 implies that $Tr_i = \emptyset$ holds so the last iteration was an iteration of the classical algorithm.

**Lemma 5.** The total work in Step 4.2.2.1 of Algorithm 2 is $O(kn)$. 

Algorithm 2 Improved Algorithm for Büchi MDP’s

**Input:** \( 1^{1/2} \)-player Büchi game \((G,T) \). **Output:** \( W_\emptyset \) and \( W_\emptyset = V \setminus W_\emptyset \).

1. \( G_0 := G \); 2. \( W_0 := \emptyset \); 3. \( i := 0 \)

4. **repeat**
   
   4.1 if \( |In(W_i, V_i \cup W_i)| \geq k \) then
      
      4.1.1 \( W_{i+1} := \text{One-Iteration-Of-The-Classical-Algorithm}(V_i) \)

   4.2 else (\( |In(W_i, V_i \cup W_i)| < k \))
      
      4.2.1 \( U := \emptyset \)

   4.2.2 **repeat**
      
      4.2.2.1 \( Tr_i := \text{Dovetail-Explore}(V_i \setminus U, \text{Source}(W_i \cup U, V_i \cup W_i)) \)

      4.2.2.2 \( U := U \cup Attr_\emptyset(Tr_i, V_i \setminus U) \)

      **until** \( |In(W_i \cup U, V_i \cup W_i)| \leq k \) and \( Tr_i \neq \emptyset \)

   4.2.3 if \( Tr_i \neq \emptyset \) then \( W_{i+1} := W_i \cup U \)

   4.2.4 else \( W_{i+1} := U \cup (\text{One-Iteration-Of-The-Classical-Algorithm}(V_i \setminus U)) \)

   4.3 \( V_{i+1} := V_i \setminus W_{i+1} \)

   4.4 \( i := i + 1 \)

   **until** \( W_i = \emptyset \)

5. \( W_\emptyset := \bigcup_{k=1}^i W_k \)

**Procedure** \textit{Dovetail-Explore}

**Input:** set \( S \subseteq V_i \) and graph \( G_i = (V_i, E_i) \). **Output:** set \( Tr_i \subseteq V_i \).

1. \( Tr_i := \emptyset \); 2. \( ec := \ell \)

3. **repeat**

   3.1 \( ec := ec - 1 \)

   3.2 **for each** vertex \( v \in S \)

   extend the sub-graph \( R_v \subseteq V_i \) by exploring a new edge

   3.3 if there is \( v \in S \) s.t. for all \( u \in R_v \), \( \text{succ}(u, V_i) \subseteq R_v \) and \( R_v \cap T \cap V_i = \emptyset \)

   then return \( Tr_i := R_v \)

   **until** \( ec = 0 \)

4. return \( Tr_i \)

**Proof.** Consider the following two cases.

1. If a nonempty set of vertices \( R_v \) is included in the set \( Tr_i \) in Step 3.3 of \textit{Dovetail-Explore}, and the number of edges in the induced subgraph \( R_v \) is \( e_i \), then the total work done in Step 3 is \( O(ke_i) \), because \( |\text{Source}(W_i \cup U, V_i \cup W_i)| \leq k \). Since \( e_i \) edges are removed from the graph and the number of all edges in the graph is \( O(n) \), the total work over all iterations of \textit{Dovetail-Explore} when a nonempty set \( R_v \) is included in a set \( Tr_i \) is \( O(kn) \).

2. If \( Tr_i = \emptyset \) after executing the procedure \textit{Dovetail-Explore}, then the work done there is \( O(k\ell) \). Whenever this happens, the subgraph induced by the set of vertices \( Tr_i \) discovered by the following iteration of the classical algorithm must have more than \( \ell \) edges. This can happen at most \( O(n/\ell) \) times, because the number of edges in the graph is \( O(n) \). Hence the total work over all iterations is \( O((n/\ell)k\ell) = O(kn) \).

**Lemma 6.** The total work in Step 4.2.2.2 of Algorithm 2 is \( O(n) \), in Step 4.2.2 it is \( O(kn) \), in Step 4.1 it is \( O(n^2/k) \), and in Step 4.2.4 it is \( O(n^2/\ell) \).
Lemma 7. Algorithm\textsuperscript{2} solves $1^1/2$-player Büchi games with $n$ vertices and $O(n)$ edges in time $O(n\sqrt{n})$.

Proof. Correctness follows from Lemma 4. By Lemma 6 the work in Steps 4.1, 4.2.2, and 4.2.4 is $O(n^2/k + kn + n^2/\ell)$. Take $k = \ell = \sqrt{n}$ to get the $O(n\sqrt{n})$ bound for the total work. \[\square\]

5 An $O(n^2/\log n)$ Algorithm for 2-Player Büchi Games

In this section we consider 2-player games of the form $(G, T)$, where $T$ for the set of target vertices. By $\mathcal{W}_{\bigotimes}$ we denote the set of vertices from which player Even has a strategy to visit a state in $T$ infinitely often, and by $\mathcal{W}_{\bigodot}$ the set of vertices from which player Odd can avoid visiting $T$ infinitely often. These are the winning sets for player Even and Odd, respectively. By determinacy of parity games \cite{7} we have $\mathcal{W}_{\bigodot} = V \setminus \mathcal{W}_{\bigotimes}$. Inspired by the algorithm of the previous section, we provide an algorithm for computing the set $\mathcal{W}_{\bigodot}$ in time $O(n^2/\log n)$ if $G$ has $n$ vertices and $O(n)$ edges, a proof of Theorem 4. A graph is binary if every vertex has at most two successors. For simplicity we present the algorithm for the case when the game graph is binary.

The classical algorithm for solving 2-player Büchi games (Algorithm\textsuperscript{3} \cite{15}) is very similar to the classical algorithm of the previous section. The only difference is that in step 1 of each iteration $i$ we compute set $R_i$ to be the set of vertices from which player Even has a strategy to reach set $T$, i.e., $R_i = Attr_{\bigotimes}(T, V_i)$; and in step 3 the set $W_{i+1}$ is the set of vertices from which player Odd has a strategy to reach set $Tr_i$. Then the winning set of player Even is obtained as the union of the sets $W_i$ over all iterations.

Note that in step 1 of every iteration $i$ an $O(n)$ backward alternating search is performed to compute the set $R_i$. The key idea of our improved algorithm (Algorithm\textsuperscript{4}) is to perform a cheap forward exploration of edges in some iterations in order to discover subsets of the winning set for player Odd. The improved algorithm for 2-player Büchi games differs from the improved algorithm of the previous section in the way the forward exploration is performed. In order to detect a trap for player Even in which player Odd has a winning strategy, we need to consider all successors of every vertex in the forward exploration. Let $S$ be the set of sources of edges entering the winning set of player Odd discovered in the previous iteration, and let $|S| \leq k$. The vertices in set $S$ are new candidates to be included in the winning set of player Odd. From these vertices a BFS of depth $\log \ell$ is performed in Step 4.2.2.1 of Algorithm\textsuperscript{4}. In step 4.2.2.4 we check if the explored subgraph contains a trap for player Even in which player Odd has a winning strategy. If no such trap is detected then one iteration of the classical algorithm is executed. The key for the subquadratic bound of our algorithm is the observation that if step 4.2.2 fails to identify a non-empty winning subset for player Odd, then the set discovered by the following iteration of the classical algorithm has at least $\log \ell$ vertices.

We say that a set of vertices $S$ is an Even trap in a graph $G$ if for all $v \in S$, we have $\text{succ}(v, G) \subseteq S$ if $v \in V_{\bigotimes}$, and $\text{succ}(v, G) \cap S \neq \emptyset$ if $v \in V_{\bigodot}$. It is easy
Algorithm 3 Classical Algorithm for 2-player Büchi Games

**Input**: 2-player Büchi game \((G, T)\). **Output**: \(W_\odot\) and \(W_\Box = V \setminus W_\odot\).

Steps 1.–4. are the same as in Algorithm 1.

4. \(W_3 := \bigcup_{k=1}^{i} W_k\)

Procedure One-Iteration-Of-The-Classical-Algorithm

**Input**: set \(V_i \subseteq V\). **Output**: set \(W_{i+1} \subseteq V_i\).

1. \(R_i := Attr_2(T \cap V_i, V_i)\); 2. \(Tr_i := V_i \setminus R_i\); 3. \(W_{i+1} := Attr_3(Tr_i, V_i)\)

Algorithm 4 Improved Algorithm for 2-player Büchi Games

**Input**: 2-player Büchi game \((G, T)\). **Output**: \(W_\odot\) and \(W_\Box = V \setminus W_\odot\).

Steps 1.–3. and 4.1 are the same as in Algorithm 2.

4. repeat
   4.2 else \(((\mid\text{In}(W_i, V_i \cup W_i)\mid < k)\)
      4.2.1 \(Tr_i := \emptyset\)
      4.2.2 for each vertex \(v \in \text{Source}(W_i, V_i \cup W_i)\)
         4.2.2.1 Find the reachable subgraph \(R_v\) by a BFS of depth \(\log \ell\)
         4.2.2.2 Let \(F_v\) denote the set of vertices at depth \(\log \ell\)
         4.2.2.3 \(T'_v := \{v \in V_\Box \cap F_v : \text{succ}(v, G_i) \cap R_v = \emptyset\} \cup (V_\Box \cap F_v)\)
         4.2.2.4 \(R'_v := Attr_2((R_v \cap T \cap V_i) \cup T'_v, R_v)\)
         4.2.2.5 \(Tr'_i := Tr_i \cup (R_v \setminus R'_v)\)
      4.2.3 if \(Tr_i \neq \emptyset\) then \(W_{i+1} := Attr_3(Tr_i, V_i)\)
      4.2.4 else \(W_{i+1} := \text{One-Iteration-Of-The-Classical-Algorithm}(V_i)\)
   4.2 until \(W_i = \emptyset\)

5. \(W_\odot := \bigcup_{k=1}^{i} W_k\)

Lemma 8. Algorithm 4 correctly computes the sets \(W_\odot\) and \(W_\Box\).

Proof. We prove by induction that \(W_i\) computed in any iteration of the improved algorithm satisfies \(W_i \subseteq W_\odot\). The proof is similar to Lemma 4 and differs only for case 2, which we prove now. If Steps 4.2 and 4.2.3 get executed in iteration \(i\), then every nonempty set \(R_v \setminus R'_v\) included in the set \(Tr_{i+1}\) is an Even trap winning for player Odd in the subgraph of \(G\) induced by the set of vertices \(V_i\). It is an Even trap because for every vertex \(u \in V_\Box \cap (R_v \setminus F_v)\), we have \(\text{succ}(u, V_i) \subseteq R_v\), and as a complement of an Even attractor it is an Even trap in the subgraph induced by set \(R_v\). The set \(R_v \setminus R'_v\) is moreover an Even trap winning for player Odd, because by step 4.2.2.4 all target vertices in the set \(R_v\) are included in the set \(R'_v\).
Lemma 9. Let $R_v$ be a set computed in Step 4.2.2.1. Let $S$ be an Even trap winning for player Odd such that $v \in S$, all vertices of $S$ are reachable from $v$, and $|S| < \log \ell$. Then $S \subseteq R_v \setminus R'_v$, and hence $S$ is discovered in Step 4.2.2.

Lemma 10. The total work in Step 4.1 of Algorithm 4 is $O(n^2/k)$, in Step 4.2.2 it is $O(k\ell n)$, and in Step 4.2.3 it is $O(n)$.

Lemma 11. Algorithm 4 solves 2-player binary Büchi games in time $O(n^2 / \log n)$.

Proof. Correctness follows from Lemma 8. The work of Step 4.2.4 is $O(n^2 / \log \ell)$ by Lemma 9. The work of Steps 4.1, 4.2.2, and 4.2.4 is $O(n^2/k + k\ell n + n^2 / \log \ell)$ by Lemma 10. Take $\ell = n^\varepsilon$ with $0 < \varepsilon < 1$ and $k = \log n$ to get the $O(n^2 / \log n)$ upper bound for the total work. By Lemma 10, the work in Step 4.2.3 is $O(n)$, hence the time complexity of Algorithm 4 is $O(n^2 / \log n)$.

References

Machine Characterizations
of the Classes of the W-Hierarchy

Yijia Chen and Jörg Flum
Abteilung für Mathematische Logik, Universität Freiburg,
Eckerstr. 1, 79104 Freiburg, Germany
chen@zermelo.mathematik.uni-freiburg.de, Joerg.Flum@math.uni-freiburg.de

Abstract. We give machine characterizations of the complexity classes
of the W-hierarchy. Moreover, for every class of this hierarchy, we present
a parameterized halting problem complete for this class.

1 Introduction

Parameterized complexity theory provides a framework for a refined complexity
analysis of algorithmic problems that are intractable in general. Central to the
theory is the notion of fixed-parameter tractability, which relaxes the classical
notion of tractability, polynomial time computability, by admitting algorithms
whose runtime is exponential, but only in terms of some parameter that is usu-
ally expected to be small. As a complexity theoretic counterpart, a theory of
parameterized intractability has been developed. In classical complexity theory,
the notion of NP-completeness is central to a nice and simple theory for in-
tractable problems. Unfortunately, the world of parameterized intractability is
more complex: there is a big variety of seemingly different classes of parameter-
ized intractability. Nevertheless, it can be argued that the classes W[1], W[2], . . .
of the W-hierarchy together with some other classes like W[P] correspond to NP
in classical complexity theory. In particular, all these classes are defined by pa-
rameterized variants of the NP-complete satisfiability problem. Unfortunately,
the definition of these classes by means of complete problems makes it not easy
to understand them. The authors of [4] tried to remedy this situation by present-
ing machine characterizations for some of the classes. But as they remarked “it
remains an interesting open problem to find natural machine characterizations
for the classes W[t] for t ≥ 2”. In this paper we obtain such characterizations.

By definition, a parameterized problem is fixed-parameter tractable, if it is
decidable by a (deterministic) algorithm in at most f(k) · p(n) steps for some
computable function f and some polynomial p. Here k denotes the size of the
parameter and n the size of the input. Problems in any of the “intractable”
parameterized complexity classes mentioned so far, also are decidable in at most
f(k) · p(n) steps but by a nondeterministic algorithm. Thus, a first attempt to
characterize one of these intractable classes by machines consists in considering
nondeterministic algorithms that perform at most f(k) · p(n) steps but restricting
further the number of nondeterministic steps. This approach led to the following

© Springer-Verlag Berlin Heidelberg 2003
results in [4]: A problem is in $W[P]$ if and only if it is decidable in at most $f(k) \cdot p(n)$ steps by an algorithm whose number of nondeterministic steps is bounded in terms of the parameter. Similarly, a problem is in $W[1]$, if in addition the nondeterministic steps are performed at the end of the computation.

Already in [3], nondeterministic random access machines turned out to be the appropriate machine model in order to get clear formulations of the characterizations. In their nondeterministic steps these machines are able to guess natural numbers ($\leq f(k) \cdot p(n)$); these numbers are considered as names of objects. To obtain machine characterizations of the classes of the W-hierarchy we have to consider the corresponding alternating random access machines, but at the same time we have to ensure that the programs only have access to (the properties of) the elements named by the guessed numbers and not to the numbers themselves.

In [3], Cesati and Di Ianni prove that the halting problem $p$-HPNMT for nondeterministic multitape Turing machines, parameterized by the number of steps, is $W[2]$-hard by reducing the parameterized dominating set problem to it: the machine first guesses the elements of a dominating set and then checks that they really constitute a dominating set. An analysis of the use of and the access to the guessed elements in this algorithm helped the authors to find the machine characterizations of the classes of the W-hierarchy. In Section 4 we present a (short) proof of membership in $W[2]$ of $p$-HPNMT by reducing it to a model-checking problem in $W[2]$, a result obtained in [2] by different means.

As the corresponding proof in [4] shows, the machine characterization of $W[1]$ is closely related to the $W[1]$-completeness of the halting problem for nondeterministic Turing machines. For $t \geq 2$, the $W[t]$-complete halting problem we present in the last section refers to alternating Turing machines with oracles and it is not so closely related to the corresponding machine characterization but to a logical model-checking problem complete for $W[t]$.

2 Preliminaries

In this section we recall some definitions and results and fix our notations.

2.1. Relational Structures and First-Order Logic. A vocabulary $\tau$ is a finite set of relation symbols. Each relation symbol has an arity. A (relational) structure $A$ of vocabulary $\tau$, or $\tau$-structure, consists of a set $A$ called the universe, and an interpretation $R^A \subseteq A^r$ of each $r$-ary relation symbol $R \in \tau$. We synonymously write $\bar{a} \in R^A$ or $R^A \bar{a}$ to denote that the tuple $\bar{a} \in A^r$ belongs to the relation $R^A$. For example, we view a directed graph as a structure $G = (G, E^G)$, whose vocabulary consists of one binary relation symbol $E$. $G$ is an (undirected) graph, if $E^G$ is irreflexive and symmetric.

The class of all first-order formulas is denoted by $FO$. They are built up from atomic formulas using the usual boolean connectives and existential and universal quantification. Recall that atomic formulas are formulas of the form $x = y$ or $Rx \ldots x_r$, where $x, y, x_1, \ldots, x_r$ are variables and $R$ is an $r$-ary relation symbol. For $t \geq 1$, $\Sigma_t$ denotes the class of all first-order formulas of the form

$$\exists x_1 \ldots \exists x_{k_1} \forall x_{21} \ldots \forall x_{2k_2} \ldots Qx_{t1} \ldots Qx_{tk_t} \psi,$$
where $Q = \forall$ if $t$ is even and $Q = \exists$ otherwise, and where $\psi$ is quantifier-free. $\Pi_t$-formulas are defined analogously starting with a block of universal quantifiers. Let $t, u \geq 1$. A formula $\varphi$ is $\Sigma_{t,u}$ if it is $\Sigma_t$ and all quantifier blocks after the leading existential block have length $\leq u$.

If $\Phi$ is a class of formulas, then $\Phi[\tau]$ denotes the class of all formulas of vocabulary $\tau$ in $\Phi$. If $A$ is a $\tau$-structure and $\psi \in \text{FO}[\tau]$ a sentence, i.e., a formula without free variables, then we write $A \models \varphi$ to denote that $A$ satisfies $\varphi$.

The proof of the following lemma is easy.

**Lemma 1.** Let $\varphi_1(\bar{x}), \ldots, \varphi_m(\bar{x})$ and $\psi_1(\bar{x}, \bar{y}), \ldots, \psi_m(\bar{x}, \bar{y})$ be formulas in $\text{FO}[\tau]$, where $\bar{x} = x_1 \ldots x_r$ and $\bar{y} = y_1 \ldots y_s$ are sequences of variables that have no variable in common. Assume $Q_1, \ldots, Q_r, Q'_1, \ldots, Q'_s \in \{\forall, \exists\}$. If $A$ is a $\tau$-structure with $A \models \forall \bar{x} \neg (\varphi_i \land \varphi_j)$ for $i \neq j$, then $A$ satisfies both or none of the formulas

\[
\begin{align*}
&Q_1 x_1 \ldots Q_r x_r \land_{i=1}^{m} (\varphi_i \rightarrow Q'_1 y_1 \ldots Q'_s y_s \psi_i), \\
&Q_1 x_1 \ldots Q_r x_r Q'_1 y_1 \ldots Q'_s y_s \land_{i=1}^{m} (\varphi_i \rightarrow \psi_i).
\end{align*}
\]

### 2.2. Parameterized Complexity

A parameterized problem is a set $Q \subseteq \Sigma^* \times \Pi^*$, where $\Sigma$ and $\Pi$ are finite alphabets. If $(x, y) \in \Sigma^* \times \Pi^*$ is an instance of a parameterized problem, we refer to $x$ as the input and to $y$ as the parameter. We usually denote the length of the input string $x$ by $n$ and the length of the parameter string $y$ by $k$.

For example, for a class $S$ of structures and a class $L$ of first-order formulas,

\[ p\text{-MC}(S, L) := \{(A, \varphi) \mid A \in S, \varphi \text{ a sentence in } L, \text{ and } A \models \varphi\} \]

is the parameterized model-checking problem for $S$ and $L$; mostly, for easier readability, we present parameterized problems in the following form:

<table>
<thead>
<tr>
<th>p-MC(S,L)</th>
<th>Input: $A \in S$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter: $\varphi$, a sentence in $L$.</td>
<td></td>
</tr>
<tr>
<td>Problem: $A \models \varphi$?</td>
<td></td>
</tr>
</tbody>
</table>

**Definition 1.** A parameterized problem $Q \subseteq \Sigma^* \times \Pi^*$ is fixed-parameter tractable, if there are a computable function $f : \mathbb{N} \rightarrow \mathbb{N}$, a polynomial $p$, and an algorithm that, given a pair $(x, y) \in \Sigma^* \times \Pi^*$, decides if $(x, y) \in Q$ in at most $f(k) \cdot p(n)$ steps.

FPT denotes the complexity class consisting of all fixed-parameter tractable parameterized problems.

Complementing the notion of fixed-parameter tractability, there is a theory of parameterized intractability. It is based on the following notion of reduction:

**Definition 2.** An FPT-reduction from the parameterized problem $Q \subseteq \Sigma^* \times \Pi^*$ to the parameterized problem $Q' \subseteq (\Sigma')^* \times (\Pi')^*$ is a mapping $R : \Sigma^* \times \Pi^* \rightarrow (\Sigma')^* \times (\Pi')^*$ such that:
1. For all \((x, y) \in \Sigma^* \times \Pi^*\): \((x, y) \in Q \iff R(x, y) \in Q'\).

2. There exists a computable function \(g : \mathbb{N} \rightarrow \mathbb{N}\) such that for all \((x, y) \in \Sigma^* \times \Pi^*\), say with \(R(x, y) = (x', y')\), we have \(k' \leq g(k)\) (where \(k = |y|\) and \(k' = |y'|\)).

3. There exist a computable function \(f : \mathbb{N} \rightarrow \mathbb{N}\) and a polynomial \(p\) such that \(R\) is computable in time \(f(k) \cdot p(n)\).

We write \(Q \leq_{\text{FPT}} Q'\) if there is an FPT-reduction from \(Q\) to \(Q'\) and set \(\mathbb{[}Q\mathbb{]}_{\text{FPT}} := \{Q' \mid Q' \leq_{\text{FPT}} Q\}\). For a class \(C\) of parameterized problems, we let \(\mathbb{[}C\mathbb{]}_{\text{FPT}} := \bigcup_{Q \in C}\mathbb{[}Q\mathbb{]}_{\text{FPT}}\).

Denote by \(\text{GRAPH}\) the class of all finite graphs and by \(\text{STR}\) the class of all finite structures. The parameterized complexity classes \(W[1], W[2], \ldots\) of the \(W\)-hierarchy are defined as the closure of a family of parameterized problems under FPT-reductions. For \(W[t]\), the defining family of problems consists of parameterized versions of the satisfiability problems for circuits of weft \(t\) (and varying depth).

For the purposes of this paper, the most appropriate way to introduce the complexity classes of the \(W\)-hierarchy is the following (cf. [7], [8]):

**Definition 3.** For \(t \geq 1\), \(W[t]\) is the class of all parameterized problems that, for some \(u \geq 1\), are FPT-reducible to \(p\text{-MC(\text{GRAPH}, \Sigma_{t,u})}\), that is,

\[
W[t] = \mathbb{[}\{p\text{-MC(\text{GRAPH}, \Sigma_{t,u})} \mid u \geq 1\}\mathbb{]}_{\text{FPT}}.
\]

The following equivalent characterization of \(W[t]\) is well-known:

**Proposition 1.** \(W[t] = \mathbb{[}\{p\text{-MC(\text{STR}, \Sigma_{t,u}[\tau])} \mid u \geq 1, \tau \text{ vocabulary}\}\mathbb{]}_{\text{FPT}}\).

At various places of this paper we tacitly make use of the following remark:

**Remark 1.** Sometimes, in order to show that a given parameterized problem is in \(W[t]\), we will present a reduction to \(p\text{-MC(\text{STR}, \Sigma_{t,u}[\tau])}\) for some vocabulary \(\tau\) also containing a fixed finite number of constant symbols or even, we will consider a reduction where the number of constants depends on the parameter. This will allow to express properties in a more readable fashion. If we have a fixed finite number of constant symbols, they can be eliminated by using appropriate unary relations that are singletons; then, in the \(\Sigma_{t,u}\)-formula to be defined, the constants are replaced by variables that are existentially quantified (in the first block) and get their right value using the corresponding relations. If the number of constants symbols depends on the parameter, e.g., we use constants for \(0, 1, \ldots, k\), in order to stay within a fixed vocabulary (independent of \(k\)) these constants can be eliminated by a singleton relation for 0 and the binary successor relation on \(\{0, 1, \ldots, k\}\) and again by existentially quantified variables.

Sometimes we refer to the complexity classes of the \(A\)-hierarchy (cf. [8]):

**Definition 4.** For \(t \geq 1\), \(A[t] = [p\text{-MC(\text{GRAPH}, \Sigma_t)}]_{\text{FPT}}\).
3 Machine Characterization

In [4], machine characterizations of the classes \( W[1] \), \( A[t] \) for \( t \geq 1 \), and \( W[P] \) using nondeterministic and alternating random access machines (RAMs) were presented. The nondeterministic RAMs are based on the standard random access machines (cf. [10]). The model was non-standard when it came to nondeterminism. Instead of allowing the machines to nondeterministically choose one bit, or an instruction of the program to be executed next, the authors allowed them to nondeterministically choose a natural number, more precisely, there was an additional instruction “GUESS \( i \ j \)” whose semantics was: Guess a natural number less than or equal to the number stored in register \( i \) and store it in register \( j \). In [4] it was remarked: “While this form of nondeterminism may seem unnatural at first sight, we would like to argue that it is very natural in many typical ‘applications’ of nondeterminism. For example, a nondeterministic algorithm for finding a clique in a graph guesses a sequence of vertices of the graph and then verifies that these vertices indeed form a clique. Such an algorithm is much easier described on a machine that can guess the numbers representing the vertices of a graph at once, rather than guessing their bits.”

The alternating RAMs, in addition to the instructions of the form “GUESS \( i \ j \)” (denoted by “EXISTS \( i \ j \)” in the context of alternating machines) also had “FORALL \( i \ j \)” instructions. The semantics was defined as usually for alternating machines. The computations of alternating machines suitable for \( A[t] \) have a parameter-bounded final part which contains all the EXISTS- and FORALL-instructions and have at most \( t−1 \) alternations (see [4] for the precise statement). For \( W[t] \) it seems not to suffice (see Section 6) to bound the length of the blocks without alternation, but we have to restrict the access to the numbers guessed in the EXISTS- and FORALL-instructions: as just remarked in the algorithm for finding a clique, we view these numbers as labels of certain objects and the type of machine we are going to introduce only has access to the properties of these objects and not directly to the labels.

We turn to the precise definition of the random access machines we are going to use and that we call W-RAMs. A W-RAM has the

- the standard registers \( 0, 1, \ldots \), their contents are denoted by \( r_0, r_1, \ldots \), respectively.
- the guess registers \( 0, 1, \ldots \), their contents are denoted by \( g_0, g_1, \ldots \), respectively.

All registers have initial value 0. Often we denote \( g_{r_i} \), i.e., the contents of the guess register whose index is the content of the \( i \)th standard register, by \( g(r_i) \).

The W-RAM has all the standard instructions for the standard registers (cf. Section 2.6 of [10]; e.g., the arithmetic operations are addition, subtraction, and division by two (rounded off)). Moreover, it has four additional instructions:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXISTS ( \uparrow j )</td>
<td>guess a natural number ( \leq r_0 ); store it in the ( r_j )th guess register</td>
</tr>
<tr>
<td>FORALL ( \uparrow j )</td>
<td>guess a natural number ( \leq r_0 ); store it in the ( r_j )th guess register</td>
</tr>
<tr>
<td>JG(=i\ j\ c)</td>
<td>if ( g(r_i) = g(r_j) ), then jump to the instruction with label ( c )</td>
</tr>
<tr>
<td>JG0(i\ j\ c)</td>
<td>if ( r(g(r_i), g(r_j)) = 0 ), then jump to the instruction with label ( c ).</td>
</tr>
</tbody>
</table>
Here, $\langle i,j \rangle: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is any simple coding of ordered pairs of natural numbers by natural numbers such that $\langle i,j \rangle \leq (1 + \max\{i,j\})^2$ and $\langle 0,0 \rangle = 0$. Of course, the semantics of the EXISTS- and FORALL-instructions are the same, but we view them as existential and universal instructions, respectively. All other instructions are said to be deterministic. If the machine stops, it accepts its input, if $r_0 = 0$, otherwise it rejects it.

The following lemma, whose proof is immediate, is crucial for the main theorem of this section; it shows that the contents of the standard registers depend only on the sequence of executed instructions:

**Lemma 2.** Assume that, for a given input, we have two (partial) computations on a W-RAM. If the same sequence of instructions is carried out in both computations, then the contents of the standard registers will be the same.

**Definition 5.** A program $P$ for a W-RAM is an AW-program, if there is a computable function $f$ and a polynomial $p$ such that for every input $(x,y)$ with $|x| = n$ and $|y| = k$ the program $P$ on every run

1. performs at most $f(k) \cdot p(n)$ steps;
2. at most $f(k)$ steps are existential or universal;
3. at most the first $f(k) \cdot p(n)$ standard registers are used;
4. at every point of the computation the registers contain numbers $\leq f(k) \cdot p(n)$.

The promised machine characterization of $W[t]$ reads as follows:

**Theorem 1.** Let $Q$ be a parameterized problem and $t \geq 1$. Then $Q$ is in $W[t]$ if and only if there is a $u \geq 1$ and there are a computable function $h$ and an AW-program $P$ for a W-RAM such that $P$ decides $Q$ and such that for every run of $P$ on an instance $(x,y)$ of $Q$ as input (with $|y| = k$)

- all existential and universal steps are among the last $h(k)$ steps of the computation,
- there are at most $t - 1$ alternations between existential and universal states, and the first guess step is existential,
- every block without alternations, besides the first one, contains at most $u$ guess steps.

**Proof.** Assume first that $Q \in W[t]$, then $Q \leq_{\text{FPT}}^{\text{p-MC}} (\text{GRAPH}, \Sigma_{t,u})$ for some $u \geq 1$. Hence there are computable functions $f$ and $g$, a polynomial $p \in \mathbb{N}[x]$, and an algorithm $A$ assigning to every $(x,y)$, in time $\leq f(k) \cdot p(n)$, a graph $G = G_{x,y}$ and a sentence $\varphi = \varphi_{x,y} \in \Sigma_{t,u}$, say,

$$\varphi = \exists x_{11} \ldots \exists x_{1k_1} \forall x_{21} \ldots \forall x_{2k_2} \ldots Q x_{t1} \ldots Q x_{tk_t} \psi,$$

with $k_2, \ldots, k_t \leq u$, with $|\varphi| \leq g(k)$, and with a quantifier-free $\psi$, such that

$$Q xy \iff G \models \varphi.$$
The claimed AW-program \( P \) for a W-RAM, on input \((x, y)\), proceeds as follows:

1. It computes the graph \( G = (G, E^G) \), say with \( G = \{1, \ldots, m\} \), and stores its adjacency matrix in the standard registers: \( r_{i,j} = 0 \iff E^G_{i,j} \).
2. It computes \( \phi \).
3. It checks whether \( G|\phi = \phi \).

To carry out point 3, the program \( P \), using the EXISTS- and FORALL-instructions guesses the values of the quantified variables. Then, it checks the quantifier-free part using the JG=-=-=-=- and JG0-instructions. The number of steps needed for point 3 can be bounded by \( h(k) \) for some computable \( h \). Hence, all existential and universal steps are among the last \( h(k) \) steps of the computation.

Now assume that \( P = (\pi_1, \ldots, \pi_m) \) is an AW-program deciding \( Q \) and that \( u \geq 1, h, \) and \( P \) have the properties stated at the right side of the equivalence claimed in the theorem. For the program \( P \) choose the function \( f \) and the polynomial \( p \) according to Definition 5. We claim that \( Q \in W[t] \). By Proposition 1, it suffices to show that \( Q \leq_{FPT} p-MC(STR, \Sigma_{t,2}.u[\tau]) \) for some \( \tau \). The set of instruction numbers of \( P \) is \( \{1, \ldots, m\} \), more precisely, \( \pi_i \) is the instruction of \( P \) with instruction number \( i \). We denote instruction numbers (= potential contents of the program counter) by \( c, c_1, \ldots \) and finite sequences of instruction numbers by \( \bar{c} \). \( \ell(\bar{c}) \) denotes the last instruction number of the sequence \( \bar{c} \), and \( [\bar{c}] \) the sequence obtained from \( \bar{c} \) by omitting its last member. Fix an instance \((x, y)\) of \( Q \). Let

\[
C := \bigcup_{0 \leq r \leq h(k)-1} \{1, \ldots, m\}^r \quad \text{and} \quad N := \{0, 1, \ldots, f(k) \cdot p(n)\}.
\]

with \( k = |y| \). We look for a structure \( A \) and a \( \Sigma_{t,2}.u \)-sentence \( \varphi \) such that

\[
P \text{ accepts } (x, y) \iff A \models \varphi.
\]

Let \( \bar{c}_0 \) be the sequence of instruction numbers of the deterministic part of \( P \) on input \((x, y)\) ending with the instruction number of the first existential instruction. As universe \( A \) of \( A \) we take \( A := C \cup N \). Moreover, in \( A \) there are the binary relation \( \leq_A \), the natural ordering on \( N \), and ternary relations \( R_A \) and \( T_A \) defined by

\[
R_A \bar{c}i\bar{j} \iff \bar{c} \in C, i, j \in N, \text{ and if } P, \text{ on input } (x, y), \text{ carries out the sequence of instructions } [\bar{c}_0 \bar{c}], \text{ then } r_i = j.
\]

\[
T_A \bar{c}i\bar{j} \iff \bar{c} \in C, i, j, (i, j) \in N, \text{ and if } P, \text{ on input } (x, y), \text{ carries out the sequence of instructions } [\bar{c}_0 \bar{c}], \text{ then } r_{(i,j)} = 0.
\]

Moreover, we have a constant for 0. This finishes the definition of \( A \), which can be constructed within the time allowed by an FPT-reduction.

We turn to the definition of \( \varphi \). First, we fix \( \bar{c} \in C \): Let \( i = i(\bar{c}) \) be the number of blocks without alternations of the sequence of instructions determined by \([\bar{c}_0 \bar{c}]\);
let $j = j(\bar{c})$ be the number of guesses in the $i$th block. If $i \leq t$ and if each block, besides the first one, has length $\leq u$, we introduce a formula

$$\varphi_{\bar{c}}(\bar{x}_1, \ldots, \bar{x}_{i-1}, x_{i,1} \ldots x_{i,j})$$

where $\bar{x}_1 := x_{1,1}, \ldots, x_{1,h(k)}$ and $\bar{x}_s := x_{s,1} \ldots x_{s,u}$ for $s = 2, \ldots, i - 1$ with the intuitive meaning

if a partial run of $\mathbb{P}$ has $\bar{c}_0 \bar{c}$ as sequence of instructions numbers and if every variable $x_{i',j'}$ displayed in $\Pi$ has, as value, the $j'$th guess of the $i'$th block (and $x_{i',j'} = 0$, if there was no such guess), then there is an accepting continuation of this run.

Then, for the empty sequence $\emptyset$ of instruction numbers and $\varphi := \varphi_{\emptyset}$, we have

$$\mathbb{P} \text{ accepts } (x, y) \iff A \models \varphi.$$  

For $\bar{c} \in C$ of maximal length, $|\bar{c}| = h(k) - 1$, we set (recall that, by definition, a computation accepts its input, if $r_0 = 0$ at the end of the computation)

$$\varphi_{\bar{c}} := \begin{cases} \text{TRUE, if } \pi_{\ell(\bar{c})} = \text{STOP (i.e., } \pi_{\ell(\bar{c})} \text{ is the STOP-instruction) and } R\bar{c}00 \\ \text{FALSE, otherwise.} \end{cases}$$

If $\bar{c} \in C$ and $|\bar{c}| < h(k) - 1$, we assume that $\varphi_{\bar{c}'}$ has already been defined for all $\bar{c}'$ with $|\bar{c}| < |\bar{c}'|$. The definition depends on the type of the instruction of $\mathbb{P}$ with instruction number $\ell(\bar{c})$.

If $\pi_{\ell(\bar{c})} = \text{STOP}$, then again

$$\varphi_{\bar{c}} := \begin{cases} \text{TRUE, if } R\bar{c}00 \\ \text{FALSE, otherwise.} \end{cases}$$

The definition of $\varphi_{\bar{c}}$ is simple for the standard instructions, e.g., if $\pi_{\ell(\bar{c})} = \text{STORE } \uparrow u$ (i.e., “$\pi_{\ell(\bar{c})} = r_{r_u} := r_0$”), then $\varphi_{\bar{c}} := \varphi_{\bar{c} \ell(\bar{c}) + 1}$.

We give the definitions for the new instructions: If $\pi_{\ell(\bar{c})} = \text{EXISTS } \uparrow v$, then (for $i = i(\bar{c})$ and $j = j(\bar{c})$)

$$\varphi_{\bar{c}} := \begin{cases} \exists x_{i,j+1} \exists y(R\bar{c}0y \land x_{i,j+1} \leq y \land \varphi_{\bar{c} \ell(\bar{c}) + 1}) \text{, if } i = 1 \text{ or } (i \text{ is odd and } j < u) \\ \exists x_{i+1,1} \exists y(R\bar{c}0y \land x_{i+1,1} \leq y \land \varphi_{\bar{c} \ell(\bar{c}) + 1} \\ \quad \quad \land x_{i,j+1} = 0 \land \ldots \land x_{i,u} = 0) \text{, if } i \text{ is even, } i < t, \text{ and } j \leq u \\ \text{FALSE, otherwise.} \end{cases}$$

The definition is similar for instructions of the type $\text{FORALL } \uparrow v$, but then the variables are quantified universally.

Assume $\pi_{\ell(\bar{c})} = \text{JG } v \ w \ c$. We need $g(r_v)$ and $g(r_w)$. Determine the actual contents $v_0$ and $w_0$ of the $v$th and the $w$th standard register, i.e., $v_0$ and $w_0$ with $R^A_{\bar{c} v}v_0$ and $R^A_{\bar{c} w}w_0$. Consider the sequence of instructions given by $\bar{c}$ and determine the last instructions in it of the form $\text{FORALL } \uparrow z$ or $\text{EXISTS } \uparrow z$ such
that at that time \( r_z = v_0 \), say, it is the \( j_0 \)th guess in the \( i_0 \)th block. Similarly, let the \( j_1 \)th guess in the \( i_1 \)th block be the last instruction of the form FORALL \( \uparrow z \) or EXISTS \( \uparrow z \) such that at that time \( r_z = w_0 \) (the case that such instructions do no exist is treated in the obvious way). Then set

\[
\varphi_{\vec{c}} := (x_{i_0,j_0} = x_{i_1,j_1} \rightarrow \varphi_{\vec{c}c}) \land (\neg x_{i_0,j_0} = x_{i_1,j_1} \rightarrow \varphi_{\vec{c}c} c+1).
\]

Assume \( \pi_{\ell(\vec{c})} = JG0 u v c \). As in the preceding case, let \( x_{i_0,j_0} \) and \( x_{i_1,j_1} \) denote the actual values of the \( r_u \)th and the \( r_v \)th guess register, respectively. Then set

\[
\varphi_{\vec{c}} := (T_{\vec{c}}x_{i_0,j_0}x_{i_1,j_1} \rightarrow \varphi_{\vec{c}c}) \land (\neg T_{\vec{c}}x_{i_0,j_0}x_{i_1,j_1} \rightarrow \varphi_{\vec{c}c} c+1).
\]

As already mentioned above, we set \( \varphi := \varphi_0 \). By Lemma 1, one easily verifies that \( \varphi \) is equivalent to a \( \Sigma_{t,2,2} \)-formula. Clearly, the size of \( \varphi \) can be bounded in terms of \( h(k) \) and

\[
Qxy \iff \mathbb{P} \text{ accepts } (x, y) \iff A \models \varphi,
\]

which gives the desired reduction showing that \( Q \in W[t] \).

4 W[2] and Multitape Machines

Among the many known W[1]-complete problems, of course, the most generic one is the halting problem p-HPN for nondeterministic Turing machines (cf. [1]):

**Proposition 2.** For some vocabulary \( \tau \), p-HPNMT \( \leq_{FPT} \) p-MC(STR, \( \Sigma_{2,3}[\tau] \)).

Here p-HPNMT denotes the halting problem for nondeterministic multitape Turing machines:

**p-HPNMT** Input: A nondeterministic Turing machine \( M \).

Parameter: \( k \in \mathbb{N} \).

Problem: Does \( M \) accept the empty word in at most \( k \) steps?
Proof. Let $M$ be a nondeterministic Turing machine with $w_0$ (work) tapes. We aim at a structure $A = A_{M,k}$ and a $\Sigma_{2,3}$-sentence $\varphi = \varphi_{M,k}$ such that

$$(M, k) \in \text{p-HPNMT} \iff A \models \varphi.$$ 

Let $\Sigma$ and $Q$ be the alphabet and the set of states of $M$, respectively. The instructions of $M$ have the form

$$q(a_1, \ldots, a_{w_0}) \rightarrow q'(a'_1, \ldots, a'_{w_0})(h_1, \ldots, h_{w_0})$$

where $q, q' \in Q$, $a_1, \ldots, a_{w_0}, a'_1, \ldots, a'_{w_0} \in \Sigma \cup \{\ast\}$ ($\ast$ is the blank symbol) and $h_1, \ldots, h_{w_0} \in \{-1, 0, 1\}$. Let $T$ be the set of tuples $(b_1, \ldots, b_{w_0}) \in (\Sigma \cup \{\ast\})^{w_0}$ and $H$ the set of tuples $(h_1, \ldots, h_{w_0}) \in \{-1, 0, 1\}^{w_0}$ occurring in instructions.

The structure $A$ has the universe

$$A := Q \cup (\Sigma \cup \{\ast\}) \cup \{-1, 0, 1, \ldots, \max\{w_0, k\}\} \cup T \cup H.$$ 

We need the natural ordering relation $\leq^A$ on $\{-1, 0, 1, \ldots, \max\{w_0, k\}\}$, the 5-ary relation $D^A$ (the “transition relation”) and the ternary relation $P^A$ (the “projection relation”) defined by

$$D^Aqtq't'h \iff qt \rightarrow q't'h$$ and $$P^A \text{wba} \iff 1 \leq w \leq w_0, b \in T \cup H, b = (b_1, \ldots, b_{w_0}), \text{ and } b_0 = a.$$ 

Moreover, we have constant symbols for the initial state $q_0$, the accepting state $q_{\text{acc}}$, for $\ast$, and for $-1, w_0, 0, 1, \ldots, k$ (cf. Remark $\dagger$).

The formula $\varphi$ we aim at, will express that there is an accepting run of length $\leq k$ (w.l.o.g. of length $= k$); among others, it will contain the variables $q_i, t_i, q'_i, t'_i, h_i$ for $i = 1, \ldots, k$, in fact, $q_i, t_i \rightarrow q'_i, t'_i, h_i$ represents the $i$th instruction applied in the run; $\varphi$ is obtained by existentially quantifying all variables in

$$(\varphi_{\text{init}}(q_1, t_1) \land \bigwedge_{i=1}^k Dq_i t_i q'_i t'_i h_i \land \bigwedge_{i=1}^{k-1} q'_i = q_{i+1} \land q_k = q_{\text{acc}} \land \psi),$$

where $\varphi_{\text{init}}(q_1, t_1) := (q_1 = q_0 \land \forall w(1 \leq w \leq w_0 \rightarrow P\text{wt}_1\ast))$ and where $\psi$ is a universal formula expressing that the sequence of instructions can be applied: For this purpose, for $i = 1, \ldots, k$, we introduce quantifier-free formulas

$$\varphi_i^L(w, p, x, \bar{v}_i) \text{ and } \varphi_i^S(w, p, \bar{v}_i)$$

with $\bar{v}_i := q_i, t_1, q'_1, t'_1, h_1, \ldots, q_{i-1}, t_{i-1}, q'_{i-1}, t'_{i-1}, h_{i-1}$ and with the meaning

if, starting with the empty tape, the sequence of instructions $\bar{v}_i$ has been carried out, then the $p$th cell of the $w$th tape contains the letter $x$, and

if, starting with the empty tape, the sequence of instructions $\bar{v}_i$ has been carried out, then the head of the $w$th tape scans the $p$th cell,
respectively. Then, as \( \psi \), we can take

\[
\psi := \forall w \forall p \forall x \bigwedge_{i=1}^{k} \left( (\varphi_i^L(w, p, x, \bar{v}_i) \land \varphi_i^S(w, p, \bar{v}_i)) \rightarrow Pwt_ix \right).
\]

The simultaneous definition of \( \varphi_i^L \) and \( \varphi_i^P \) by induction on \( i \) is routine, e.g.,

\[
\varphi_1^L(w, p, x) := (1 \leq w \leq w_0 \land 1 \leq p \leq k \land x = *)
\]

\[
\varphi_{i+1}^L(w, p, x, \bar{v}_{i+1}) := (1 \leq w \leq w_0 \land 1 \leq p \leq k) \land \\
((\neg \varphi_i^S(w, p, \bar{v}_i) \land \varphi_i^L(w, p, x, \bar{v}_i)) \lor (\varphi_i^S(w, p, \bar{v}_i) \land Pwt'_ix)).
\]

One easily verifies that \( \varphi \) is (logically equivalent to) a \( \Sigma_{2,3} \)-sentence.

\[\square\]

**Corollary 1.** The halting problem p-HPNMT for multitape machines is \( W[2] \)-complete.

**Proof.** By Proposition\[1\] and the preceding proposition, p-HPNMT is in \( W[2] \). To show that p-HPNMT is \( W[2] \)-hard we recall the proof of \[3\] that the parameterized dominating set problem p-DS can be reduced to p-HPNMT. The essential problem is to obtain a corresponding machine in the time allowed by an FPT-reduction.

Suppose \((G, k)\) is an instance of p-DS with \( G = \{a_1, \ldots, a_n\} \). Let the multi-tape machine \( M \) have \( n+1 \) tapes, numbered by 0 to \( n \), and let \( \Sigma := G \cup \{\text{yes}\} \) be its alphabet. In the first step all heads move one cell to the right. In the next \( 2 \cdot k \) steps, only the 0th head is active: it (nondeterministically) writes \( k \) elements of \( G \) on its tape, say \( b_1, \ldots, b_k \) (the elements of the intended dominating set), and goes back to the cell containing \( b_1 \). In the next \( k \) steps the 0th head again reads these elements; at the same time, in the \( j \)th step, the \( i \)th head checks whether \( a_i = b_j \) or \( Ea_i b_j \); in the affirmative case the \( i \)th head prints “yes” and moves to the right, in the negative case it neither prints nor moves; finally, the machine moves all heads one cell to the left and accepts, if the heads on the tapes number 1 to \( n \) read “yes”. Clearly,

\[
(G, k) \in \text{p-DS} \iff (M, 3 \cdot (k + 1)) \in \text{p-HPNMT}. \quad \square
\]

## 5 Complete Halting Problems for the W-Hierarchy

As already mentioned at the beginning of Section\[4\] the halting problem for nondeterministic Turing machines, parameterized by the number of steps, is a quite generic \( W[1] \)-complete problem. The corresponding halting problems for alternating machines yield complete problems for the classes of the A-hierarchy. Indeed, for \( t \geq 1 \), we have \( A[t] = [p-\text{HPA}_t]^{\text{FPT}} \) (cf. \[8\]), where

<table>
<thead>
<tr>
<th>p-\text{HPA}_t</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> An alternating Turing machine ( M ) whose initial state is existential.</td>
</tr>
<tr>
<td><strong>Parameter:</strong> ( k \in \mathbb{N} ).</td>
</tr>
<tr>
<td><strong>Problem:</strong> Does ( M ) accept the empty word in at most ( k ) steps with at most ( t - 1 ) alternations?</td>
</tr>
</tbody>
</table>
One would expect that in order to obtain complete problems for the classes of the W-hierarchy one has to bound the number of steps of all non-alternating blocks but the first one. More precisely, for \( t, u \geq 1 \), consider the following problem:

\[
p\text{-HPA}_{t,u} \quad \text{Input:} \text{ An alternating Turing machine } M \text{ whose initial state is existential.}
\]

\[
\text{Parameter:} \ k \in \mathbb{N}.
\]

\[
\text{Problem:} \text{ Does } M \text{ accept the empty word in at most } k \text{ steps with at most } t - 1 \text{ alternations, where every block of steps without alternation, besides the first one, has length } \leq u?
\]

Essentially along the lines of the proof of \( p\text{-HPA}_t \in \text{A}[t] \) in [8], one can show that \( p\text{-HPA}_{t,u} \in W[t] \). But the corresponding hardness proof does not seem to go through. Recall that in order to establish that \( p\text{-MC}(\text{GRAPH}, \Sigma_t) \leq_{\text{FPT}} p\text{-HPA}_t \), given a graph and a \( \Sigma_t \)-sentence \( \varphi \), one constructs an alternating Turing machine that first associates values to the quantifiers in \( \varphi \) (the values for existentially quantified variables are chosen in existential states, the values for universally quantified variables in universal states) and that then checks whether the selected variables satisfy the quantifier-free part of \( \varphi \), the quantifier-free check. Of course, a \( \Sigma_{t,u} \)-prefix would yield an alternation sequence according to \( p\text{-HPA}_{t,u} \), but the number of steps needed for the quantifier-free check cannot be bounded in terms of \( t \) and \( u \) as required in \( p\text{-HPA}_{t,u} \), it depends on \( \varphi \). Therefore, we add suitable oracles to the Turing machines that carry out the quantifier-free check in a single step. Of course, we have to add them in such a way that the corresponding halting problem still is in \( W[t] \).

As in an alternating machine, the set \( Q \) of states of an alternating Turing machine \( M \) with oracle is the disjoint union of the set of universal states \( Q_u \) and the set of existential states \( Q_e \). The “oracle states” \( q_? \), \( q_y \), and \( q_n \) are all contained in \( Q_u \) or all in \( Q_e \). Let \( \Sigma \) be the vocabulary of \( M \) and let \( O \subseteq \Sigma^* \) be a language, the “oracle language”. \( M^O \), the machine \( M \) with oracle \( O \), in state \( q? \) will check if the word to the left of the cell scanned by its head is in \( O \) and will change to the “yes state” \( q_y \) or to the “no state” \( q_n \) (without printing a letter nor moving its head).

For a graph \( G = (G, E^G) \) and a first-order formula \( \psi(x_1, \ldots, x_p) \), let \( O(G, \psi) \) be the following set of words over \( G \), \( O(G, \psi) \subseteq G^* \):

\[
O(G, \psi) := \{a_1 \ldots a_p \mid G \models \psi(a_1, \ldots, a_p)\}.
\]

Due to space limitations we omit the proof of the following theorem:

\textbf{Theorem 2.} For \( t \geq 1 \),

\[
W[t] = \{\{p\text{-HPAO}_{t,u} \mid u \geq 1\}\}^{\text{FPT}},
\]

where \( p\text{-HPAO}_{t,u} \) is the parameterized halting problem for alternating Turing machines with oracle:
p-HPAO\textsubscript{t,u} \textbf{Input:} An alternating Turing machine \(M\) with oracle whose initial state is existential and a graph \(G = (G, E^p)\) such that \(G\) is a subset of the alphabet of \(M\).

\textbf{Parameter:} \(k \in \mathbb{N}\) and a quantifier-free formula \(\psi(x_1, \ldots, x_p)\).

\textbf{Problem:} Does \(M^{O(G, \psi)}\) accept the empty word in at most \(k\) steps with at most \(t - 1\) alternations, where every block of steps without alternation, besides the first one, has length \(\leq u\)?

6 Conclusions

Most standard complexity classes like LOGSPACE, NLOGSPACE, PTIME, NPTIME, the classes of the polynomial hierarchy, or PSPACE have definitions in terms of machines. By contrast, originally nearly all parameterized complexity classes containing intractable problems were defined via complete problems. In [4], machine characterizations of \(W[1], W[P]\), and of \(A[t]\) for \(t \geq 1\) were presented; in this paper we derive such characterizations for \(W[t]\) for \(t \geq 2\).

As mentioned at the beginning of Section 3 in [4] AW-programs for alternating RAMs were introduced, which have “unrestricted access” to the guessed numbers. For \(t \geq 1\), denote by \(L[t]\) the class that satisfies Theorem [11] if we replace W-RAM by alternating RAM, i.e., \(L[t]\) is the class of parameterized problems \(Q\) such that there is a computable function \(h\) and an AW-program \(P\) for an alternating RAM deciding \(Q\) such that for every run of \(P\) on an instance \((x, y)\) of \(Q\) as input (with \(k = |y|\))

– all existential and universal steps are among the last \(h(k)\) steps of the computation,
– there are at most \(t - 1\) alternations between existential and universal states, and the first guess step is existential,
– every block without alternations, besides the first one, contains at most \(u\) guess steps.

Clearly, by [4] and Theorem [11] we have

\[ W[t] \subseteq L[t] \subseteq A[t]. \]

We do not know, if \(W[t] = L[t]\) or if \(L[t] = A[t]\). Let an \(f\)-\textit{vocabulary} be a finite set of relation symbols, function symbols, and constant symbols. By an appropriate refinement of the proof of Theorem [11] one can show:

\textbf{Theorem 3.} For \(t \geq 1\),

\[ L[t] = \left\{ \{p-MC(STR, \Sigma_{t,u}[\tau]) \mid u \geq 1, \tau f\text{-vocabulary}\} \right\}^{\text{FPT}}. \]

For every \(f\)-\textit{vocabulary} \(\tau\), the classical problem \(MC(STR, \Sigma_{t,u}[\tau])\) is in NP. Therefore, \(\text{FPT} \subseteq W[t] \subseteq L[t] \subseteq \text{para-NP}\) (compare [9] for the definition of para-NP). As \(\text{FPT} \neq \text{para-NP}\) is equivalent to \(P \neq NP\) (cf. [9]), we obtain from the preceding theorem:
Corollary 2. If $W[t] \neq L[t]$ for some $t \geq 1$, then $P \neq NP$.

References

Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions

Yannick Chevalier\textsuperscript{1}, Ralf Küsters\textsuperscript{2}, Michaël Rusinowitch\textsuperscript{1}, Mathieu Turuani\textsuperscript{1}, and Laurent Vigneron\textsuperscript{1}

\textsuperscript{1} LORIA-INRIA-Universités Henri Poincaré, Nancy 2
54506 Vandoeuvre-les-Nancy cedex, France
\{chevalie,rusi,turuani,vigneron\}@loria.fr
\textsuperscript{2} Department of Computer Science
Stanford University, Stanford CA 94305, USA
kuesters@theory.stanford.edu

Abstract. We propose a protocol model which integrates two different ways of analyzing cryptographic protocols: i) analysis w.r.t. an unbounded number of sessions and bounded message size, and ii) analysis w.r.t. an a priori bounded number of sessions but with messages of unbounded size. We show that in this model secrecy is DEXPTIME-complete. This result is obtained by extending the Dolev-Yao intruder to simulate unbounded number of sessions.

1 Introduction

Formal analysis has been very successful in finding flaws in published cryptographic protocols \cite{7}. Even fully automatic analysis of such protocols is possible, based on models for which security is decidable or based on approximations (see, e.g., \cite{3,19,18,11,16}, and \cite{17,9} for an overview of the different approaches, decidability, and complexity theoretic results).

The decidability of security, or more precisely secrecy, of protocols heavily depends on whether in the analysis an unbounded number of sessions of a protocol is taken into account or only an a priori bounded number. In the former case, secrecy is in general undecidable \cite{11,13,14}, with only a few exceptions \cite{13,11,2,8}. One such exception, which is of particular interest in this paper, is that secrecy is DEXPTIME-complete when the message size is bounded and nonces, i.e., newly generated constants, are disallowed \cite{13}. In what follows, let us call this setting the \textit{bounded message model}. In the latter case, in which the number of sessions is bounded, secrecy is known to be NP-complete \cite{19}, even when there is no bound on the size of messages, complex keys are allowed, i.e., keys that may be complex messages, and messages can be paired to form larger messages. We will refer to this setting as the \textit{unbounded message model}.

\* This work was partially supported by PROCOPE and IST AVISPA. The second author was also supported by the DFG.
In this paper, we integrate the two models — the bounded and the unbounded message model, and thus, integrate two different approaches for protocol analysis: i) analysis w.r.t. an unbounded number of sessions, which has the advantage that the exact sessions to be analyzed do not need to be provided beforehand, but where a bound on the size of messages is put, and ii) analysis which is rather detailed since the size of messages is not bounded, but where only explicitly given sessions are analyzed. More precisely, we consider a protocol model in which there are two kinds of principals, \textit{bounded message} and \textit{unbounded message} principals, or \textit{bounded} and \textit{unbounded} principals for short, which only accept messages of bounded size from the environment or messages of unbounded size, respectively. Conversely, in a protocol run, bounded principals may be involved in an unbounded number of sessions while unbounded principals run in at most one session. The communication between the principals is controlled by the standard Dolev-Yao intruder, in particular, the size of the messages the intruder may produce is not bounded. Just as in the bounded and unbounded message model, the principals and the intruder are not allowed to generate nonces. Our model, in what follows referred to as \textit{integrated model}, comprises both the bounded and the unbounded message model: If in the integrated model the set of bounded principals is empty, then the model coincides with the unbounded message model, and if the set of unbounded principals is empty, then this gives the bounded message model.

The main result shown in this paper is that secrecy in the integrated model is \textsc{DEXPTIME}-complete. The main difficulty is to establish the complexity upper bound. The key idea is as follows: To deal with the bounded principals in the integrated model, and thus, the unbounded number of sessions, the bounded principals are turned into intruder rules, and thus, they extend the ability of the intruder to derive new messages. These intruder rules can be applied by the intruder an arbitrary number of times and in this way simulate the unbounded number of sessions. More precisely, we will extend the standard Dolev-Yao intruder by oracle rules, i.e., intruder rules that satisfy certain properties, and show that insecurity w.r.t. a set of \textit{unbounded} principals and the extended intruder is in \textsc{NP} (given an oracle for applying the oracle rules). — This result is obtained in a similar way as the one in [5], although the kind of oracle rules considered in [5] is quite different from the rules studied here. — We then turn the bounded principals into oracle rules, show that these rules in fact simulate the bounded principals, and prove that the rules can be applied in exponential time. These steps are non-trivial. From this, we conclude the desired complexity upper bound, i.e., obtain a deterministic exponential time algorithm for deciding secrecy in the integrated model.

As we will see in Section 3 the integrated model is not more powerful than the unbounded message model in the sense that from every protocol in the integrated model one can construct a protocol in the unbounded message model such that one protocol preserves secrecy only if the other one does. Moreover, feeding this constructed protocol into an algorithm for analyzing protocols in the unbounded message model, yields an alternative way of deciding secrecy in
the integrated model. However, since the number of (unbounded) principals in
the constructed protocol grows exponentially, using the NP-completeness result
shown in [19], this reduction only provides an NEXPTIME decision algorithm.
(Note that, together with the main result of this paper and the result shown in
[19], the existence of a polynomial time reduction from the integrated model to
the unbounded message model would imply NP=EXPTIME.) More importantly,
the constructed protocol is too big for current analysis tools in the unbounded
messages model, e.g., [3,18], since they can only handle a small number of prin-
cipals. Conversely, in our decision algorithm, we not only reduce secrecy in the
integrated model to secrecy in the unbounded message model but in addition ex-
tend the Dolev-Yao intruder to simulate the bounded principals. In this way, we
avoid creating new (unbounded) principals. In addition to the improved com-
plexity theoretic result, this approach seems to be much better amenable to
practical implementations. In fact, in [6] an implementation is presented for an
intruder with capabilities similar to those needed here.

Structure of the paper. In the following section, the protocol and intruder model
is presented. Then we state the main result (Section 3). In Section 4 the intruder
extended by oracle rules is introduced and it is shown that insecurity is in NP
given an oracle for applying oracle rules. We then, Section 5 turn bounded
principals into oracle rules, and by applying the result from Section 4 establish
the complexity upper bound. We conclude in Section 6.

We refer the reader to our technical report [4] for full proofs and a formal
description of the Three-Pass Mutual Authentication ISO Protocol in our model.

2 Problem Definition

We now provide a formal definition of our protocol and intruder model. We first
define terms and messages, then protocols, and finally the intruder and attacks.

2.1 Terms and Messages

Terms are defined according to the following grammar:

\[
\text{term ::= } A | V | \langle \text{term}, \text{term} \rangle | \{\text{term}\}_s | \{\text{term}\}_p
\]

where \( A \) is a finite set of constants (atomic messages), containing principal
names, nonces, keys, and the atomic messages secret and \( I \) (the intruder’s name);
\( K \) is a subset of \( A \) denoting the set of public and private keys; and \( V \) is a finite set
of variables. We assume that there is a bijection \( -1 \) on \( K \) which maps every public
(private) key \( k \) to its corresponding private (public) key \( k^{-1} \). The binary symbol
\( \langle \cdot, \cdot \rangle \) is called pairing, the binary symbol \( \{\cdot\}_s \) is called symmetric encryption, the
binary symbol \( \{\cdot\}_p \) is public key encryption. Note that a symmetric key can be
any term and that for public key encryption only atomic keys (namely, public
and private keys from \( K \)) can be used.
Variables are denoted by $x, y$, terms are denoted by $s, t, u, v$, and finite sets of terms are written $E, F, \ldots$, and decorations thereof, respectively. 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 cardinality of a set $S$ is denoted by $\text{card}(S)$.

For a term $t$ and a set of terms $E$, $\text{Var}(t)$ and $\text{Var}(E)$ denote the set of variables occurring in $t$ and $E$, respectively.

A ground term (also called message) is a term without variables. A (ground) substitution is a mapping from $\mathcal{V}$ to the set of (ground) terms. The application of a substitution $\sigma$ to a term $t$ (a set of terms $E$) is written $t\sigma$ ($E\sigma$), and is defined as usual.

The set of subterms of a term $t$, denoted by $\text{Sub}(t)$, is defined as follows:

- If $t \in A \cup \mathcal{V}$, then $\text{Sub}(t) = \{t\}$.
- If $t = \langle u, v \rangle$, $\{u\}^s$, or $\{u\}^p$, then $\text{Sub}(t) = \{t\} \cup \text{Sub}(u) \cup \text{Sub}(v)$.

Let $\text{Sub}(E) = \bigcup_{t \in E} \text{Sub}(t)$. We define the size of a term and a set of terms basically as the size of the representation as a dag. That is, the size $|t|$ ($|E|$) of a term $t$ (a set of terms $E$) is $\text{card}(\text{Sub}(t))$ ($\text{card}(\text{Sub}(E))$).

### 2.2 Protocols

We now define principals and protocols.

**Definition 1.** A principal $\Pi$ is a finite linear ordering of rules of the form $R \rightarrow S$ where $R$ and $S$ are terms. We assume that every variable in $S$ occurs in $R$ or on the left-hand side of a rule preceding $R \rightarrow S$. The rules are called principal rules.

A protocol $P$ is a tuple $(F_u, F_b, E_I, D)$ where $F_u$ and $F_b$ are finite unions of principals, and thus, partially ordered sets, $E_I$ is a finite set of messages with $I \in E_I$, and $D$ is some representation of a finite set of messages such that the dag size of messages in the set represented by $D$ is linearly bounded in the size of the representation of $D$.

Given a protocol $P$, in the following we will assume that $A$ is the set of constants occurring in $P$. We define the size $|P|$ of $P$ as the number of different subterms in $F_u$, $F_b$, and $E_I$ plus the size of the representation of $D$. For instance, $D$ may be a non-negative integer $n$ (encoded in unary) representing the set of all messages of dag size $\leq n$. This implies that the dag size of the set of messages represented by $D$ is exponentially bounded in the size $|P|$ of the protocol. We define $\text{Var}(P)$ to be the set of variables occurring in $P$.

The idea behind the definition of a protocol is as follows. In an attack on $P$ the intruder may use every principal in $F_u$ at most once but the principals in $F_b$ may be used as often as the intruder wishes. In other words, the principals in $F_b$ may be involved in an unbounded number of sessions in one attack while the principals in $F_u$ only participate in at most one session. It is well-known that deciding the security of a protocol w.r.t. an intruder who may use an unbounded number of sessions and may produce messages of unbounded size is undecidable.
For this reason, we will restrict the messages that can be substituted for variables of rules in $F_b$ to belong to the finite domain $D$. However, we put no restrictions on the variables of rules in $F_u$, i.e., these variables can be substituted by messages of unbounded size. We therefore refer to principals in $F_u$ as unbounded and to those in $F_b$ as bounded. A rule of an unbounded principal is called unbounded and analogously a rule of a bounded principal is bounded. In the following section, attacks are defined formally and the relationship to other models is further discussed. As mentioned, our technical report [4] contains a formal description of a protocol in our protocol model.

2.3 The Intruder and Attacks

Our intruder model follows the Dolev-Yao intruder [12]. That is, the intruder has complete control over the network and he can derive new messages from his initial knowledge and the messages received from honest principals during protocol runs. To derive a new message, the intruder can compose and decompose, encrypt and decrypt messages, in case he knows the key. What distinguishes our model from most other models in which security is decidable is that the intruder may use the (bounded) principals as often as he wishes to perform his attack. As mentioned in the introduction, to deal with this, in Section 4 we will extend the intruder by so-called oracle rules.

The intruder derives new messages from a given (finite) set of messages by applying rewrite rules. A rewrite rule (or t-rule) $L$ is of the form $M \rightarrow t$ where $M$ is a finite set of messages and $t$ is a message. Given a finite set $E$ of messages, the rule $L$ can be applied to $E$ if $M \subseteq E$. We define the step relation $\rightarrow_L$ induced by $L$ as a binary relation on finite sets of messages. For every finite set of messages $E: E \rightarrow_L E, t$ (recall that $E, t$ stands for $E \cup \{t\}$) if $L$ is a t-rule and $L$ can be applied to $E$. If $L$ denotes a (finite or infinite) set of intruder rules, then $\rightarrow_L$ denotes the union $\bigcup_{L \in L} \rightarrow_L$ of the step relations $\rightarrow_L$ with $L \in L$. With $\rightarrow_L^*$ we denote the reflexive and transitive closure of $\rightarrow_L$.

The set of rewrite rules the intruder can use is listed in Table 1. These rules are called (Dolev-Yao) intruder rules. In the table, $a, b$ denote (arbitrary) messages, $K$ is an element of $K$, and $E$ is a finite set of messages.

The intruder rules are denoted as shown in Table 1. We consider $L_{p_1}(\langle a, b \rangle)$, $\ldots, L_{sd}(\{a\}_b^s)$ and $L_c(\langle a, b \rangle), \ldots, L_c(\{a\}_b^s)$ as singletons. Note that the number of decomposition and composition rules is always infinite since there are infinitely many messages $a, b$.

We further group the intruder rules as follows. In the following, $t$ ranges over all messages.

- $L_d(t) := L_{p_1}(t) \cup L_{p_2}(t) \cup L_{qd}(t) \cup L_{sd}(t)$. In case, for instance, $L_{p_1}(t)$ is not defined, i.e., the head symbol of $t$ is not a pair, then $L_{p_1}(t) = \emptyset$; analogously for the other rule sets,
- $L_d := \bigcup_t L_d(t)$, $L_c := \bigcup_t L_c(t)$,
- $L_{DY} := L_d \cup L_c$ (where $DY$ stands for “Dolev and Yao”).
Table 1. Intruder Rules

<table>
<thead>
<tr>
<th></th>
<th>Decomposition rules</th>
<th>Composition rules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pair</strong></td>
<td>$L_{p1}((a, b))$: $(a, b) \rightarrow a$</td>
<td>$L_{c}((a, b))$: $a, b \rightarrow (a, b)$</td>
</tr>
<tr>
<td></td>
<td>$L_{p2}((a, b))$: $(a, b) \rightarrow b$</td>
<td></td>
</tr>
<tr>
<td><strong>Asymmetric</strong></td>
<td>$L_{ad}(({a}^r_K)_K)$: ${a}^r_K \rightarrow a$</td>
<td>$L_{c}(({a}^r_K)_K)$: $a, K \rightarrow {a}^r_K$</td>
</tr>
<tr>
<td><strong>Symmetric</strong></td>
<td>$L_{sd}(({a}^r_b)_b)$: ${a}^r_b \rightarrow a$</td>
<td>$L_{c}(({a}^r_b)_b)$: $a, b \rightarrow {a}^r_b$</td>
</tr>
</tbody>
</table>

The set of messages the intruder can derive from a (finite) set $E$ of messages is:

$$d_{DY}(E) := \bigcup \{E' \mid E \rightarrow^*_{L_{DY}} E' \}.$$ 

Before we can define attacks on a protocol $P = (F_u, F_b, E_I, D)$, we need some new notions.

Given a partially ordered set $F$ of principal rules with associated ordering $<$, an execution ordering $\pi$ for $F$ is a bijective mapping from some subset $F'$ of $F$ into $\{1, \ldots, \text{card}(F')\}$ such that $L < L'$ implies $\pi(L) < \pi(L')$ for every $L, L' \in F'$. The size of $\pi$ is $\text{card}(F')$.

The partially ordered set of instantiations of the bounded principals in $P$ is $F_b^D := \{\Pi\sigma' \mid \Pi \in F_b \text{ and } \sigma' : \text{Var}((\Pi)) \rightarrow D\}$. The partially ordered set induced by $P$ is $F_P := F_u \cup F_b^D$.

We are now prepared to define attacks. In an attack, a principal $\Pi$ performs his sequence (linear ordering) of principal rules $R_1 \rightarrow S_1, \ldots, R_n \rightarrow S_n$ one after the other. Note that the different rules may share variables which are substituted by the same message and in this way model the (unbounded) memory of a principal. When in step $i$ a message $m$ is received, then $m$ is matched against $R_i$ yielding a matcher $\sigma$ (if any) with $R_i\sigma = m$ and $\Pi$ returns $S_i\sigma$ as output. Variables in $R_i$ and $S_i$ which occurred in a previous step, and thus, have been assigned a message already, are substituted by this message. As mentioned in Section 2.2, the intruder may use an unbounded principal, i.e., a principal in $F_u$, at most once, and he may use every bounded principal, i.e., a principal in $F_b$, as often has he wishes (any time with a possibly different matching). The difference between unbounded and bounded principals is as follows: While an unbounded principal accepts every message as long as it matches the current input pattern $R_i$, a bounded principal expects that the variables are filled with elements of the domain represented by $D$. Thus, $F_b^D$ is the set of instances of bounded principals the intruder may use to perform an attack. Note that subsequent use of an instance after the first time does not yield new knowledge. Therefore, we assume w.l.o.g. that bounded principal instances in $F_b^D$ are used only once. Note, however, that $F_b^D$ contains different (an exponential number of) instances of one bounded principal. Altogether, the intruder may use every principal in $F_P$ once. For a subset of these principals he (nondeterministically) chooses some execution ordering and then tries to produce input messages for the principal rules. These input messages are derived from the intruder’s initial knowledge and the output messages produced by executing the principal rules.
The aim of the intruder is to derive the message secret. Formally, attacks are defined as follows.

**Definition 2.** Let $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, \mathcal{D})$ be a protocol and let $\mathcal{F}_P$ be the partially ordered set induced by $P$. An $\mathcal{L}_{DY}$-attack (or simply attack) on $P$ is a tuple $(\pi, \sigma)$ where $\pi$ is an execution ordering on $\mathcal{F}_P$, of size $k$, and $\sigma$ is a ground substitution of the variables occurring in $P$ such that

$$R_i \sigma \in d_{DY}(S_0, S_1 \sigma, ..., S_{i-1} \sigma)$$

for every $i \in \{1, \ldots, k\}$ where $R_i \rightarrow S_i = \pi^{-1}(i)$, and

$$\text{secret} \in d_{DY}(S_0, S_1 \sigma, ..., S_k \sigma).$$

The decision problem we are interested in is the following set of protocols where we assume the terms occurring in a protocol to be given as dags.

**INSECURE := \{P \mid \text{there exists an } \mathcal{L}_{DY} \text{-attack on } P\}.$$

If we restrict the set $\mathcal{F}_b$ of bounded principals to be the empty set (and in this case we do not need $\mathcal{D}$), then this is the case of protocol analysis w.r.t. a bounded number of sessions and unbounded message size as considered, for instance, in [15,19,18], and called unbounded message model in the introduction. On the other hand, if we restrict $\mathcal{F}_u$ to be an empty set, then this is basically the case of protocol analysis w.r.t. an unbounded number of sessions but with bounded message size as studied in [13], and called bounded message model in the introduction. We note, however, that in contrast to [13], here we allow the intruder to derive messages of arbitrary size, only the size of messages accepted by the (bounded) principals is bounded. Also, we allow complex rather than only atomic keys.

Summing up, with the protocol and the intruder model considered here, we integrate the bounded and the unbounded message models.

### 3 Main Result

The main result of this paper is:

**Theorem 1.** The problem $\text{INSECURE}$ is $\text{DEXPTIME}$-complete.

In [13], it has been shown that deciding secrecy in the bounded model, i.e., an unbounded number of sessions but bounded messages, is $\text{DEXPTIME}$-complete. Since here we extend this setting, it is not surprising that for $\text{INSECURE}$ we also obtain $\text{DEXPTIME}$-hardness. In fact, one can use the same reduction, namely a reduction from the recognition problem for Datalog programs [10], as in [13].

In [19], it has been shown that deciding $\text{INSECURE}$ for protocols $P$ without bounded principals (i.e., $\mathcal{F}_b = \emptyset$) is $\text{NP}$-complete. We can use this result to also obtain an upper bound for $\text{INSECURE}$ in the general case: Let $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, \mathcal{D})$. Observe that $P \in \text{INSECURE}$ iff $P' = (\mathcal{F}_u \cup \mathcal{F}_b^D, \emptyset, E_I, \emptyset) \in$
Insecure. The protocol $P'$ can be handled with the algorithm proposed in [19]. However, since $P'$ may be of size exponential in the size of $P$ this only shows that Insecure is in NEXPTIME. Thus, the main problem in proving Theorem 1 is to establish the tight upper bound.

The main idea of this proof is as follows: We first extend capabilities of the Dolev-Yao intruder by so-called oracle rules, i.e., intruder rules which satisfy certain conditions. For this extended intruder we show that insecurity for protocols without bounded principals is in NP given an oracle for performing oracle rules (Theorem 2). We then turn the set $F_b^D$ of instantiated bounded principals into intruder rules and show that these rules are in fact oracle rules. This will yield the claimed exponential time upper bound (Section 5).

In the following section oracle rules are introduced and the NP-decision algorithm is presented.

4 A General Framework

We now extend the Dolev-Yao intruder by oracle rules, which are intruder rules satisfying certain conditions, and show that insecurity in presence of such an extended intruder for protocols without bounded principals is in NP given a procedure for applying oracle rules. We first introduce oracle rules and then present the NP algorithm.

4.1 Extending the Dolev-Yao Intruder by Oracle Rules

In the rest of this paper, let $L_o$ denote a (finite or infinite) set of rewrite rules of the form $M \rightarrow t$ where $M$ is a finite set of messages and $t$ is a message. In Definition 4, we will impose restrictions on this set and then call it the set of oracle rules. The subset of $L_o$ consisting of $t$-rules is denoted by $L_o(t)$. The union of the Dolev-Yao intruder rules and the oracle rules is denoted by $L_{DYO}$ and called oracle intruder rules. Define $L_c := L_c \cup L_o$ to be the set of composition rules, $L_c(t) := L_c(t) \cup L_o(t)$, and $L_d(t)$ to be the set of all decomposition $t$-rules in Table 1.

The set $d_{DYO}(E)$ of messages the intruder can derive from $E$ using the rules $L_{DYO}$ is defined analogously to $d_{DY}(E)$. Also, $L_{DYO}$-attacks are defined analogously to $L_{DY}$-attacks.

Given finite sets of messages $E, E'$, an ($L_{DYO}$-)derivation $D$ of length $n$, $n \geq 0$ from $E$ to $E'$ is a sequence of steps of the form $E \rightarrow L_1 E, t_1 \rightarrow L_2 \cdots \rightarrow L_n E, t_1, \ldots, t_n$ with messages $t_1, \ldots, t_n$, $E' = E \cup \{t_1, \ldots, t_n\}$, and $L_i \in L_{DYO}$ such that $E, t_1, \ldots, t_{i-1} \rightarrow L_i E, t_1, \ldots, t_i$ and $t_i \not\in E \cup \{t_1, \ldots, t_{i-1}\}$, for every $i \in \{1, \ldots, n\}$. The rule $L_i$ is called the $i$th rule in $D$ and the step $E, t_1, \ldots, t_{i-1} \rightarrow L_i E, t_1, \ldots, t_i$ is called the $i$th step in $D$. We write $L \in D$ to say that $L \in \{L_1, \ldots, L_n\}$. If $L'$ is a set of rewrite rules, then we write $L' \not\in D$ to say $L' \cap \{L_1, \ldots, L_n\} = \emptyset$. The message $t_n$ is called the goal of $D$.

We also need well formed derivations which are derivations where every message generated by an oracle intruder rule is a subterm of the goal or a subterm of a term in the initial set of messages.
Definition 3. Let \( D = E \rightarrow_{L_1} \ldots \rightarrow_{L_n} E' \) be a derivation with goal \( t \). Then, \( D \) is well formed if for every \( L \in D \) and every \( t' \): \( L \in \mathcal{L}_c(t') \) implies \( t' \in \text{Sub}(E, t) \), and \( L \in \mathcal{L}_d(t') \) implies \( t' \in \text{Sub}(E) \).

We can now define oracle rules. Condition 1. in the following definition requires the existence of well formed derivations. This will allow us to bound the length of derivations and the size of messages needed in derivations. The remaining conditions are later used to bound the size of the substitution \( \sigma \) of an attack.

Definition 4. Let \( L_o \) be a (finite or infinite) set of rules and \( P \) be a protocol. Then, \( L_o \) is a set of oracle rules (w.r.t. \( L_c \cup L_d \) as defined above) iff there exists a polynomial \( p(\cdot) \) such that:

1. For every message \( t \), if \( t \in d_{DYO}(E) \), then there exists a well formed derivation from \( E \) with goal \( t \).
2. If \( F \rightarrow_{L_o(t)} F, t \) and \( F, t \rightarrow_{L_o(t)} F, t, a \), then there exists a derivation \( D \) from \( F \) with goal \( a \) such that \( L_d(t) \notin D \).
3. For every rule \( F \rightarrow t \in L_o(t) \) we have \( |t| \leq p(|P|) \) and for all \( t' \in F \), \( |t'| \leq p(|P|) \).

In what follows, we always assume that \( L_o \) is a set of oracle rules. We call a protocol \( P \) of the form \((F_u, \emptyset, E_I, \emptyset)\) restricted. We want to decide the insecurity of a restricted protocol w.r.t. an intruder using \( \mathcal{L}_{DYO} \), i.e., the Dolev-Yao intruder rules plus the oracle rules. Formally, the decision problem we are interested in is the following set of restricted protocols \( P \):

\[
\text{Insecure}_O := \{ P \mid \text{there exists an} \ \mathcal{L}_{DYO}-\text{attack on the restricted protocol} \ \ P \}
\]

4.2 An NP Decision Algorithm

The following theorem is used to prove Theorem 1.

Theorem 2. Let \( L_o \) be a set of oracle rules. Given a procedure (an oracle) for deciding \( E \rightarrow_{L_o} t \) for every finite set \( E \) of messages and message \( t \) in constant time, \( \text{Insecure}_O \) can be decided by a nondeterministic polynomial time algorithm.

The NP decision procedure is given in Figure 1. In (1) and (2) of the procedure, an attack \((\pi, \sigma)\) is guessed of size polynomially bounded in \( n \). Then, it is checked whether this is in fact an attack.

Obviously, the procedure is sound. As for completeness, one needs to show that it suffices to only consider substitutions bounded as done in the procedure. This is proved in Section 4.3 Theorem 3.

To show that the procedure is in fact an NP procedure given a procedure for deciding \( E \rightarrow t \in L_o \), we prove that (3) and (4) can be decided by an NP algorithm. Given that \( |R_i\sigma, S_0\sigma, \ldots, S_{i-1}\sigma| \) is polynomially bounded in \( |P| \) for every \( i \leq k \) (see Corollary 1), it suffices to show that the following problem belongs to NP (given the decision procedure for \( E \rightarrow t \in L_o \)):

\[
\text{Derive} := \{ (E, t) \mid \text{there exists an} \ \mathcal{L}_{DYO}-\text{derivation from} \ E \ \text{with goal} \ t \}.
\]
Input: restricted protocol $P = (\mathcal{F}_u, \emptyset, S_0, \emptyset)$ with $n = p(|P|)$, where $p(\cdot)$ is the polynomial associated to the oracle rules, and $V = \text{Var}(P)$.

1. Guess an execution ordering $\pi$ for $P$. Let $k$ be the size of $\pi$. Let $R_i \rightarrow S_i = \pi^{-1}(i)$ for $i \in \{1, \ldots, k\}$.
2. Guess a normalized ground substitution $\sigma$ such that $|\sigma(x)| \leq 3n^2$ for all $x \in V$.
3. Test that $R_i \sigma \in d_{DYO}(\{S_0 \sigma, \ldots, S_{i-1} \sigma\})$ for every $i < k$.
4. Test that $\text{secret} \in d_{DYO}(\{S_0 \sigma, \ldots, S_k \sigma\})$.
5. If each test is successful, then answer “yes”, and otherwise, “no”.

Fig. 1. NP Decision Procedure for Insecurity

In this problem, $E$ and $t$ are assumed to be represented as dags. The following lemma follows quite easily from the existence of well formed derivations (see [4] for the proof).

**Lemma 1.** Given a procedure for deciding $E \rightarrow^2 t \in L_\alpha$, DERIVE can be decided in nondeterministic polynomial time.

From Theorem [8] proved in the following section, completeness of the procedure depicted in Figure [1] follows.

**4.3 Polynomial Bounds on Attacks**

To show completeness of the NP decision algorithm depicted in Figure [1] we need to prove that it suffices to consider substitutions bounded as in the second step of this algorithm. To this end, we consider an attack of minimal size, a so-called normal attack, and show that the size of this attack can be bounded as stated in the algorithm.

Given an attack $(\pi, \sigma)$ on a protocol $P$ define $|\sigma| := \Sigma_{x \in \text{Var}(P)}|\sigma(x)|$. We say that the attack $(\pi, \sigma)$ is normal if $|\sigma|$ is minimal, i.e., for every attack $(\pi', \sigma')$, $|\sigma| \leq |\sigma'|$. Clearly, if there is an attack, there is a normal attack. Note also that a normal attack is not necessarily uniquely determined.

The next lemma says that normal attacks can always be constructed by linking subterms that are initially occurring in the problem specification or by terms bounded by $p(|P|)$. This will allow us to bound the size of attacks as desired (Theorem [8] and Corollary [1]). To state the lemma, we need some notation.

Let $P$, $R_i$, $S_i$, $(\pi, \sigma)$, $V$, $p(\cdot)$, and $k$ be defined as in Figure [1]. Let $SP = \text{Sub}(\{R_j | j \in \{1, \ldots, k\}\} \cup \{S_j | j \in \{0, \ldots, k\}\})$. We recall that $A \subseteq SP$.

**Definition 5.** Let $t$ and $t'$ be two terms and $\theta$ a ground substitution. Then, $t$ is a $\theta$-match of $t'$, denoted $t \sqsubseteq_\theta t'$, if $t$ is not a variable, and $t \theta = t'$.

In [4], we prove:

**Lemma 2.** Given a normal attack $(\pi, \sigma)$, for all variables $x$: $|\sigma(x)| \leq p(|P|)$ or there exists $t \in SP$ such that $t \sqsubseteq_\sigma \sigma(x)$.
Using this lemma, it is now easy to bound the size of every $\sigma(x)$ (see [4] for the proof):

**Theorem 3.** For every protocol $P$, if $(\pi, \sigma)$ is a normal attack on $P$, then $|\{\sigma(x) | x \in \text{Var}\}| \leq 3 \cdot p(|P|^2)$, where $|P|$ is the size of $P$ as defined in Section 2.2 and $p(\cdot)$ is the polynomial associated to the set of oracle rules.

From this, we easily obtain:

**Corollary 1.** For every protocol $P$ and normal attack $(\pi, \sigma)$ on $P$: $|R_i \sigma, S_0 \sigma, \ldots, S_{i-1} \sigma|$ and $|\text{Secret}, S_0 \sigma, \ldots, S_k \sigma|$ can be bounded by a polynomial in $|P|$ for every $i \in \{1, \ldots, k\}$.

## 5 Proof of the Complexity Upper Bound

We now show the complexity upper bound claimed in Theorem 1. In what follows, let $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, \mathcal{D})$ be a protocol.

The idea of the proof is to turn the partially ordered set $\mathcal{F}_b^D$ of instantiated bounded principals into oracle rules and then use Theorem 2.

The conversion of $\mathcal{F}_b^D$ is carried out in two steps. First, this set is turned into a set of so-called aggregated rules. Then, the rules are turned into oracle rules.

### 5.1 Aggregated Rules

The set $\mathcal{F}_b^D$ consists of a finite set of (instantiated) principals $\Pi$. Assume that the linear ordering associated to $\Pi$ is $<$, $\Pi = \{R_0 \rightarrow S_0, \ldots, R_{n-1} \rightarrow S_{n-1}\}$, and $R_i \rightarrow S_i < R_j \rightarrow S_j$ for every $i < j$.

Now, replace every $R_i \rightarrow S_i$ in $\Pi$ by a rewrite rule $\{R_0, \ldots, R_i\} \rightarrow S_i$. We denote the resulting set by $\Pi_{agg}$ and call this set the aggregated version of $\Pi$. Let $\mathcal{F}_{agg}$ denote the set obtained from $\mathcal{F}_b^D$ by replacing every principal by its aggregated version. We call this set the set of aggregated rules induced by $P$. Define $\mathcal{L}_{agg} := \mathcal{L}_{DY} \cup \mathcal{F}_{agg}$, the set of aggregated intruder rules (induced by $P$). Note that $\mathcal{L}_{agg}$ depends on $P$. However, for simplicity, we omit $P$ in the notation of this set.

The set of terms the intruder can derive from $E$ using $\mathcal{L}_{agg}$ is defined as:

$$d_{agg}(E) := \bigcup \{E' | E \rightarrow^{*}_{\mathcal{L}_{agg}} E'\}.$$

An $\mathcal{L}_{agg}$-attack on $P$ is defined analogously to $\mathcal{L}_{DY}$-attacks.

The following lemma states that there is an $\mathcal{L}_{DY}$-attack on $P$ iff there exists an $\mathcal{L}_{agg}$-attack on $P$ when the bounded principals of $P$ are removed. In other words, the bounded principals are moved to the intruder. The proof of this lemma is straightforward.

**Lemma 3.** There exists an $\mathcal{L}_{DY}$-attack on $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, \mathcal{D})$ iff there exists an $\mathcal{L}_{agg}$-attack on $(\mathcal{F}_u, \emptyset, E_I, \emptyset)$. 

Extending the Dolev-Yao Intruder 139

From this and if \( \mathcal{F}_{agg} \) were oracle rules (in the sense of Definition 4), \( \text{INSECURE} \in \text{DEXPTIME} \) would immediately follow from Theorem 2. In general the set \( \mathcal{F}_{agg} \) does not meet the restrictions on oracle rules. Therefore, we define principal oracle rules meeting the restrictions on oracle rules. In what follows, they are formally defined and it is shown that whether \( E \rightarrow t \) is such a rule can be decided in exponential time. Then, we show that these rules can replace aggregated rules and that they are oracle rules. Together with Theorem 2, this will yield Theorem 1.

5.2 Principal Oracle Rules

Let \( \text{Sub}_r(\mathcal{F}_{agg}) \) denote the set of subterms occurring on the right hand-side of rewrite rules in \( \mathcal{F}_{agg} \).

**Definition 6.** A principal oracle rule induced by a protocol \( P \) is a rewrite rule of the form \( E \rightarrow t \) where \( E \) is some finite set of ground terms with \( |u| \leq |P|^2 \) for every \( u \in E \) and \( t \in \text{Sub}_r(\mathcal{F}_{agg}) \) such that \( t \in d_{agg}(E) \). Let \( \mathcal{F}_p \) denote the set of principal oracle rules induced by \( P \).

Note that in the above definition \( |t| \leq |P|^2 \) and \( \mathcal{F}_{agg} \subseteq \mathcal{F}_p \).

We now show that principal oracle rules can be decided in exponential time.

**Proposition 1.** For every \( E \) and \( t \), it can be decided in exponential time in the dag size of \( E \) and \( P \) whether \( E \rightarrow t \in \mathcal{F}_p \).

The key to the proof of this proposition is the following lemma, which is proved in [4]. Intuitively, it states that \( \mathcal{L}_{agg} \)-derivations are well-formed in the sense that the messages produced in each step of the derivation are subterms of a certain set of messages. Note that \( \mathcal{L}_{agg} \)-derivations are derivations, as defined in Section 4.1, which use only intruder rules from \( \mathcal{L}_{agg} \). In what follows, let \( \mathcal{H} \) denote the set of subterms of \( \mathcal{F}_{agg} \).

**Lemma 4.** Assume that \( E \rightarrow t \in \mathcal{F}_p \). Let \( D \) denote a derivation from \( E \) with goal \( t \) over \( \mathcal{L}_{agg} \) of minimal length. Then, \( u \in \text{Sub}(E,t,\mathcal{H}) \) for every message \( u \) such that there exists a \( u \)-rule in \( D \).

Now to test whether \( E \rightarrow t \in \mathcal{F}_p \) one can iteratively apply rules in \( \mathcal{L}_{agg} \) to \( E \) that create subterms of \( E, t, \mathcal{H} \). Let \( E' \) be the resulting set of terms. Then, Lemma 4 ensures that \( t \in E' \) iff \( E \rightarrow t \in \mathcal{F}_p \). It is easy to see that \( E' \) can be computed in time exponential in the size of \( P \). This completes the proof of Proposition 1.

5.3 Principal Oracle Rules can Replace Aggregated Rules

Let \( \mathcal{L}_p := \mathcal{L}_{DY} \cup \mathcal{F}_p \) be the set of principal intruder rules. The set of terms the intruder can derive from \( E \) using \( \mathcal{L}_p \) is defined as \( d_p(E) := \bigcup \{ E' \mid E \rightarrow^* \mathcal{L}_p \ E' \} \). An \( \mathcal{L}_p \)-attack on \( P \) is defined analogously to \( \mathcal{L}_{DY} \)-attacks.

Obviously, \( d_{agg}(E) = d_p(E) \) for every finite set \( E \) of messages. As an immediate consequence, we obtain:
Lemma 5. Let $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, D)$ be a protocol and let $\mathcal{L}_{agg}$ and $\mathcal{L}_p$ be the aggregated and principal intruder rules induced by $P$. Then, there exists an $\mathcal{L}_{agg}$-attack on $(\mathcal{F}_u, \emptyset, E_I, \emptyset)$ iff there exists an $\mathcal{L}_p$-attack on this protocol.

Together with Lemma 3 this yields:

Proposition 2. Let $P = (\mathcal{F}_u, \mathcal{F}_b, E_I, D)$ be a protocol and let $\mathcal{L}_p$ be the principal intruder rules induced by $P$. Then, there exists an $\mathcal{L}_{DY}$-attack on $P$ iff there exists an $\mathcal{L}_p$-attack on $(\mathcal{F}_u, \emptyset, E_I, \emptyset)$.

5.4 Principal Oracle Rules Are Oracle Rules

In what follows, we identify $L_o$ with $\mathcal{F}_p$, and show that $L_o$ is a set of oracle rules. By definition of $\mathcal{F}_p$, the last condition on oracle rules (Definition 4.3.) is met with $p(n) = n^2$.

The following lemma shows the second condition in the definition of oracle rules.

Lemma 6. If $F \rightarrow_{L_o(t)} F, t$ and $F, t \rightarrow_{L_d(t)} F, t, a$, then there exists a derivation $D$ from $F$ with goal $a$ such that $L_d(t) \notin D$.

Proof. The proof is obvious. It suffices to observe that $a \in d_p(F) \cap \text{Sub}_r(\mathcal{F}_{agg})$, and thus, $F \rightarrow a \in \mathcal{F}_p$. ⊓⊔

The next lemma, shown in [4], states that if a derivation exists, then also a well formed derivation.

Lemma 7. If $t \in d_p(E)$, then there exists a well formed derivation with goal $t$.

The two lemmas imply:

Proposition 3. The set $L_o$ of principal oracle rules is a set of oracle rules.

Now, together with Theorem 2 and Proposition 1 this shows the complexity upper bound claimed in Theorem 1.

6 Conclusion

We have proposed a protocol model which integrates what we have called the unbounded and the bounded message models, and we have shown that deciding secrecy in our model is EXPTIME-complete. For this purpose we have extended the Dolev-Yao intruder in a general framework by oracle rules and applied this framework to handle an unbounded number of sessions.

In future work, we will investigate in how far this framework can be applied to yield other interesting extensions of the Dolev-Yao intruder. Another question is whether the oracle rules introduced here can be combined with those considered in [5], with the potential of even more powerful intruders, e.g., those combining unbounded number of sessions with the XOR operator.
References

On Relativisation and Complexity Gap for Resolution-Based Proof Systems

Stefan Dantchev\textsuperscript{1} and Søren Riis\textsuperscript{2}

\textsuperscript{1} Dept. of Mathematics and Computer Science, University of Leicester
University Road, Leicester, LE1 7RH, UK
dantchev@mcs.le.ac.uk

\textsuperscript{2} Dept. of Computer Science, Queen Mary, University of London
Mile End Road, London E1 4NS, UK
smriis@dcs.qmul.ac.uk

Abstract. We study the proof complexity of $\text{Rel}(\text{Taut})$, the class of Second-Order Existential (SO\textsubscript{\exists}) logical sentences which fail in all finite models. The Complexity-Gap theorem for Tree-like Resolution says that the shortest Tree-like Resolution refutation of any such sentence $\Phi$ is either fully exponential, $2^{\Omega(n)}$, or polynomial, $n^{O(1)}$, where $n$ is the size of the finite model. Moreover, there is a very simple model-theoretical criteria which separates the two cases: the exponential lower bound holds if and only if $\Phi$ holds in some infinite model.

In the present paper we prove several generalisations and extensions of the Complexity-Gap theorem.

1. For a natural subclass of $\text{Taut}$, $\text{Rel}(\text{Taut})$, there is a gap between polynomial Tree-like Resolution proofs and sub-exponential, $2^{\Omega(n^{\varepsilon})}$, general (DAG-like) Resolution proofs, whilst the separating model-theoretic criteria is the same as before. $\text{Rel}(\text{Taut})$ is the set of all sentences in $\text{Taut}$, relativised with respect to a unary predicate.

2. The gap for stronger systems, $\text{Res}^+(k)$, is between polynomial and $\exp \left( \Omega \left( \frac{k^n}{k^k} \right) \right)$ for every $k$, $1 \leq k \leq n$. $\text{Res}^+(k)$ is an extension of Tree-like Resolution, in which literals are replaced by terms (i.e. conjunctions of literals) of size at most $k$. The lower bound is tight.

3. There is (as expected) no gap for any propositional proof system (including Tree-like Resolution) if we enrich the language of SO logic by a built-in order.

1 Introduction

In $[1]$ a new kind of results for propositional logic was introduced. Expressed somewhat informally, it was shown that any sequence $\psi_n$ of tautologies which expresses the validity of a fixed combinatorial principle either is “easy” i.e. has polynomial size tree-resolution proofs or is “difficult” i.e requires full exponential size tree-resolution proofs. It was shown that the class of tautologies which are hard (for tree-resolution) is identical to the class of tautologies which are based on combinatorial principles which are violated for infinite sets.
According to this result the proof complexity of a combinatorial principle never have intermediate growth rates like for example $2^\log^k(n)$, $k = 2, 3, \ldots$. In this paper we extend this result to a number of related resolution-based systems.

A central question in the theory of proof complexity concerns to the amount of resources (usually proof length) which is needed to prove a certain sequence of tautologies. Usually, the sequence of tautologies consists of Tautologies which are similar except for their size. The paper is organised as follows:

Firstly, we consider the resolution proof system in a setting of DAG-like proofs rather than tree-like structure. In a DAG-like proof a once derived disjunction can be used any number of times later in the proof. This cannot happen in the tree like cases. Thus a given tautology might have a DAG-like proof which is substantially shorter than the shortest tree-like proof [2].

The class of combinatorial problems which are hard for DAG-like resolution differs from the class of combinatorial problems which are hard for tree like resolution. ”Minimal element” is a principle separates the two systems [3].

In this paper we show the class of combinatorial problems which are hard for DAG resolution is identical to the class which is hard for tree resolution provided the combinatorial principles are being relativised. This answers an open question by Krajíček [4].

Secondly, we consider $\text{Res}(k)$ which is similar but stronger than resolution. In $\text{Res}(k)$ clauses i.e. disjunctions, are replaced by disjunctions of conjunctions of $\leq k$ literals. The rules are strengthened so one can resolve not just a single variable (like in resolution), but also conjunctions of up to $k$ variables. An easy extension of [1] gives a complexity jump from polynomial to $\text{exp}(\Omega(n^{\log k}))$ for these problems (when proofs are represented as trees). This lower bound is also implicit in [4] by Krajíček. We improve this, and show that the jump is from polynomial to $\text{exp}(\Omega(n^{\log^k(n)}))$. Moreover, we show that the lower bound is tight.

Finally, we show that there is no complexity gap if we enrich the language of SO logic by a built-in order. Even though this result is expected it has a less obvious consequence. It allows us to answer an open question from [5] by showing that there is no complexity gap for tree resolution above $2^n \log(n)$. We expect that there is a complexity gap above $2^n$, but are not sure if the gap jumps the whole way up to $2^n \log(n)$.

2 Preliminaries

2.1 Resolution with Bounded Conjunction, $\text{Res}(k)$

In this section we recall some of the basic concepts related to resolution proofs.

A literal is a propositional variable or the negation of a propositional variable. A $k$-conjunction $\bigwedge_j l_j$ is a conjunction of at most $k$ literals. A $k$-DNF is a disjunction of $k$-conjunctions.

The $\text{Res}(k)$ proof system is a refutation system designed to provide certificates (i.e. proofs) that a system of $k$-DNF’s is unsatisfiable. This is done by means of the following four derivation rules. The $\bigwedge$-introduction rule is
\[
\frac{C_1 \lor \bigwedge_{j \in J_1} l_j}{C_1 \lor C_2 \lor \bigwedge_{j \in J_1 \cup J_2} l_j},
\]
provided that \(|J_1 \cup J_2| \leq k\). The cut (or resolution) rule is
\[
\frac{C_1 \lor \bigvee_{j \in J} l_j \quad C_2 \lor \bigwedge_{j \in J} \neg l_j}{C_1 \lor C_2}.
\]

The two weakening rules are
\[
\frac{C}{C \lor \bigwedge_{j \in J} l_j},
\]
provided that \(|J| \leq k\), and
\[
\frac{C \lor \bigwedge_{j \in J_1 \cup J_2} l_j}{C \lor \bigwedge_{j \in J_1} l_j}.
\]

Here \(C\)'s are \(k\)-DNFs, and \(l\)'s are literals.

The given DNF's are often referred to as axioms, and the task is to derive the empty clause (the contradiction) from the axioms. In Tree-like Res\((k)\), denoted \(\text{Res}^* (k)\), the proof is organised as a binary tree with the axioms in the leaves and the empty clause in the root. In DAG-like Res\((k)\) denoted just Res\((k)\), the proof is given as a linear sequence \(C_1, C_2, \ldots, C_u\) of clauses, where each clause either is an axiom or can be obtained by means of the resolution rule (applied to two already derived clauses). In a Resolution proof, clauses can be reused more the once. A tree-like proof do not allow this.

### 2.2 Proving Lower Bounds for Resolution and Tree-Like Res\((k)\)

We will first describe the search problem, associated to an inconsistent set of clauses as defined in \([6]\): Given a truth assignment, find a clause, falsified under the assignment.

We can use a refutation of the set of clauses to solve the search problem as follows. We first turn around all the edges of the graph of the proof. The contradiction now becomes the only root (source) of the new graph, and the axioms and the initial formulae become the leaves (sinks). We perform a search in the new graph, starting from the root, which is falsified by any assignment, and always going to a vertex which is falsified under the given assignment. Such a vertex always exists as the inference rules are sound. We end up at a leaf, which is one of the initial clauses.

Thus, if we want to prove the existence of a particular kind of clauses in any proof, we can use an adversary argument against the refutation, solving the search problem as described above. The argument is particularly nice for Resolution (Res\((1)\)) as developed by Pudlak in \([7]\). There are two players, named Prover and Adversary. An unsatisfiable set of clauses is given. Adversary claims wrongly that there is a satisfying assignment. Prover holds a Resolution refutation, and uses it to solve the search problem. A position in the game is a partial assignment of the propositional variables. The positions can be viewed as conjunctions of
literals. All the possible positions in the game are exactly negations of all the clauses in the Prover’s refutation. The game start from the empty position (which corresponds to $\top$, the negation of the empty clause). Prover has two kind of moves:

1. She queries a variable, whose value is unknown in the current position. Adversary answers, and the position then is extended with the answer.
2. She forgets a value of a variable, which is known. The current position is then reduced, i.e., the variable value becomes unknown.

The game is over, when the current partial assignment falsifies one of the clauses. Prover then wins, having shown a contradiction.

We will be interested in deterministic Adversary’s strategies which allows to prove the existence of certain kind of clauses in a Resolution refutation.

In order to prove lower bounds on the size of a Resolution proof, we will use the known technique, “bottleneck counting”. It was introduced by Haken in his seminal paper [8] (for the modern treatment see [9]). We first define the concept of big clause. We then design random restrictions, so that they “kill” (i.e. evaluate to $\top$) any big clause with high probability (whp). By the union bound principle, if there are few big clauses, there is a restriction which kills them all. We now consider the restricted set of clauses, and using Prover-Adversary game, show that there has to be at least one big clause in the restricted proof, which is a contradiction and completes the argument.

The case of Tree-like proofs, either Resolution or Res($d$), is much simpler as a tree-like proof of a given set of clause is equivalent to a decision tree, solving the search problem [6]. We can use pretty straightforward adversary argument against a decision tree, in order to show that it has to have many nodes.

### 2.3 Relativising Combinatorial Principles

Let $\Psi$ be a $\forall\exists$ logical sentence. Informally, the statement $\Psi$ states that some property holds for the whole universe. Informally, the relativised sentence $\text{Rel}(\Psi)$ say that the principle $\Psi$ holds for any non-empty subset if the universe. The relativised principle not only state the validity of the principle for models of size $n$, but also implies that the principles holds for all models of size $\leq n$. Thus the relativised principle is in general harder to prove than its non-relativised counter part.

We will briefly describe the translation of a $\forall\exists$ sentence into a set of clauses. Assume first that there is a single relation symbol $F$ which is quantified existentially, and the FO part of the sentence is in a prenex normal form, i.e.

$$\forall x_1 \exists y_1 \ldots \forall x_m \exists y_m F(x_1, y_1, \ldots x_m, y_m).$$

We first introduce Skolem relations $s$,

$$\bigvee_{k=1}^{n} s_{j_1, \ldots j_i, k}^i \quad \text{for all } 1 \leq i \leq m \text{ and all } j_1, \ldots j_i \in [n],$$
and then the sentence translates into
\[ \bigwedge_{i=1}^{m} s_{j_1,j_2,...,j_i,k_i}^i \rightarrow f_{j_1,k_1,...,j_m,k_m} \text{ for all } j_1, k_1, \ldots j_m, k_m \in [n]. \]

The case when the quantifier-free part of the \( \Psi, F \) is not an atomic formula is as easy as this one. We just rewrite \( F \) in CNF, and then the clause above becomes a set of clauses. The number of these clauses is a constant, i.e. independent from the size of the universe, \( n \).

Let us now consider the relativisation, \( \text{Rel}(\Psi) \). It is
\[
\forall x_1 \in R \exists y_1 \in R \ldots \forall x_m \in R \exists y_m \in R F(x_1,y_1,\ldots x_m,y_m),
\]
and can be rewritten as
\[
\forall x_1 (R(x_1) \rightarrow \exists y_1 (R(y_1) \land \ldots \forall x_m (R(x_m) \rightarrow \exists y_m (R(y_m) \land F(x_1,\ldots y_m))))).
\]

It is not hard to see that the latter formula translates into the following sets of clauses:
\[
\bigwedge_{i=1}^{l} r_{j_i} \land s_{j_1,j_2,...,j_i,k_i}^i \rightarrow r_k \text{ for all } 1 \leq l \leq m \text{ and all } j_1, \ldots j_l, k \in [n]
\]
and
\[
\bigwedge_{i=1}^{m} r_{j_i} \land \bigwedge_{i=1}^{m} s_{j_1,j_2,...,j_i,k_i}^i \rightarrow f_{j_1,k_1,...,j_m,k_m} \text{ for all } j_1, k_1, \ldots j_m, k_m \in [n].
\]

Finally we add the clause saying that the unary predicate, we relativise with respect to, should not define the empty set, i.e.
\[
\bigvee_{i=1}^{n} r_i.
\]

3 Complexity Gap for \( \text{Rel}(\text{Taut}) \)

Let us first introduce some conventions. Recall that \( \varphi_n \) is built upon two kinds of variables, \( r \)'s and \( s \)'s. \( r \)-variables correspond to the relation symbols in the original sentence \( \Phi \). Suppose there are \( m \) such variables, \( r^1, r^2, \ldots, r^m \), having arities \( p_1, p_2, \ldots p_m \), and let us denote \( p = \max_i p_i \). Given the \( r \)-variable \( r_{j_1,j_2,...,j_{p_i}}^i \), we say it mentions the elements \( j_1, j_2, \ldots j_{p_i} \). \( s \)-variables correspond to the Skolem relations we use to encode \( \Phi \) as a set of clauses. Suppose there are \( l \) such variables \( s^1, s^2, \ldots s^l \). Given the \( s \)-variable \( s_{j_1,j_2,...,j_{q_i},j_0}^i \), we say it mentions only its arguments \( j_1, j_2, \ldots j_{q_i} \), but not the witness, \( j_0 \). We also define its arity to be \( q_i \), not \( q_i + 1 \). Let us denote \( q = \max_i q_i \) as well as \( t = \max \{p, q\} \).

We will first prove that we need big clauses to refute \( \Phi \).
Lemma 1. Any Resolution refutation of $\varphi_n$ contains a clause which mentions at least $n^{1/q}/(2l^{1/q}) - t$ elements.

Proof. We will describe a deterministic Adversary's strategy which enforces a big clause. As usual Adversary holds an infinite model $M$ which satisfies $\Phi$. We say that an element is busy in the current position (i.e. clause) iff it is mentioned by the clause. We say an element is hidden iff it is not busy, but is the witness of a Skolem relation, having all its arguments busy. An element, which is neither busy nor hidden, is said to be free. At each stage in the Prover-Adversary game Adversary maintains two disjoint sets $B$ and $H$. $B$ is the set of all the busy elements, and $H$ is the set of the hidden elements.

The Adversary's strategy is now clear. At any stage in the game all the elements from the disjoint union $B \uplus H$ have interpretations in $M$. Initially $B = H = \emptyset$. There are two kinds of Prover's moves:

1. She queries a new propositional variable. The easier case is when the variable can be evaluated under the current interpretation. Adversary replies with the value, and does not change $B$ and $H$. If the variable cannot be evaluated, Adversary needs to enlarge $B$ with all the new elements mentioned by the variable. Note that there is a constant number of such elements, namely at most $t$. $H$ then has to be enlarged as well with the witnesses of all the new tuples in $B$.

2. She forgets a propositional variable. Some elements from $B$ may then become non-busy. Adversary removes these from $B$. Some of them may become hidden, i.e. they are witnesses of a Skolem relation with all its arguments from $B$, and go to $H$. The rest become free. Some elements from $H$, namely the witnesses of the tuples which contained at least one of the just forgotten elements, become free as well.

In any case no contradiction can be achieved as far as $|B| + |H| \leq n$. On the other hand there are $l$ Skolem relation, each with arity at most $q$, so at any time $|H| \leq l |B|^q$. Thus, without loss of generality, we can assume that at any stage in the game, before a contradiction is achieved, we have

$$|B| + l |B|^q \leq n.$$ 

Since $|B| = n^{1/q}/(2l^{1/q})$ satisfies the inequality, and $|B|$ increases by at most $t$ after any stage, there should be a point where $|B| > n^{1/q}/(2l^{1/q}) - t$ as claimed. \hfill $\square$

Let us now consider $\text{Rel} (\varphi_n)$. We will first describe a distribution of random restrictions which kills any big clause in any Resolution refutation of $\text{Rel} (\varphi_n)$ with high probability (whp). The idea is to randomly divide the universe $U = [n]$ into two approximately equal parts. One of them, $R$, will represent the predicate, we relativise by, $\hat{R}^0$; all the variables within it will remain unset. The rest, $C$, will be the “chaotic” part; all the variables within $C$ and most of the variables between $C$ and $R$ will be set entirely at random.
More precisely, the random restrictions are as follows.

1. We first set all the variables $r_j^0$ to either $\top$ or $\bot$ independently at random with equal probabilities, $1/2$. Let us denote the set of variables with $r_j^0 = \top$ by $R$, and the set of variables with $r_j^0 = \bot$ by $C$, $C = U \setminus R$.

2. We now set all the variables $r_{j_1, j_2, \ldots, j_{p_i}}$, $i > 0$, which mention at least one element of $C$, i.e. $\{j_1, j_2, \ldots, j_{p_i}\} \cap C \neq \emptyset$, to either $\top$ or $\bot$ independently at random with equal probabilities, $1/2$.

3. We set all the variables $s_{j_1, j_2, \ldots, j_{q_i}, j_0}$, which mention only elements from $R$, but the witness is in $C$, i.e. $\{j_1, j_2, \ldots, j_{q_i}\} \subseteq R$ and $j_0 \in C$.

It is very important to note that the variables and clauses, which survive the random restriction, define exactly $\Phi$ on $R$, i.e. $\varphi|_R$.

There are few minor problems. The third case of the above description may violate an axiom as well as the first case may make $R$ very small. We can however see that these bad events happen with exponentially small probability.

**Lemma 2.** The probability that the random restrictions are inconsistent with the axioms or $|R| \leq n/4$ is at most $\ln q / 2^n + 1/e^{n/16}$.

**Proof.** Indeed, an axiom is violated iff in the third case there is a $q_i$-tuple $j_1, j_2, \ldots, j_{q_i}$ such that for every $j_0$, $s_{j_1, j_2, \ldots, j_{q_i}, j_0} = \bot$, i.e. there is no witness for the tuple. The probability for this is $1/2^n$ and there are at most $\ln q$ such tuples, so that the union-bound gives $\ln q / 2^n$. By the Chernoff bound the probability that $|R| \leq n/4$ is at most $e^{-n/16}$. \qed

The next step is to show that the random restrictions kill any clause with exponential probability in the number of the elements mentioned.

**Lemma 3.** Given a clause, which mentions at least $k$ elements, the probability it does not evaluate to $\top$ under the random restrictions is at most $(3/4)^{k/t}$.

**Proof.** Let us denote the clause by $A$, and perform the following experiment. We pick up a literal $l_1$ from $A$. The probability that at least one of the elements, mentioned by $l_1$, is in the chaotic set $C$ is at least $1/2$. Given such an element, the probability that $l_1$ evaluates to $\top$ under the random restrictions is $1/2$. Thus the probability $l_1$ does not evaluate to $\top$ is at most $3/4$. We now take all the elements, mentioned by $l_1$ and mark them.

We pick another literal $l_2$ from $A$ which mentions at least one unmarked element, and proceed as we have done with $l_1$. We then pick yet another literal $l_3$ and so on.

The clause $A$ mentions at least $k$ elements, whilst after having considered a literal we mark at most $t$ elements, so that we can repeat the above procedure at least $k/t$ times. These trials have been independent by the construction of the random restrictions. Therefore the probability that $A$ does not evaluate to $\top$ is at most $(3/4)^{k/t}$ as claimed. \qed
We can now prove the main result of the section.

**Theorem 1.** A $SO^3$ sentence $\Phi$ is given which fails in all finite models, but holds in some infinite model. Let us denote $\text{Rel}(\Phi)$ the relativisation of $\Phi$ with respect to a unary predicate. Let $\text{Rel}(\varphi_n)$ be the translation of the latter into set of clauses, assuming a finite model of size $n$. Then there is a constant $\varepsilon$, depending only on $\Phi$, such that any Resolution refutation of $\text{Rel}(\varphi_n)$ is of size $\exp(\Omega(n^\varepsilon))$.

**Proof.** Let us denote $k = \frac{n^{1/\rho}}{2(4l)^{1/\rho}} - t$, and say a clause is big, iff it mentions at least $k$ elements. We will prove that any Resolution refutation of $\text{Rel}(\varphi_n)$ contains exponentially many in $k$ big clauses, which would give the desired lower bound.

Assume, for the sake of contradiction, there is a refutation $\Gamma$ which contains at most $(4/3)^{k/2}$ big clauses. We hit $\Gamma$ by random restrictions. By the lemma 2 and the lemma 3 + union-bound, the probability that the restrictions are “bad” is at most

$$\frac{\ln q}{2^n} + \frac{1}{e^{n/16}} + \left(\frac{3}{4}\right)^{n^{1/\rho}/(2(4l)^{1/\rho}) - t}.$$ 

As this quantity is smaller than 1 for big enough $n$ (recall that $l$, $p$, $q$ and $t$ depend on $\Phi$, but not on $n$), there is a set of good restrictions, “good” meaning that they kill all the big clauses, and moreover what survives is $\varphi_m$ for some $m > n/4$. But now by the lemma 4 there has to be at least one big clause in the restricted refutation which is a contradiction. $\Box$

Note that we do not claim that the bound proven, $\exp(\Omega(n^{1/\rho}))$ is tight. As a matter of fact, we believe the right lower bound is $\exp(\Omega(n))$, but we also believe this might be very hard to prove, say as hard as proving the Complexity Gap theorem for (DAG-like) Resolution.

---

**4 Complexity Gap for Res $^*$ (k)**

**Theorem 2.** A $SO^3$ sentence $\Phi$ is given which fails in all finite models, but holds in some infinite model. Let us denote by $\varphi_n$ the translation of $\Phi$ into propositional logic, assuming a finite model of size $n$. Then for any $k$, $2 \leq k \leq n$, any $R^*(k)$ refutation of $\varphi_n$ is of size $\exp\left(\Omega\left(\frac{\log k}{k} n\right)\right)$.

**Proof.** We will describe the modifications of the original proof, Section 4 of [1]. Recall that the proof goes as follows. Prover holds the Tree-like Resolution refutation which is equivalent to a decision tree solving the search problem for $\varphi_n$. Adversary’s task to force a big subtree in the Prover’s tree. In doing so, he uses an infinite model $M$ in which $\Phi$ holds. At any stage in the game some elements of the universe are interpreted in $M$, and it is clear that no contradiction can be achieved by Prover, unless she forces Adversary to interpret all the $n$ elements. When a variable is queried by Prover, Adversary answer as follows.
1. If the truth value can be derived from the current interpretation, i.e. the partial assignment, he replies with the value. The current interpretation is not extended as this was a forced question.

2. If the value cannot be derived from the current interpretation, it follows that both $\top$ and $\bot$ answers are consistent with some extension of it. Thus Adversary is free to choose an answer, and in both possible cases there is a consistent extension of the current interpretation by at most $r$ new elements, where $r$ is a constant, the maximal arity of relation symbols in $\varphi_n$.

This shows that Prover’s decision tree has to contain a complete binary subtree of height $n/r$ which implies a $2^{n/r}$ lower bound.

Let us now consider the case of $Res^*(k)$ instead of Tree-like Resolution. A Prover’s query is a $k$-disjunction instead of a single variable. Adversary first simplifies the query, using the current interpretation. That is, if a literal evaluates to $\bot$ it vanishes; if all the literals vanish, the query itself is forced, and the answer is $\bot$. If a literal evaluates to $\top$ so does the entire disjunction; the query is forced, and the answer is $\top$.

The non-trivial case is when the query is not forced, i.e. after having been simplified, it can still be answered both $\top$ and $\bot$. For the positive answer it is enough to force a single literal to $\top$, and therefore to interpret at most $r$ new elements. For the negative answer Adversary should force all the literals to $\bot$, and therefore he has to interpret all the mentioned elements which are at most $kr$.

We will show that a subtree rooted at a given node can be lower-bounded by a function $S$ in the number of free elements at the node. If the number of free elements at the current node is $u$, the $\top$ successor has at least $u - r$ such elements whilst the $\bot$ successor has at most $u - kr$. Thus we have

$$S(u) \geq S(u - r) + S(u - kr) + 1.$$ 

We will prove that $S(u) \geq x_k^{u/r} - 1$ where $x_k$ is the biggest positive real root of the equation

$$x^k - x^{k-1} - 1 = 0.$$ 

The induction step is trivial. By the induction hypothesis we get

$$S(u) \geq (x_k^{(u/r) - 1}) + (x_k^{(u/r) - k} - 1) + 1 = x_k^{u/r} - 1.$$ 

Let us now observe that there are positive constants $a$ and $b$ such that for every $k \geq 1$, $x_k$ is in the interval $(1 + a \frac{\ln k}{k}, 1 + b \frac{\ln k}{k})$. Indeed, let us denote $f(x) = x^k - x^{k-1} - 1$. We have $f(1 + c \frac{\ln k}{k}) = (1 + c \frac{\ln k}{k})^{k-1} c \frac{\ln k}{k} - 1$ where $c$ is a constant. Since $\lim_{k \to \infty} (1 + c \frac{\ln k}{k})^{k-1} k^{-c} = 1$ we can conclude that for every constant $\varepsilon > 0$ there is $k_\varepsilon$ so that for every $k \geq k_\varepsilon$, $f(1 + (1 - \varepsilon) \frac{\ln k}{k}) < 0$ and $f(1 + (1 + \varepsilon) \frac{\ln k}{k}) > 0$. 


It is now clear how to get the desired lower bound. At the root of every decision tree we have all the $n$ elements free, so that the decision tree has to be of size at least $(1 + a \ln \frac{k}{n})^{n/r} - 1$ which is $\sim e^{\frac{a \ln k}{n/r}}$ and therefore $\exp \left( \Omega \left( \frac{\log k}{k} n \right) \right)$.

What remains to be checked is the basis case, $n < kr$. In this case the size of the tree is at least $n/r$ (as this is the minimal number of queries required to force interpretation of all the elements), whilst the lower bound expression is $\leq e^{a \ln k} = k^a$. Clearly $n/r > k^a$ for big enough $n$ as $k \leq n$ and $r, a, a < 1$ are constants independent from $n$. This completes the proof. ⊓⊔

It is important to note that the lower bound we have proven is tight. The SO $\exists$ sentence which shows this is Minimal Element Principle ($\text{MEP}_n$), saying that a finite (partially) ordered $n$-element set has a minimal element. Its negation is

$$\exists P \quad ((\forall x \neg P(x, x))$$

$$\land (\forall x, y, z \quad (P(x, y) \land P(y, z)) \rightarrow P(x, z))$$

$$\land (\forall x \exists y P(y, x))).$$

It is not hard to see that there is a $\text{Res}^*(k)$ refutation of $\text{MEP}_n$ of size $\exp \left( \Omega \left( \frac{\log k}{k} n \right) \right)$ for any $k, 2 \leq k \leq n - 1$. Note also that the $\text{Res}^*(n - 1)$ proof of $\text{MEP}_n$ is essentially the same as the (DAG-like) Resolution proof of the principle, so that our result is consistent with the known fact that $\text{MEP}_n$ is hard for Tree-like Resolution, but easy for Resolution.

5 Built-in Order “Kills” the Gap

We will first show that there is no Complexity gap for Tree-like Resolution if we enrich the SO$\exists$-language with a built-in order predicate.

**Theorem 3.** There is no tree-resolution complexity gap for the logical sentences in the language SO$\exists$ + built-in order.

**Proof.** Let us first describe the argument very informally. Assume a finite model of size $n$. There are know tautologies which requires size $2^{\Omega(n)}$ to refute in Tree-like Resolution. The most natural of them is Minimal Element Principle (MEP) which has already been mentioned (note that the partial order, defined by MEP, is entirely independent from the built-in total order predicate).

As we have built-in order, we can interpret the elements of the universe as the first $n$ natural numbers. We can define in the SO$\exists$ language a broad class of functions, such as $\log x$, $x^{p/q}$ ($p$ and $q$ integers) and so on. We will show that it is easy, i.e. polynomially size doable, to verify that a given element $k$ of the universe is $f(n)$, where $f$ is a function from the class. Then we will restrict Minimal Element Principle to the first $k$ elements (in the built-in order) to get a sentence in the SO$\exists$ language + built-in order predicate whose optimal refutation is of size $2^{\Omega(f(n))}$, provided $f(n) = \Omega(\log n)$.

What remains is to show how to define and verify $f(\cdot)$ in polynomial in $n$ size. As an example, we will give the definitions of $\sqrt{\cdot}$ and $\log(\cdot)$.
In what follows, all the relation symbols are quantified existentially and the free variables are quantified universally. Moreover, the definitions are nested in the order they appear, starting with the above definition of a total order, being outermost one. We denote by $L$ the built-in order predicate, i.e. $L(x, y)$ stands for $x < y$.

We first define the successor function as the relation $S(x, y)$ standing for $y$ is the successor of $x$.

$$S(x, y) \equiv (L(x, y) \land (\forall z \neg (L(x, z) \land L(z, y))))$$

We can define any constants, by the relations $C_a(x)$, meaning $x = a$.

$$C_0(x) \equiv (\forall y \neg S(y, x))$$
$$C_a(x) \equiv (\exists y (C_{a-1}(y) \land S(y, x)))$$

We are now ready to define addition and multiplication recursively by the following relations, $A(x, y, z)$ and $M(x, y, z)$ standing for $x + y = z$ and $x \times y = z$, respectively.

$$A(x, y, z) \equiv ((C_0(x) \land C_0(y) \land C_0(z))$$
$$\lor \exists u, v (S(v, z))$$
$$\land ((S(u, x) \land A(u, y, v))$$
$$\lor (S(u, y) \land A(x, u, v))))$$

$$M(x, y, z) \equiv (((C_0(x) \lor C_0(y)) \land C_0(z))$$
$$\lor \exists u, v ((S(u, x) \land A(y, v, z) \land M(u, y, v))$$
$$\lor (S(u, y) \land A(x, v, z) \land M(x, u, v))))$$

We can now define $y = \lfloor \sqrt{x} \rfloor$ by $y = \lfloor \sqrt{x} \rfloor$ if and only if either $y^2 \leq x < (y + 1)^2$ or $y^2 \leq x$, but $(y + 1)^2 > n$.

$$R_2(x, y) \equiv (M(y, y, x)$$
$$\lor \exists u (M(y, y, u) \land L(u, x)$$
$$\land (\exists z, w (S(y, z) \land M(z, z, w) \land L(z, w))))$$
$$\lor \forall z, w (S(y, z) \rightarrow \neg M(z, z, w))))$$

We also define $y = \lfloor \log x \rfloor$ as

$$D_2(x, y) \equiv \exists u, v, w (M(u, x, v) \land A(v, w, u)$$
$$\land C_2(u) \land (C_0(v) \lor C_1(v)))$$

and finally $y = \lfloor \log x \rfloor$, using the recursion

$$\lfloor \log x \rfloor = \begin{cases} 
0 & x = 1 \\
1 + \lfloor \log \left(\frac{x}{2}\right) \rfloor & x > 1
\end{cases}$$
\[ L_2(x, y) \equiv (C_1(x) \land C_0(y) \lor (\exists u, v (S(v, y) \land D_2(x, u) \land L_2(u, v)))) \, . \]

We shall not formally prove that it is possible (in a straightforward way) to verify all the functions defined above within the decision tree computational model. The sketch of the proof is, however, pretty clear: as all the definitions are inductive and there are no mutual recursive relations, we check the functions in the order they are defined, starting from the basis cases. Thus all the queries are forced in the sense that if the assignment does not satisfy the definition of the given relation then the “wrong” answer leads to an immediate contradiction. Thus the size of the initial part of the tree, verifying the definitions of these functions, is polynomial in \( n \) as claimed. \( \square \)

Of course, the result holds for many other propositional proof systems as Tree-like Resolution is a very weak system, and can be polynomially-simulated by them. As an important consequence we get the following

**Theorem 4.** There is no Tree-like Resolution Complexity gap above \( 2^{\theta(n \log n)} \).

**Proof.** Let us first observe that there are \( \text{SO}\exists \) statements with optimal proofs of size \( 2^{\theta(np)} \) for every \( p \geq 1 \). It is enough to take \( \text{MEP}_{np} \), i.e. defined on \( p \)-tuples instead of single elements of the universe. To get the intermediate complexities, we could restrict the last element of any such \( p \)-tuple to be less than or equal to \( f(n) \) where \( f \) is some function and \( n \) is the size of the universe (model). The optimal tree resolution proof of the obtained in this way statement would be \( 2^{\theta(np-1,f(n))} \). However we have not any predefined functions in our language. The definition of \( f \) therefore should be a part of the sentence and the proof should “verify” the definition of the function \( f \). We shall use the same argument as we have done in the previous proof. However we have not total order either, so we have to define it within the \( \text{SO}\exists \) language, and to verify it by a decision tree. In the rest of the section we will show how to do this, which would complete the proof. \( \square \)

The total order can be defined as
\[
\exists L \quad (\forall x \lnot L(x, x) \\
\land \forall x, y \ ((x = y) \lor L(x, y) \lor L(y, x)) \\
\land \forall x, y, z \ ((L(x, y) \land L(y, z)) \rightarrow L(x, z)) \, ,
\]
and we complete the argument by proving the following lemma.

**Lemma 4.** The relation \( L \) can be optimally verified by a decision tree of size \( 2^{\theta(n \log n)} \).

**Proof.** The sentence translates into the following set of clauses:
\[
-\ell_{ii} i \\
\ell_{ij} \lor \ell_{ji} \ i, j \\
-\ell_{ij} \lor -\ell_{jk} \lor \ell_{ik} \ i, j, k.
\]
It is clear that every permutation $\pi$ of $\{1, 2, \ldots, n\}$ defines a satisfying assignment by setting $l_{ij}$ to $\top$ if and only if $\pi(i) < \pi(j)$ and vice versa. This observation immediately implies a lower bound of $n! = 2^{\Omega(n \log n)}$.

A decision tree which verifies $L$ can be constructed by incrementally ordering the elements of the universe. Suppose we already have a decision tree which orders the first $j$ elements, i.e. each leaf corresponds either to a contradiction or to a permutation of $\{1, 2, \ldots, j\}$ as explained above. We can now expand the latter leaves by finding the place of the $j + 1$-th element. In doing so, we use binary search which uses $O(\log (j + 1))$ free-choice queries. Once the place of the $j + 1$-th element has been found, all the queries involving it and some of the previous $j$ elements are forced, i.e. one of the answers leads to an immediate contradiction. Thus the forced queries contribute a polynomial factor to the size of the subtree consisting of the free-choice queries only. The depth of this subtree is $\sum_{j=1}^{n-1} O(\log (j + 1)) = O(n \log n)$, and therefore its size and the size of the entire decision tree is $2^{O(n \log n)}$. \hfill \Box

References

Strong Normalization of the Typed $\lambda_{ws}$-Calculus

René David¹ and Bruno Guillaume²

¹ Université de Savoie, Campus Scientifique, F-73376 Le Bourget du Lac
david@univ-savoie.fr

² LORIA / INRIA Lorraine, Campus Scientifique, F-54506 Vandœuvre-lès-Nancy
Bruno.Guillaume@loria.fr

Abstract. The $\lambda_{ws}$-calculus is a $\lambda$-calculus with explicit substitutions introduced in [4]. It satisfies the desired properties of such a calculus: step by step simulation of $\beta$, confluence on terms with meta-variables and preservation of the strong normalization. It was conjectured in [4] that simply typed terms of $\lambda_{ws}$ are strongly normalizable. This was proved in [7] by Di Cosmo & al. by using a translation of $\lambda_{ws}$ into the proof nets of linear logic. We give here a direct and elementary proof of this result. The strong normalization is also proved for terms typable with second order types (the extension of Girard’s system F). This is a new result.

1 Introduction

Explicit substitutions provide an intermediate formalism which, by decomposing the $\beta$ rule of the $\lambda$-calculus into more atomic steps, gives a better understanding of the execution models. The pioneer calculus with explicit substitutions, $\lambda\sigma$, was introduced by Curien & al. in [1] as a bridge between the classical $\lambda$-calculus and concrete implementations of functional programming languages. Since Melliès [6] has shown that this calculus does not preserve strong normalization, even for typed terms, finding a system satisfying the following properties became a challenge:

- step by step simulation of $\beta$,
- confluence on terms with meta-variables,
- strong normalization of the calculus of substitutions,
- preservation of strong normalization of the $\beta$-reduction.

During the last decade, various systems were presented in the literature but none of them satisfied simultaneously the previous properties. $\lambda_{ws}$, the calculus we introduced in [4], has been the first satisfying all of them. In addition to explicit substitutions, the terms of $\lambda_{ws}$ are decorated with “labels”. The typed version of the calculus (also introduced in [4]) shows that there is a strong link between the computational and the logical points of view: substitutions correspond to cuts and labels to weakenings. The proof that any pure $\lambda$-term which is $\beta$-strongly normalizable is still strongly normalizable in the $\lambda_{ws}$-calculus was highly technical and uses ad-hoc methods. We conjectured that the typed
terms are strongly normalizable (SN). Di Cosmo, Kesner and Polonovsky \cite{7} understood the relation between \( \lambda_{ws} \) and linear logic and, by using a translation of \( \lambda_{ws} \) into proof nets, they proved this conjecture. We give here a direct and arithmetical proof of SN for simply typed terms. This proof is based on the one for the (usual) \( \lambda \)-calculus due to the first author \cite{2,3}. We also prove, by using the standard notion of reducibility candidates, that terms typable with second order types (the extension of Girard’s system \( F \)) are strongly normalizable. This result is new.

The general idea of the proofs is the following. We first give a simple characterization of strongly normalizing terms (theorem \( \textup{3} \)). This result, which is only concerned with the untyped calculus, is interesting by itself and may be used to prove other results on \( \lambda_{ws} \). It can be seen as a kind of standardization result. Theorem \( \textup{3} \) mainly consists of commutation results. Note that permutation of rules is also the main ingredient in the proof of \cite{7}. Then, for \( \mathcal{S} \), we use this characterization to prove, by a tricky induction, a substitution lemma (theorem \( \textup{6} \)) from which the result follows immediately. For \( \mathcal{F} \), we use this characterization to prove that if a term is typed then it belongs to the interpretation of its type.

The paper is organized as follows. Section \( \textup{2} \) gives the main notations. In section \( \textup{3} \) we introduce some useful notions and we prove the key technical result. It is used in section \( \textup{4} \) to prove SN for simply typed terms and in section \( \textup{5} \) for second order types.

2 The \( \lambda_{ws} \)-Calculus

2.1 The Untyped Calculus

We define here a variant of \( \lambda_{ws} \) which is equivalent to the one in \cite{4}: \( \langle k \rangle \) is no more primitive but becomes the abbreviation of \( \langle \rangle \ldots \langle \rangle \), \( k \) many times and \( n \) is coded by \( \langle n \rangle 0 \). Since the strong normalization of both formulations are equivalent (see proposition \( \textup{1} \) below) and the proof is a bit simpler for the new one, we introduce here this calculus.

**Definition 1.** The set of terms of \( \lambda_{ws} \) is defined by the following grammar:

\[ T = 0 \mid \lambda T \mid (TT) \mid \langle \rangle T \mid [i/T, j]T \text{ where } i, j \in \mathbb{N}. \]

and the reduction rules of the \( \lambda_{ws} \)-calculus are given in fig.1.

**Remark 1.** – The “logical” meaning of \( \langle \rangle \) and \( [i/u, j]t \) is given by the typing rules. The “algorithmic” meaning is, intuitively, the following: \( \langle k \rangle t \) means that each de Bruijn index in \( t \) is increased by \( k \) (as a consequence, there is no variable with de Bruijn indices less than \( k \) in \( t \)) and \( [i/u, j]t \) represents the term \( t \) in which the variable indexed by \( i \) is substituted by \( u \) with a re-indexing commanded by \( j \).

– It is clear that the version of \( \lambda_{ws} \) presented here is a restriction of the one in \cite{4}. For self completeness the terms and the rules of this calculus are given in the appendix. The translation \( \phi \) from the latter to the present one is given by: \( \phi(t) \) is obtained from \( t \) by replacing \( n \) by \( \langle n \rangle 0 \) and then \( \langle k \rangle \) by \( \langle \rangle \ldots \langle \rangle \), \( k \) many times. In particular, \( \langle 0 \rangle \) is empty.
| b  | $((k)\lambda t u) \rightarrow [0/u, k]t$ |
| l  | $[i/u, j]t \lambda \rightarrow \lambda[i + 1/u, j]t$ |
| a  | $[i/u, j](t v) \rightarrow (([i/u, j]t) ([i/u, j]v))$ |
| $e_1$ | $[0/u, j][k]t \rightarrow (j)t$ |
| $e_2$ | $[i/u, j][k]t \rightarrow (i)[0/u, j]t$ |
| $n_1$ | $[i/u, j]0 \rightarrow 0$ |
| $n_2$ | $[0/u, j]0 \rightarrow u$ |
| $c_1$ | $[i/u, j][k/v, l]t \rightarrow [k/[i - k/u, j]v, j + l - 1]t$ |
| $c_2$ | $[i/u, j][k/v, l]t \rightarrow [k/[i - k/u, j]v, l][i - l + 1/u, j]t$ |

**Fig. 1.** Reduction rules of $\lambda_{ws}$

- Note that, in this variant, the reduction rules become a bit simpler and some of them ($m$ and $n_3$ in the original calculus) even disappear. Also note that rules $b_1$ and $b_2$ give a unique rule $b$ which is in fact a family of rules since $\langle k \rangle$ represents a family of symbols.

**Proposition 1.** If $t \rightarrow t'$ then $\phi(t) \rightarrow^+ \phi(t')$. In particular, the strong normalization of both versions of $\lambda_{ws}$ are equivalent.

**Proof.** Straightforward. $\square$

### 2.2 The Typed Calculus

**Definition 2.** Let $\mathcal{V}$ be a set of type variables.
- The set $\mathcal{S}$ of simple types is defined by: $\mathcal{S} ::= \mathcal{V} | \mathcal{S} \rightarrow \mathcal{S}$
- The set $\mathcal{F}$ of second-order types is defined by: $\mathcal{F} ::= \mathcal{V} | \mathcal{F} \rightarrow \mathcal{F} | \forall \mathcal{V}.\mathcal{F}$

**Definition 3.** A basis $\Gamma$ is an (ordered) list of types. The length of $\Gamma$ is denoted by $\|\Gamma\|$.
- The typing rules for $\mathcal{F}$ are the given in fig[2]. Note that the first element (on the left) of $\Gamma$ corresponds to the variable with de Bruijn index 0. For $\mathcal{S}$, just forget $\forall_i$ and $\forall_e$.

**Proposition 2.** Both systems have subject reduction: if $\Gamma \vdash t : A$ and $t \rightarrow u$, then $\Gamma \vdash u : A$.

**Proof.** We have to check that, for each rule, the typing is preserved after reduction. We give below the example of rule $b$. The proof is detailed in [5] for the original version of the calculus.

The typing of the $b$-redex $\langle (k)\lambda t u \rangle$ is given on the left and the typing of its reduct $[0/u, k]t$ is given on the right. We assume that $\|\Gamma\| = k$ and the last element of $\Gamma$ is $C$. 

3 Characterization of Strongly Normalizable Terms

This section gives a characterization (Theorem 3) of strongly normalizable terms. This is the key of the proof of the strong normalization for both systems. We first need some definitions.

3.1 Some Definitions

Definition 4. The set \( S \) of substitutions and the set \( \Sigma \) are defined by the following grammars:

\[
S ::= \emptyset | [i/T, j]S \\
\Sigma ::= \emptyset | \emptyset | i/T, j | \Sigma
\]

Definition 5. Some particular contexts are defined by the following grammars where \( * \) denotes a hole and, if \( H \) is a context, \( H[t] \) denotes the term obtained by replacing \( * \) by \( t \) in \( H \).

\[
C_i ::= * | \Sigma C_i | \lambda C_i \\
C_e ::= * | \Sigma C_e | (C_e T) \\
C ::= C_i | C_e
\]

Note that these contexts have a unique hole at the leftmost position. The elements of \( C \) (resp. \( C_i, C_e \)) are called head contexts (resp. i-contexts, e-contexts). Elements of \( T \) (resp. \( S, \Sigma, C \)) will be denoted by \( t, u, v, w \) (resp. by \( s \), by \( \sigma \), by \( H,K \)).
**Notation 1.**
1. We denote by $\rightarrow$ the least congruence on $T \cup C$ containing the rules of fig.\(\Dagger\) As usual, $t \rightarrow^* t'$ (resp. $t \rightarrow^+ t'$) means that $t$ reduces to $t'$ by some steps (resp. at least one step) of reduction.
2. The set of strongly normalizable terms (i.e. such that every sequence of $\rightarrow$ reductions is finite) is denoted by $SN$.

**Lemma 1 (and notation).** Every term in $T$ can uniquely be written as $H[0]$ or $H[(\sigma \lambda u \, \nu)]$ where $H$ is an head context. The head of $t$ (denoted by $hd(t)$) is:

$$
hd(H[0]) = H \quad \text{and} \quad \hd(H[(\sigma \lambda u \, \nu)]) = H[(\sigma \star \, v)]
$$

**Proof.** Straightforward. \(\square\)

**Notation 2.** Say $t \rightarrow^\star t'$ if $t \rightarrow t'$ with the following restrictions: use only the rules $a, e, c$ and only in $hd(t)$ either at the top level or, recursively, for $[i/u, j]$ in $hd(t)$, only in $hd(u)$. The rule $l$ is also permitted but only in $H_i$ where $hd(t) = H_i[H_e]$ with $H_i \in C_i$ and $H_e \in C_e$.

**Example 1.**

$$
[0/b, 0][\lambda[0/c, 0][1 \ [0/d, 1][0] \rightarrow^\star \ ([0/b, 0][\lambda[0/c, 0][1 \ [0/0/b, 0][d, 1][0]]
$$

$$
[0/a, 0][\lambda b \, c] \rightarrow^\star \lambda([1/a, 0][b \ [1/a, 0][c])
$$

$$
[0/0/a, 0][\lambda c, 0][b \not\rightarrow^\star [0/\lambda[1/a, 0][c, 0][b

$$(\lambda[0/a, 0]([b \, c] \not\rightarrow^\star (\lambda b \, c)
$$

**Lemma 2 (and notation).** The reduction $\rightarrow^\star$ is locally confluent and thus is confluent for terms such that $hd(t) \in SN$. The r-normal form of $t$ will be denoted by $r(t)$.

**Proof.** Straightforward. \(\square\)

**Remark 2.** The $r$-reduction is actually strongly normalizing for every term and thus confluent. This follows immediately from the strong normalization of the calculus of substitution (i.e. all the rules except $b$) which is proved in [4]. We have stated the previous lemma in this way to keep this paper self contained, i.e. our proof does not need this result. Thus, in the rest of the paper, when we use $r(t)$ or the confluence of $r$ we have to check that $hd(t) \in SN$. We will not mention this since this is always straightforward.

**Definition 6.**
1. Let $H$ be an head context. Let $R(H) \subset T$, $L(H) \subset C_i$ and, if $H \in C_i$, $I(H) \in T$ be defined by the following rules:
   - $R(\star) = \emptyset$, $R(\lambda H) = \lambda R(H)$, $R(\sigma H) = \sigma R(H)$ and $R((H t)) = R(H) \cup \{t\}$.
   - $L(\star) = \emptyset$, $L(\lambda H) = \lambda L(H)$, $L(\sigma H) = \sigma L(H)$ and $L((H t)) = L(H)$.
   - $I(\star) = 0$, $I(H[\lambda \star]) = I(H[[\lambda] \star]) = I(H)$ and $I(H[[i/u, j] \star]) = H[[i/u].$
2. An head context is pure if $L(H)$ has no substitutions.
3. Let \( t \) be a term in \( T \). The set \( \text{arg}(t) \subset T \) is defined by:
- \( \text{arg}(H[0]) = R(H) \cup \{ I(L(H)) \} \)
- \( \text{arg}(H[(\sigma \lambda u v)]) = R(H[[* v]]) \cup L(H)[\sigma \lambda u] \)

**Remark 3.** In the previous definition, the equation \( R(\lambda H) = \lambda R(H) \) actually means, since \( R(H) \) is a set of terms, \( R(\lambda H) = \{ \lambda t / t \in R(H) \} \) and similarly for \( R(\sigma H) = \sigma R(H) \).

**Example 2.**
\[
\text{arg}([4/0, j](\langle 2 \rangle \lambda 3 0)) = \{ [4/0, j](\lambda 3, [4/0, j]0 ) \}
\]
\[
\text{arg}([2/0, j][0/v, 2]0) = \{ [2/0, j]v \}
\]

**Lemma 3.**
- Let \( t = H[0] \). Then \( \text{r}(t) \) can be uniquely written as \( K[s_{0}] \) where \( K \) is pure.
- Let \( t = H[(\sigma \lambda u v)] \). Then \( \text{r}(t) \) can be uniquely written as \( K[((k)s_{\lambda u v_{1}})] \) where \( K \) is pure.

**Proof.** Straightforward. \( \square \)

**Definition 7.** Let \( s \in S \) be a substitution, we define \( s^{+} \in S \) and \( s^{\downarrow} \in T \) as follows:
- \( s^{+} \) is defined by: \( \emptyset ^{+} = \emptyset \) and \( ([i/u, j]s)^{+} = [i+1/u, j]s^{+} \).
- \( s^{\downarrow} \) is defined by: \( \emptyset ^{\downarrow} = 0 \) and \( (s[i/u, j])^{\downarrow} = su \) if \( i = 0 \) and \( s^{\downarrow} \) otherwise.

**Definition 8.** Let \( t \) be a term in \( T \). The head reduct of \( t \) (denoted as \( \text{hred}(t) \)) is defined as follows:
- If \( t = H[0] \) and \( \text{r}(t) = K[s_{0}] \) then \( \text{hred}(t) = K[s_{\downarrow}] \).
- If \( t = H[(\sigma \lambda u v)] \) and \( \text{r}(t) = K[((k)s_{\lambda u v_{1}})] \) then \( \text{hred}(t) = K[[0/v_{1}, k]s^{+}u] \).

**Example 3.** With terms as in the previous example, we have:
\[
\text{hred}([4/0, j](\langle 2 \rangle \lambda 3 0)) = [0/4/0, j0, 2][3/0, j]3
\]
\[
\text{hred}([2/0, j][0/v, 2]0) = 2
\]

**Theorem 3.** Let \( t \in T \) be such that \( \text{arg}(t) \subset SN \).

1. Assume \( t \rightarrow^{*} t' \) and \( t' \in SN \). Then \( t \in SN \).
2. Assume \( \text{hred}(t) \in SN \). Then \( t \in SN \).

### 3.2 Proof of Theorem 3

We first need some notations and lemmas.

**Notation 4.**
1. If \( t \in SN \), \( \eta(t) \) is the length of the longest reduction starting from \( t \) and \( \eta_{0}(t) \) is the maximum number of \( b \) or \( n \) steps in a reduction starting from \( t \).
2. The complexity of a term $t$ (denoted by $\text{ctxy}(t)$) is defined by: $\text{ctxy}(\ast) = \text{ctxy}(0) = 0$, $\text{ctxy}(\lambda t) = \text{ctxy}(\langle t \rangle) = \text{ctxy}(t) + 1$, $\text{ctxy}(t', t'') = \text{ctxy}(t) + \text{ctxy}(t') + 1$ and finally $\text{ctxy}([i/t', j]t) = \text{ctxy}(t) + \text{ctxy}(t') + i + 1$.

Note that the unusual definition of $\text{ctxy}([i/t', j]t)$ is due to the fact that $\text{ctxy}([k]) = k$. It ensures that $\text{ctxy}([i/u, j]) > \text{ctxy}(\langle i \rangle u)$ and thus, except for $t = 0$, $\text{ctxy}(u) < \text{ctxy}(t)$ for any $u \in \text{arg}(t)$.

**Lemma 4.** Let $H$ be an head context, $u$ be a term and $w \in \text{arg}(H[u])$. Then,

- either $w \in R(H)$,
- or $w = L(H)[v]$ for some $v \in \text{arg}(u)$,
- or $H$ is not an $i$-context, $u = \sigma\lambda u'$ and $w = L(H)[u]$.

**Proof.** Straightforward. □

**Lemma 5.** Let $H \in C$ be pure.

1. If $t = H[u] \in \text{SN}$ and $s \in \text{SN}$, then $H[[0/u,j][s^+0]] \in \text{SN}$.
2. If $t = H[[0/v,k][s^+u]] \in \text{SN}$, then $H[[\langle k \rangle s \lambda u v]] \in \text{SN}$.

**Proof.** By induction on $\eta(t) + \eta(s^+0)$ for (1) and $\eta(t) + \text{ctxy}(s)$ for (2). □

**Lemma 6.** Let $K$ be an head context. Assume that

- either $k \geq i + j$ and $w = [i/[k - i/v, l]u, j][k - j + 1/v, l]K \rightarrow^r w_1 = K_1[0/k - i/v, l][u, j][s^+]\ast$,
- or $i \leq k < i + j$ and $w = [i/[k - i/v, l]u, k + j - 1]K \rightarrow^r w_1 = K_1[[0/k - i/v, l][u, j][s^+]\ast]$.

Then, there is an head context $K_2$ such that $[i/u, j]K \rightarrow^r K_2[[0/u, j][s^+]\ast]$ and $[k/v, l]K_2[[0/u, j][s^+]\ast] \rightarrow^r w_1$.

**Proof.** By induction on the length of the reduction $w \rightarrow^r w_1$. □

**Lemma 7.** Assume $w = [i/u, j]K_1[[k/v', l]K_2] \rightarrow^r w_1 = K_3[[0/u, j][s^+]\ast]$ and $v \rightarrow v'$. Then, $[i/u, j]K_1[[k/v, l]K_2] \rightarrow^r K_4[[0/u, j][s^+]\ast] \rightarrow^r K_3[[0/u, j][s^+]\ast]$ for some $K_4, s_1$.

**Proof.** By induction on the length of the reduction $w \rightarrow^r w_1$. □

**Lemma 8.**

1. Assume $t = H[\sigma\lambda u v] \rightarrow^* t_0 = H_0[[\langle k_0 \rangle \lambda u_0 v_0]]$. Then, there is a term $t_1 = H_1[[\langle k_1 \rangle s_1 \lambda u v_1]]$ such that $t \rightarrow^r t_1 \rightarrow^* t_0$.
2. Assume $t = H[0] \rightarrow^* t_0 = H_0[[0/u_0, j_0][s^+]0]$. Then, $H$ can be written as $K[[i/u, j]K_0]$ such that $[i/u, j]K_0 \rightarrow^* K'_1 = K_1[[0/u, j][s^+]\ast]$ and $t_1 = K[K'_1][0] \rightarrow^* t_0$.

**Proof.** First note that we should be a bit more precise in the terms of the lemma: we implicitly assume that the potential $b$-redex (resp. $n$-redex) at the end of the left branch of $t$ is not reduced during the reduction $t \rightarrow^* t_0$. The lemma is proved by induction on the length of the reduction $t \rightarrow^* t_0$. We give some details only for (2). They are similar and simpler for (1).

The result is clear for $t = t_0$. Assume $t \rightarrow^* t_0$. By the induction hypothesis, $H \rightarrow H_1 = K[[i/u, j]K_0]$ for some $K, u, K_0$ such that $[i/u, j]K_0 \rightarrow^* K'_1 = K_1[[0/u, j][s^+]\ast]$ and $t_1 = K[K'_1][0] \rightarrow^* t_0$. $H$ can be written as $K_3[[i/u_1, j_1]K_2]$...
- if \( K_3 \rightarrow K \) or \( u_1 \rightarrow u \) the result is trivial,
- if \( K_2 = (\ast \ v) \) and \( K = K_3[(\ast \ [i/u, j])v] \) the result is trivial,
- if \( [i/u, j]K_2 \rightarrow_r [i/u, j]K_0 \) the result is trivial,
- if \( K_2 \rightarrow K_0 \) but the reduction is not an \( r \)-reduction, the result follows from lemma \[7\]
- if \( K_3 = K[[k/v, l]s] \) and, either \( [i/u, j] = [i/[k - i/v, l]u_1, j_1] \) and \( K_0 = [k - j_1 + 1/v, l]K_2 \), or \( [i/u, j] = [i/[k - i/v, l]u_1, l + j_1 - 1] \) and \( K_0 = K_2 \), the result follows from lemma \[6\].

**Lemma 9.**
1. Assume \( t_1 = H_1[(\sigma_1 \lambda u_1 \ v_1)] \rightarrow_1^* t_0 = H_0[(\langle k_0 \rangle \lambda u_0 \ v_0)] \). Then,
   \( H_1[0/v_1, k_1]s^+_1u_1 \rightarrow_1^* H_0[0/v_0, k_0]s^+_0u_0 \) where \( r(\sigma_1) = \langle k_1 \rangle s_1 \).
2. Assume \( t_1 = H_1[[0/u_1, j_1]s^+_10] \rightarrow_1^* t_0 = H_0[[0/u_0, j_0]s^+_00] \). Then, \( H_1[u_1] \rightarrow_1^* H_0[u_0] \).

**Proof.** By induction on the length of the reduction \( t_1 \rightarrow_1^* t_0 \). Look at the first reduction. Note that there is no simple relation between the original and the resulting reduction sequence and, in particular, the latter may be longer than the original.

**Lemma 10.**
1. Assume \( H[(\sigma \lambda u \ v)] \rightarrow_1^* t_0 \). Then \( t_0 \) has the form \( H_0[(\sigma_0 \lambda u \ v_0)] \) and \( H[0/v, k]s^+u \rightarrow_1^* H_0[0/v_0, k_0]s^+_0u \) where \( r(\sigma) = \langle k \rangle s \) and \( r(\sigma_0) = \langle k_0 \rangle s_0 \).
2. Assume \( H_0[0/u, j]s^+_00 \rightarrow_1^* t_0 \). Then \( t_0 \) has the form \( H_1[[0/u_1, j_1]s^+_10] \) where \( H_0 \rightarrow_1^* H_1[s_2^+] \) for some \( s_2 \) such that \( s_2[0/u, j]s^+_0 \rightarrow_1^* [0/u_1, j_1]s^+_1 \).

**Proof.** Straightforward.

**Lemma 11.** Let \( K \) be an i-context. Then, \( K \in SN \) iff \( I(K) \in SN \) and, in this case, \( \eta_0(I(K)) \leq \eta_0(K) \).

**Proof.** This follows immediately from the following result. Let \( K \) be an i-context, then: \( K[[i/u, j]s] \in SN \Leftrightarrow K[[i]u] \in SN \) and, in this case, \( \eta_0(K[[i]u]) \leq \eta_0(K[[i/u, j]s]) \).

\( \Rightarrow \) Prove, by induction on \( (\eta(t), cxty(K)) \) that if \( t = K[s[i/u, j]s] \in SN \) then \( K[d(s, i)u] \in SN \) where \( d(s, i) \rightarrow \) is the result of moving down \( s \) through \( (i) \). It is enough to prove that, if \( K[d(s, i)u] \rightarrow t' \) then \( t' \in SN \). This is done by a straightforward case analysis.

\( \Leftarrow \) This is proved by showing that to any sequence of reductions of \( t' = K[[i]u] \) corresponds a sequence of reductions of \( t \) with the same \( b \) or \( n \) steps. Define for \( s \in S, \delta(s) \in Z \) by: \( \delta(0) = 0 \) and \( \delta([k/v, l]s) = \delta(s) + l - 1 \). We show that, to a term of the form \( K'[\langle i'\rangle u'\rangle \) coming from \( t' \) corresponds, for some \( s \) such that \( \delta(s) < i' \), the term \( K'[s[i' - \delta(s)/u', l]s] \) coming from \( t \). This is done by a straightforward case analysis. For example, if \( t' \rightarrow_1^* K'[k/v, l][i'u'] \rightarrow K'[(l + i' - 1)u'] \) then \( t \rightarrow_1^* K'[k/v, l][s[i' - \delta(s)/u', l]s] = K'[s[i' - \delta(s)/u', l]s] \) where \( s' = [k/v, l]s \).

It is important to note that the result on \( \eta_0 \) would not be true with \( \eta \). This is essentially because \([k/v, l]\) can always go through \( (i) \) whereas \([k/v, l]\) cannot move down in \([i/u, j]\) if \( k < i \).
Lemma 12. Let $t \in T$ be such that $\text{arg}(t) \subset SN$ and $t \not\in SN$. Then,

1. If $t = H[\langle \sigma \lambda u \, v \rangle]$, there is a term $t_1 = H_1[\langle (k_1)s_1 \lambda u \, v_1 \rangle]$ such that $t \rightarrow^*_r t_1$ and $H_1[0/v_1, k_1]s_1^+ u \not\in SN$.

2. If $t = H[0]$ there is a term $t_1 = K[K_1[0/u, j]s^+_1 0]$ such that $t \rightarrow^*_r t_1$ can be written as $K[0/v_0, k_0]u \not\in SN$.

Proof. 1. Since $\text{arg}(t) \subset SN$, the potential $b$-redex must be reduced in an infinite reduction of $t$ and thus such a reduction looks like: $t \rightarrow^* H_0[\langle (k_0) \lambda u_0 \, v_0 \rangle] \rightarrow H_0[0/v_0, k_0]u_0 \rightarrow \ldots$ and the result follows from lemmas 8 and 9.

2. Since $\text{arg}(t) \subset SN$ and thus, by lemma 11 $H \in SN$, an infinite reduction of $t$ looks like: $t \rightarrow^* H_0[0/v_0, j_0]s^+_0 0 \rightarrow H_0[u_0] \rightarrow \ldots$ and the result follows from lemma 8 and 9.

Proof of theorem $\Box$

1. By induction on $(\eta_0(t'), \text{ctxy}(t))$. Note that the proof is by contradiction. We tried to find a constructive proof but we have been unable to find a correct one.

- Assume first $t = H[\langle \sigma \lambda u \, v \rangle]$ and $t \not\in SN$. By lemma 12 let $t \rightarrow^*_r t_0 = H_0[\langle (k_0)s_0 \lambda u \, v_0 \rangle]$ be such that $t_1 = H_0[0/v_0, k_0]s_0^+ u \not\in SN$. By the confluence of $\rightarrow^*_r$, let $t'_0$ be such that $t' \rightarrow^*_r t'_0$ and $t_0 \rightarrow^*_r t'_0$. By lemma 10 with the reduction $t_0 \rightarrow^*_r t_0$, $t'_0 = H'[\langle \sigma' \lambda u \, v' \rangle]$. Let $t'_1 = H'[0/v', k']s^+ u]$ where $r(\sigma') = \langle k' \rangle$. Then $\eta_0(t'_1) < \eta_0(t')$ and, by lemma 10, $t_1 \rightarrow^*_r t'_1$. It is thus enough to show that $\text{arg}(t_1) \subset SN$ to get a contradiction from the induction hypothesis.

Let $w_1 \in \text{arg}(t_1)$. By lemma 4 either $w_1 \in \text{arg}(t_0)$ and the result is trivial or $w_1 = L(H_0)[0/v_0, k_0]s_0^+ w$ for some $w \in \text{arg}(u)$ or $H$ is not an $i$-context and $w_1 = L(H_0)[0/v_0, k_0]s_0^+ u$.

Since the second case is similar, we consider only the first one. Let $a = L(H)[\langle \sigma \lambda w \, v \rangle]$ and $a' = L(H')[\langle \sigma' \lambda w \, v' \rangle]$. Then, $a \rightarrow^*_r a'$ and $\eta_0(a') \leq \eta_0(t')$ (use lemma 11 for the difficult case, i.e. when $u = K[0]$ and $w = I(L(K))$). If it is not the case that $H$ is an $i$-context and $u = 0$, then $\text{ctxy}(a) < \text{ctxy}(t)$ and, by the induction hypothesis, $a \in SN$ and the result follows since $a \rightarrow^* w_1$. Otherwise, the result is trivial since it is easily seen (by induction on $(\eta(H), \text{ctxy}(H))$) that, if $t = H[\langle \sigma \lambda 0 \, v \rangle]$ (where $H$ is an $i$-context), $r(\sigma) = \langle k \rangle s$ and $\text{arg}(t) \subset SN$, then $H[0/v, k]s^+ 0 \not\in SN$.

- Assume $t = H[0]$ and $t \not\in SN$. By lemma 12 let $t = K[\langle i/u, j \rangle K_0][0] \rightarrow^*_r t_0 = K[0/v_0, j s^+_0 0]$ be such that $t_1 = K[0/u, j]u \not\in SN$. By the confluence of $\rightarrow^*_r$, let $t'_0$ be such that $t' \rightarrow^*_r t'_0$ and $t_0 \rightarrow^*_r t'_0$. By lemma 10 with the reduction $t_0 \rightarrow^*_r t'_0$, $t'_0 = H'[0/u', j']s^+ u]$ where $K[H_0] \rightarrow^*_r H'[s_1 s]$ for some $s_1$ such that $s_1[0/u, j]s^+_0 \rightarrow^*_r [0/u', j]s^+$. Let $t'_1 = H'[u']$. Then $\eta_0(t'_1) < \eta_0(t')$ and, by lemma 10, $t_1 \rightarrow^*_r t'_1$. It is thus enough to show that $\text{arg}(t_1) \subset SN$ to get a contradiction from the induction hypothesis.
Let \( w_1 \in \arg(t_1) \). By lemma \( \square \) either \( w_1 \in \arg(t_0) \) and the result is trivial or \( w_1 = L(K[H_0])[w] \) for some \( w \in \arg(u) \) or \( H \) is not an \( i \)-context and \( w_1 = L(K[H_0])[w] \).

Since the second case is similar, we consider only the first one. Let \( a = L(K[[i/w,j]K_0])[0] \). Since \( s_1 u \rightarrow^* u' \), it is easy to find \( w' \) such that \( s_1 w \rightarrow^* w' \) and, letting \( a' = L(H')(0/w', j's\, )s_0][0], a \rightarrow^* a' \) and \( \eta_0(a') \leq \eta_0(t') \) (use lemma \( \square \) for the difficult case, i.e. when \( u = K[0] \) and \( w = I(L(K)) \)). Since \( cxty(a) < cxty(t) \) (except if \( H \) is an \( i \)-context and \( u = 0 \) but in this case again the result is trivial), by the induction hypothesis, \( a \in SN \) and the result follows since \( a \rightarrow^* w_1 \).

2. This follows immediately from (1) and lemma \( \boxdot \).

4 Strong Normalization for \( S \)

Theorem 5 below has first been proved in [7] by Di Cosmo & al. It is of course a trivial consequence of theorem 7 of section 5. However, the proof presented below is interesting in itself because it is purely arithmetical whereas the one of section 5 is not.

**Theorem 5.** Typed terms of \( T \) are strongly normalizing.

**Proof.** By induction on \( cxty(t) \). The cases \( t = 0 \), \( t = \lambda t' \) and \( t = (\,)t' \) are immediate. The case \( t = [i/u,j]t' \) follows immediately from theorem 6 below. The remaining case is \( t = (u v) \). By the induction hypothesis, \( u \) and \( (0(\,1)v) \) are in \( SN \). Thus, by theorem 6, \([0/u,0](0(\,1)v) \in SN \) and since \([0/u,0](0\langle 1\rangle v) \rightarrow^* t \) it follows that \( t \in SN \).

**Theorem 6.** Assume \( u, t \in T \cap SN \). Then \([i/u,j]t \in SN \).

**Proof.** We prove the following. Let \( u \in T \cap SN \). Then,

1. \( t = [i/u,j]t' \). The fact that \( \arg(t) \subset SN \) follows immediately from the IH.

   By theorem \( \square \) it is thus enough to show that \( \text{hred}(t) \in SN \).

   (a) If \( t' = H[(\sigma\lambda v_1\, )v_2] \); since \( \eta_0(\text{hred}(t')) < \eta_0(t') \), it follows from the IH that \([i/u,j]\text{hred}(t') \in SN \) and the result follows since \([i/u,j]\text{hred}(t') \rightarrow^* \text{hred}(t) \).

   (b) If \( t' = H[0] \); let \( r(t') = K[s0] \).

      • If \( s^* \neq 0 \); since \( \eta_0(\text{hred}(t')) < \eta_0(t') \), it follows from the IH that \([i/u,j]\text{hred}(t') \in SN \) and the result follows since \([i/u,j]\text{hred}(t') \rightarrow^* \text{hred}(t) \).
Definition 9. Assume $\lambda$. The proof uses the same lines as the one for the (ordinary) Strong Normalization for $\lambda$-calculus. We first define the interpretation of a type and we show that if we define the candidates of reducibility and show some of their properties. Then, $H$.

Proof. We need (2) in the proof of (1) for the following reason: we cannot conveniently it is not difficult to get $\arg(t_1) \subseteq SN$. It is thus enough to show that $\hred(t_1) \subseteq SN$.

- If $u' = \langle k' \rangle \lambda u'_0$ and thus $\hred(t_1) = H'[w]$ where $w = [0/t_0, k + k']u'_0$. Since $\type(t_0) < \type(u)$, by the $IH$, $w \in SN$. By the $IH$, $H'[w] \in SN$ since $\type(w) < \type(u)$. Note that, here, we use (2).

- Else $\hred(t_1) = H'[(\langle k \rangle \hred(u')) t_0] = \hred([i/\hred(u'), j]t')$. If $u' \rightarrow^+ \hred(u')$, the result follows from the $IH$. Otherwise, the result is trivial.

2. $t = H[u]$. If $H$ is an $i$-context, the result is immediate. Otherwise, $H = H'[\langle (k) \star t' \rangle]$. It is clear that $\arg(t) \subseteq SN$. It remains to prove that $\hred(t) \subseteq SN$.

(a) If $u = \sigma \lambda u'$: then $\hred(t) = r(H')[[0/t', k + k']s^+u']$ where $r(\sigma) = \langle k' \rangle s$. Since $u \in SN$, $s^+u' \in SN$. By the $IH$ since $\type(t') < \type(u)$, $[0/t', k + k']s^+u' \in SN$. Finally $\hred(t) \subseteq SN$ since $\type([0/t', k + k']s^+u') < \type(u)$.

(b) Otherwise $\hred(t) = H[\hred(u)]$. If $u \rightarrow^+ \hred(u)$ the result follows from the $IH$ and otherwise the result is trivial.

Remark 4. We need (2) in the proof of (1) for the following reason: we cannot always find $H'$ and $i, j$ such that $[i/v, 0]H'[(j)0] \rightarrow^* H[v]$. By choosing $i$ large enough and $j$ conveniently it is not difficult to get $[i/v, 0]H[(j)0] \rightarrow^* H[(j)v]$ but we do not know how to get rid of $\langle j \rangle$. This is rather strange since, in the $\lambda$-calculus, this corresponds to the trivial fact that $(u v)$ can be written as $(x v)[x := u]$ where $x$ is a fresh variable.

5 Strong Normalization for $\mathcal{F}$

The proof uses the same lines as the one for the (ordinary) $\lambda$-calculus. We first define the candidates of reducibility and show some of their properties. Then, we define the interpretation of a type and we show that if $t$ has type $A$ then $t$ belongs to the interpretation of $A$.

Definition 9. 1. If $X$ and $Y$ are subsets of $T$, $X \rightarrow Y$ denotes the set of $t$ such that, for all $u \in X$, $(t u) \in Y$.

2. The set $C$ of candidates of reducibility is the smallest set which contains $SN$ and is closed by $\rightarrow$ and intersection.

3. $N_0$ is the set of terms of the form $(0 u_1 \ldots u_n)$ where $u_i \in SN$ for each $i$.

Lemma 13. Assume $C \in C$. Then, $N_0 \subseteq C \subseteq SN$.

Proof. By induction on $C$.  □
Definition 10. An interpretation $I$ is a function from $V$ to $C$. $I$ is extended to $\mathcal{F}$ by: $|\alpha|_I = I(\alpha)$, $|A \rightarrow B|_I = |A|_I \rightarrow |B|_I$ and $|\forall \alpha.A|_I = \bigcap_{C \in C} |A|_{I\{\alpha:=C\}}$ (where $J = I\{\alpha:=C\}$ is such that $J(\alpha) = C$ and $J(\beta) = I(\beta)$ for $\beta \neq \alpha$).

Definition 11. Let $u_0, \ldots, u_{n-1}$ be a sequence of terms. We denote by $[i/u]$ the substitution $[i/u_0, 0][i+1/u_1, 0] \cdots [i+n-1/u_{n-1}, 0]$.

Lemma 14. Let $w$ be a sequence of terms in $SN$, $s \in S$ be regular and $C \in C$. Assume either $t' \rightarrow^* t$ or $t' = [0/t, j]s^+0$ or $t' = (s\lambda u v)$ and $t = [0/v, 0]s^+u$. If $(t \ w) \in C$, then $(t' \ w) \in C$.

Proof. By induction on $C$. The case $C = SN$ follows immediately from theorem 5. The other cases are straightforward.

Lemma 15. $|A\{\alpha:=B\}|_I = |A|_{I\{\alpha:=|B|_I\}}$ and thus $|A|_{I\{\alpha:=B\}} = |A|_I$ if $\alpha \notin A$.

Proof. Straightforward.

Lemma 16. Let $I$ be an interpretation. Assume $\Gamma \vdash t : B$ and $u \in |\Gamma|_I$ then $[0/u]t \in |B|_I$.

Proof. By induction on $\Gamma \vdash t : B$. For simplicity, we write $|A|$ instead of $|A|_I$. Assume $u \in |\Gamma|$ and look at the last rule used in the typing derivation:

- rule $Ax$:

  $A, \Gamma \vdash 0 : A$

  Let $v \in |A|$. By lemma 13, $v, u \in SN$ and the result follows from lemma 14.

- rule $\rightarrow_i$:

  $A, \Gamma \vdash t : B$

  $G \vdash \lambda t : A \rightarrow B$

  Let $v \in |A|$ and $w = ([0/u]a v)$. By the IH, $[0/v, 0][1/u]t \in |B|$ and the result follows from lemma 14.

- rule $\rightarrow_e$:

  $\Gamma \vdash t : A \rightarrow B$  $\Gamma \vdash v : A$

  $\Gamma \vdash (tv) : B$

  By the IH, $[0/u]t \in |A \rightarrow B|$ and $[0/u]v \in |A|$. Thus $([0/u]t [0/u]v) \in |B|$ and the result follows from lemma 14.

- rule $Weak$:

  $\Gamma \vdash t : A$

  $B, \Gamma \vdash (t) : A$

  Let $v \in |B|$. By the IH $[0/u]t \in |A|$ and the result follows from lemma 14.
Strong Normalization of the Typed \( \lambda_{ws} \)-Calculus

- rule \textit{Cut}:

\[
\frac{\Gamma, \Delta, \Phi \vdash t : B \quad \Delta, \Phi \vdash v : A}{\Gamma, \Delta, \Phi \vdash [i/v, j]t : B}
\]

where \( i = \|\Gamma\| \) and \( j = \|\Delta\| \)

Let \( u_1 \in |\Delta|, u_2 \in |\Phi| \) and \( w' = [0, u][i/u_1][i + j/u_2][i/v, j]t \). By the \textit{IH} (on the second premise), \([0/u][i/u_1][i + j/u_2][v, 0][i + 1/u_2]t \in |B| \). Since \( w' \to^*_r w \), The result follows from lemma 14.

- rule \( \forall_i \):

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall \alpha.A}
\]

if \( \alpha \not\in \Gamma \)

Let \( C \in C \). Since \( \alpha \not\in \Gamma \), by lemma 15, \( u \in |\Gamma|_{I\{\alpha := C\}} \) and thus, by the \textit{IH}, \([0/u]t \in |A|_{I\{\alpha := C\}} \). It follows that \([0/u]t \in |\forall \alpha.A|_I \).

- rule \( \forall_e \):

\[
\frac{\Gamma \vdash t : \forall \alpha.A}{\Gamma \vdash t : A\{\alpha := B\}}
\]

By the \textit{IH}, \([0/u]t \in |\forall \alpha.A|_I \) and thus \([0/u]t \in |A|_{I\{\alpha := B|_I\}} = |A\{\alpha := B\}|_I \) (by lemma 15).

\( \Box \)

Theorem 7. Every typed term is strongly normalizing.

\textbf{Proof.} Assume \( \Gamma \vdash t : B \). By lemma 13, \( 0 \in |\Gamma| \) and thus, by lemma 16, \([0/0]t \in |B| \). By lemma 13, \([0/0]t \in SN \) and thus, since \( SN \) is closed by subterms, \( t \in SN \).

\( \Box \)

References

A Appendix

The set of terms and the reduction rules of the original calculus of [4] are:

Terms

\[ T = n | \lambda T | (T T) | \langle k \rangle T | [i/T, j]T \] where \( n, k, i, j \in \mathbb{N} \).

Rules

\begin{align*}
\text{b}_1 \quad (\lambda t u) & \rightarrow [0/u, 0]t \\
\text{b}_2 \quad (\langle k \rangle \lambda t u) & \rightarrow [0/u, k]t \\
\text{l} \quad [i/u, j]\lambda t & \rightarrow \lambda [i + 1/u, j]t \\
\text{a} \quad [i/u, j](t v) & \rightarrow ([i/u, j]t)([i/u, j]v)) \\
\text{e}_1 \quad [i/u, j]\langle k \rangle t & \rightarrow (j + k - 1)t \quad i < k \\
\text{e}_2 \quad [i/u, j]\langle k \rangle t & \rightarrow \langle k \rangle [i - 1/u, j]t \quad k \leq i \\
\text{n}_1 \quad [i/u, j]n & \rightarrow n \\
\text{n}_2 \quad [i/u, j]n & \rightarrow \langle i \rangle u \\
\text{n}_3 \quad [i/u, j]n & \rightarrow n + j - 1 \quad i < n \\
\text{c}_1 \quad [i/u, j][k/v, l]t & \rightarrow [k/i - k/u, j]v, j + l - 1]t \quad k \leq i < k + l \\
\text{c}_2 \quad [i/u, j][k/v, l]t & \rightarrow [k/i - k/u, j]v, l][i - l + 1/u, j]t \quad k + l \leq i \\
\text{m} \quad \langle i \rangle \langle j \rangle t & \rightarrow \langle i + j \rangle t
\end{align*}
**A Fixed-Point Logic with Symmetric Choice**

Anuj Dawar and David Richerby

University of Cambridge Computer Laboratory, William Gates Building, J.J. Thomson Avenue, Cambridge, CB3 0FD, UK

\{Anuj.Dawar,David.Richerby\}@cl.cam.ac.uk

**Abstract.** Gire and Hoang introduce a fixed-point logic with a ‘symmetric’ choice operator that makes a nondeterministic choice from a definable set of tuples at each stage in the inductive construction of a relation, as long as the set of tuples is an automorphism class of the structure. We present a clean definition of the syntax and semantics of this logic and investigate its expressive power. We extend the logic of Gire and Hoang with parameterized and nested fixed points and first-order combinations of fixed points. We show that the ability to supply parameters to fixed points strictly increases the power of the logic. Our logic can express the graph isomorphism problem and we show that, on almost all structures, it captures $P^{GI}$, the class of problems decidable in polynomial time by a deterministic Turing machine with an oracle for graph isomorphism.

1 Introduction

Descriptive complexity classifies problems according to the richness of the logical language required to describe them, offering a view of complexity that is independent of any machine model. Fagin’s result that the problems describable in the existential fragment of second-order logic are exactly those in $NP$ invites the natural question of whether there is a logic for $P$. First-order logic can only express queries of low computational complexity. This is because it is unable to express many fundamental algorithmic tools such as iteration, counting, arithmetic and the selection of a single element with some property.

Fixed-point logics such as LFP and IFP address the first of these deficiencies. With the proviso that every structure be equipped with a linear ordering of its vertices, these logics capture the class $P$ \cite{Imm86,Var82}. (A logic $\mathcal{L}$ is said to capture a complexity class $\mathcal{C}$ if every problem in $\mathcal{C}$ is definable by a formula in $\mathcal{L}$ and the problem of evaluating $\mathcal{L}$-formulae is, itself, in $\mathcal{C}$.) Given a linear order, iteration allows counting, arithmetic and choice to be performed. The successive elements of the ordering can be used to simulate numbers and choice from a set can be achieved by taking the element that is least according to the order.

The imposition of an extrinsic linear ordering is undesirable as it allows the expression of queries such as, ‘there are no edges to the last vertex in the graph,’

\* Research supported by EPSRC grant GR/S06721.
which is as much a property of the ordering as it is of the graph. A compromise would be to allow only those queries which are invariant under changes to the ordering but this attempt fails because the set of formulae giving rise to order-invariant queries is undecidable. In the absence of a linear order, LFP and IFP cannot express simple counting properties like the parity of a set.

The ability to count alone does not suffice as first-order logic with counting quantifiers cannot define the class of connected graphs. However, the combination of inflationary fixed points and counting is a widely-studied and reasonably powerful logic. Immerman had conjectured that IFP + C would capture \( P \) but this was refuted by Cai, Fürrer and Immerman [CFI92]. Nonetheless, the logic does capture \( P \) on many classes of graphs, such as trees [IL90], planar graphs [Gro98], and graphs of bounded tree width [GM99] or genus [Gro00].

In [DR], we considered the combination of fixed points and nondeterministic choice, extending the work that appears in [AB87, GH98]. Iteration with choice can be used to construct a linear order on the universe of a structure but, in general, formulae are nondeterministic: they define sets of relations rather than single relations. The queries defined by deterministic formulae (those defining a single relation) are exactly \( P \) but this set of formulae is undecidable so cannot reasonably be called a logic. We also considered several notions of satisfaction for nondeterministic formulae and showed that the resulting logics capture a wide range of complexity classes including \( NP \) and \( co-NP \).

Assuming that \( P \neq NP \), unrestricted choice is, therefore, too powerful to capture \( P \). In [GH98], Gire and Hoang consider a combination of fixed points and symmetric choice, a restriction in which choices may be made only from automorphism classes of the structure on which the formula is being evaluated (that is, sets in which every pair of elements is exchanged by some automorphism). They define a fixed-point operator that takes as arguments formulae \( \varphi \) and \( \psi \). At each stage in evaluating the fixed point, \( \psi \) defines a set of tuples, the choice set. If this set is an automorphism class, one tuple is nondeterministically chosen from it; otherwise, no tuple is chosen. The chosen tuple (or lack of such) is then used by \( \varphi \) to add tuples to the relation being constructed.

This fixed-point operator is ‘semideterministic’: while an induction may still define a set of relations, these relations are pairwise isomorphic. This allows the definition of a logic that we call SC-IFP. This is still not a serious candidate for capturing \( P \) as, even if it were shown to express all polynomial-time properties, evaluating formulae requires testing that sets are automorphism classes and there is no known polynomial-time algorithm for doing this.

Gire and Hoang deal with this problem by defining a sublogic in which the fixed-point operator takes as an argument an extra formula that is supposed to provide a witness automorphism for each pair of tuples in the choice set; if the witness formula does not show that the choice set is an automorphism class, no element is chosen. This guarantees that formulae can be evaluated in polynomial time while maintaining semideterminism of the fixed-point operator. It is this logic (or, rather, its closure under an operator that performs interpretations between structures) that Gire and Hoang show to be strictly more expressive.
than IFP + C while still defining only polynomial-time properties. While it seems unlikely that this logic captures \( P \), this remains to be proved.

In this paper, we concentrate on SC-IFP. Showing that there are polynomial-time properties not definable in this logic would settle the status of the sublogic and, with this aim in mind, we investigate the power of SC-IFP. The logic we define is broadly similar to that of Gire and Hoang but there are some important differences in the syntax (we allow parameterized and nested fixed points) and semantics (Gire and Hoang require the choice set to be an automorphism class of the structure; we require it only to be an automorphism class up to those relations mentioned in the formula). In particular, we show that the ability to supply first-order parameters to fixed points strictly increases the power of the logic. This is significant as it involves the first inexpressibility result for SC-IFP and the usual techniques for proving such results (embedding into infinitary logics or using games) seem to be unavailable. We proceed by choosing a class of structures on which the parameterless fragment is weak.

The expressive power of SC-IFP is closely related to the graph isomorphism problem, GI, which it can express. GI is clearly in \( NP \) (guess a mapping between two graphs and check that it is an isomorphism) but not known to be either \( NP \)-complete, in \( P \) or even \( P \)-hard. There are, nonetheless, many results concerning the complexity of this problem. For example, it is known to be \( NL \)-hard under logarithmic space reductions \cite{Lor00} and it is unlikely to be \( NP \)-complete as, if it were, the polynomial hierarchy would collapse at its second level \cite{BHZ87, Sch88}.

Our main result is that, on almost all structures, the closure of SC-IFP under first-order reductions captures \( P^{GI} \), the class of problems decidable in polynomial time by a Turing machine with an oracle for GI. In particular, it captures \( P^{GI} \) on any class of structures on which IFP + C captures \( P \). Independent of GI, it is known from \cite{GH98} that the logic is strictly more expressive than IFP + C.

The rest of this section contains background definitions. SC-IFP is introduced in Section \( \exists \) and we discuss the effects of allowing parameterized definitions in Section \( \exists \). In Section \( \exists \) we investigate the expressive power of the logic.

**Preliminaries.** All structures in this paper are finite and all vocabularies are finite and purely relational, though constant symbols are omitted for notational convenience only. We write \( |\mathcal{A}| \) for the universe of structure \( \mathcal{A} \) and \( ||\mathcal{A}|| \) for the cardinality of \( |\mathcal{A}| \).

Classes of structures are assumed to be isomorphism-closed: if a structure is in a class, all images of that structure under isomorphisms are in the class. If \( \mathcal{C} \) is a class of structures, a \( k \)-ary query \( Q \) on \( \mathcal{C} \) maps each structure \( \mathcal{A} \in \mathcal{C} \) to a \( k \)-ary relation on \( |\mathcal{A}| \) such that if \( \rho : \mathcal{A} \rightarrow \mathcal{B} \) is an isomorphism, \( Q(\mathcal{B}) = \rho(Q(\mathcal{A})) \). The case \( k = 0 \) is known as a boolean query; \( \emptyset \) represents false and \( \{ \langle \rangle \} \), the relation containing the unique empty tuple, represents true. A boolean query \( Q \) on a class \( \mathcal{C} \) may be associated with the class \( \{ \mathcal{A} \in \mathcal{C} : Q(\mathcal{A}) \text{ is true} \} \).

A query \( Q \) on a class \( \mathcal{C} \) of structures is \( \mathcal{L} \)-definable for some logic \( \mathcal{L} \) if there is a formula \( \varphi \in \mathcal{L} \) such that \( \varphi^{\mathcal{A}} = Q(\mathcal{A}) \) for all \( \mathcal{A} \in \mathcal{C} \). We write \( \mathcal{L}_1 \leq \mathcal{L}_2 \) if every \( \mathcal{L}_1 \)-definable query is also \( \mathcal{L}_2 \)-definable and \( \mathcal{L}_1 = \mathcal{L}_2 \) if \( \mathcal{L}_1 \leq \mathcal{L}_2 \) and \( \mathcal{L}_2 \leq \mathcal{L}_1 \). We denote by \( \mathcal{L}^k \) the fragment of first-order logic in which no formula contains
more than $k$ distinct variables and we assume familiarity with the conventional fixed-point logics LFP and IFP and with logics with counting. See [EF99] for an introduction to these topics.

2 Symmetric Choice

In this section, we introduce the sc-ifp fixed-point operator and establish some of its basic properties. sc-ifp takes two operands, respectively the induction formula and the choice formula. We consider the fixed point as being built up in a series of stages. At each stage of evaluation on some structure $\mathfrak{A}$, the choice formula is evaluated to produce a set of tuples, the choice set. If the choice set is an automorphism class of $\mathfrak{A}$, one of its tuples is nondeterministically chosen; if it is not, no element is chosen. The induction formula is then evaluated using the chosen tuple, if any, along with the relation built up so far to define the tuples to be added to the relation being constructed.

To avoid circularity of definition, we define the operator to take a general class of maps from relations to relations as its arguments. We also introduce parameters to the definitions from the outset as the definitions of conventional fixed-point logics. This complicates the definitions a little but the reader may treat all tuples of parameters as empty at a first reading. Throughout, we use the notation $\bar{z}$ to denote variables that are treated as parameters.

On a structure $\mathfrak{A}$ and for any interpretation of the $r$-ary relation $X$ and the $n$-tuple of parameters $\bar{z}$, a formula $\varphi(X, \bar{x}, \bar{z})$, where $|\bar{x}| = r$, defines an $r$-ary relation. We can, therefore, associate with $\varphi$ a map $f^\mathfrak{A}_A : |\mathfrak{A}|^n \times \mathcal{P}(|\mathfrak{A}|^r) \to \mathcal{P}(|\mathfrak{A}|^r)$ and we shall define our operator in terms of maps such as this.

Definition 1. Let $\mathfrak{A}$ be a structure and let $X$ and $Y$ be new relation symbols of arity $r$ and $s$, respectively. A pair of maps $f^\mathfrak{A}_A$ and $g^\mathfrak{A}_A$ is appropriate for $X$ and $Y$ on $\mathfrak{A}$ if:

- both have domain $|\mathfrak{A}|^n \times \mathcal{P}(|\mathfrak{A}|^r) \times \mathcal{P}(|\mathfrak{A}|^s)$;
- $f^\mathfrak{A}_A$ has range $\mathcal{P}(|\mathfrak{A}|^r)$ and $g^\mathfrak{A}_A$ has range $\mathcal{P}(|\mathfrak{A}|^s)$;
- for any fixed $\bar{a} \in |\mathfrak{A}|^n$ and interpretations for $X$ and $Y$ in $\mathfrak{A}$, $f^\mathfrak{A}_A(\bar{a}, X, Y)$ and $g^\mathfrak{A}_A(\bar{a}, X, Y)$ are invariant under any automorphism of $(\mathfrak{A}, X, Y, \bar{a})$.

To cope with parameterized definitions, we need to restrict attention to automorphisms that fix the interpretations of the parameters and we define automorphism classes appropriately. This is a technical condition which we will explain in Example 7 after we have defined the sc-ifp operator.

Definition 2. Let $\mathfrak{A}$ be a structure and $n, s \in \mathbb{N}$. For a tuple $\bar{a} \in |\mathfrak{A}|^n$, an $\bar{a}$-respecting automorphism class of $|\mathfrak{A}|^s$ is a maximal set $C \subseteq |\mathfrak{A}|^s$ such that, for all $\bar{b}, \bar{c} \in C$, there is an automorphism $\rho$ of $\mathfrak{A}$ such that $\rho(\bar{a}) = \bar{a}$ and $\rho(\bar{b}) = \bar{c}$.

Definition 3. Let $\mathfrak{A}$ be a structure and let $f^\mathfrak{A}_A$ and $g^\mathfrak{A}_A$ be maps appropriate for new relation symbols $X$ and $Y$ on $\mathfrak{A}$. For any fixed tuple of parameters $\bar{a} \in |\mathfrak{A}|^n$, let $T_{f,g}^{\mathfrak{A},\bar{a}}$ be the least tree with the following properties:
the root is labelled \(\langle \emptyset, \emptyset \rangle\);  
- if \(g^A(\bar{a}, R, S)\) is a non-trivial \(\bar{a}\)-respecting automorphism class of \((A, R, S)\),  
a node labelled \(\langle R, S \rangle\) has a child \(\langle R \cup f^A(\bar{a}, R, \{\bar{s}\}), \{\bar{s}\} \rangle\) for each \(\bar{s} \in g^A(\bar{a}, R, S)\);  
- otherwise, a node labelled \(\langle R, S \rangle\) has a child labelled \(\langle R \cup f(\bar{a}, R, \emptyset), \emptyset \rangle\).

Call a path in \(T^A_{f, g}\) \(R\)-dense if infinitely many of its nodes have \(R\) as the first component of their label and define \(\text{sc-ifp}(f^A, g^A)(\bar{a})\) to be the set of relations \(R\) for which there is an \(R\)-dense path in the tree.

The tree is equivalent to a nondeterministic computation. At each stage, if the map \(g^A\) defines a single automorphism class of \(A\), an element of that class is nondeterministically chosen and fed into the map \(f^A\) to generate the next stage. Of course, there is no guarantee that the process defines a unique relation.

**Example 4.** On a pure set \(\mathfrak{A}\), consider the maps \(f^A\) and \(g^A\) defined by

\[
\begin{align*}
\varphi(X, Y, x_1 x_2) &\equiv Y(x_2) \land (x_1 = x_2 \lor X(x_1 x_1)) \\
\psi(X, Y, y) &\equiv \neg X(yy),
\end{align*}
\]

respectively. \(\text{sc-ifp}(f^A, g^A)\) defines the set of linear orders on \(|\mathfrak{A}|\). Initially, the relations \(X\) and \(Y\) are empty. At the first iteration, \(\psi\) defines the whole of \(|\mathfrak{A}|\), which is an automorphism class of the structure \(\langle |\mathfrak{A}|, X, Y \rangle\) so an element is nondeterministically selected — call it ‘1’ — and we set \(Y = \{1\}\). We then add to the relation \(X\) all tuples satisfying \(\varphi\). In this case, there is just one such tuple, \(11\). At the second iteration, \(\psi\) defines all members of \(|\mathfrak{A}|\) other than 1. So long as \(\mathfrak{A}\) is not a singleton, this is an automorphism class, so an element that we shall call ‘2’ is nondeterministically selected and \(Y\) is set to \(\{2\}\). Now \(\varphi\) is satisfied by the tuples 12 and 22 and these are added to \(X\). At each subsequent stage, the choice set contains all elements that have not yet been ordered. If this set is non-empty, it is an automorphism class so one unordered element is selected to be the new maximal element of the ordering. Once all elements have been ordered, the choice set will always be empty so no more tuples will be added.

Although Definition \(\mathfrak{A}\) refers to labels occurring infinitely often in infinite trees, on a finite structure \(\mathfrak{A}\), we need consider only a finite portion of the tree.

**Lemma 5 (Finite Evaluation Lemma).** Let \(\mathfrak{A} \in \text{STRUC}[\sigma]\) and \(\bar{a} \in |\mathfrak{A}|^n\). There is an \(R\)-dense path in \(\text{sc-ifp}(f^A, g^A)(\bar{a})\) if, and only if, there is an \(S\) such that \(\langle R, S \rangle\) labels two nodes on a path in the tree between depth \(d\) and \(2d\), for some \(d\) bounded by a polynomial in \(|\mathfrak{A}|\).

This follows from the finiteness of \(\mathfrak{A}\) and the fact that if \(\langle R_i, S_i \rangle_{i \geq 0}\) labels a maximal path in the tree, \(R_i \subseteq R_{i+1} \subseteq |\mathfrak{A}|^r\) for some \(r\) and \(|S_i| \leq 1\) for all \(i\). We omit the details as the proof is identical to that of Lemma 17 in [DR].

While \(\text{sc-ifp}\) does not, in general, define a unique relation, the relations defined by an application are pairwise isomorphic: the operator is *semideterministic* in the terminology of Gyssens, Van den Bussche and Van Gucht [GVV94].
Proposition 6. If \( R, R' \in \text{sc-ifp} (f^A, g^A) \) then \( (A, R) \cong (A, R') \).

Proof. We show that, whenever \( \langle R, S \rangle \) and \( \langle R', S' \rangle \) label nodes at depth \( d \) in the tree, \( (A, R, S) \cong (A, R', S') \). For \( d = 0 \), this is trivial.

Suppose \( \langle R, S \rangle \) and \( \langle R', S' \rangle \) label two (not necessarily distinct) nodes at depth \( d \) and that the former has a child labelled \( \langle T, U \rangle \) and the latter a child labelled \( \langle T', U' \rangle \). By the inductive hypothesis, there is an isomorphism \( \rho : (A, R, S) \rightarrow (A, R', S') \). (We assume that all isomorphisms and automorphisms mentioned in this proof fix any first-order parameters to the maps.)

Since \( g^A \) is invariant under automorphisms, it defines an automorphism class on \( (A, R, S) \) if, and only if, it does on \( (A, R', S') \). If it does not define an automorphism class, \( U = U' = \emptyset \). If it does, we have \( U = \{ \bar{u} \} \) and \( U' = \{ \bar{u}' \} \) and there must be an automorphism \( \alpha \) of \( (A, R, S) \) such that \( \rho(\alpha(\bar{u})) = \bar{u}' \). In both cases, \( (A, R, U) \cong (A, R', U') \).

Since \( f^A \) is invariant under automorphisms, it follows that \( (A, T, U) = (A, R \cup f^A(\bar{a}, R, U), U) \cong (A, R' \cup f^A(\bar{a}, R', U'), U') = (A, T', U') \).

The restriction to automorphisms that fix parameters is necessary in order to guarantee semideterminism. This is illustrated by the following example.

Example 7. Consider a pure set \( A \) with at least two elements and the maps \( f^A \) and \( g^A \) defined, respectively, by the formulae

\[
\varphi(X, Y, xz) \equiv Y(x) \lor x = z \quad \text{and} \quad \psi(X, Y, x) \equiv \text{true},
\]

treating \( z \) as a parameter to \( \varphi \). Fix an interpretation \( a \in \vert A \vert \) for \( z \). Initially, \( X \) and \( Y \) are both empty. At every iteration of the fixed point, \( \psi \) defines the whole of the set. At the first iteration, this is an automorphism class of the structure \( \langle \vert A \vert, X, Y \rangle \) so some element ‘\( b \)’ is nondeterministically chosen and \( Y \) is set to \( \{ b \} \). \( \varphi \) is now satisfied by \( b \) and \( a \) (which are not necessarily distinct) and these elements are added to \( X \). At subsequent iterations, the structure \( \langle \vert A \vert, X, Y \rangle \) is not a single automorphism class so nothing more will happen: no more elements will be chosen and no more elements will be added to \( X \). Therefore, \( \text{sc-ifp} (f^A, g^A) \) contains the relation \( \{ a \} \) and the relation \( \{ a, b \} \) for each \( b \neq a \in \vert A \vert \): these relations are not isomorphic to each other.

We may use the \( \text{sc-ifp} \) operator to define a deterministic logic but, in doing so, we need to be careful. Since the semideterminism of the operator guarantees that the relations defined are isomorphic to each other, a sentence such as \( \exists \bar{x} R(\bar{x}) \), where the variables occurring in an atomic formulae involving \( R \) are quantified, is either true in \( (A, R) \) for all \( R \in \text{sc-ifp} (f^A, g^A) \) or false for all of them. However, suppose the formula of Example 7 is evaluated in a graph \( G = (V, E) \). The relation \( E \) does not appear in the formula, but the meaning of the formula would appear to be different in \( G \) than in its reduct to the empty vocabulary, as \( G \) may have fewer automorphisms than its reduct. Thus, the logic would fail to have Ebbinghaus’s ‘reduct property’ \[Ebb85\].

To avoid defining something which, by Ebbinghaus’s criteria, is not a logic at all, our semantics of the \( \text{sc-ifp} \) operator requires invariance under automorphisms of the relevant reduct of the structure. That is, if the operands only
mention vocabulary $\tau$, the expression is evaluated in $\mathfrak{A} \models \tau$. This means that the operator is only semideterministic with respect to this reduct but this is still sufficient to guarantee that formulae are deterministic. We denote by $\text{voc}\varphi$ the vocabulary of relations mentioned in $\varphi$.

**Definition 8.** Let $\sigma$ be a vocabulary. The formulae of $\text{SC-IFP}[\sigma]$ are the least set containing all atomic $\sigma$-formulae and closed under first-order operations and the following rule.

- If $\varphi, \psi \in \text{SC-IFP}[\sigma, X, Y]$, $\bar{x}$ and $\bar{y}$ are tuples of variables with $|\bar{x}| = \text{ar}(X) = r$ and $|\bar{y}| = \text{ar}(Y)$, $\bar{u}$ is an $r$-tuple of variables drawn from $u_1 \ldots u_k$ and each $Q_i$ is a quantifier $\exists$ or $\forall$ then $Q_1u_1 \ldots Q_ku_k \text{sc-ifp}_{X,Y,\bar{x},\bar{y}}(\varphi; \psi)(\bar{u})$ is a formula of $\text{SC-IFP}[\sigma]$. The free variables of this formula are the free variables of $\varphi$ and $\psi$ except those in $\bar{x}$ or $\bar{y}$.

The semantics is that of first-order logic with the addition of the following rule.

- Let $\Phi \equiv \bar{Q} \bar{u} \text{sc-ifp}_{X,Y,\bar{x}}(\varphi; \psi)(\bar{u})$ with free variables $z_1 \ldots z_n$ and let $\tau = \text{voc}\Phi$. If $\bar{a} \in |\mathfrak{A}|^n$, we write $(\mathfrak{A}, \bar{a}) \models \Phi$ if, and only if, $(\mathfrak{A}, R) \models \bar{Q} \bar{u} \text{R}^{\bar{u}}(\bar{u})$ for all $R \in \text{sc-ifp}$ $(f^{\mathfrak{A}}_{\varphi} \upharpoonright \tau(a, -, -), f^{\mathfrak{A}}_{\psi} \upharpoonright \tau(a, -, -))$.

Note that, since $\text{sc-ifp}$ is semideterministic, if $(\mathfrak{A}, R) \models \bar{Q} \bar{u} \text{R}^{\bar{u}}(\bar{u})$ for at least one $R$ defined by the fixed point, then it is true for all of them.

The logic we have defined is broadly the same as $\text{FO+IFP}_{c,s}$ defined by Gire and Hoang in [GH98], but there are a number of important differences. Firstly, their logic could be more properly denoted $\text{IFP}_{c,s}(\text{IFP})$ as it has formulae only of the form $\bar{Q} \bar{u} \text{sc-ifp}_{X,Y,\bar{x}}(\varphi; \psi)(\bar{u})$ where $\varphi$ and $\psi$ are IFP formulae. That is, they do not allow either nesting or first-order combinations of fixed points, whereas our definitions permit both freely. Secondly, $\text{FO+IFP}_{c,s}$ does not allow parameters to fixed-point expressions which, as we show in the next section, has a significant effect on the expressive power of the logic. Thirdly, Gire and Hoang consider automorphisms of the whole structure rather than the reduct to the vocabulary of the formula. This has the undesirable effect that the meaning of a formula may depend on relations not mentioned in the formula.

In addition to the formal differences above, it is also convenient to allow simultaneous fixed-points, such as $\bar{Q} \bar{u} \text{sc-ifp}_{X_1, \ldots, X_n, Y, \bar{x}}(\varphi_1, \ldots, \varphi_n; \psi)(\bar{u})$. As usual, the fixed point defines relation $X_1$ with $X_2$, $\ldots$, $X_n$ considered as auxiliary relations. Note that we permit only one choice formula, as making more than one choice per stage would destroy semideterminism. (Consider making two simultaneous choices from a pure set; the effect is essentially the same as in Example [7] Simultaneous definitions of the form indicated do not affect the expressive power of the language, as can be shown by standard techniques for combining several relations into a single relation of wider arity, though we must use a coding, such as that in [DR], that does not interfere with automorphisms of the structure. We shall freely use simultaneous definitions from this point.

In [DR], we showed that the logics C-IFP, NIO and IFP $+$ $\delta$ (from [BG00]) have equal expressive power. It is natural to ask why we consider here the symmetric version only of C-IFP. The answer is that the symmetric versions of the
other two do not lead to sensible logics. IFP with symmetric δ is not semideterministic as it is possible to write a formula that makes two simultaneous choices with the effects discussed in the previous paragraph. On the other hand, a symmetric nio operator seems to be too restricted. When constructing a relation, at each stage, it would only be able to add a tuple from a definable automorphism class: it is not even obvious that such an operator could simulate IFP.

3 The Rôle of Parameters

In the previous section, we defined the logic SC-IFP, allowing fixed-point expressions to take first-order parameters and observed that Gire and Hoang’s logic did not allow such parameters. Here, we show that the parameterless fragment of SC-IFP is much weaker than the whole logic. We denote by pSC-IFP this fragment, which is defined as the set of formulae of SC-IFP in which no occurrence of the fixed-point operator takes first-order parameters.

Consider structures over a vocabulary with a single binary relation ≈, interpreted as an equivalence relation. Call an equivalence relation even if all of its classes are of even cardinality. The following formula defines the class of even equivalence relations, by saying that no element z is in a class of odd cardinality.

\[
\chi \equiv \theta \land \neg \exists z \text{sc-ifp}_{P,Q,R,xy}(\varphi_P, \varphi_Q; \psi_R),
\]

where θ states that ≈ is an equivalence relation, P, Q and R are new relation symbols of arity 0, 1 and 2 respectively and

\[
\begin{align*}
\varphi_P & \equiv \forall u \left( u \approx z \rightarrow Q(u) \right) \\
\varphi_Q(x) & \equiv x = z \lor \exists u \left( R(ux) \lor R(xu) \right) \\
\psi_R(xy) & \equiv x \neq y \land x \neq z \land y \neq z \land x \approx y \approx z \land \neg Q(x) \land \neg Q(y).
\end{align*}
\]

At each stage of the evaluation, a pair of elements xy is selected such that x, y and z are equivalent but distinct and neither x nor y is in Q. They and z are added to Q. If all elements of z’s equivalence class are added to Q, it follows that the number of members of the class distinct from z is even, i.e., that the class is of odd cardinality. In contrast, the main result of this section is:

**Theorem 9.** The class of even equivalence relations is not pSC-IFP-definable.

From this and the formula χ above, it is immediate that:

**Corollary 10.** pSC-IFP < SC-IFP.

Towards a proof of Theorem 9, call an equivalence relation k-large if it has more than k equivalence classes and each class contains more than k elements.

**Lemma 11.** On the class of k-large equivalence relations, every \( L^k \) formula is equivalent to one without quantifiers.
Proof. Let \( \mathfrak{A} \) be a \( k \)-large equivalence relation and \( \varphi(\bar{x}) \in \mathcal{L}^k \). Since \( \mathcal{L}^k \) cannot distinguish between \( \mathfrak{A} \)'s equivalence classes, if \( (\mathfrak{A}, \bar{a}) \models \varphi \), the same is true for all \( \bar{b} \) of the same atomic type so \( \varphi \) is equivalent to a disjunction of atomic types. \( \square \)

Lemma 12. Let \( \varphi(\bar{x}) \in \mathcal{L}^k \). Either there is a formula in the language of equality to which \( \varphi \) is equivalent on all \( k \)-large equivalence relations or \( \approx^\mathfrak{A} \) is uniformly \( \mathcal{L}^k \)-definable in \( \langle |\mathfrak{A}|, \varphi^\mathfrak{A} \rangle \) for all \( k \)-large equivalence relations \( \mathfrak{A} \).

Proof. We may assume \( R = \varphi^\mathfrak{A} \) is at least binary; if it is unary, it must be either empty or \( |\mathfrak{A}| \) as \( \mathcal{L}^k \) cannot distinguish between the elements of \( \mathfrak{A} \). By the previous lemma, we may assume that \( \varphi \) is a disjunction of quantifier-free types, which we may write as \( \bigvee_i (\eta_i \land \bigvee_j \alpha_{ij}) \), where the \( \eta_i \) are an enumeration of all possible equality types of the relevant arity and the \( \alpha_{ij} \) are \( \approx \)-types.

If all the \( \bigvee_j \alpha_{ij} \) are trivial, \( R \) is a union of equality types; otherwise, there must be \( i \) and \( m \neq n \) such that \( \mathfrak{A} \models \forall \bar{x} \left[ (\varphi_1(\bar{x}) \rightarrow \varphi(\bar{x})) \land (\varphi_2(\bar{x}) \rightarrow \neg \varphi(\bar{x})) \right] \), where \( \varphi_1(\bar{x}) \equiv \eta_i \land x_m \approx x_n \land \beta \) and \( \varphi_2(\bar{x}) \equiv \eta_i \land x_m \not\approx x_n \land \beta \) or vice-versa, for some quantifier-free formula \( \beta \). We may assume that \( m = 1 \) and \( n = 2 \).

Because \( \varphi_1 \) and \( \varphi_2 \) are both satisfiable, it follows from the symmetry of \( \mathfrak{A} \) that any pair \( a_1 \approx a_2 \) can be extended to a tuple \( \bar{a} \) satisfying \( \varphi_1 \) (and, hence, satisfying \( \varphi \)) and any pair \( a_1 \not\approx a_2 \) to a tuple satisfying \( \varphi_2 \) (and, hence, not satisfying \( \varphi \)) or vice-versa. So, on \( \langle |\mathfrak{A}|, R \rangle \), the following formula, which is equivalent to one in \( \mathcal{L}^k \), defines either \( \approx \) or \( \not\approx \): \( \exists \bar{u} \left[ \eta_i(x y u) \land R(x y u) \land \exists z (\eta_i(x z u) \land \neg R(x z u)) \right] \). \( \square \)

We are now ready to prove Theorem \( \| \). We show that, for any \( k \)-variable pSC-IFP formula \( \Phi \), there are \( k \)-large equivalence relations \( \mathfrak{A} \), which is even, and \( \mathfrak{A}' \), which is not, between which \( \Phi \) cannot distinguish. This is done by showing that the relations defined by every stage of the evaluation of every fixed point \( \Phi \) are defined by the same \( \mathcal{L}^k \) formulae on both equivalence relations. Since no formula of \( \mathcal{L}^k \) can distinguish between \( \mathfrak{A} \) and \( \mathfrak{A}' \), \( \Phi \) cannot either. We shall denote by pSC-IFP\( _n^k \) the fragment of pSC-IFP in which formulae contain at most \( k \) distinct variables and at most \( n \) nested fixed points.

Proof (Theorem \( \| \)). Fix \( k \in \mathbb{N} \). Let \( \mathfrak{A} \) and \( \mathfrak{A}' \) be \( k \)-large equivalence relations, with \( \mathfrak{A} \) having classes of size \( 2k, 2k + 2, \ldots, 4k \) and \( \mathfrak{A}' \) having classes of size \( 2k - 1, 2k + 3, 2k + 4, 2k + 6, \ldots, 4k \). Note that \( \mathfrak{A} \) is even and \( \mathfrak{A}' \) is not.

Let \( \varphi_1, \ldots, \varphi_r \in \mathcal{L}^k \) and \( \mathfrak{A}^* = \langle \mathfrak{A}, \varphi_1^\mathfrak{A}, \ldots, \varphi_r^\mathfrak{A} \rangle \); define \( \mathfrak{A}'^* \) similarly. We claim that, for any formula \( \Phi \in \text{pSC-IFP}_n^k \) and any \( r \), there is an \( \mathcal{L}^k \) formula \( \hat{\Phi} \) such that \( \mathfrak{A}^* \models \Phi \iff \hat{\Phi} \) and \( \mathfrak{A}'^* \models \Phi \iff \hat{\Phi} \). This is proved by induction on the depth \( n \) of nesting of fixed points, assuming inductively that any relations defined so far by fixed points within which \( \Phi \) is nested are defined by the \( \varphi_i \).

The base case \( n = 0 \) is trivial since pSC-IFP\( _0^k = \mathcal{L}^k \)

Assume the claim is true for pSC-IFP\( _{n-1}^k \) and consider \( \Phi \in \text{pSC-IFP}_n^k \). Let \( \Psi \equiv Q \bar{u} \text{sc-ifp}_{X,Y,z}(\varphi; \psi)(\bar{u}) \) be a subformula of \( \Phi \). We may assume that \( \approx \) is either mentioned in \( \Psi \) or is \( \mathcal{L}^k \)-definable from the relations that are. If not, by Lemma 12 the relations mentioned in \( \Psi \) are unions of equality types so \( \Psi \) cannot possibly distinguish between \( \mathfrak{A}^* \) and \( \mathfrak{A}'^* \) and we are done.
To evaluate $\Psi$ on $A^*$, we must consider automorphisms of $B = A^* \upharpoonright \text{voc} \Psi$. The automorphisms of $B$ are exactly those of $A$ as $A$ and $B$ are $L^k$-definable in each other. Since no two equivalence classes of $A$ are the same size, two tuples $a, b \in |A|^k$ are in the same automorphism class if, and only if, they have the same equality type and $a_i \approx b_i$ for $1 \leq i \leq k$.

As the automorphisms of $B$ are exactly the automorphisms of $A$, it suffices to consider the evaluation of $\Psi$ in $A$ (and, of course, $A'$) and we shall assume that all references in $\Psi$ to relations defined by the $\varphi_i$ are replaced with the formulae defining them. After this substitution, $\Psi$ still has at most $k$ distinct variables.

Let $\langle P_i, Q_i \rangle_{i \geq 0}$ be the labels on a maximal path in $T^A_\Psi$ and $\langle R_i, S_i \rangle_{i \geq 0}$ the labels on a maximal path in $T^{A'}_\Psi$. Our second claim is that, for all $i \geq 0$, $Q_i = S_i = \emptyset$ and there are formulae $\theta_i \in L^k$ such that $P_i = \theta_i^A$ and $R_i = \theta_i^{A'}$.

This is trivial for $i = 0$ as $P_0 = Q_0 = R_0 = S_0 = \emptyset$ by definition. Suppose $P_i = \theta_i^A$, $R_i = \theta_i^{A'}$ and $Q_i = S_i = \emptyset$. $Q_{i+1}$ is the result of evaluating $\psi$ in the structure $(A, P_i, Q_i)$. Since $\psi \in pSC-IFP^k_n$, it follows by the inductive hypothesis of the first claim that it is equivalent to some $\hat{\psi} \in L^k$ which, by Lemma 11, we may assume to be quantifier-free. As no quantifier-free formula defines an automorphism class of $A$, $Q_{i+1} = \emptyset$. Similarly, $S_{i+1} = \emptyset$.

$P_{i+1}$ is the result of evaluating $\varphi$ in $(A, P_i, Q_{i+1})$. By the inductive hypothesis of the first claim, $\varphi$ is equivalent on $A$ and $A'$ to some $\hat{\varphi} \in L^k$; by the inductive hypothesis of the second claim, $P_i = \theta_i^A$. It follows that $P_{i+1} = (\theta_i \vee \hat{\varphi})^A$. Similarly, $R_{i+1} = (\theta_i \vee \hat{\varphi})^{A'}$ and the second claim is proven.

By Lemma 5, there is a $d$ such that, for all $i \geq d$, $P_i = P_d$ and $R_i = R_d$. Therefore, $\Psi \equiv Q_d \theta_d(u)$ and the first claim is proven. A simple pebble-game argument shows that no $L^k$ formula can distinguish between $A$ and $A'$. D

It can be shown, using similar techniques, that there is no formula of SC-IFP$_1$ (even with parameters) that defines the class of graphs all of whose components are of even cardinality. This class can, however, be defined by nesting applications of fixed-point operators. Therefore, SC-IFP$_1 <$ SC-IFP$_2$: the ability to nest fixed points also increases the expressive power of the logic. We conjecture that the SC-IFP nesting hierarchy is strict, i.e., that SC-IFP$_n <$ SC-IFP$_{n+1}$ for all $n$. There is no obvious way to simulate a formula using $n$ nested fixed points with one using fewer as each level of nesting may mention a different vocabulary and hence be sensitive to automorphisms of different reducts of the structure.

The results of this section are in contrast to, for example, LFP, IFP, IFP + C, C-IFP and NIO, where neither the ability to supply first-order parameters to fixed points nor to nest fixed points increases the expressive power of the resulting logic (see, e.g., [DR]; the proofs are essentially the same for all of the logics mentioned). Parameters and nesting alter the expressiveness of SC-IFP because they alter the number of automorphisms available to formulae by fixing certain elements or increasing or reducing the number of relevant relations. Although SC-IFP is somewhat unusual in this respect, it remains a reasonable logic. It is, for example, regular in the sense of Ebbinghaus [Eb85].
4 Expressive Power

It is easy to see that SC-IFP is strictly more powerful than IFP. Any formula of IFP is equivalent to one of the form $\exists y (\text{ifp}_X, \bar{x}) (\bar{y} \ldots \bar{y})$ for some first-order $\varphi$ and this can be translated directly as $\exists y \text{sc-ifp}_{X,Y,\bar{z}} (\varphi; \psi) (\bar{y} \ldots \bar{y})$, for any syntactically appropriate $\psi$, as the choice part of the induction is ignored. We saw in the previous section that the class of even equivalence relations is SC-IFP-definable but it is easy to show that it is not definable in IFP.

In the remainder of this section, we consider the close relationship between SC-IFP and the graph isomorphism problem, GI. This is the following query:

**Definition 13.** Let $\sigma = \langle A^1, B^1, E^2 \rangle$ and let $A$ and $B$ be graphs on disjoint vertex sets. Denote by $A + B$ the $\sigma$-structure $\langle |A| \cup |B|, |A|, |B|, E_A \cup E_B \rangle$. The graph isomorphism problem is the boolean query $\{ A + B : A \cong B \} \subset \text{STRUC}[\sigma]$.

For our purposes, nothing is lost in restricting our attention to graph isomorphism rather than considering the isomorphism problem for general structures. The reader who would prefer to think in terms of general structure isomorphism may view graphs as a notational convenience.

**Lemma 14.** Let $\sigma$ be a vocabulary. There is a formula $\rho_\sigma(\bar{u}, \bar{v})$ of SC-IFP such that, for any structure $A$ in a vocabulary including $\sigma$, $(A, \bar{a}, \bar{b}) \models \rho_\sigma$ if, and only if, $\bar{a}$ and $\bar{b}$ are exchanged by some automorphism of $A \upharpoonright \sigma$.

**Proof.** Let $|\bar{u}| = |\bar{v}| = k$. To determine whether there is an automorphism of $A \upharpoonright \sigma$ that exchanges $\bar{u}$ and $\bar{v}$, we define a new $k$-ary relation $P$ containing just those two tuples and ask if this relation is an automorphism class. Since $P$ is a new relation, it will be an automorphism class of $(A \upharpoonright \sigma, P)$ if, and only if, there is an automorphism of $A \upharpoonright \sigma$ that exchanges $\bar{u}$ and $\bar{v}$.

Let $X, Y, P$ and $Z$ be new relation symbols of arity $0, 0, k$ and $k$, respectively.

$$\rho_\sigma(\bar{u}, \bar{v}) \equiv \text{ifp}_{X,P,\bar{x}}(\theta, \bar{x} = \bar{u} \lor \bar{x} = \bar{v})$$

$$\theta \equiv \text{sc-ifp}_{Y,Z,\bar{y}}(\text{true} \land \exists \bar{x} Z(\bar{x}); P(\bar{y})),$$

where ‘true’ is some tautology mentioning all of the relation symbols in $\sigma$. $\theta$ attempts to choose a tuple from $P$: this will succeed only if $P$ is a non-trivial automorphism class of $A \upharpoonright \sigma$. (Note that $\bar{u}$ and $\bar{v}$ are not parameters to $\theta$ so we are not restricted to automorphisms that fix these tuples.) Before the first iteration of $\rho_\sigma$, $X$ is false and $P$ is empty. Since $P$ is not a non-trivial automorphism class, $X$ remains false after the first iteration but $P$ is set to $\{ \bar{u}, \bar{v} \}$. At the second iteration, $X$ will become true if $P$ is an automorphism class but nothing else will change at any future iteration. Hence, $\rho_\sigma$ is satisfied if, and only if, the tuples interpreting $\bar{u}$ and $\bar{v}$ are exchanged by some automorphism of $A \upharpoonright \sigma$. \qed

**Theorem 15.** The graph isomorphism problem is SC-IFP-definable.

**Proof (sketch).** Let $A$ be a structure in the vocabulary of Definition 13. $A$ is an instance of the graph isomorphism problem if $A^A$ and $B^A$ partition $|A|$ and every...
edge in $E^A$ is within one partition: this is first-order definable. To determine whether $\mathfrak{A}$ is a yes instance, we first, construct the relation $C = (A^\mathfrak{A} \times A^\mathfrak{A}) \cup (B^\mathfrak{A} \times B^\mathfrak{A})$. $\mathfrak{A}$ is a yes instance if, and only if, $\langle |\mathfrak{A}|, C, E \rangle$ contains two vertices, one in each $C$-component, that are exchanged by an automorphism. This is SC-IFP-definable by the previous lemma.

Recall that a query $Q$ is *Cook-reducible* to a query $R$ if $Q$ is decided by a polynomial-time Turing machine with an oracle for $R$, i.e., $Q \in P^R$. We denote by $P^{\text{GI}}$ the class of problems that are Cook-reducible to graph isomorphism. SC-IFP $\leq P^{\text{GI}}$ as any SC-IFP formula requires at most a polynomial number of iterations, each requiring polynomially many calls to the oracle.

Let $\text{IFP}(\text{GI})$ be the logic obtained by closing first-order logic under inflationary fixed points and vectorized generalized quantifiers for graph isomorphism (see [Daw95,EF99] for a definition of vectorized quantifiers). By [Daw95, Theorem 5.6], $\text{IFP}(\text{GI})$ captures $P^{\text{GI}}$ on ordered structures. It is immediate from Theorem 15 that $\text{IFP}(\text{GI}) \leq \text{SC-IFP}^*$, the closure of SC-IFP under first-order reductions. (We require closure under reductions to deal with vectorizations; it is not clear whether SC-IFP is so closed.) We conjecture that SC-IFP* is strictly more powerful than IFP(GI) as there seems to be no way for the latter to simulate choice on arbitrary structures. However, on ordered structures, choice can be simulated by taking the least element of any set. This gives,

**Theorem 16.** On ordered structures, SC-IFP* = IFP(GI) = $P^{\text{GI}}$. □

What, then, is the expressive power of SC-IFP* on unordered structures? Suppose we can define an order of the automorphism classes of a structure: that is, a linear pre-order whose equivalence classes are precisely the automorphism classes. The elements of each automorphism class can be linearly ordered using the techniques of Example 4 so the pre-order can be refined to a linear order. This linear order can be used to express any $P^{\text{GI}}$ property of the structure.

It is easy to see that SC-IFP* captures $P^{\text{GI}}$ on pure sets and on any class of structures with only unary relations. We have seen in Example 4 how to linearly order a pure set. In the second case, the atomic types of elements are automorphism classes and, for any fixed vocabulary, there are only finitely many atomic types and they can be ordered even by a first-order formula.

Results on IFP + C allow us to show that SC-IFP* captures $P^{\text{GI}}$ on large classes of graphs. A linear ordering of the $C^k$ types realized in a structure can be produced by an iterated refinement scheme known as the $(k-1)$-dimensional Weisfeiler–Lehman method, which is expressible in IFP + C [CFI92, Ott97]. It can be shown that this ordering of $C^k$ types is also SC-IFP-definable.

A class of $G$ graphs contains almost all graphs if, as $n \to \infty$, the proportion of all $n$-vertex graphs that are in $G$ tends to 1. A logic captures a complexity class $C$ on almost all graphs if it captures $C$ on a class containing almost all graphs.

**Theorem 17.** SC-IFP* captures $P^{\text{GI}}$ on almost all graphs.

*Proof (sketch).* Immerman and Lander show that the $C^2$-equivalence classes of almost all graphs are exactly the automorphism classes [IL90]. For any $k$, the
ordering of the $C^k$ types is SC-IFP-definable and the result follows from the observations above. □

**Theorem 18.** SC-IFP$^*$ captures $P^{GI}$ on any class of structures which is closed under disjoint unions and on which IFP + C captures $P$.

*Proof.* It follows from [Ott97, Theorem 4.22] that, if IFP + C captures $P$ on a class of structures which is closed under disjoint unions, there is a $k$ such that the $C^k$-equivalence classes of $k$-tuples are automorphism classes. □

It follows from Theorems 17 and 18 that SC-IFP$^*$ captures $P^{GI}$ on many natural classes of graphs, such as trees [IL90], planar graphs [Gro98] and graphs of bounded tree width [GM99] or genus [Gro00].

Any class of structures on which IFP + C captures $P$ has a polynomial time isomorphism algorithm (namely, the $k$-dimensional Weisfeiler–Lehman method, for some $k$) so one might hope that Theorem 18 could be improved to SC-IFP$^*$ capturing $P$ on such a class of structures. Although the isomorphism problem for these structures is in $P$, SC-IFP formulae may construct new structures internally and there might not be a polynomial-time isomorphism algorithm for these structures. For example, IFP + C captures $P$ on ordered graphs but we have already shown that SC-IFP$^*$ captures $P^{GI}$ on that class. In particular, an SC-IFP formula can take an ordered instance of GI, ‘forget’ the ordering and ask if the two graphs encoded in the resulting structure are isomorphic. Hence, we cannot expect SC-IFP$^*$ to capture $P$ on all classes of structures on which IFP + C does, unless graph isomorphism is itself in $P$.

**Conclusion.** Gire and Hoang’s specified symmetric choice logic is a possible candidate for a logic for isomorphism-invariant polynomial-time properties. That is, it is one of a few logics that have been shown to properly extend IFP + C while being contained in $P$. There is no known example of a polynomial-time property that it cannot express. Towards a better understanding of this logic, in this paper we develop the semantics of symmetric choice and investigate its expressive power. We show that it is closely related to the complexity class $P^{GI}$. We also develop some techniques for establishing lower bounds, using them to demonstrate the power of parameters and of nesting.

**References**


Positive Games and Persistent Strategies

Jacques Duparc
LuFG Mathematische Grundlagen der Informatik,
RWTH-Aachen, D-52056 Aachen
Fax: +49/241/8022-215
duparc@informatik.rwth-aachen.de

Abstract. At CSL 2002, Jerzy Marcinkowski and Tomasz Truderung presented the notions of positive games and persistent strategies [8]. A strategy is persistent if, given any finite or infinite run played on a game graph, each time the player visits some vertex already encountered, this player repeats the decision made when visiting this vertex for the first time. Such strategies require memory, but once a choice is made, it is made for ever. So, persistent strategies are a weakening of memoryless strategies.

The same authors established a direct relation between positive games and the existence of persistent winning strategies. We give a description of such games by means of their topological complexity. In games played on finite graphs, positive games are unexpectedly simple. On the contrary, infinite game graphs, as well as infinite alphabets, yield positive sets involved in non determined games.

Last, we discuss positive Muller winning conditions. Although they do not help to discriminate between memoryless and LAR winning strategies, they bear a strong topological characterization.

1 Introduction

The theoretical framework of infinite two-player games has found growing interest in theoretical computer science. With applications to verification and synthesis of reactive programs, the usual setting are the finite-state games where two players move a token along the edges of a finite game graph. The player (0 or 1) to whom the current vertex belongs, pushes the token to one of the successors of this vertex. Practical applications imposing simple winning conditions, these games are usually determined, and one can decide the winner, and compute a winning strategy. Here, for algorithmic concerns, how much memory is needed to win plays a crucial role. Indeed, memoryless (or positional) strategies for Muller games can be verified in polynomial time, provided winning conditions for the opponent, together with the game graph itself, are given as input.

Recently, a weakening of the notion of memoryless strategy was introduced to obtain results on complexity of deciding graph games with winning conditions defined by formulas from fragments of $LTL\mathbb{N}$.

* The author sincerely thanks Erich Grädel for numerous remarks and corrections.
Definition 1. A strategy $\sigma$ for player $P$ is persistent if for each play $v_1v_2\ldots v_k$, played according to $\sigma$, $v_i = v_j$ and $P$ is to move at $v_i$, then $v_{i+1} = v_{j+1}$.

A persistent strategy behaves like a becoming positional one. Uncertainty is confined to the vertices not encountered yet. But, once every vertex has been visited, the strategy definitively becomes positional.

It is very useful to know of some general property of the winning conditions that gives the existence of persistent strategies among the winning ones. In [8], such a very effective property is pointed out.

Theorem 1 (Marcinkowski & Truderung). If a game on a finite graph is positive for player $P$, and his opponent has a winning strategy, then he has a persistent winning strategy.

Here, a game is positive, if for any two infinite plays $x, y$, where $x$ is a subsequence of $y$, whenever $x$ is winning, $y$ is also winning.

Intuitively, theorem says that, in a positive game for the opponent, if, in the picture below, the left infinite path is winning for the other player, then the right one also.

\[ \text{In other words, there exists a persistent winning strategy if removing loops that are subpaths still preserves the winning condition.} \]

Not all games that admit persistent winning strategies are positive (for one of the players). For example, most parity games are not positive for either player, however the winner has a memoryless - hence persistent - winning strategy.

This paper gives a description of these positive games, by means of their topological complexity. We show that they are quite simple - $\Pi^0_2$ - when the game graph is finite. On the contrary, when the constraint of finite graphs is abandoned, these games can be extremely complicated, and can even give rise to non determined games. Last, we study the prominent example of Muller winning condition. In this context, we show that positive games and $\Pi^0_2$ games coincide.

2 Preliminaries

Given two - finite or infinite - sequences $x, y$, the relation $x \sqsubseteq y$ means that $x$ is a subsequence of $y$. We write $x \sqsubseteq y$ when $x \sqsubseteq y$ holds, but $y \sqsubseteq x$ fails.

Unless mentionned, throughout this paper we consider infinite two-player games played on a directed graph $\mathcal{G} = (v_\infty, V_0, V_1, E)$, where $\{V_0, V_1\}$ is a partition (between the two players 0 and 1) of the set of vertices $V$. The initial vertex where the play starts is $v_\infty$. An infinite play $x$ is nothing but an infinite
sequence of vertices: \( x = v_0v_1v_2 \ldots v_iv_{i+1} \ldots \) such that \( v_0 = v_n \), and for every integer \( i \), \((v_i,v_{i+1}) \in E\). The choice of \( v_{i+1} \) being made by the player to whom \( v_i \) belongs to.

We set \( \mathcal{G}^* \) as the tree of all possible positions inside \( \mathcal{G} \).

\[
\mathcal{G}^* = \{v_0v_1v_2 \ldots v_n \in V^* : v_0 = v_n \land \forall i < n \ (v_i,v_{i+1}) \in E\}.
\]

We write \( \mathcal{G}^\omega \) for the set of infinite plays (the set of infinites branches of \( \mathcal{G}^* \)).

With these notations, the definition of a positive game becomes.

**Definition 2** (Marcinkowski & Truderung [8]). Let \( A \subseteq \mathcal{G}^\omega \),

\( A \) is positive \( \iff \forall x, y \in \mathcal{G}^\omega \ (x \in A \land x \sqsubseteq y) \rightarrow y \in A \).

We equip \( \mathcal{G}^\omega \) with the usual topology - that is the topology induced by the usual topology on \( V^\omega \): the product topology of the discrete topology on \( V \).

In other words, non empty open sets of \( \mathcal{G}^\omega \) take the form \( W V^\omega \cap \mathcal{G}^\omega \) for some set \( W \subseteq \mathcal{G}^* \) of finite words.

We recall that the finite Borel Hierarchy is a sequence \( \Sigma^0_1, \Pi^0_1, \Sigma^0_2, \Pi^0_2, \ldots \) of classes of \( \omega \)-languages over spaces of the form \( \mathcal{G}^\omega \) inductively defined by:

- \( \Sigma^0_1 = \{\text{Open sets}\} \)
- \( \Pi^0_n = \left\{ A^c : A \in \Sigma^0_n \right\} \) \((n > 0)\), where \( A^c \) stands for the complement of \( A \).
- \( \Sigma^0_{n+1} = \left\{ \bigcup_{i \in \mathbb{N}} A_i : \forall i \in \mathbb{N} \ A_i \in \Pi^0_n \right\} \) - countable unions of sets in \( \Pi^0_n \).

This hierarchy classifies sets with respect to their topological complexity, it is ordered by inclusion in the sense that \( \Sigma^0_n \) and \( \Pi^0_n \) are both strictly included in \( \Sigma^0_{n+1} \) and \( \Pi^0_{n+1} \), while \( \Sigma^0_n \) and \( \Pi^0_n \) are incomparable for inclusion. The class \( \Sigma^0_n \cap \Pi^0_n \) is denoted \( \Delta^0_n \).

### 3 Finite Game Graphs

For the general study of positive games, we need to consider the following basic sets.

**Definition 3.** Let \( \alpha = (u, \Lambda) \in \mathcal{G}^* \times \mathcal{P}(V) \),

\[
\Omega_\alpha = \bigcup_{u \sqsubseteq v} (v\mathbb{I}_A \cap \mathcal{G}^\omega) = \left( \bigcup_{u \sqsubseteq v} v\mathbb{I}_A \right) \cap \mathcal{G}^\omega
\]

where \( \mathbb{I}_A = \{y \in V^\omega : \forall a \in A \ a \text{ occurs infinitely often in } y\} \).

So, \( \Omega_\alpha \) is the set of all infinite sequences in \( \mathcal{G}^\omega \) that contain \( u \) as a subsequence, and infinitely many times all vertices of \( \Lambda \). It is \( \omega \)-rational and topologically simple:

**Proposition 1.** Let \( \alpha = (u, \Lambda) \in \mathcal{G}^* \times \mathcal{P}(V) \),

\[
\Omega_\alpha \in \Pi^0_2.
\]
Proof of proposition 1. We assume \( \Lambda = \{a_0, \ldots, a_k\} \). To show that \( \Omega_\alpha \in \Pi_2^0 \), it is enough to see that

\[
\Omega_\alpha = \left( \bigcap_{n \in \omega} \left( \bigcup_{u \subseteq v \in V} v (V^* a_0 V^* a_1 \cdots V^* a_k V^*)^n V^\omega \right) \right) \cap \mathcal{G}^\omega.
\]

We equip \( \mathcal{G}^* \times \mathcal{P}(V) \) with the following partial ordering.

**Definition 4.** Let \( \alpha = (u, \Lambda), \beta = (v, \Gamma) \in \mathcal{G}^* \times \mathcal{P}(V) \),

\[
\alpha \leq_s \beta \iff u \subseteq v \land \Lambda \subseteq \Gamma.
\]

**Lemma 1.**

\( \langle \mathcal{G}^* \times \mathcal{P}(V), \leq_s \rangle \) is a well quasi ordering (WQO).

**Proof of lemma 1.** We first recall Higman’s lemma ([5][11]) which states that given \( \leq \) any WQO on some alphabet \( \Sigma \), the partial ordering on \( \Sigma^* \) defined by \( u \leq v \iff x_i \leq y_{\varphi(i)} \) holds for some strictly increasing \( \varphi : lh(u) \mapsto lh(v) \), is also a WQO.

Clearly, \( \langle V, = \rangle \) is a WQO (only because \( V \) is finite!). Hence, by Higman’s lemma, \( \subseteq \) is a WQO on \( V^* \), from which we derive that \( \subseteq \) is a WQO on \( \mathcal{G}^* \).

- Towards a contradiction, we assume that there exists an infinite decreasing sequence \( \alpha_0 >_s \alpha_1 >_s \alpha_2 >_s \alpha_3 >_s \ldots \). Since \( \mathcal{P}_+(V) \) is finite, there is an infinite subsequence \( \alpha_{i_0} >_s \alpha_{i_1} >_s \alpha_{i_2} >_s \alpha_{i_3} >_s \ldots \) where all elements have same second projection. Let \( u_j \) denotes the first projection of \( \alpha_j \), we obtain \( u_{i_0} \not\subseteq u_{i_1} \not\subseteq u_{i_2} \not\subseteq u_{i_3} \not\subseteq \ldots \), which contradicts the fact \( \subseteq \) is a WQO on \( V^* \).

- Towards a contradiction, let us assume that there exists \( (\alpha_i)_{i \in \mathbb{N}} \) an infinite antichain for \( \leq_s \). Since \( \mathcal{P}_+(V) \) is finite, there exists an infinite subset \( I \subseteq \mathbb{N} \) such that all elements in \( (\alpha_i)_{i \in I} \) have same second projection. Let \( u_i \) denotes the first projection of \( \alpha_i \), we obtain that \( (u_i)_{i \in I} \) is an infinite antichain for \( \subseteq \), which also contradicts the fact \( \subseteq \) is a WQO on \( V^* \).

So \( \leq_s \) is a WQO on \( V^* \times \mathcal{P}(V) \), therefore \( \leq_s \) is also a WQO on \( \mathcal{G}^* \times \mathcal{P}(V) \).

**Definition 5.** We define the mapping \( x \mapsto \alpha_x \) from \( \mathcal{G}^\omega \) to \( \mathcal{G}^* \times \mathcal{P}(V) \) by:

\[
\alpha_x = (u, \Lambda) \in \mathcal{G}^* \times \mathcal{P}(V) \text{ where:}
\]

- \( \Lambda \) is the non empty set of all letters occurring infinitely often in \( x \).
- \( u \) is the shortest sequence such that \( u^{-1} x \in \Lambda^\omega \). (i.e. \( x = uy \) with \( y \in \Lambda^\omega \)).

Or equivalently, \( u \) is the longest prefix of \( u \) that ends with a letter not in \( \Lambda \).

This mapping, the sets \( \Omega_\alpha \), and the WQO \( \leq_s \), are all we need to prove:
Theorem 2. Let $A \subseteq \mathcal{G}^\omega$ be positive, and $x \in \mathcal{G}^\omega$,

1. $x \in A \implies x \in \Omega_{\alpha_x} \subseteq A$,
2. $A = \bigcup_{x \in A} \Omega_{\alpha_x}$,
3. there exists some finite set $\{x_0, \ldots, x_k\} \subseteq A$ such that $A = \bigcup_{i \leq k} \Omega_{\alpha_{x_i}}$,
4. $A \in \Pi^0_2$, and $A$ is an $\omega$-rational set.

Proof of theorem 2: (1) and (2) are immediate from the definition of both $\alpha_x$ and $\Omega_{\alpha}$. (3) is an immediate consequence of lemma (1): we set

$$\min_B = \{\alpha_x : x \in A\}.$$ We let $\min_B$ be the set of all $\leq_s$-minimal elements of $B$. Clearly $\min_B$ forms an antichain for $\leq_s$, hence, by lemma (1) it is finite. Since $\Omega_\beta \subseteq \Omega_\alpha$ clearly follows from $\alpha \leq_s \beta$, we obtain

$$A = \bigcup_{x \in A} \Omega_{\alpha_x} = \bigcup_{\alpha \in B} \Omega_{\alpha} \subseteq \bigcup_{\alpha \in \min_B} \Omega_{\alpha} \subseteq A.$$ Finally (4) comes from the fact $A = \bigcup_{\alpha \in \min_B} \Omega_{\alpha}$, where each $\Omega_{\alpha}$ is both $\omega$-rational and a $\Pi^0_2$-set. Hence $A$ is $\omega$-rational and $\Pi^0_2$ as a finite union of $\omega$-rational $\Pi^0_2$-sets.

This yields the following characterization of positive games played on finite graphs:

Proposition 2. Let $A \subseteq \mathcal{G}^\omega$,

$$A \text{ positive } \iff A = \bigcup_{i \leq k} \Omega_{\alpha_i} \text{ for finitely many } \alpha_0, \ldots, \alpha_k \in \mathcal{G}^* \times \mathcal{P}(V).$$

Proof of proposition 2: ($\implies$) is case (3) of previous theorem. The other direction relies on:

- the class of positive subsets of $\mathcal{G}^\omega$ is closed under union,
- each $\Omega_{\alpha}$ is positive.

Similarly to computational complexity, there are notions of reduction and completeness for topological complexity. What plays there the role of polynomial time reduction, is continuous reduction: given two $\omega$-languages, $A$ and $B$, one says $A$ continuously reduces to $B$ (denoted $A \leq_W B$ since introduced by Wadge in 1974 [15]) if there is a continuous mapping $\varphi$ such that $x \in A \iff \varphi(x) \in B$ holds. Where continuity for such a mapping $\varphi : \mathcal{G}^\omega \rightarrow \mathcal{G'}^\omega$, means that every inverse image of an open set is open - for any $W_B \subseteq \mathcal{G'}^\omega$ there exists some $W_A \subseteq \mathcal{G}^*$ such that $y \in (W_A V^\omega) \cap \mathcal{G}^\omega \iff \varphi(y) \in (W_B V^{\prime\omega}) \cap \mathcal{G'}^\omega$ holds for all $y \in \mathcal{G'}^\omega$. 

\[\varphi\]
The relation $\leq_W$ is a partial ordering which measures the topological complexity. Intuitively $A \leq_W B$ means that $A$ is less complicated than $B$ with regard to the topological structure. This notion of continuous reduction yields the one of completeness for topological classes. Indeed, a set $A$ is $\Sigma_n^0$-complete (resp. $\Pi_n^0$-complete) if it belongs to the class $\Sigma_n^0$ (resp. $\Pi_n^0$) and reduces all sets in the class. For example, $0^*1\{0,1\}^\omega$ is $\Sigma_1^0$-complete, and the set $\emptyset$ of all infinite words over the alphabet $\{0,1\}$ that have finitely many 1 is $\Sigma_2^0$-complete. As a matter of fact, if $A$ is $\Sigma_n^0$-complete, then its complement $A^c$ is $\Pi_n^0$-complete, and both $A$ and $A^c$ are incomparable for $\leq_W$. We write $A \equiv_W B$ when both $A \leq_W B$ and $B \leq_W A$ hold; and $A <_W B$ when $A \leq_W B$ is verified but not $B \leq_W A$. (For background see e.g. [6]).

The main device in working with this measure of complexity is due to Wadge [15]. It is a game that links the existence of a winning strategy for a player to the existence of a continuous function that witnesses the relation $A \leq_W B$:

**Definition 6 (Wadge game).** Given $A \subseteq \Sigma^*_A$, $B \subseteq \Sigma^*_B$, $W(A,B)$ is an infinite two player game between players $I$ and $II$, where players take turns. Player $I$ plays letters in $\Sigma_A$, and $II$ plays finite words over the alphabet $\Sigma_B$. At the end of an infinite play (in $\omega$ moves), $I$ has produced an $\omega$-sequence $x \in \Sigma_A^\omega$ of letters and $II$ has produced an $\omega$-sequence of finite words which concatenated give rise to a finite or $\omega$-word $y \in \Sigma_B^* \cup \Sigma_B^\omega$. The winning condition on the resulting play, denoted here $x^*y$, is the following:

$II$ wins the play $x^*y \iff \text{def } y \text{ is infinite } \land (x \in A \iff y \in B)$.

**Proposition 3 ([15]).** $II$ has a winning strategy in $W(A,B) \iff A \leq_W B$.

We recall that an ordinal is the set of its predecessors. i.e. $\xi + 1 = \{0,1,2,\ldots, \xi\}$. In order to get a finer characterization of the positive sets, we introduce the following sets.

**Definition 7.** Given $\xi$ any countable ordinal, $i \in \{0,1,(0,1)\}$, we define $D^\xi_i$ by:

- $D^\xi_0 = \{x \in (\xi + 1)^\omega : \operatorname{parity}(\min x) \text{ is even}\}$, where $\min x$ is the least ordinal in the longest decreasing initial sequence of $x$, and $\alpha$ is even iff $\alpha = \omega \cdot \beta + n$, for some even $n \in \mathbb{N}$.
- $D^\xi_i = (\xi + 1)^\omega \setminus D^\xi_0$, and
- $D^\xi_{(0,1)} = 0D^\xi_0 \cup 1D^\xi_1$.

The space $(\xi + 1)^\omega$ is equipped with the usual topology, that is basic open sets are of the form $u(\xi + 1)^\omega$ for some finite sequence $u$ of ordinals below $\xi + 1$.

We need two technical results. For $u \in G^\omega$, and $u \in G^*$, $A_\langle u \rangle$ denotes the set of infinite words $x$ such that $ux \in A$

**Lemma 2.** Let $A \subseteq \Sigma^\omega$,

$$A \in \Delta^0_2 \iff \exists ! \xi < \omega_1 \exists ! i \in \{0,1,(0,1)\} \left(A \equiv_W D^\xi_i\right).$$
Lemma 3. Let $A \subseteq G^\omega$, $\theta < \lambda$ be countable ordinals with $\lambda$ limit, $i \in \{0, 1, (0, 1)\}$, and $n \in \mathbb{N}$,

$$D^\theta_0 \leq_W A \equiv_W D^{\lambda+n}_i \implies \exists u \in G^* \begin{cases} A(u) \equiv_W D^{\theta+1}_0 \\
 or \\
 A(u) \equiv_W D^\theta_0.
\end{cases}$$

The previous two lemmas are very similar to results published in [3]. For completeness, we give direct proofs in the appendix. $D_n(\Sigma^0)$ is the class of sets $A = (A_{n-1} \setminus A_{n-2}) \cup (A_{n-3} \setminus A_{n-4}) \cup \ldots$ for open sets $A_0 \subseteq A_1 \subseteq \ldots \subseteq A_{n-1}$.

Lemma 4. Let $A \subseteq G^\omega$ be positive,

$$A \in \Delta^0_2 \implies \exists n \in \mathbb{N} \ A \in D_n(\Sigma^0_1).$$

(i.e. $A \equiv_W D^n_i$ for some $i \in \{0, 1, (0, 1)\}$ and $n \in \mathbb{N}$).

Proof of lemma 4 Towards a contradiction, we assume that $A \in D_n(\Sigma^0_1)$ fails for all integer $n$. This is equivalent to say $D^0_\theta \leq_W A$ holds for any $n$. From proposition [2] we can assume that there exists an integer $k$ such that:

$$A = \bigcup_{i \leq k} \Omega_{\alpha_i} \text{ with } \alpha_0 = (u_0, \ldots), \alpha_k = (u_k, A_k) \in G^* \times P_i(V).$$

Since $A \in \Delta^0_2 \setminus \left( \bigcup_{n \in \mathbb{N}} D_n(\Sigma^0_1) \right)$, there exists some limit cardinal $\lambda$, some integer $n$ and some $i \in \{0, 1, (0, 1)\}$ such that $A \equiv_W D^{\lambda+n}_i$. By lemma [3] given some non limit ordinal $\theta < \lambda$, there exists a sequence $(w_i)_{i \in \mathbb{N}}$ of finite words of $G^\ast$, together with a sequence $(c_i)_{i \in \mathbb{N}} \in 2^\omega$ such that $A(w_i) \equiv_W D^{\theta+1}_{c_i}$. So, in particular $A(w_i) < A(w_j)$ holds for any $i < j$. It follows that there exists an infinite subset $I \subseteq \mathbb{N}$ such that, given any two $i, j \in I$:

$$w_i(\ell h(w_i) - 1) = w_j(\ell h(w_j) - 1) \text{ and } \forall i \leq k \ \forall w \subseteq u_i \ (w \subseteq w_i \iff w \subseteq w_j).$$

In other words, $w_i$ and $w_j$ both end with the same vertex (let us call it $v_{end}$), and they extend exactly the same prefixes of the same words among $\{u_0, \ldots, u_k\}$.

Now consider two different $i, j \in I$, such that $i < j$, one has $A(w_i) < A(w_j)$. Get $\sigma$ any winning strategy for $II$ in the game $\mathbb{W}(D^{\theta+j}_{c_i}, A(w_j))$. We claim that $\sigma$ is winning for $II$ in the game $\mathbb{W}(D^{\theta+j}_{c_j}, A(w_i))$. For this, we let $G^\omega_{v_{end}}$ denote the set of infinite paths in the graph $\mathcal{G}$ with initial vertex $v_{end}$. It is then enough to notice the following:

$$\forall x \in G^\omega_{v_{end}} \ x \in A(w_j) \iff \exists l \leq k \ (u_l \subseteq w_j x \ \land \ x \in I_{A_l}) \iff \exists l \leq k \ (u_l \subseteq w_i x \ \land \ x \in I_{A_l}).$$

Which proves $II$ wins $\mathbb{W}(D^{\theta+j}_{c_j}, A(w_i))$, hence $A(w_j) \leq_W D^{\theta+j}_{c_j} \leq_W A(w_i)$ holds, which contradicts $A(w_i) < W A(w_j)$.

These technical results put together give the following characterization of positive games.
Theorem 3. Let $V$ be finite, and $A \subseteq G^\omega$,

$A$ positive $\implies \begin{cases} A \equiv_W D_i^n \text{ for some } n \in \mathbb{N}, \text{ and } i \in \{0, 1, (0, 1)\} \\
\text{or} \\
A \equiv_W Q^\mathcal{G} \end{cases}$

Proof of theorem \[\triangleright\] From theorem 2(4), we know that if $A$ is positive, then $A$ belongs to $\Pi_0^2$, hence $A \leq_W Q^\mathcal{G}$ holds. If $A <_W Q^\mathcal{G}$ is verified, then $A \in \Delta_0^2$, and by lemma \[\triangleright\] this implies that $A$ is a finite difference of open sets. Hence, the only possibilities are $A \equiv_W D_i^n$ for some integer $n$ and some $i \in \{0, 1, (0, 1)\}$.

To say it differently, any infinite play will sooner or later be restricted to some Strictly Connected Component (SCC) of the game graph. Assume $ux$ and $uy$ are two infinite plays with final parts $x$ and $y$ taking place in the same SCC. The first condition $A \equiv_W D_i^n$ means that $ux \in A$ iff $uy \in A$. The second condition $A \equiv_W Q^\mathcal{G}$ means that for at least one SCC, there exists such particular $u, x, y$ such that $ux \notin A$, $uy \in A$, and $x \sqsubseteq y$.

Remark 1. The other direction of the implication in theorem \[\triangleright\] does not hold: consider the open set $0102^\omega$ inside the following graph. It is not positive, and its complement is not positive either.

![Graph](image)

To conclude with finite graphs, we would like to focus on the particular case of positive subsets of $\Sigma^\omega$, for some finite alphabet $\Sigma$. For the rest of this section, positive set means set that satisfy $\forall x, y \in \Sigma^\omega \ ( (x \in A \land x \sqsubseteq y) \implies y \in A )$. Replacing $G$ by $\Sigma$, Theorem 1 still holds for such winning sets. Amazingly, getting rid of the constraint of the graph yields even sharper results. First we need two easy lemmas.

Lemma 5. Let $A \subseteq \Sigma^\omega$ be positive, $B \subseteq \Gamma^\omega$ any non empty set, and $b \notin \Gamma$,

$B \leq_W A \implies (B \cup \Gamma^* b \Gamma^\omega) \leq_W A$

where $B'$ is the subset of $(\Gamma \cup \{b\})^\omega$, defined by $B' = (B \cup \Gamma^* b \Gamma^\omega)$.

In particular, $A \equiv_W D_1^\xi$ always fails for $\xi > 0$.

Proof of lemma \[\triangleright\] Since $B$ is non empty, $A$ must be non empty, hence there exists some $\alpha = (u, A) \in \Sigma^* \times \mathcal{P}(\Sigma)$ such that $\Omega_\alpha \subseteq A$. Now consider any winning strategy $\sigma$ for $II$ in $W(B, A)$, the following strategy is also winning for $II$ in $W(B', A)$: play $\sigma$ as long as $II$ does not play $b$, play some sequence $y \in u \mathcal{I}_A$ when $I$ plays $b$. \[\triangleright\]
Lemma 6. Let $A \subseteq \Sigma^\omega$,

$$A \text{ positive } \implies A \not\subseteq_W A^C.$$  

Proof of lemma 6: First of all, in $\Sigma^0_1 \cup \Pi^0_1$ the only sets that verify $A \subseteq_W A^C$ (called self-dual) are the $\Delta^0_1$-complete ones \[1,5\]. But if $A$ is both positive, and satisfies $u\Sigma^0 \subseteq A^C$ (for some finite word $u$), then $A = \emptyset$. This shows that there exists no positive self-dual set inside $\Sigma^0_1 \cup \Pi^0_1$. Since $\Pi^0_2$-complete sets are non self-dual (see \[2,15\]), the only remaining problem are sets in $\Delta^0_2 \setminus (\Sigma^0_1 \cup \Pi^0_1)$. By lemma 4 this is the one of the sets of the form $A \in D_n(\Sigma^0_1)$ for some integer $n$. So, towards a contradiction, we assume that both $A \subseteq_d A^C$, and $A \in \bigcup_{n \in \mathbb{N}} D_n(\Sigma^0_1) \setminus (\Sigma^0_1 \cup \Pi^0_1)$ hold. It follows that there exists some integer $n > 1$ such that $D_n(0,1) \equiv_W A$. By lemma 5 this leads to $D_n(0,1) \subseteq_W A$, where $D_n(0,1)$ is the subset of $\{0,1,\ldots,n,b\}^\omega$, defined by $D_n(0,1) = D_n(0,1) \cup \{0,1,\ldots,n\}^* b(0,1,\ldots,n,b)\omega$. But clearly, $D_n(0,1) \equiv_W D_0^{n+1}$ holds, which contradicts the fact $A \equiv_W D_n(0,1)$. \[6\]

Theorem 4. Let $\Sigma$ be finite, and $A \subseteq \Sigma^\omega$,

$$A \text{ positive } \implies \begin{cases} A \equiv_W \emptyset \text{ or } A \equiv_W \mathcal{D}^n \ (n \in \mathbb{N}) \text{ or } A \equiv_W \mathcal{Q}^C. \end{cases}$$

Proof of theorem 4: From remark 2,4, we know that if $A$ is positive, then $A$ belongs to $\Pi^0_2$, hence $A \subseteq_W \mathcal{Q}^C$ holds. If $\mathcal{Q}^C \subseteq_W A$ fails, then $A \in \Delta^0_2$. Hence by lemma 4 $A$ is a finite difference of open sets, or in other words: $A \equiv_W \mathcal{D}^n_i$ for some integer $n$ and some $i \in \{0,1,\ldots,1\}$. But lemmas 6 and 5 prohibit respectively the case $i = (0,1)$, and the case $i = 1$ (except for $n = 0$, which corresponds to $A = \emptyset$). So the only remaining possibilities are $A \equiv_W \mathcal{D}^n_0$ for all $n \in \mathbb{N}$, with $A = \emptyset \equiv_W \mathcal{D}^0_0$. Examples for the first and last case are $A = \emptyset$, and $A = \mathcal{Q}^C$. The only one example for the case $A \equiv_W \mathcal{D}^0_0$ is $A = \Sigma^\omega$. Finally, for $A \equiv_W \mathcal{D}^n_0$ with $n > 0$, set $A = \Omega_{\alpha_0} \cup \Omega_{\beta_1}$ where $\alpha_0$ and $\alpha_1$ are defined by:

$$\alpha_0 = (u_0, \Lambda_0) \text{ with } u_0 = \varepsilon, \Lambda_0 = \{0\},$$

$$\alpha^n_1 = (u^n_1, \Lambda_1) \text{ with } u^n_1 = \overbrace{01010}^{n \text{ letters}}, \Lambda_1 = \{1\}.$$  

By $u^n_1 = \overbrace{01010}^{n \text{ letters}}$, we mean $u^n_1 = 0, u^n_1^{k+1} = 0u^n_1^{2k},$ and $u^n_1^{2k+2} = 1u^n_1^{2k+1}$. The set $A = \Omega_{\alpha_0} \cup \Omega_{\beta_1}$ is both positive, and satisfies $A \equiv_W \mathcal{D}^n_0$. \[4\]

4 Infinite Game Graphs

While working with finite graphs, we saw that positive sets are very simple objects. On the contrary, when working with infinite game graphs, a first remark
is that any set is homeomorphic to a positive one. Precisely, given any alphabet $\Sigma$, one can design a game graph $G = (v_0, V_0, V_1, E)$ such that $V = \Sigma^*$, $v_0 = \varepsilon$ (the empty word) and $E = \{(u, ua) : \text{ any } u \in \Sigma^*, a \in \Sigma\}$ - the partition $V = \{V_0, V_1\}$ doesn’t matter. Clearly $G^\omega$ and $\Sigma^\omega$ are homeomorphic, and for two infinite sequences $x, y \in G^\omega$, $x \subseteq y$ implies $x = y$; so that any subset of $G^\omega$ is positive. Therefore working with positive sets in this general setting is totally useless.

The fundamental reason of this vacuity is the capacity of the graph to control positiveness. So we may ask the question in a different setting. We saw that positive subsets of $\Sigma^\omega$ are simpler than subsets of $G^\omega$ when the alphabet, respectively the graph, is simple. Therefore, the idea of considering infinite alphabets instead of infinite graphs should here also, lead to a completely different answer. It certainly does, however, even if the condition $\forall x, y \in \Sigma^\omega (\exists x \in A \land x \subseteq y \rightarrow y \in A)$ is rarely satisfied, even these positive sets may be extremely complicated. Amazingly, when the alphabet is finite, all such sets are in $\Pi^0_2$, but as soon as $\Sigma$ becomes infinite, they reach such high levels of complexity that they make determinacy fail.

We recall that, given a tree $T$ on an alphabet $\Sigma$, and a winning set $A \subseteq \Sigma^\omega$, the Gale-Stewart game $G(T, A)$ is an infinite two player game, where players (I and II) take turn playing letters in $\Sigma$. Player I begins, and all positions must remain inside $T$ (otherwise the first player to exit this tree loses). Then, if both players only reach positions inside $T$, the play $(x \in \Sigma^\omega)$ is an infinite branch of this tree. Player I wins if $x \in A$, player II wins otherwise. It is well known that this game is determined as soon as $A$ is Borel [9], and for sets of higher level in the projective hierarchy, determinacy was shown to be equivalent to large cardinal hypotheses [10].

**Proposition 4.** (in ZF+BPI) there exists a positive set $A \subseteq \mathbb{N}^\omega$, and a tree $T \subseteq \mathbb{N}^*$ such that $G(T, A)$ is not determined.

The proof - widely inspired by folklore - takes place in usual Set Theory (ZF), with some additional choice, namely BPI (Boolean prime ideal theorem). BPI stands for the assumption that every Boolean algebra has a prime ideal. It is a common statement in algebra, much weaker than the full Axiom of Choice. However it is elsewhere independent from DC (Dependent Choice) another weak version of the axiom of choice, used to prove large amounts of mathematics.

**Proof of proposition 4** We describe the tree $T$, and the winning set $A$ later. First, we let $U$ be a free ultrafilter over $\mathbb{N}$. We recall that filters over $\mathbb{N}$ are subsets $\mathcal{F} \subseteq \mathcal{P}(\mathbb{N})$ that satisfy:

- $U \in \mathcal{F} \land V \in \mathcal{F} \implies U \cap V \in \mathcal{F}$
- $U \in \mathcal{F} \land U \subseteq V \implies V \in \mathcal{F}$
- $\emptyset \notin \mathcal{F}$

An ultrafilter is a filter with the additional property that for all $U \subseteq \mathbb{N}$, either $U \in \mathcal{F}$ or $(\mathbb{N} \setminus U) \in \mathcal{F}$ holds.
The filter \( \{ U \subseteq \mathbb{N} : \mathbb{N} \setminus U \text{ is finite} \} \) of all co-finite subsets of \( \mathbb{N} \) is called the Fréchet filter, and any ultrafilter that contains (extends) the Fréchet filter is called a free ultrafilter. We need the Boolean Prime Ideal theorem to ensure that the Fréchet filter can be embedded into an ultrafilter. So, we assume that there exists \( U \), some free ultrafilter over \( \mathbb{N} \). We set \( A \) as the following set

\[
A = \{ y \in \mathbb{N}^\omega : \mathcal{I}(y) \in U \},
\]

where \( \mathcal{I}(y) \) denotes the image of \( y \in \mathbb{N}^\omega \) (regarded as a function from \( \mathbb{N} \) to \( \mathbb{N} \)).

It is immediate to see that \( A \) is positive, since given any two infinite sequence \( x, y \in \mathbb{N}^\omega, x \subseteq y \) implies \( \mathcal{I}(x) \subseteq \mathcal{I}(y) \). Therefore, if \( x \) is in \( A \), then \( \mathcal{I}(x) \) is in \( U \), hence \( \mathcal{I}(y) \) is also in \( U \), therefore \( y \in U \).

Instead of formally describing the tree \( T \), we explicit the rules of the play that it allows and forbids:

- As first move, player \( I \) is allowed to play any integer \( x_0 \). But once this is done, the next \( x_0 \) moves of both players are restricted. Player \( I \) must precisely play \( x_0 - 1, x_0 - 2, \ldots, 0 \), while player \( II \) can only play \( 0, 0, \ldots, 0 \).
- Then player \( II \) can play any \( x_1 > x_0 \), forcing player \( I \) to play \( x_2 > x_1 \) as next move. From this point, the next \( x_1 - x_0 \) moves are also restricted to \( 0, 0, \ldots, 0 \) for player \( II \), and precisely \( x_2 - 1, x_2 - 2, \ldots, x_1 \), for player \( I \).
- Then player \( I \) can play any \( x_3 > x_2 \), forcing player \( II \) to play \( x_4 > x_2 \). The next \( x_4 - x_3 \) moves being \( 0, 0, \ldots, 0 \) for player \( II \), and \( x_4 - 1, x_4 - 2, \ldots, x_3 \) for player \( I \).

And so on... We remark that

- if one skips all occurrences of \( 0 \), it appears that \( II \) played exactly the infinite strictly increasing subsequence \( x_1 < x_3 < x_5 < \ldots \)
- \( I \) played \( x_0, x_0 - 1, \ldots, 0, x_2, x_2 - 1, \ldots, x_1, x_4, x_4 - 1, \ldots, x_3, x_5 \ldots \)
- The integers played during this run \( x \) are precisely \( [0, x_0] \cup \bigcup_{i \in \mathbb{N}} [x_{2i+1}, x_{2i+2}] \).

In the rest of the proof we simply forget about the part of the play that is totally imposed. We concentrate on the subsequence \( x_0, x_1, x_2, \ldots \) - the full play being easily reconstructed from it.

To show that \( G(T, A) \) is not determined, we first assume, towards a contradiction, that \( I \) has a winning strategy \( \sigma \). We let \( II \) play as follows: after \( I \) plays \( x_0 \), \( II \) plays any \( x_1 > x_0 \), then \( I \) answers with \( x_2 \), but then \( II \) considers a second play - let’s call it fake - of the game \( G(T, A) \) where \( I \) still applies \( \sigma \), but \( II \), in this fake game, copies \( I \)'s answers in the original game with a shift, skipping \( x_0 \). Similarly, player \( II \), in the original game, copies \( I \)'s answers in the fake game, with here also a shift. See next picture where \( x_i' \) stands for \( 1 + x_i \).

Since \( \sigma \) is a winning strategy for \( I \), it follows: \( \mathcal{I}(x_{\text{original}}) = \{0, x_0\} \cup [x_1, x_2] \cup \bigcup_{i \in \mathbb{N}} [x_{2i+3}, x_{2i+4}] \in U \), and \( \mathcal{I}(x_{\text{fake}}) = \{0, x_0\} \cup \bigcup_{i \in \mathbb{N}} [x_{2i+2}, x_{2i+3}] \in U \). Since \( U \) is a filter, \( \mathcal{I}(x_{\text{original}}) \cap \mathcal{I}(x_{\text{fake}}) = \{0, x_0\} \in U \). Elsewhere \( \mathbb{N} \setminus \{0, x_0\} \) belongs to \( U \) by very definition. This yields \( [0, x_0] \cap (\mathbb{N} \setminus [0, x_0]) = \emptyset \in U \), a contradiction.
We have proved that player I cannot have a winning strategy, the assumption that II has a w.s. in \( G(T,A) \) leads exactly to the same contradiction. \( \dashv \) [4]

**Remark 2.** In a game on some graph \( G \) with infinitely many vertices, if a player \( P \) has a winning strategy, and his opponent’s winning set is positive, then \( P \) does not necessarily have a *persistent* winning strategy. Consider the following counterexample where there is only one vertex (the initial one) for player 0, and player 1 wins if and only if the sequence of 1-vertices (the ones with two circles) is not strictly increasing:

Since any subsequence of a strictly increasing sequence is also strictly increasing, \( A \) (the winning set for player 1) satisfies \( \forall x, y \in G^\omega \ (x \subseteq y \land x \in A) \implies y \in A \), hence this game is positive. Clearly, player 0 has a winning strategy, but certainly do not have any *persistent* winning strategy. In fact, every winning strategy must be *persistently changing*.

## 5 Muller Acceptance Conditions

For the games where the winning set is defined in terms of Muller acceptance condition - i.e. an infinite play is accepted iff the set of vertices visited infinitely often belongs to some set of the form \( \mathcal{F} \subseteq \mathcal{P}(V) \) - topological properties do help.

**Theorem 5.** Let \( V \) be finite, and \( A \subseteq G^\omega \) defined by Muller acceptance conditions \( \mathcal{F} \subseteq \mathcal{P}(V) \), the following are equivalent:

1. \( A \) is positive,
2. \( A \equiv_W Q^0 \) or \( A \equiv_W D^n_i \) for some \( n \in \mathbb{N} \), and \( i \in \{0,1,(0,1)\} \),
3. \( A \in \Pi^0_2 \).
In particular, this theorem gives an amazing proof of an already known result that all $\Delta^0_2$-sets defined in terms of Muller acceptance conditions are finite differences of open sets (see [16], [3]).

**Proof of theorem 5.** The direction $1 \Rightarrow 2$ is theorem 3. For $2 \Rightarrow 3$, it is enough to say that if $A \equiv W^Q$, then $A$ is $\Pi^0_1$-complete, and if $A \equiv D^n_i$, then $A$ is in $\Delta^0_2$. So that in both cases, $A$ belongs to $\Pi^0_2$.

For $3 \Rightarrow 1$, towards a contradiction, we assume both $A \in \Pi^0_2$, and $A$ is not positive. This means, in particular, that there exists two infinite sequences $x, y \in G^{\omega}$, that verify $x \sqsubseteq y$, $x \in A$, and $y \notin A$. Let $A_x$, and $A_y$ be the set of vertices visited infinitely often by respectively $x$ and $y$. Since $x \sqsubseteq y$ holds, we infer that $A_x \subseteq A_y$ holds too. So there is at least one vertex $v \in A_x$ that is accessible from $v$, and from which there is, inside $G$, both a path $w_x$ going through precisely all vertices in $A_x$ and coming back to $v$, and another path $w_y$ going through precisely all vertices in $A_y$ and also coming back to $v$. This yields the following winning strategy for player $II$ in $W(Q, A)$:

- first reach vertex $v$, then
  - each time player $I$ plays a 0, play the whole path $w_x$,
  - each time player $I$ plays a 1, play the whole path $w_y$.

The sequence played by $I$ belongs to $Q$ iff it contains finitely many 1, in which case the set of vertices visited infinitely often by $II$ is exactly $A_x$, showing that the sequence played by $II$ also belongs to $A$.

If $I$’s play contains infinitely many 1, then it is not in $Q$, but in this case $II$ visited infinitely often precisely all vertices in $A_y$, so that $II$ also wins this case.

We have shown that $Q \leq_W A$ holds. Since $Q$ is $\Sigma^0_2$-complete, it follows that $A \notin \Pi^0_2$, a contradiction.

**Remark 3.** If a player’s winning set is a $\Sigma^0_2$-set defined in terms of Muller acceptance conditions, and this player has a winning strategy, then this player also has a persistent winning strategy. However, this fact does not help much, since this player not only has a persistent winning strategy but also a memoryless one.

**Lemma 7.** Let $V$ be finite, and $A \subseteq G^{\omega}$ defined by Muller acceptance conditions $F \subseteq P(V)$,

1. A positive $\implies F$ is closed under union.
2. A positive winning set for player 1 (resp. 0), and player 0 (resp. 1) has a w.s. $\implies$ player 0 (resp. 1) has a memoryless winning strategy.

**Proof of lemma 7.**

1. It is immediate to see that given $Q, Q' \in F$ such that $Q \cup Q' \notin F$, and given any sequence $x$ that visits infinitely often the set of vertices $Q$, one can build a sequence $y$ that visits infinitely often all vertices in $Q \cup Q'$ and satisfies $x \sqsubseteq y$. Which leads to both $x \in A$ and $y \notin A$, a contradiction.
2. This is an immediate consequence of theorem 17 in [17].

$\square$
References

Generating All Abductive Explanations for Queries on Propositional Horn Theories

Thomas Eiter\textsuperscript{1} and Kazuhisa Makino\textsuperscript{2}

\textsuperscript{1} Institut für Informationssysteme, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria
\texttt{eiter@kr.tuwien.ac.at}

\textsuperscript{2} Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan
\texttt{makino@sys.es.osaka-u.ac.jp}

\section*{Abstract}
Abduction is a fundamental mode of reasoning, which has taken on increasing importance in Artificial Intelligence (AI) and related disciplines. Computing abductive explanations is an important problem, and there is a growing literature on this subject. We contribute to this endeavor by presenting new results on computing multiple resp. all of the possibly exponentially many explanations of an abductive query from a propositional Horn theory represented by a Horn CNF. Here the issues are whether a few explanations can be generated efficiently and, in case of all explanations, whether the computation is possible in \emph{polynomial total time} (or \emph{output-polynomial time}), i.e., in time polynomial in the combined size of the input and the output. We explore these issues for queries in CNF and important restrictions thereof. Among the results, we show that computing all explanations for a negative query literal from a Horn CNF is not feasible in polynomial total time unless $\text{P} = \text{NP}$, which settles an open issue. However, we show how to compute under restriction to acyclic Horn theories polynomially many explanations in input polynomial time and all explanations in polynomial total time, respectively. Complementing and extending previous results, this draws a detailed picture of the computational complexity of computing multiple explanations for queries on Horn theories.

\textbf{Keywords}: Computational logic, abduction, propositional logic, Horn theories, polynomial total time computation, NP-hardness.

\section{Introduction}
Abduction is a fundamental mode of reasoning, which was extensively studied by C.S. Peirce \cite{19}. It has taken on increasing importance in Artificial Intelligence (AI) and related disciplines, where it has been recognized as an important principle of common-sense reasoning (see e.g. \cite{3}). Abduction has applications in many areas of AI and Computer Science including diagnosis, database updates, planning, natural language understanding, learning etc. (see e.g. references in \cite{10}), where it is primarily used for generating explanations.

\textsuperscript{*} This work was supported in part by the Austrian Science Fund (FWF) Project Z29-N04, by a TU Wien collaboration grant, and by the Scientific Grant in Aid of the Ministry of Education, Science, Sports, Culture and Technology of Japan.
In a logic-based setting, abduction can be viewed as the task to find, given a set of formulas $\Sigma$ (the background theory) and a formula $\chi$ (the query), a minimal set of formulas $E$ (an explanation) from a set of hypotheses $H$ such that $\Sigma$ plus $E$ is satisfiable and logically entails $\chi$. Often considered is a scenario where $\Sigma$ is a propositional Horn theory, $\chi$ is a single literal or a conjunction of literals, and $H$ contains literals (see [24,10] and references therein). For use in practice, the computation of abductive explanations in this setting is an important problem, for which well-known early systems such as Theorist [20] or ATMS solvers [6,22] have been devised. Since then, there has been a growing literature on this subject, indicating the need for efficient abductive procedures. We refer to [18], which gives an excellent survey on intimately closely related problems in computational logic. Note that much effort has been spent on studying various input restrictions, cf. [14,4,13,25,8,7,10,23,24].

While computing some explanation of a query $\chi$ has been studied extensively in the literature, the issue of computing multiple or even all explanations for $\chi$ has received less attention. This problem is important since often one would like to select one out of a set of alternative explanations according to a preference or plausibility relation; this relation may be based on subjective intuition and thus difficult to formalize. As easily seen, exponentially many explanations may exist for a query, and thus computing all explanations inevitably requires exponential time in general, even in propositional logic. However, it is of interest whether the computation is possible in polynomial total time (or output-polynomial time), i.e., in time polynomial in the combined size of the input and the output. Furthermore, if exponential space is prohibitive, it is of interest to know whether a few explanations (e.g., polynomially many) can be generated in polynomial time, as studied by Selman and Levesque [24].

Computing some explanation for a query $\chi$ which is a literal from a Horn theory is a well-known polynomial problem. Selman and Levesque conjectured [24] that generating $O(n)$ many explanations for a positive literal is NP-hard, where $n$ is the number of propositional atoms in the language, even if it is guaranteed that there are only few explanations overall. As shown in [11], this conjecture is not true unless $P=NP$. This follows from the result of [11] that all explanations for an atom can be generated in polynomial time.

The status of generating all explanations for a negative literal $\chi = \overline{q}$ from a Horn CNF $\varphi$, however, remained open in [11]. Moreover, it was unclear whether a resolution-style procedure similar to the one for query atoms in [11] could solve the problem in polynomial total time. In this paper, we provide a negative answer to this question, by showing that given a collection of explanations for a query $\chi = \overline{q}$ from a Horn CNF $\varphi$, deciding whether there is an additional explanation is NP-complete. Consequently, the existence of a polynomial total time algorithm for computing all explanations implies $P=NP$. However, for the well-known class of acyclic Horn theories (see e.g. [5,24,21,11]) we present an algorithm which enumerates all explanations for $\overline{q}$ with incremental polynomial delay (i.e., in time polynomial in the size of the input and output so far), and thus solves the problem in polynomial total time. Compared to explanations for an atomic query $q$, intuitively cyclic dependencies between atoms make the problem difficult. For completeness, a resolution-style procedure as in [11] needs to consider besides the input...
and output clauses also auxiliary clauses (see Example 7), whose derivation may cause a lot of overhead, since it is not a priori clear which such clauses are needed.

We furthermore address computing all explanations for queries \( \chi \) beyond literals, where we consider CNF and important special cases such as a clause and a term (i.e., a conjunction of literals). Note that the explanations for single clause queries correspond to the minimal support clauses for a clause in Clause Management Systems [22]. In the light of the negative results from above, we aim at elucidating the tractability frontier and present positive as well as negative results for such queries.

Our results shed new light on the computational nature of abduction and Horn theories in particular. They imply that, e.g., generating all minimal support clauses for a given clause (cf. [22]) from an acyclic Horn CNF is feasible in polynomial total time.

The intractability result for negative literal queries \( \overline{\eta} \) is somewhat unexpected, and the tractability result for acyclic Horn theories is more difficult to obtain than in case of atomic queries. As a byproduct, we also obtain results for computing all prime implicates of Horn theories containing a certain literal, which complement and refine previous results for computing all prime implicates of a Horn theory [2].

For space reasons, some proofs are omitted; we refer to the extended version [12].

2 Preliminaries and Notation

We assume a standard propositional language with atoms \( x_1, x_2, \ldots, x_n \) from a set \( At \), where each \( x_i \) takes either value 1 (true) or 0 (false). Negated atoms are denoted by \( \overline{x_i} \), and the opposite of a literal \( \ell \) by \( \overline{\ell} \). Furthermore, we use \( \overline{A} = \{ \overline{\ell} \mid \ell \in A \} \) for any set of literals \( A \) and set \( Lit = At \cup \overline{At} \).

A clause is a disjunction \( c = \bigvee_{p \in P(c)} p \lor \bigvee_{t \in N(c)} \overline{p} \) of literals, where \( P(c) \) and \( N(c) \) are the sets of atoms occurring positively and negated in \( c \) and \( P(c) \cap N(c) = \emptyset \). Dually, a term is conjunction \( t = \bigwedge_{p \in P(t)} p \land \bigwedge_{t \in N(t)} \overline{p} \) of literals, where \( P(t) \) and \( N(t) \) are similarly defined. We also view clauses and terms as sets of literals (resp., definite, negative, positive clauses). A clause \( c \) is Horn, if \( |P(c)| \leq 1 \); definite, if \( |P(c)| = 1 \); and negative (resp., positive), if \( |P(c)| = 0 \) (resp., \( |N(c)| = 0 \)). A conjunctive normal form (CNF) is a conjunction of clauses. It is Horn (resp., definite, negative, positive), if it contains only Horn clauses (resp., definite, negative, positive clauses). A theory \( \Sigma \) is any finite set of formulas; it is Horn, if it is a set of Horn clauses. As usual, we identify \( \Sigma \) with \( \varphi = \bigwedge_{c \in \Sigma} c \), and write \( c \in \varphi \) etc.

A model is a vector \( v \in \{0, 1\}^n \), whose \( i \)-th component is denoted by \( v_i \). For \( B \subseteq \{1, \ldots, n\} \), we let \( x^B \) be the model \( v \) such that \( v_i = 1 \), if \( i \in B \) and \( v_i = 0 \), if \( i \notin B \), for \( i \in \{1, \ldots, n\} \). Satisfaction \( v \models \varphi \) and logical consequence \( \varphi \models c \), \( \varphi \models \psi \) etc. are defined as usual (i.e., \( \varphi(v) = 1 \) etc.)

Example 1. The CNF \( \varphi = (x_1 \lor x_4) \land (x_4 \lor x_3) \land (x_1 \lor x_2) \land (x_4 \lor x_5 \lor x_1) \land (x_2 \lor x_5 \lor x_3) \) over \( At = \{x_1, x_2, \ldots, x_5\} \) is Horn. The vector \( u = (0, 1, 0, 1, 0) \) is a model of \( \varphi \). ■

The following proposition is well-known.

Proposition 1. Given a Horn CNF \( \varphi \) and a clause \( c \), deciding whether \( \varphi \models c \) is possible in polynomial time (in fact, in linear time, cf. [2]).
Recall that two clauses $c$ and $c'$ resolve on a pair of literals $x, \overline{x}$ if $x, \overline{x} \in c \cup c'$ and $c \cup c' \setminus \{x, \overline{x}\}$ is a legal clause (thus, $x$ must occur in exactly one of $c$ and $c'$, and same for $\overline{x}$); $c$ and $c'$ resolve if their is a pair of literals $x, \overline{x}$ on which they resolve. Note that this pair, if it exists, is unique. In that case, we denote by $c \oplus c'$ the clause $c \cup c' \setminus \{x, \overline{x}\}$, which is their resolvent (otherwise, $c \oplus c'$ is undefined). A resolution proof of a clause $c$ from a CNF $\phi$ is a sequence $c_1, c_2, ..., c_l$ of clauses such that $c_l = c$ and, for all $i = 1, ..., l$, either $c_i \in \phi$ or $c_i = c_j \oplus c_k$ for clauses $c_j$ and $c_k$ such that $j, k < i$.

It is well-known that resolution proofs are sound and complete with respect to clause inference in the following sense (cf. [18]): For any CNF $\phi$ and clause $c$, $\phi \models c$ holds iff there is a clause $c' \subseteq c$ which has a resolution proof from $\phi$. For further background on resolution, we refer to [17,16].

### 2.1 Abductive Explanations

The notion of an abductive explanation can be formalized as follows (cf. [24,10]).

**Definition 1.** Given a (Horn) theory $\Sigma$, called the background theory, a CNF $\chi$ (called query), an explanation of $\chi$ is a minimal set of literals $E \subseteq \text{Lit}$ such that

(i) $\Sigma \cup E \models \chi$, and

(ii) $\Sigma \cup E$ is satisfiable.

**Example 2.** Reconsider the Horn CNF $\varphi = (\overline{x}_1 \lor \overline{x}_4) \land (\overline{x}_4 \lor \overline{x}_3) \land (\overline{x}_1 \lor x_2) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_1) \land (\overline{x}_2 \lor \overline{x}_5 \lor x_3)$ from above. Suppose we want to explain $\chi = x_2$ from $A = \{x_1, x_4\}$. Then, we find that $E = \{x_1\}$ is an explanation. Indeed, $\Sigma \cup \{x_1\} \models x_2$, and $\Sigma \cup \{x_1\}$ is satisfiable; moreover, $E$ is minimal. On the other hand, $E' = \{x_1, \overline{x}_4\}$ satisfies (i) and (ii) for $\chi = x_2$, but is not minimal. \[ \square \]

More restricted forms of explanations require that $E$ must be formed over a given set of abducible letters (cf. [24]); however, in such a setting, generating all explanations is easily seen to be coNP-hard for the cases that we consider from results in the literature.

The following characterization of explanations is immediate by the monotonicity of classical logic.

**Proposition 2.** For any theory $\Sigma$, any query $\chi$, and any $E \subseteq \text{Lit}$, $E$ is an explanation for $\chi$ from $\Sigma$ iff the following conditions hold: (i) $\Sigma \cup E$ is satisfiable, (ii) $\Sigma \cup E \models \chi$, and (iii) $\Sigma \cup (E \setminus \{\ell\}) \not\models \chi$, for every $\ell \in E$.

From Proposition 2 we thus obtain the following easy lemma.

**Lemma 1.** Given a Horn CNF $\varphi$, a set $E \subseteq \text{Lit}$, and a CNF query $\chi$, deciding whether $E$ is an explanation for $\chi$ w.r.t. $A$ is possible in polynomial time.

### 3 Intractability of Negative Literal Queries

In this section, we show that computing all explanations of a negative query $\chi = \overline{q}$ is not possible in polynomial total time unless $P = NP$. This result follows by standard arguments from the following theorem.
Theorem 1. Given a Horn CNF \( \varphi \), a query \( \chi = \overline{q} \), and explanations \( E_1, E_2, \ldots, E_k \) for \( \chi \), deciding whether \( \chi \) has some additional explanation \( E_{k+1} \) different from each \( E_i \), \( 1 \leq i \leq k \), is NP-complete. Hardness holds even if \( \varphi \) is definite Horn.

In the proof of Theorem 1 we use the following well-known lemma, which links prime implicants of a theory to explanations. Recall that a prime implicate of a theory \( \Sigma \) is a minimal (w.r.t. inclusion) clause \( c \) such that \( \Sigma \models c \). Let us call an explanation \( E \) for a literal query \( \chi = \ell \) trivial, if \( E = \{ \ell \} \).

**Lemma 2 (cf. [22,15]).** Given a theory \( \Sigma \), a set \( E \subseteq \text{Lit} \) is a nontrivial explanation of a query literal \( \chi \) iff the clause \( c = \bigwedge_{\ell \in E} \overline{\ell} \lor \chi \) is a prime implicate of \( \Sigma \).

Note \( \chi = \ell \) has a trivial explanation iff \( \Sigma \models \ell \) and \( \Sigma \notmodels \overline{\ell} \), which can be checked in polynomial time. Hence, as for NP-hardness we can without loss of generality focus on generating the nontrivial explanations of \( \overline{q} \), i.e., all prime implicants containing \( \overline{q} \).

**Proof of Theorem 1** As for membership in NP, an additional explanation \( E_{k+1} \) can be guessed and, by Lemma 2, be verified in polynomial time.

We show the NP-hardness by a reduction from 3SAT. Let \( \gamma = c_1 \land \cdots \land c_m, m \geq 2 \), be a 3CNF over atoms \( x_1, \ldots, x_n \), where \( c_i = \ell_{i,1} \lor \ell_{i,2} \lor \ell_{i,3} \). We introduce for each clause \( c_i \) a new atom \( y_i \), for each \( x_j \) a new atom \( x_j' \) (which intuitively corresponds to \( \overline{x}_i \)), and special atoms \( q \) and \( z \). The Horn CNF \( \varphi \) contains the following clauses:

1. \( c_{i,j} = \overline{q} \lor \ell_{i,j} \lor y_j \), for all \( i = 1, \ldots, m \) and \( j = 1, 2, 3 \);
2. \( d_{i,j} = \ell_{i,j} \lor y_j \lor \overline{y}_{i+1} \), for all \( i = 1, \ldots, m \) and \( j = 1, 2, 3 \);
3. \( \overline{x}_i \lor x_i' \lor z \), for all \( i = 1, \ldots, n \);
4. \( e = \overline{y}_1 \lor \overline{y}_2 \lor \cdots \lor \overline{y}_m \lor z \),

where \( \ell_{i,j} = x_k \) if \( \ell_{i,j} = x_k \) and \( \ell_{i,j}^* = x_k' \) if \( \ell_{i,j} = \overline{x}_i \), and \( y_{m+1} = y_1 \).

Note that \( \varphi \) is definite Horn, and thus all prime implicants of \( \varphi \) are definite Horn. Informally, the clauses \( c_{i,j} \) and \( d_{i,j} \) stand for selection of literal \( \ell_{i,j} \) in clause \( c_i \). The clause in 4., which is needed to produce any negative prime implicate \( c \) containing \( \overline{q} \), and the minimality of a prime implicate will effect that a literal is chosen from each clause \( c_i \), and the clauses in 3. will ensure that the choice is consistent, such that \( \gamma \) is satisfied. Since the positive prime implicants containing \( \overline{q} \) are just the clauses \( c_{i,j} \), the a further prime implicate of \( \varphi \) containing \( \overline{q} \) exists iff \( \gamma \) is satisfiable.

We establish the following properties of \( \varphi \).

**Lemma 3.** Any prime implicate \( c \) of \( \varphi \) such that \( q \in N(c) \) and \( P(c) \neq \{ z \} \) is of the form \( c_{i,j} \), where \( i \in \{ 1, \ldots, m \} \) and \( j \in \{ 1, 2, 3 \} \).

**Lemma 4.** Any prime implicate \( c \) of \( \varphi \) such that \( P(c) = \{ z \} \) and \( q \in N(c) \) satisfies (i) \( \{ x_i, x_i' \} \not\subseteq N(c) \), for all \( i = 1, \ldots, n \), and (ii) \( y_i \not\in N(c) \), for all \( i = 1, \ldots, m \).

From Lemma 3 it is now easy to see that all prime implicants of \( \varphi \) given by the clauses in 1. correspond to nontrivial explanations of \( \overline{q} \), and from Lemma 4 that an additional nontrivial explanation for \( \overline{q} \) exists if and only if some prime implicate of \( \varphi \) of form \( c = \overline{q} \lor \bigvee_{x_i \in X} \overline{x}_i \lor \bigvee_{x_i' \in X} x_i' \lor z \) exists iff the CNF \( \gamma \) is satisfiable. As for the last
equivalence, note that for each smallest (w.r.t. $\subseteq$) choice $\ell_i \in c_i$ of a consistent collection of literals $\ell_1, \ldots, \ell_m$, we have an additional prime implicate of $\varphi$ of form $\overline{q} \lor \bigvee_{i} \overline{\ell_i} \lor z$. Conversely, each additional prime implicate $c$ containing $\overline{q}$ gives rise to a consistent set of literals $\{x_j \mid x_{i,j} \in N(c)\} \cup \{x_j' \mid x_{i,j}' \in N(c)\}$ which satisfies $\gamma$.

Clearly, $\varphi$ is constructible in polynomial time from $\gamma$. Since $\varphi$ is definite, this proves the NP-hardness under the asserted restriction. \hfill \Box

We note that $\varphi$ in the hardness proof of Theorem 1 remains Horn upon switching the polarity of $z$. From this easily NP-completeness of deciding the existence of an explanation for $\chi = \overline{q}$ of only positive literals follows, even if all other explanations are given. This contrasts with respective tractability results for acyclic theories (implied by the next section) and for atomic queries $\chi = q$ on arbitrary Horn CNFs [11].

4 Negative Literal Queries on Acyclic Horn Theories

Since as shown in the previous section, a polynomial total time procedure for generating all explanations of a negative literal query is infeasible in general unless P=NP, it becomes an issue to find restricted input classes for which this is feasible. In this section, we show a positive result for the important class of acyclic Horn theories, which has been studied extensively in the literature (see, e.g., [5,24,21,1]).

We first recall the concept of acyclic Horn theories (see e.g. [5,24]).

**Definition 2.** For any Horn CNF $\varphi$ over atom set $At$, its dependency graph is the directed graph $G(\varphi) = (V, E)$, where $V = At$ and $E = \{x_i \rightarrow x_j \mid c \in \varphi, x_i \in N(c), x_j \in P(c)\}$, i.e., $E$ contains an arc from each atom in a negative literal to the positive literal in a clause (if such a literal exists). A Horn CNF $\varphi$ is acyclic if $G(\varphi)$ has no directed cycle.

**Example 3.** As easily seen, the edges of $G(\varphi)$ for the CNF $\varphi$ in Examples 1 and 2 are $x_1 \rightarrow x_2, x_4 \rightarrow x_1, x_5 \rightarrow x_1, x_2 \rightarrow x_3$, and $x_5 \rightarrow x_3$. Hence, $\varphi$ is acyclic. \hfill \Box

Since the trivial explanation $E = \{\overline{q}\}$ can be easily generated (if it applies), we focus on generating all nontrivial explanations. For a negative query on an acyclic Horn theory, this is accomplished by Algorithm n-EXPLANATIONS in Figure 1. It first converts the input into an equivalent prime Horn CNF $\varphi^*$, and then applies a restricted resolution procedure, in which pairs $(c, c')$ of clauses are considered of which at least one is a prime implicate containing $\overline{q}$ and the other is either a clause of this form or a clause from the converted input $\varphi^*$. In case their resolvent $d' = c \oplus c'$ exists and, as implied by condition (ii) in Definition 1 includes only prime implicants containing $\overline{q}$, any such prime implicate $d'$ is computed. If $d'$ is recognized as a new prime implicate which has not been generated so far, a corresponding explanation is output and the set of candidate pairs is enlarged.

**Example 4.** Reconsider $\varphi = (\overline{x_1} \lor \overline{x_4}) \land (\overline{x_4} \lor \overline{x_3}) \land (\overline{x_1} \lor x_2) \land (\overline{x_4} \lor \overline{x_5} \lor x_1) \land (\overline{x_2} \lor \overline{x_5} \lor x_3)$, and apply n-EXPLANATIONS for $\chi = \overline{x_1}$. All clauses of $\varphi$ are prime except $\overline{x_4} \lor \overline{x_5} \lor x_1$, which contains the prime implicate $\overline{x_4} \lor \overline{x_5}$. Thus, $\varphi^* = (\overline{x_1} \lor \overline{x_4}) \land (\overline{x_4} \lor \overline{x_3}) \land (\overline{x_1} \lor x_2) \land (\overline{x_4} \lor \overline{x_5}) \land (\overline{x_2} \lor \overline{x_5} \lor x_3)$, and $S$ contains the clauses $\overline{x_1} \lor \overline{x_4}$.
Algorithm n-Explanations

Input: An acyclic Horn CNF $\varphi$ and an atom $q$.

Output: All nontrivial explanations of the query $\chi = \overline{q}$ from $\varphi$.

Step 1. $\varphi^* := \emptyset$, $S := \emptyset$, and $O := \emptyset$;

Step 2. for each $c \in \varphi$ do
  add any prime implicate $c' \subseteq c$ of $\varphi$ to $\varphi^*$;
  for each $c' \in \varphi^*$ with $q \in N(c')$ and $c' \notin S$ do
    begin
      output ($\{\overline{\ell} \mid \ell \in c' \setminus \{\overline{q}\}\}$);
      $S := S \cup \{c'\}$; $O := O \cup \{(c, c') \mid c \in \varphi^*, q \notin P(c)\}$
    end;

Step 3. while some $(c, c') \in O$ exists do
  begin
    $O := O \setminus \{(c, c')\}$;
    if (1) $c$ and $c'$ resolve and (2) $\varphi^* \not\models (c \oplus c' \setminus \{\overline{q}\})$
      then begin
        $d := c \oplus c'$;
        compute any prime implicate $d' \subseteq d$ of $\varphi$;
        if $d' \notin S$ then
          begin
            output ($\{\overline{\ell} \mid \ell \in d' \setminus \{\overline{q}\}\}$);
            $S := S \cup \{d'\}$;
            $O := O \cup \{(d'', d') \mid d'' \in \varphi^*, q \notin P(d'')\} \cup \{(d'', d') \mid d'' \in S\}$
          end
        end
      end

Fig. 1. Algorithm computing all nontrivial explanations of a query $\chi = \overline{q}$ on an acyclic Horn theory

and $\overline{x_1} \lor x_2$; the corresponding explanations $E_1 = \{x_4\}$ and $E_2 = \{\overline{x}_2\}$ are output. In Step 2, the pair $($$\overline{x}_2 \lor \overline{x}_5 \lor x_3, \overline{x}_1 \lor x_2$$)$ is found in $O$ which satisfies condition (i) in Def.\ref{def:prime} Moreover, (ii) is satisfied, since $\varphi^* \not\models \overline{x}_5 \lor x_3$. Thus, a prime implicate within $d = \overline{x}_1 \lor \overline{x}_5 \lor x_3$ is computed; in fact, $d$ is prime. Therefore, $E_3 = \{x_5, \overline{x}_3\}$ is output and $d$ is added to $S$, and then $O$ is enlarged. Eventually, the pair $($$\overline{x}_3 \lor \overline{x}_4, \overline{x}_1 \lor \overline{x}_5 \lor x_3$$)$ from $O$, which satisfies condition (i), will be considered. However, $\varphi^* \models \overline{x}_4 \lor \overline{x}_5$, and thus $S$ remains unchanged. Hence, the output of n-Explanations is $E_1, E_2,$ and $E_3$. As can be seen, these are all nontrivial explanations for $\overline{x}_1$ from $\varphi$.

We remark that our algorithm is similar in spirit to an algorithm for computing all prime implicates of a Horn CNF in polynomial total time \cite[2]{2}. Our algorithm solves a more constrained problem, though.

In the rest of this section, we show that Algorithm n-Explanations generates all explanations in polynomial total time. For that, we first show its correctness, which splits into a soundness and completeness part, and then analyze its time complexity.

As for soundness, it is easily seen that Algorithm n-Explanations produces output only if $d'$ is some prime implicate of $\varphi^*$ (and thus of $\varphi$) such that $q \in N(d')$. Thus, from Lemma\ref{lemma:soundness} we immediately obtain

Lemma 5 (Soundness of n-Explanations). Algorithm n-Explanations outputs only nontrivial explanations for $\overline{q}$ from $\varphi$.

It is much more difficult to show the completeness, i.e., that Algorithm n-Explanations actually generates all nontrivial explanations. Intuitively, the difficulty stems from
We define an ordering on Horn clauses as follows. Suppose that $\leq$ imposes a total ordering on the atoms ($x_{i_1} \leq x_{i_2} \leq \cdots \leq x_{i_n}$). Then, for any Horn clauses $c_1$ and $c_2$, define $c_1 \leq c_2$ iff $c_1 = c_2$ or one of the following conditions holds:

- The fact that the restricted resolution procedure retains only prime clauses containing $\overline{q}$, and, moreover, may skip relevant prime implicates $d' \subseteq c \oplus c'$ in Step 3 if condition (ii) fails, i.e., $c \oplus c'$ is an implicate of $\varphi$ (which is tantamount to the condition that $c \oplus c'$ contains some prime implicate of $\varphi$ that does not contain $\overline{q}$). To see that no explanation is missed requires a careful analysis of how the desired explanations are generated, and leads to a nontrivial argument which takes the complex interaction between clauses into account.

We need a number of preliminary technical lemmas on which our proof builds, which are interesting in their own right. In what follows, we call a Horn clause $c$ definite, if $P(c) \neq \emptyset$. Furthermore, for any literal $\ell$, a clause $c$ is a $\ell$-clause if $c$ contains $\ell$.

The following propositions are well-known.

**Proposition 3.** Let $c_1$, $c_2$ be Horn implicates of a Horn CNF $\varphi$ that resolve. Then, $c = c_1 \oplus c_2$ is Horn, and if $c_1$ contains a negative implicate of $\varphi$, then also $c_1 \oplus c_2$ contains a negative implicate of $\varphi$.

**Proposition 4 (cf. [2]).** Every prime implicate $c$ of a Horn CNF $\varphi$ has an input resolution proof from it, i.e., a resolution proof $c_1, c_2, \ldots, c_l (= c)$ such that either $c_i \in \varphi$ or $c_i = c_j \oplus c_k$ where $j, k < l$ and either $c_j \in \varphi$ or $c_k \in \varphi$, for all $i \in \{1, \ldots, l\}$.

We start with the following lemma.

**Lemma 6.** Let $\varphi$ be a prime Horn CNF, and let $c$ be any prime implicate of $\varphi$ such that $c \notin \varphi$. Then, $c = c_1 \oplus c_2$, where $c_1$ is a prime implicate contained in $\varphi$, and either (i) $c_2$ is a prime implicate of $\varphi$, or (ii) $c_2 = c \cup \{\ell\}$ where $c_1 \setminus \{\overline{\ell}\} \subseteq c$ and $c$ is the unique prime implicate of $\varphi$ contained in $c_2$.

Note that item (ii) is needed in this lemma, as shown by the following example.

**Example 5.** Consider the Horn CNF $\varphi = (\overline{\overline{x}_0} \lor \overline{x}_1 \lor x_2)(\overline{x}_2 \lor \overline{x}_3)(\overline{x}_3 \lor x_0)$. As easily checked, $\varphi$ is prime and has a further prime implicate $\overline{x}_1 \lor \overline{x}_3$, which can not be derived as the resolvent of any two prime implicates of $\varphi$. Note that $\varphi$ is acyclic.

Next we state some important properties of acyclic Horn CNFs under resolution.

**Proposition 5.** Let $\varphi$ be an acyclic Horn CNF, and let $c = c_1 \oplus c_2$ where $c_1, c_2 \in \varphi$. Then, $\varphi' = \varphi \land c$ is acyclic Horn, and the dependency graphs $G(\varphi)$ and $G(\varphi')$ have the same transitive closure. Furthermore, any subformula $\varphi'' \subseteq \varphi$ is acyclic Horn.

Thus, adding repeatedly clauses derived by resolution preserves the acyclicity of a CNF, and, moreover, the possible topological sortings of the dependency graph.

The following proposition captures that for an acyclic Horn CNF, resolution cannot be blocked because of multiple resolving pairs of $x_i$ and $\overline{x}_i$ of literals.

**Proposition 6.** Let $\varphi$ be an acyclic Horn CNF, and let $c_1$ and $c_2$ be any implicates of $\varphi$ derived from $\varphi$ by resolution. Then, $c_1$ and $c_2$ do not resolve iff $P(c_1) \cap N(c_2) = \emptyset$ and $P(c_2) \cap N(c_1) = \emptyset$.
(i) \(P(c_1) \neq \emptyset\) and \(P(c_2) = \emptyset\);
(ii) \(P(c_1) = \{x_i\}\) and \(P(c_2) = \{x_j\}\) and \(x_i < x_j\);
(iii) \(P(c_1) = P(c_2)\) and \(\max N(c_1) \triangle N(c_2) \in c_1\), where “\(\triangle\)” denotes standard symmetric difference (i.e., \(S_1 \triangle S_2 = (S_1 \cup S_2) \setminus (S_1 \cap S_2)\)).

As usual, we write \(c_1 < c_2\) if \(c_1 \leq c_2\) and \(c_1 \neq c_2\), \(c_1 > c_2\) for \(c_2 < c_1\) etc. Note that \(\leq\) orders first all definite Horn clauses along their positive literals, followed by the negative clauses. Notice that \(c_1 \subseteq c_2\) implies \(c_2 < c_1\), for any Horn clauses \(c_1\) and \(c_2\).

The following proposition is not difficult to establish:

**Proposition 7.** Every total ordering \(\leq\) of the atoms \(At\) induces a total ordering \(\leq\) of all Horn clauses over \(At\) as described.

With respect to acyclic Horn CNFs \(\varphi\), in the rest of this paper we assume an arbitrary but fixed total ordering \(\leq\) of the atoms which is compatible with some topological sorting of the dependency graph \(G(\varphi)\).

**Proposition 8.** Let \(c_1\) and \(c_2\) be Horn clauses such that \(c = c_1 \oplus c_2\) exists. Then, \(c_1 < c\) and \(c_2 < c\) hold.

**Corollary 1.** Let \(\varphi\) be an acyclic Horn CNF, and let \(c, c_1\) and \(c_2\) be any implicates of \(\varphi\) derived from \(\varphi\) such that \(c \succeq c_1 \oplus c_2\). Then, \(c > c_1\) and \(c > c_2\) holds.

Consequently, in any input resolution proof of a clause from an acyclic Horn CNF the derived clauses increase monotonically. As for the derivation of prime implicates, we find for such CNFs a more general form than in Lemma 6.

**Lemma 7.** Let \(\varphi\) be an acyclic prime Horn CNF, and let \(c\) be any prime implicate of \(\varphi\) such that \(c \not\in \varphi\). Then, there are prime implicates \(c_1\) and \(c_2\) of \(\varphi\) and, for some \(k \geq 0\), prime implicates \(d_1, d_2, \ldots, d_k\) and literals \(\ell_1, \ell_2, \ldots, \ell_k\), respectively, such that: (i) \(c_1, d_1, \ldots, d_k \in \varphi\), and (ii) \(c = c_1 \oplus e_1\), where \(e_i = c \cup \{\ell_i\} = d_i \oplus e_{i+1}\), for \(i \in \{1, \ldots, k\}\), and \(e_{k+1} = c_2\), such that \(e_i\) contains the single prime implicate \(c\).

An immediate consequence of this result is that prime implicates of an acyclic Horn CNF can be generated from two prime implicates as follows.

**Corollary 2.** Let \(\varphi\) be an acyclic prime Horn CNF, and let \(c\) be any prime implicate of \(\varphi\) such that \(c \not\in \varphi\). Then, there exist prime implicates \(c_1\) and \(c_2\) of \(\varphi\) which resolve such that either (i) \(c = c_1 \oplus c_2\) or (ii) \(c_1 \oplus c_2 = c \cup \{\ell\}\), where \(\ell \not\in c\) and \(c\) is the unique prime implicate of \(\varphi\) contained in \(c_1 \oplus c_2\).

In Example 5 the further prime implicate \(\overline{x}_1 \lor \overline{x}_3\) can be derived as in case (ii) of Corollary 2. For \(c_1 = \overline{x}_0 \lor \overline{x}_1 \lor x_2\) and \(c_2 = \overline{x}_2 \lor \overline{x}_3\), we have \(c_1 \oplus c_2 = \overline{x}_1 \lor \overline{x}_3 \lor \overline{x}_0\), and \(c = \overline{x}_1 \lor \overline{x}_3\) is the unique prime implicate of \(\varphi\) contained in \(c_1 \oplus c_2\).

After the preparatory results, we now show that Algorithm n-EXPLANATIONS is complete. Using an inductive argument on clause orderings, we show that all explanations are generated by taking into account possible derivations of prime implicates as established in Lemma 7 and Corollary 2. However, an inductive proof along \(\leq\) encounters two major difficulties: First, the resolvent \(c = c_1 \oplus c_2\) of two clauses is larger than \(c_1\).
and \(c_2\), thus we cannot simply rearrange resolution steps and appeal to smaller clauses. Second, Algorithm n-Explanations does not generate prime implicates \(d'\) by a resolution step alone, but using minimization in Step 3; that is, a prime implicate included in the resolvent \(d = c \lor c'\). A respective statement is much more difficult to prove than the one if \(d\) were prime.

In order to overcome these difficulties, we use a more sophisticated ordering of clause pairs \((c, c')\) and establish as a stepping stone the following key lemma. For ease of reference, let us say that resolvable implicates \(c_1\) and \(c_2\) of a Horn CNF \(\varphi\) satisfy the (technical) property \((\star)\), if the following conditions hold:

1. At least one of \(c_1\) and \(c_2\) is prime.
2. If \(c_i\) is not prime, then it is of form \(c_i = c' \lor \{\ell\}\), where \(c'\) is the unique prime implicate of \(\varphi\) contained in \(c_i\) \((i \in \{1, 2\}\) ), and \(c_i\) occurs in some derivation of \(c'\) as in Lemma 7.
3. There is no implicate \(c'_1 \subset c_1\) (resp., \(c'_2 \subset c_2\)) of \(\varphi\) such that \(c = c'_1 \lor c_2\) (resp., \(c = c_1 \lor c'_2\)).

**Lemma 8 (Key Lemma).** Let \(\varphi\) be a prime acyclic Horn CNF, and let \(c_1\) and \(c_2\) be resolvable clauses satisfying \((\star)\) such that \(\overline{q} \in c := c_1 \lor c_2\). Suppose that \(c_i \in S\) if \(c_i\) is prime and \(q \in N(c_i)\) (resp., \(c'_i \in S\) if \(c_i = c'_i \lor \{\ell_i\}\) where \(c'_i\) is prime and \(q \in N(c'_i)\)) for \(i \in \{1, 2\}\). Then at least one of the following conditions hold: (i) \(c \setminus \{\overline{q}\}\) is an implicate of \(\varphi\), or (ii) \(c\) contains a \(\overline{q}\)-clause from \(S\).

**Proof.** (Outline) We prove the statement using an inductive argument which involves clause orderings and takes into account how the clauses \(c_1\) and \(c_2\) are recursively generated. Depending on the shape of \(c_1\) and \(c_2\), we consider different cases.

Consider first the case in which both \(c_1\) and \(c_2\) contain \(\overline{q}\). Then, w.l.o.g. \(c = \overline{q} \lor \overline{a} \lor \overline{b} \lor (\lor x_i)\), \(c_1 = \overline{q} \lor \overline{a} \lor \overline{\pi} (\lor x_i)\), and \(c_2 = \overline{b} \lor x \lor \overline{q}\). Here, \(a\) and \(\overline{b}\) are disjunctions of negative literals, while \(x\) is a single atom; “\((\lor x_i)\)” means the optional presence of \(x_i\).

Both \(c_1\) and \(c_2\) contain a unique prime implicate \(c'_1\) resp. \(c'_2\) of \(\varphi\) (where possibly \(c'_1 = c_1\) ). If \(\overline{q} \in c'_1\), then by assertion we have \(c'_1 \in S\). Thus, if both \(c'_1\) and \(c'_2\) contain \(\overline{q}\), Algorithm n-Explanations considers \(c'_1 \lor c'_2\), which implies the statement. No other cases are possible, since either \(c'_1\) or \(c'_2\) must contain \(\overline{q}\) (since \(c_1\) or \(c_2\) is prime) and condition 3 of \((\star)\) excludes that exactly one of \(c'_1\) and \(c'_2\) contains \(\overline{q}\). This proves the statement if both \(c_1\) and \(c_2\) contain \(\overline{q}\).

For the other cases, assume that \(\overline{q} \in c_1\) and \(\overline{q} \notin c_2\) and prove the statement by induction along the lexicographic ordering of the pairs \((c_1, c_2)\), where the clauses \(c_1\) are in reverse ordering \(\geq\) and the clauses \(c_2\) in regular ordering \(\leq\). We distinguish the following cases:

**Definite/Negative Case 1 (DN1):** \(c = \overline{q} \lor \overline{a} \lor \overline{\pi} (\lor x_i)\), \(c_1 = \overline{q} \lor \overline{a} \lor \overline{\pi} (\lor x_i)\), and \(c_2 = \overline{b} \lor x\). That is, the \(\overline{q}\)-clause \(c\) is generated by resolving a \(\overline{q}\)-clause \(c_1\) with a non-\(\overline{q}\)-clause \(c_2\), where the positive resolution literal \(x\) is in \(c_2\).

**Definite/Negative Case 2 (DN2):** \(c = \overline{q} \lor \overline{a} \lor \overline{\pi} (\lor x_i)\), \(c_1 = \overline{q} \lor \overline{a} \lor x\), and \(c_2 = \overline{b} \lor \overline{\pi} (\lor x_i)\). That is, the \(\overline{q}\)-clause \(c\) is generated by resolving a \(\overline{q}\)-clause \(c_1\) with a non-\(\overline{q}\)-clause \(c_2\), where the positive resolution literal \(x\) is in \(c_1\).
We conclude this section with some remarks on Algorithm 2.

The statement is shown by a careful analysis of parent clauses of $c_1$ and $c_2$, and by reordering and adapting resolution steps. DN1 recursively only involves cases of the same kind (in fact, for negative $c_1$ we need to appeal only to smaller instances ($c'_1$, $c'_2$) where $c'_1$ is negative), while DN2 recursively involves itself as well as DN1.

By combining Lemma 8 with Proposition 8 and Corollary 2, we obtain by an inductive argument on the clause ordering $\leq$ the desired completeness result.

**Lemma 9 (Completeness of n-Explanations).** Algorithm n-Explanations outputs all nontrivial explanations for a query $\chi = \bar{\eta}$ from an acyclic Horn CNF $\phi$.

**Proof.** We prove by induction on $\leq$ that $S$ contains each $\bar{\eta}$-prime implicate $c$ of $\phi$.

**(Basis)** Let $c$ be the least prime implicate of $\phi$ which contains $\bar{\eta}$. From Proposition 8 and Corollary 2 we conclude that $c \in \phi$ must hold. Hence, $c \in S$.

**(Induction)** Suppose the claim holds for all $\bar{\eta}$-prime implicants $c'$ of $\phi$ such that $c' < c$, and consider $c$. By Corollary 2 there exist prime implicants $c_1$ and $c_2$ such that either (i) $c = c_1 \oplus c_2$ or (ii) $c$ is the unique prime implicate contained in $c_1 \oplus c_2 = c \cup \{\ell\}$ where $\ell \notin c$. By Proposition 8 and the induction hypothesis, we have $c_1 \in S$ if $q \in N(c_i)$ holds for $i \in \{1, 2\}$. Consequently, $c_1$ and $c_2$ satisfy the conditions of Lemma 8. Hence, either (a) $c_1 \oplus c_2 \setminus \{\bar{\eta}\}$ is an implicate of $\phi$, or (b) $c_1 \oplus c_2$ contains a $\bar{\eta}$-clause $c'$ from $S$. Since $q \in N(c)$ and $c$ is the unique prime implicate contained in $c_1 \oplus c_2$, we have (b). It follows from the uniqueness of $c$ that $c' = c$, which proves the statement.

We are now in a position to establish the main result of this section. Let $\|\phi\|$ denote the size (number of symbols) of any CNF $\phi$.

**Theorem 2.** Algorithm n-Explanations incrementally outputs, without duplicates, all nontrivial explanations of $\chi = \bar{\eta}$ from $\phi$. Moreover, the next output (respectively termination) occurs within $O(s \cdot (s + m) \cdot n \cdot \|\phi\|)$ time, where $m$ is the number of clauses in $\phi$, $n$ the number of atoms, and $s$ the number of explanations output so far.

**Proof.** By Lemmas 5 and 9 it remains to verify the time bound. Computing a prime implicate $c' \subseteq c$ and $d' \subseteq d$ of $\phi$ in Steps 2 and 3, respectively, is feasible in time $O(n \cdot \|\phi\|)$ (cf. Proposition 1), and thus the outputs in Step 2 occur with $O(m \cdot n \cdot \|\phi\|)$ delay. As for Step 3, note that $O$ contains only pairs $(c, c')$ where $c \in \phi^* \cup S$ and $c' \in S$ such that the explanation corresponding to $c'$ was generated, and each such pair is added to $O$ only once. Thus, the next output or termination follows within $s \cdot (s + m)$ runs of the while-loop, where $s$ is the number of solutions output so far. The body of the loop can be done, using proper data structures, in $O(n \cdot \|\phi\|)$ time (for checking $d' \notin S$ efficiently, we may store $S$ in a prefix tree). Thus, the time until the next output resp. termination is bounded by $O(s \cdot (s + m) \cdot n \cdot \|\phi\|)$.

**Corollary 3.** Computing polynomially many explanations for a negative query $\chi = \bar{\eta}$ from an acyclic Horn CNF $\phi$ is feasible in polynomial time (in the size of the input).

We conclude this section with some remarks on Algorithm n-Explanations.

1. As for implementation, standard data structures and marking methods can be used to realize efficient update of the sets $O$ and $S$, to determine resolvable clauses, and to eliminate symmetric pairs $(c, c')$ an $(c', c)$ in $O$. 

(2) Algorithm $n$-EXPLANATIONS is incomplete for cyclic Horn theories, as shown by the following example.

**Example 6.** Consider the Horn CNF $\varphi = (\overline{x}_0 \lor \overline{x}_1 \lor x_2)(\overline{x}_0 \lor \overline{x}_1 \lor x_3)(\overline{x}_1 \lor \overline{x}_2 \lor x_3)(\overline{x}_1 \lor x_2 \lor \overline{x}_3)(\overline{x}_2 \lor \overline{x}_3 \lor x_4)$ over $x_0, \ldots, x_4$. Note that all clauses in $\varphi$ are prime, and that $x_2$ and $x_3$ are symmetric. There are three further prime implicates, viz. $c_1 = \overline{x}_1 \lor \overline{x}_2 \lor x_4$, $c_2 = \overline{x}_1 \lor \overline{x}_3 \lor x_4$, and $c_3 = \overline{x}_0 \lor \overline{x}_1 \lor x_4$. Thus, $\overline{q} = \overline{x}_0$ has the nontrivial explanations $E_1 = \{x_1, \overline{x}_2\}$, $E_2 = \{x_3, \overline{x}_3\}$, and $E_3 = \{x_1, \overline{x}_4\}$. Apply then algorithm $n$-EXPLANATIONS on input $\varphi$ and $q = x_0$. While it outputs $E_1$ and $E_2$, it misses explanation $E_3$. □

Algorithm $n$-EXPLANATIONS may be extended to handle this example and others correctly by adding in Step 2 prime implicates to $\varphi^*$ which are generated in polynomial time (e.g., by minimizing clauses derived by resolution proofs from $\varphi^*$ whose number of steps is bounded by a constant).

(3) Algorithm $n$-EXPLANATIONS is no longer complete if we constrain the resolution process to input resolution, i.e., consider only pairs $(c, c')$ in Step 3 where at least one of $c$ and $c'$ is from $\varphi$ (which means that in the update of $O$ in Step 3, the part \{$(d'', d') \mid d'' \in S$\} is omitted). This is shown by the following example.

**Example 7.** Consider the Horn CNF $\varphi = (\overline{x}_0 \lor x_1)(\overline{x}_1 \lor \overline{x}_2 \lor x_3)(\overline{x}_1 \lor \overline{x}_3 \lor x_4)$ over $x_0, \ldots, x_4$. As easily seen, $\varphi$ is acyclic. Moreover, $\varphi$ is prime. There are three further prime implicates containing $\overline{x}_0$, viz. $c_1 = \overline{x}_0 \lor \overline{x}_2 \lor x_3$, $c_2 = \overline{x}_0 \lor x_3 \lor x_4$, and $c_3 = \overline{x}_0 \lor \overline{x}_2 \lor x_4$. Hence, $\overline{q} = \overline{x}_0$ has the nontrivial explanations $E_1 = \{\overline{x}_1\}$, $E_2 = \{x_2, \overline{x}_3\}$, $E_3 = \{x_3, \overline{x}_4\}$, and $E_4 = \{x_2, \overline{x}_4\}$. If at least one of the clauses $(c, c')$ in Step 3 must be from $\varphi$, then $E_2$ and $E_3$ are generated from $(\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3, \overline{x}_0 \lor x_1)$ and $(\overline{x}_1 \lor x_3 \lor x_4, \overline{x}_0 \lor x_1)$, respectively, while $E_4$ is missed: The pairs $(\overline{x}_1 \lor x_3 \lor x_4, \overline{x}_0 \lor \overline{x}_2 \lor x_3)$ and $(\overline{x}_1 \lor \overline{x}_2 \lor x_3, \overline{x}_0 \lor \overline{x}_3 \lor x_4)$ yield the same resolvent $\overline{x}_1 \lor \overline{x}_1 \lor \overline{x}_2 \lor x_4$, for which $\varphi^* \not\models (c \lor c' \setminus \{\overline{q}\})$ fails since $\overline{x}_1 \lor \overline{x}_2 \lor x_4$, which is the resolvent of the last two clauses in $\varphi$, is an implicite. Note that $E_4$ is generated from each of the excluded symmetric pairs $(\overline{x}_0 \lor \overline{x}_2 \lor x_3, \overline{x}_0 \lor \overline{x}_3 \lor x_4)$ and $(\overline{x}_0 \lor \overline{x}_3 \lor x_4, \overline{x}_0 \lor \overline{x}_2 \lor x_3)$. □

In terms of generating prime implicates, this contrasts with the cases of computing all prime implicates of a Horn CNF and all prime implicates that contain a positive literal $q$, for which input-resolution style procedures are complete, cf. [234].

## 5 Compound Queries

In this section, we consider generating all explanations for queries beyond literals. Theorem [11] implies that this problem is intractable for any common class of CNF queries which admits a negative literal. However, also for positive CNFs, it is intractable.

**Theorem 3.** Deciding whether a given CNF $\chi$ has an explanation from a Horn CNF $\varphi$ is NP-complete. Hardness holds even if $\chi$ is positive and $\varphi$ is negative (thus acyclic).

**Proof.** Membership in NP easily follows from Lemma [11] Hardness is shown via a reduction from the classical EXACT HITTING SET problem. Let $S = \{S_1, \ldots, S_m\}$ be a collection of subsets $S_i \subseteq U$ of a finite set $U$. Construct $\chi = \bigwedge_i \left( \bigvee_{x \in S_i} \overline{x} \right)$ and $\varphi = \bigwedge_i \bigwedge_{x \neq y \in S_i} (x \lor \overline{y})$. Then $\chi$ has an explanation from $\varphi$ iff there exists an exact hitting set for $S$, i.e., a set $H \subseteq U$ such that $|H \cap S_i| = 1$ for all $i \in \{1, \ldots, m\}$. □
For important special cases of positive CNFs, we obtain positive results. In particular, this holds if the query $\chi$ is restricted to be a clause or a term.

**Theorem 4.** Computing polynomially many (resp., all) explanations for a query $\chi$ which is either a positive clause or a positive term from a Horn CNF $\varphi$ is feasible in polynomial time (resp., polynomial total time).

**Proof.** Let us first consider the case in which $\chi$ is a positive clause $c = \bigvee_{x \in P(c)} x$. Then let $\varphi^* = \varphi \land \bigwedge_{x \in P(c)} (\overline{x} \lor x^*)$, where $x^*$ is a new letter. As easily seen, $\varphi^*$ is a Horn CNF and there is a one-to-one correspondence between explanations for a query $\chi$ from $\varphi$ and the ones for $x^*$ form $\varphi^*$ (except for a trivial explanation $x^*$). This, together with the result in [11] that all explanations for a query $\chi = q$ where $q$ is an atom from a Horn CNF can be generated with incremental polynomial delay, proves the theorem.

Similarly, if $\chi$ is a positive term $t = \bigwedge_{x \in P(t)} x$, one can consider explanations for $x^*$ from the Horn CNF $\varphi^* = \varphi \land (\bigvee_{x \in P(t)} \overline{x} \lor x^*)$, where $x^*$ is a new letter. $\square$

In case of acyclic Horn theories, the positive result holds also in the case where negative literals are present in a clause query.

**Theorem 5.** Computing polynomially many (resp., all) explanations for a query $\chi = c$ where $c$ is a clause from an acyclic Horn CNF $\varphi$ is feasible in polynomial time (resp., polynomial total time).

**Proof.** Let $\chi = \bigvee_{x \in P(c)} x \lor \bigvee_{x \in N(c)} \overline{x}$. Then let $\varphi^* = \varphi \land \bigwedge_{x \in P(c)} (\overline{x} \lor x^*) \land \bigwedge_{x \in N(c)} (x \lor \overline{x^*})$, where $x^*$ is a new letter. It is not difficult to see that $\varphi^*$ is an acyclic Horn CNF, and there is a one-to-one correspondence between explanations for a query $\chi$ from $\varphi$ and the ones for $\overline{x^*}$ from $\varphi^*$ (except for a trivial explanation $\overline{x^*}$). This together with Theorem 2 proves the theorem.

$\square$

Note that explanations for a single clause query $\chi = c$ correspond to the minimal support clauses for $c$ as used in Clause Management Systems [22]. Thus, from Theorems 1 and 5 we obtain that while in general, generating all minimal support clauses for a given clause $c$ is not possible in polynomial total time unless P = NP, it is feasible with incremental polynomial delay for acyclic Horn theories.

The presence of negative literals in a query $\chi = t$ for a term $t$ from an acyclic Horn theory is more involved; a similar reduction technique as for a clause to a single literal seems not to work. We can show that generating all nontrivial explanations $E$ (i.e., $E \cap \chi = \emptyset$) for a term is intractable; the case of all explanations is currently open.

### 6 Conclusion

We considered computing all abductive explanations for a query $\chi$ from a propositional Horn CNF $\varphi$, which is an important problem that has many applications in AI and Computer Science. We presented a number of new complexity results, which complement and extend previous results in the literature; they are compactly summarized in Table 1.

We showed the intractability of computing all abductive explanations for a negative literal query $\chi$ from a general Horn CNF $\varphi$ (thus closing an open issue), while we
Table 1. Complexity of computing all abductive explanations for a query $\chi$ from a Horn theory (PTT = polynomial total time, NPTT = not polynomial total time unless P=NP)

<table>
<thead>
<tr>
<th>Query $\chi$</th>
<th>CNF</th>
<th>single literal</th>
<th>single clause</th>
<th>single term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horn theory $\Sigma$, by general positive</td>
<td>atom $q$ $\overline{q}$</td>
<td>positive general</td>
<td>positive general</td>
<td></td>
</tr>
<tr>
<td>Horn CNF $\varphi$ NPTT NPTT PTT$^a$ NPTT PTT NPTT PTT NPTT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acyclic Horn CNF $\varphi$ NPTT NPTT PTT$^a$ PTT PTT PTT –</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$^a$ By the results of [11].

presented a polynomial total time algorithm for acyclic Horn CNFs. Since this amounts to computing all prime implicates of $\varphi$ which contain $q$, we have obtained as a byproduct also new results on computing all such prime implicates from a Horn CNF. Note that our intractability result contrasts with the result in [2] that all prime implicates of a Horn CNF are computable in polynomial total time. Furthermore, our results on clause queries imply analogous results for generating all minimal support clauses for a clause in a Clause Management System [22].

It remains for further work to complete the picture and to find further meaningful input classes of cyclic Horn theories which permit generating a few resp. all explanations in polynomial total time. For example, this holds for clause queries from quadratic Horn CNFs (i.e., each clause is Horn and has at most 2 literals) and for literal queries from Horn CNFs in which each clause contains the query literal. Another issue is a similar study for the case of predicate logic.

Acknowledgments

We thank the reviewers for helpful suggestions.

References

12. T. Eiter and K. Makino. Generating all abductive explanations for queries on propositional
14. G. Friedrich, G. Gottlob, and W. Nejdl. Hypothesis classification, abductive diagnosis, and
16. H. Kleine Bünning and T. Lettmann. *Aussagenlogik - Deduktion und Algorithmen*. B.G. Teub-
18. P. Marquis. Consequence finding algorithms. In D. Gabbay, Ph. Smets (eds), *Handbook of
22. R. Reiter and J. de Kleer. Foundations of assumption-based truth maintenance systems:
Refined Complexity Analysis of Cut Elimination

Philipp Gerhardy

BRICS*
Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Aarhus C, Denmark
peegee@daimi.au.dk

Abstract. In [1,2] Zhang shows how the complexity of cut elimination depends on the nesting of quantifiers in cut formulas. By studying the role of contractions we can refine that analysis and show how the complexity depends on a combination of contractions and quantifier nesting. With the refined analysis the upper bound on cut elimination coincides with Statman’s lower bound. Every non-elementary growth example must display a combination of nesting of quantifiers and contractions similar to Statman’s lower bound example. The upper and lower bounds on cut elimination immediately translate into bounds on Herbrand’s theorem. Finally we discuss the role of quantifier alternations and show an elementary upper bound for the $\forall - \land$-case (resp. $\exists - \lor$-case).

1 Introduction

The most commonly used proofs of cut elimination by Schwichtenberg[3] and Buss[4] give an estimate of the depth, resp. size, of a cut free proof in terms of the (logical) depth of the largest cut formula and the depth, resp. size, of the original proof. As shown by Zhang[1,2] this bound can be refined by distinguishing between propositional connectives and quantifiers in the complexity analysis of cut elimination, showing that the elimination of quantifiers causes the non-elementary complexity. We derive a proof of Zhang’s improved result from Buss’s (clear and understandable) proof of cut elimination in [4]. In [5,6] Luckhardt discusses simplifications of Zhang’s result, which originally motivated the proof given below.

Analysing contractions we can show that all non-elementary growth examples for cut elimination must display both nested quantifiers and also contractions on ancestors of cut formulas. If the proof is contraction free or if one restricts contraction w.r.t. the nesting of quantifiers in contracted formulas, cut elimination can be shown to be elementary. We discuss how Statman’s lower bound example displays an optimal combination of nested quantifiers and contractions, and how Statman’s lower and our refined upper bound coincide. These bounds translate directly into bounds on the size of a Herbrand disjunction, i.e. the number of

* Basic Research in Computer Science (www.brics.dk), funded by the Danish National Research Foundation.
different formula instances in the disjunction. Finally we discuss a further refinement of cut elimination based upon eliminating arbitrary $\forall - \land$-cuts (resp. $\exists - \lor$-cuts) with exponential complexity.

2 Notation and Definitions

2.1 The Calculus LK

We use a version of the sequent calculus LK. We write $x, y, z$ for bound and $\alpha, \beta, \gamma$ for free variables. Terms and formulas are defined in the usual way. An LK-proof is a rooted tree in which the nodes are sequents. The root of the tree is called the end-sequent, the leaves $A \vdash A$ for atomic formulas $A$ are called initial sequents. All nodes but the initial sequents must be inferred by one of the rules below. Proofs are denoted by $\phi, \chi$ and $\psi$. $\Gamma, \Delta, \Pi$ and $\Lambda$ serve as metavariables for multisets of formulas. The multiset of formulas to the left of the separation symbol $\vdash$ is called antecedent, the multiset to the right is called succedent.

The logical rules are:

\[
\frac{A, B, \Gamma \vdash \Delta}{A \land B, \Gamma \vdash \Delta} \quad \land : l
\]
\[
\frac{\Gamma \vdash \Delta, A \Pi \vdash A, B}{\Gamma, \Pi \vdash \Delta, A \land B} \quad \land : r
\]
\[
\frac{A, \Gamma \vdash \Delta \quad B, \Pi \vdash A}{A \lor B, \Gamma, \Pi \vdash \Delta, A} \quad \lor : l
\]
\[
\frac{\Gamma \vdash \Delta, A \quad B, \Pi \vdash A}{\Gamma \vdash \Delta, A \lor B} \quad \lor : r
\]
\[
\frac{\Gamma \vdash \Delta, A}{\neg A, \Gamma \vdash \Delta} \quad \neg : l
\]
\[
\frac{A, \Gamma \vdash \Delta}{\Gamma, \neg A \vdash \Delta} \quad \neg : r
\]
\[
\frac{\Gamma, (\forall x) A \vdash \Delta}{\Gamma, A\{x \leftarrow t\} \vdash \Delta} \quad \forall : l
\]
\[
\frac{\Gamma \vdash \Delta, A\{x \leftarrow \alpha\}}{\Gamma \vdash \Delta, (\forall x) A} \quad \forall : r, \ \alpha \notin FV(\forall x A(x))
\]
\[
\frac{\Gamma \vdash \Delta, A\{x \leftarrow t\}}{\Gamma, (\exists x) A \vdash \Delta} \quad \exists : l
\]
\[
\frac{\Gamma, A\{x \leftarrow \alpha\} \vdash \Delta}{\Gamma, (\exists x) A \vdash \Delta} \quad \exists : r, \ \alpha \notin FV(\exists x A(x))
\]
\[
\forall : r \text{ (resp. } \exists : l) \text{ must fulfill the eigenvariable condition, i.e. the free variable } \alpha \text{ does not occur in } \Gamma \vdash \Delta. \text{ In } \forall : l \text{ (resp. } \exists : r) \text{ } t \text{ may be an arbitrary term, but admitting only free variables.}
\]

The structural rules are:

\[
\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \quad w : l
\]
\[
\frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \quad w : r
\]
\[
\frac{A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \quad c : l
\]
\[
\frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, A} \quad c : r
\]
\[
\frac{\Gamma \vdash \Delta, A \Pi \vdash A}{\Gamma, \Pi \vdash \Delta, A} \quad \text{cut}
\]
Weakening and contraction inferences are called *weak* inferences. All other inferences are called *strong*. In the contraction free case the above multiplicative version of LK corresponds to the multiplicative fragment of affine linear logic \[7,8\], which also is discussed as direct predicate calculus in \[9,10\].

2.2 Definitions: Properties of Terms, Formulas and Proofs

**Definition 1.** (free variable normal form). A proof \(\phi\) is in free variable normal form if no variable that is free in the end-sequent is used as an eigenvariable somewhere in the proof and every other free variable appearing in \(\phi\) is used exactly once as an eigenvariable and appears in \(\phi\) only in sequents above the inference in which it is used as an eigenvariable.

**Definition 2.** mid-sequent A proof \(\phi\) is in mid-sequent form if it can be divided in an upper, quantifier-free part and a lower part, consisting only of quantifier inferences and contractions. The last sequent before the first quantifier inference is called the mid-sequent.

**Definition 3.** (depth and size of a proof). The depth \(|\phi|\) of a proof \(\phi\) is the depth of the proof tree, counting only strong inferences and axioms. The size \(||\phi||\) of a proof \(\phi\) is the number of strong inferences and axioms in the proof.

**Definition 4.** (depth of a formula). The depth \(|A|\) of an atomic formula \(A\) is defined as \(|A| = 0\). For formulas \(A\) and \(B\) we define \(|A \lor B| = |A \land B| = |A \to B| = \max(|A|, |B|) + 1\), and \(|\neg A| = |\forall x A(x)| = |\exists x A(x)| = |A| + 1\).

**Definition 5.** (cut-rank \(\rho\) of a proof). The cut-rank \(\rho(\phi)\) of a proof \(\phi\) is the supremum of the depths of the cut formulas in \(\phi\).

**Definition 6.** (nested quantifier depth of formulas and proofs). The nested quantifier depth \(nqf(A)\) of an atomic formula \(A\) is defined as \(nqf(A) = 0\). For formulas \(A\) and \(B\) we define \(nqf(A \lor B) = nqf(A \land B) = nqf(A \to B) = \max(nqf(A), nqf(B))\), \(nqf(\neg A) = nqf(A)\), and \(nqf(\forall x A(x)) = nqf(\exists x A(x)) = nqf(A) + 1\). The nested quantifier depth \(nqf(\phi)\) of a proof \(\phi\) is the supremum of the nested quantifier depths of the cut formulas in \(\phi\).

**Definition 7.** (propositional depth of formulas). Let \(B\) be a formula occurrence in \(A\) and let \(A\) be constructed from \(B\) and other formulas by propositional connectives only. Then the propositional depth of \(B\) in \(A\) is the number of propositional connectives which have to be removed from \(A\) to obtain \(B\).

**Definition 8.** (deepest quantified formulas and proofs). For a formula \(A\) we define \(dqf(A)\) as the supremum over the propositional depths of subformulas \(B\) of \(A\) which have \(nqf(B) = nqf(A)\). Let \(\phi\) be a proof of \(\Gamma \vdash \Delta\). Then \(dqf(\phi)\) is the supremum over \(dqf(A_i)\) for cut formulas \(A_i\) in \(\phi\) which have \(nqf(A_i) = nqf(\phi)\).

**Definition 9.** (the hyper-exponential function). The hyper-exponential function \(2^x_n\) is defined as \(2^x_0 \equiv x\) and \(2^x_{n+1} \equiv 2^{2^x_n}\). We write \(2^n_n\) for \(2^0_0\).
3 Cut Elimination

3.1 Zhang’s Refined Bound on Cut Elimination

To prove Zhang’s refined upper bound on cut elimination we state the following
Refined Reduction Lemma:

Refined Reduction Lemma. Let $\phi$ be an $LK$-proof of a sequent $\Gamma \vdash \Delta$ with
the final inference a cut with cut formula $A$. Then if for all other cut formulas $B$

(i) $nqf(A) \geq nqf(B)$ and $dqf(\phi) = dqf(A) > dqf(B)$, then there exists a
proof $\phi'$ of the same sequent with $dqf(\phi') \leq dqf(\phi) - 1$ and $|\phi'| \leq |\phi| + 1$.

(ii) $nqf(\phi) = nqf(A) > nqf(B)$ and $dqf(A) = 0$, then there exists a proof $\phi'$ of the same sequent with $nqf(\phi') \leq nqf(\phi) - 1$ and $|\phi'| < 2 \cdot |\phi|$.

If the cut formula $A$ is atomic and both subproofs are cut free, then there is
a cut free proof $\phi'$ with $|\phi'| < 2 \cdot |\phi|$.

Proof: The proof is by cases on the structure of the cut formula $A$. We assume
w.l.o.g. that $\phi$ is is free variable normal form and that for no cut formula all
direct ancestors in either subproof are introduced by weakenings only, as the
reduction is trivial in that case. We implicitly skip weakening inferences that
introduce a direct ancestor to a cut formula. This can be done without harming
the validity of the proof transformations given below.

Case 1: $A \equiv \neg B$. We then have the final inference of $\phi$:

$$
\frac{
\Gamma \vdash \Delta, \neg B \quad \neg B, \Pi \vdash A
}{
\Gamma, \Pi \vdash \Delta, A \quad \text{cut}
}$$

We transform the proofs $\chi$ and $\psi$ into proofs $\chi'$ and $\psi'$ of the sequents $B, \Gamma \vdash \Delta$ and $\Pi \vdash \Delta, A$ respectively. In $\chi$ we find all $\neg : r$ inferences which introduce a
direct ancestor of $\neg B$. By skipping these inferences, and replacing contractions
to the right on $\neg B$ with contractions to the left on $B$, we obtain a proof of $B, \Gamma \vdash \Delta$. The transformation of $\psi$ into $\psi'$ is similar. The final inference of the
modified proof $\phi'$ is then:

$$
\frac{
\Pi \vdash \Delta, A, B \quad B, \Gamma \vdash \Delta
}{
\Gamma, \Pi \vdash \Delta, A \quad \text{cut}
}$$

Trivially $dqf(\phi') \leq dqf(\phi) - 1$ and $|\phi'| \leq |\phi|$.

Case 2a: $A \equiv B \lor C$. We then have the final inference of $\phi$:

$$
\frac{
\Gamma \vdash \Delta, B \lor C \quad B \lor C, \Pi \vdash A
}{
\Gamma, \Pi \vdash \Delta, A \quad \text{cut}
}$$

We transform the proofs $\chi$ and $\psi$ into proofs $\chi'$ of the sequent $\Gamma \vdash \Delta, B, C$, and
into proofs $\psi_B$ and $\psi_C$ of the sequents $B, \Pi \vdash \Delta, A$ and $C, \Pi \vdash \Delta$ respectively. In
χ we skip the ∨ : r inferences where direct ancestors of the cut formula $B \lor C$ are introduced and replace contractions on $B \lor C$ by contractions on $B$ and $C$. For the transformed proof $|\chi'| \leq |\chi|$. To transform $\psi$ into $\psi_B$ we replace all inferences where a direct ancestor of the cut formula $B \lor C$ has been introduced

$$B, \Pi_0 \vdash A_0 \quad C, \Pi_1 \vdash A_1$$

$$B \lor C, \Pi' \vdash \Lambda'$$

(with $\Pi' = \Pi_0 \cup \Pi_1$ and $\Lambda' = A_0 \cup A_1$) with inferences

$$B, \Pi_0 \vdash A_0 \quad \Pi' \vdash \Lambda'$$

We replace contractions on $B \lor C$ with contractions on $B$. The proof for $\psi_C$ is obtained in a similar way. In both instances no strong inferences have been added to the proof, so $|\psi_B| \leq |\psi|$ and $|\psi_C| \leq |\psi|$. The transformed proofs can then be combined to a new proof $\phi'$ in the following way:

$$\Gamma \vdash \Delta, B, C \quad B, \Pi \vdash \Lambda \quad \psi_B$$

$$\Gamma, \Pi \vdash \Delta, \Lambda, C \quad \text{cut} \quad C, \Pi \vdash \Lambda \quad \psi_C$$

$$\Gamma, \Pi \vdash \Delta, \Lambda \quad \text{cut}, c : r, l$$

Then $dqf(\phi') \leq dqf(\phi) - 1$ and $|\phi'| = \sup(|\chi'|, |\psi_B|, |\psi_C|) + 2 \leq |\phi| + 1$.

**Case 2b+c:** $A \equiv B \land C$, $A \equiv B \rightarrow C$. These cases are symmetrical to 2a.

**Case 3a:** $A \equiv \exists x B(x)$. We then have the final inference of the proof:

$$\Gamma \vdash \Delta, \exists x B(x) \quad \exists x B(x), \Pi \vdash \Lambda \quad \psi$$

$$\Gamma, \Pi \vdash \Delta, \Lambda \quad \text{cut}$$

In the subproof $\chi$ we find the $k \exists : r$ inferences where direct ancestors of the formula $\exists x B(x)$ have been introduced and enumerate these (for $1 \leq i \leq k$)

$$\Gamma_i \vdash \Delta_i, B(t_i) \quad \exists: r$$

Similarly in $\psi$ we enumerate all $l \exists : l$ inferences which introduce a direct ancestor of the cut formula as (for $1 \leq j \leq l$)

$$B(\alpha_j), \Pi_j \vdash A_j \quad \exists x B(x), \Pi_j \vdash A_j \quad \exists: r$$

We obtain proofs $\psi_i$ of the sequents $B(t_i), \Pi \vdash \Lambda$ by replacing in $\psi$ all $l$ variables $\alpha_j$ with the term $t_i$. This is unproblematic since the proof is in free variable normal form. We replace the contractions on $\exists x B(x)$ by contractions on $B(t_i)$. 
We construct a proof \( \phi' \) by replacing the \( k \) enumerated quantifier introductions in \( \chi \) with cuts, so that the inferences become

\[
\frac{\Gamma_i \vdash \Delta_i, B(t_i) \quad \psi_i}{\Gamma_i, \Pi \vdash \Delta_i, A} \quad \exists : r
\]

We contract multiple copies of \( \Pi \) and \( \Lambda \) in the end sequent to make \( \phi' \) a proof of \( \Gamma, \Pi \vdash \Delta, \Lambda \). Trivially \( nqf(\phi') \leq nqf(\phi) - 1 \) and since \( |\chi_i| \leq |\chi| < |\phi| \) and \( |\psi_i| \leq |\psi| < |\phi| \) and \( |\phi'| \leq |\chi| + \sup\{|\psi_i|\} \), also \( |\phi'| < 2 \cdot |\phi| \).

**Case 3b**: \( A \equiv \forall x B(x) \). This case is symmetrical to 3a.

**Case 4**: \( A \) is atomic. We then have the final inference:

\[
\frac{\chi}{\Gamma \vdash \Delta, A} \quad \frac{\psi}{A, \Pi \vdash \Delta, A} \quad \text{cut}
\]

By assumption \( \chi \) and \( \psi \) are cut free. We obtain the desired proof by modifying the proof \( \psi \) of \( A, \Pi \vdash \Lambda \). We remove from \( \psi \) all direct ancestors of the cut formula \( A \) and add \( \Gamma \) and \( \Delta \) to the antecedent, resp. succedent, of each sequent, where an ancestor of the cut formula \( A \) has been removed. We then ensure all initial sequents in \( \psi \) have valid proofs. The initial sequents \( B \vdash B \) for \( B \neq A \) do not need new proofs. For \( \Gamma \vdash \Delta, A \) we use the (cut free) subproof \( \chi \), thus obtaining a proof of \( \Gamma^k, \Pi \vdash \Delta^k, A \), where \( k \) is the number of initial sequents we had to “repair”. By contractions we arrive at a proof \( \phi' \) of \( \Gamma, \Delta \vdash \Pi, \Lambda \), where \( \phi' \) is cut free and \( |\phi'| < 2 \cdot |\phi| \).

We use the following lemmas to prove Zhang’s refined cut elimination theorem:

**Lemma 10.** Let \( \phi \) be an \( \text{LK} \)-proof of a sequent \( \Gamma \vdash \Delta \). If \( dqf(\phi) = d > 0 \), then there is a proof \( \phi' \) of the same sequent with \( dqf(\phi') = 0 \) and \( |\phi'| \leq 2^d \cdot |\phi| \).

**Proof:** By the Refined Reduction Lemma and induction on \( d \).

**Lemma 11.** Let \( \phi \) be an \( \text{LK} \)-proof of a sequent \( \Gamma \vdash \Delta \). If \( dqf(\phi) = 0 \) and \( nqf(\phi) = d > 0 \), then there is a proof \( \phi' \) of the same sequent with \( nqf(\phi') \leq d - 1 \) and \( |\phi'| < 2^{d|\phi|} \).

**Proof:** By the Refined Reduction Lemma and induction on \( |\phi| \).

**First Refined Cut Elimination Theorem.** Let \( \phi \) be an \( \text{LK} \)-proof of a sequent \( \Gamma \vdash \Delta \). If \( nqf(\phi) = d > 0 \), then there is a proof \( \phi' \) of the same sequent and a constant \( c \), depending only on the propositional nesting of the cut formulas, so that \( nqf(\phi') \leq d - 1 \) and \( |\phi'| \leq 2^{c|\phi|} \).

**Proof:** Assume we have a proof with \( dqf(\phi) = k \) and \( nqf(\phi) = d \). Then by Lemma 10 we get a proof \( \phi'' \) with \( dqf(\phi'') = 0 \) and \( |\phi''| \leq 2^k \cdot |\phi| \). Let \( c \geq 2^k \), then by Lemma 11 we get a proof \( \phi' \) with \( nqf(\phi') \leq d - 1 \) and \( |\phi'| \leq 2^{c|\phi|} \).
Corollary 12. Let $\phi$ be an $\text{LK}$-proof of a sequent $\Gamma \vdash \Delta$ and let $\text{nqf}(\phi) = d$. Then there is a constant $c$, depending only on the propositional nesting of the cut formulas, and a proof $\phi'$ of the same sequent where $\phi'$ is cut free and $|\phi'| \leq 2^{c|\phi|}d + 1$.

Proof: By the First Refined Cut Elimination Theorem and induction on $d$. $\square$

3.2 The Role of Contractions in Cut Elimination

We show that if no direct ancestor of a cut formula has been contracted, the cut can be reduced by a mere rearrangement of the proof. We define $\text{cnqf}$, the contracted nested quantifier depth, of formulas and proofs.

Definition 13. Let $A$ be a cut formula in a proof and let $B_1, B_2, \ldots, B_k$ be ancestors of $A$, s.t. $B_i$ is a principal formula in a contraction inference. Then the contracted nested quantifier depth $\text{cnqf}(A)$, is the supremum over $\text{nqf}(B_i)$.

For $\phi \text{cnqf}(\phi)$ is the supremum over $\text{cnqf}(A_i)$ for cut formulas $A_i$.

We state the following variant of the reduction lemma:

Lemma 14. Let $\phi$ be a proof with the final inference a cut with cut formula $A$, where there have been no contractions on direct ancestors of the cut formula. Then if for all other cut formulas $B(i)$ $\text{nqf}(A) > \text{nqf}(B)$ and $\text{dqf}(A) = 0$, then we can find a proof $\phi'$ of the same sequent where $\text{nqf}(\phi') \leq \text{nqf}(\phi) - 1$.

Case 1: $A \equiv \neg B$. We use the same proof transformation as before. No new contractions have been added to $\phi'$, so $\text{dqf}(\phi') \leq \text{dqf}(\phi)$ and $||\phi'|| \leq ||\phi||$.

Case 2a: $A \equiv B \lor C$. Since no contractions have been made on direct ancestors of the cut formula, we can find the unique inferences where the $\lor$-connective has been introduced. Those inferences are (in $\chi$, resp.$\psi$):

$$
\frac{\chi_0}{\Gamma' \vdash \Delta', \lor B, C} \quad \frac{\psi_0}{\Gamma' \vdash \Delta', B \lor C} \quad \frac{\psi_1}{\Gamma' \vdash \Delta', B \lor C}
$$

where $\Pi' \equiv \Pi_0 \cup \Pi_1$ and $\Lambda' \equiv A_0 \cup A_1$. We construct the following new proof:

$$
\frac{\chi_0}{\Gamma' \vdash \Delta', \lor B, C} \quad \frac{\psi_0}{\Gamma', \Pi_0 \vdash A_0} \quad \frac{\psi_1}{\Gamma', \Pi_1 \vdash A_1}
$$
The remaining steps to get a proof of $\Gamma, \Pi \vdash \Delta, \Lambda$ are as in the subproofs $\chi$ and $\psi$ respectively. In the new proof $\phi'$ we replaced two $\lor$-introductions and a cut by two new cuts. No new contractions have been added to $\phi'$, so $dqf(\phi') \leq dqf(\phi) - 1$ and $||\phi'|| \leq ||\phi||$.

**Case 2b+c:** $A \equiv B \land C$, $A \equiv B \rightarrow C$. These cases are symmetrical to 2a.

**Case 3a:** $A \equiv \exists x B(x)$. We again know that in $\chi$ there is a unique inference where the quantified variable $x$ has been introduced for some term $t$:

$$
\frac{\Gamma', \Delta', B(t)}{\Gamma', \Delta', \exists x B(x)} \exists : r
$$

The subproof $\psi$ can be transformed to a proof $\psi'$ of $B(t), \Pi \vdash \Lambda$ as described earlier. These proofs can now be combined to a new proof $\phi'$ by replacing the inference in $\chi$ with a cut

$$
\frac{\chi', B(t)}{\Gamma', \Pi \vdash \Delta', \Lambda} \text{cut}
$$

and then continuing with $\chi$ as before. We skip some inferences and then rearrange the proof, adding no new contractions, so $nqf(\phi') \leq nqf(\phi) - 1$ and $||\phi'|| \leq ||\phi||$.

**Case 3b:** $A \equiv \forall x B(x)$. This case is symmetrical to 3a.

**Case 4:** $A$ is atomic. Again we know there is exactly one ancestor to the cut formula in each subproof. We showed earlier how to obtain a cut free proof of $\Gamma^k, \Pi \vdash \Delta^k, \Lambda$, where $k$ was the number of direct ancestors of the cut formula $A$ in the left subproof $\psi$. Since now $k = 1$ we automatically have a proof $\phi'$ of $\Gamma, \Pi \vdash \Delta, \Lambda$, and $||\phi'|| \leq ||\phi||$.

**Contraction Lemma.** Let $\phi$ be an LK-proof of a sequent $\Gamma \vdash \Delta$, with $nqf(\phi) > cnqf(\phi)$ then there is proof $\phi'$ of the same sequent with $nqf(\phi') = cnqf(\phi')$ and $||\phi'|| \leq ||\phi||$. As a consequence also $|\phi'| \leq 2^{|\phi|}$.

**Proof:** By induction on the number of uncontracted cuts and Lemma 14 we get a proof $\phi'$ s.t. $nqf(\phi') = cnqf(\phi')$ and $||\phi'|| \leq ||\phi||$. By $||\phi|| \leq 2^{|\phi|}$ and $|\phi'| \leq ||\phi'||$ we get $|\phi'| \leq 2^{|\phi|}$.

**Second Refined Cut Elimination Theorem.** Let $\phi$ be an LK-proof of a sequent $\Gamma \vdash \Delta$. Then there is a constant $c$ depending only on the propositional nesting of the cut formulas and a cut free proof $\phi'$ of the same sequent where $|\phi'| \leq 2^{c|\phi|}$. $c=2$.

**Remark.** In the contraction free case, i.e. for the multiplicative fragment of affine linear logic, we get an exponential upper bound on cut elimination. A comparable result was shown by Bellin and Ketonen in [10].
If we compare the results of this section with previous results by Schwichtenberg\textsuperscript{[3]}, Buss\textsuperscript{[4]} and Zhang\textsuperscript{[1,2]}, we get the following table:

<table>
<thead>
<tr>
<th>bound</th>
<th>complexity</th>
<th>complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schwichtenberg</td>
<td>$2^{</td>
<td>φ</td>
</tr>
<tr>
<td>Buss</td>
<td>$2^{</td>
<td>φ</td>
</tr>
<tr>
<td>Zhang</td>
<td>$2^{</td>
<td>nqf(φ)</td>
</tr>
<tr>
<td>Gerhardy</td>
<td>$2^{</td>
<td>cnqf(φ)</td>
</tr>
</tbody>
</table>

(*) In the case that $cnqf(φ) = nqf(φ)$ we can, as discussed, use the estimates based on $nqf(φ)$ to match Zhang’s bound.

### 3.3 Statman’s Theorem

A Kalmar non-elementary lower bound on Herbrand’s theorem, and hence also cut elimination, was first proved by Statman\textsuperscript{[11]}. Statman’s theorem states that there exist simple sequents $S_n$, for which we have short proofs (of depth linear in $n$), but for which every cut free proof must have depth at least $2n$. The original example used by Statman is formulated in relational logic. Later presentations are due to Orevkov\textsuperscript{[12]} and Pudlak\textsuperscript{[13]}. We will present Pudlak’s version of the theorem, which uses a simple fragment $T$ of arithmetic.

Let the theory $T$ be the language with function symbols $+, \cdot$, relation symbols $=$, $I(\cdot)$ and constants $0, 1$. The theory $T$ has the following non-logical axioms: $(x + y) + z = x + (y + z)$, $y + 0 = y$, $2^0 = 1$, $2^x + 2^x = 2^{1+x}$, $I(0)$ and $I(x) \to I(1+x)$.

We want to prove the sequents $S_n \equiv A_{T+eq} \vdash I(2_n)$, expressing that $T$ with some equality proves $2_n$ is an integer. Here $\bigwedge A_{T+eq}$ is the universal closure of a finite conjunction of axioms from $T$ and equality axioms over $T$. For the short proofs of $S_n$ we make use of recursively defined relations $R_i : R_0 \equiv I$, $R_{i+1}(x) : \forall y (R_i(y) \to R_i(2^x + y))$. The proofs will employ subproofs $φ_i$ and $ψ_i$ as presented in \textsuperscript{[14]}. Both $φ_i$ and $ψ_i$ have short, almost cut-free proofs, as only some cuts on instances of equality axioms are needed.

We give a sketch of short proofs $π_n$ of the sequents $S_n$. For convenience and readability $A_{T+eq}$ is only written in the end-sequent, though it is used throughout the proof. The cut and contraction inferences that are critical for the non-elementary complexity are marked with a $\ast$.

The proof continues in a symmetric fashion, ending with the top line:
Throughout the proof. For more details on the proofs by the number of formula instances of $\psi$

The subproofs for $\phi_i$ and $\psi_i$ are simple proofs of depth linear in $n$. We show the proof of $\psi_i$, as it is in these proofs that ancestors of cut formulas $R_i(0)$ are contracted. The proofs $\phi_i$ are contraction free. The proof $\psi_i$ is:

$$
\frac{R_{i-2}(2^\beta + \alpha) \vdash R_{i-2}(2^{\beta} + \alpha)}{R_{i-2}(2^{\beta} + \alpha) \vdash R_{i-2}(2^{1+\beta} + \alpha), R_{i-2}(2^\beta + \alpha) \vdash R_{i-2}(2^{1+\beta} + \alpha)} \vdash l
$$

$$
\frac{R_{i-1}(\beta), R_{i-2}(2^{\beta} + \alpha) \vdash R_{i-2}(2^{1+\beta} + \alpha)}{R_{i-1}(\beta), R_{i-2}(\alpha) \vdash R_{i-2}(2^{\beta} + \alpha) \vdash R_{i-2}(2^{1+\beta} + \alpha) \vdash r}
$$

The proof $\psi_n^*$ necessary for the simplification of the top line of the proof $\pi_n$ is obtained from $\psi_n$ by skipping the last two inferences and substituting 0 for $\beta$ throughout the proof. For more details on the proofs $\phi_i$ and $\psi_i$ see [14] and [15].

**Proposition 15.** There exist short proofs $\pi_n$ of sequents $S_n$ s.t. $|\pi_n| \leq \Theta(n)$.

**Proof:** See Pudlak [13], Baaz/Leitsch [14] or Gerhardy [15].

To estimate the lower bound for the complexity of cut elimination we will use a property of mid-sequents. It is common to extract mid-sequents from cut free proofs, but it can be shown that proofs with all cuts quantifier free suffice.

**Proposition 16.** Let $\phi$ be a proof of a prenex formula $A$ with at most quantifier free cuts, then we can extract a mid-sequent and bound the number of non-identical formula instances of $A$ in the mid-sequent by $2^{\beta|\phi|}$.

**Proof:** In Appendix.

Since we can derive a Herbrand disjunction from a mid-sequent we immediately have the following theorem:

**Theorem 17.** Let $\phi$ be a proof of a prenex formula $A$, then for some constant $c$ the number of formula instances of $A$ in a Herbrand disjunction can be bounded by $2^{-\beta|\phi|}c_{nqf}(\phi)+2^1$, resp. $2^{c|\phi|}nqf(\phi)+1$.

**Proof:** By the Second Refined Cut Elim. Theorem and Proposition 10.

**Lemma 18.** Let $S_n \equiv \exists \ldots \exists (\exists A_{T+eq} \rightarrow I(2_n))$ be the sequents as described above. Let $\chi_n$ be a proof in mid-sequent form of the sequent $S_n$. Then the mid-sequent must contain at least $2^n$ different instances of the axiom $I(x) \rightarrow I(1+x)$.
Proof: See Pudlak[13].

Theorem 19. Let the sequents $S_n$ be as described above. Then for every proof $\chi_n$ of $S_n$ with quantifier free cuts $|\chi_n| \geq 2^{n-1}$.

Proof: By Proposition[16] we obtain a mid-sequent from a proof $\phi$ with quantifier free cuts. By Lemma[18] a mid-sequent for $S_n$ must have at least $2^n$ different instances of the axiom $I(x) \rightarrow I(1 + x)$, and hence the proof must have depth at least $2^{n-1}$.

To compare the obtained lower bound for cut elimination with our upper bound for cut elimination, we state the following theorem:

Theorem 20. There are proofs $\chi_n$ of the sequents $S_n$ with at most quantifier free cuts, so that the depth of $\chi_n$ can be bounded by $2^{O(n)} n^{n-1}$, i.e. $|\chi_n| \leq 2^{O(n)} n^{n-1}$.

Proof: By Proposition[15] there exist short proofs $\pi_n$, of depth linear in $n$, of the sequents $S_n$. The cut formulas involving $R_i$, $i \leq n - 1$ in these short proofs have $\text{nqf}(R_i) \leq n - 1$. There are no cut formulas with deeper nesting of quantifiers in the proof. From the proof of Corollary[12] it follows that we can bound the depth of a proof with quantifier free cuts by $|\chi_n| \leq 2^{O(n)} n^{n-1}$.

The Second Improved Cut Elimination Theorem gives no further improvement of the upper bound, since sufficiently complex subformulas of cut formulas $R_i(0)$ are contracted in the subproofs $\psi_i$, i.e. $\text{cnqf}(\pi_n) = \text{nqf}(\pi_n)$. This demonstrates, how efficient these short proofs are in their use of cuts and contractions. Every non-elementary growth example must employ a similar combination of nesting of quantifiers and contractions, as we have shown that cut elimination in the absence of either becomes elementary.

Remark. In [16] Carbone defines the so-called bridge-groups of proofs which capture the structure of a proof w.r.t. cuts and contractions. It is shown that short proofs of Statman’s example correspond to the so-called Gersten-group for which distortion into cyclic subgroups is hyper-exponential. Carbone conjectures that for bridge-groups the complexity of distortion into cyclic subgroups characterizes the complexity of cut elimination for the corresponding proofs. Since the Gersten group is the only known 1-relator group with hyper-exponential distortion this further suggests that Statman’s lower bound example is a characteristic example of the non-elementary complexity of cut elimination.

In comparison to the refined upper bound, the bounds obtained from Schwichtenberg’s and Buss’s cut elimination theorem both estimate the complexity in the logical depth of the cut formulas $R_i$, where $|R_i| = 2i$. Comparing Statman’s lower bound to Schwichtenberg’s and Buss’s upper bounds, we notice a substantial gap between the upper and the lower bound. With the improved analysis of cut elimination the gap between our upper and Statman’s lower bound is now closed. In tabular form the bounds on proofs with quantifier free cuts compare as follows. Note that the bounds on the size of a proof equivalently give a bound on the size of a Herbrand disjunction:
Statman/Pudlak (lower bound) \( \frac{2^{n-1}}{2^n} \)
Schwichtenberg (upper bound) \( \frac{2^{\Theta(n)}}{2^{2n-1}} \)
Buss (upper bound) \( \frac{2^{\Theta(n)}}{2^{2n-2}} \)
Zhang/Gerhardy (upper bound) \( \frac{2^{\Theta(n)}}{2^{n-1}} \)

Remark. In [1] Zhang also states that the upper and lower bound coincide. His statement relies on two propositions: that there are, as shown above, proofs of the sequents \( S_n \) of depth linear in \( n \) with \( \text{nqf}(\) \( \leq n - 1 \), and that every cut free proof of the sequents \( S_n \) must have hyper-exponentially in \( n \) many quantifier inferences in serial. For the first proposition no proof is given in Zhang’s paper, for the second proposition a counterexample is presented in [15].

3.4 Further Improvement of the Cut Elimination Theorem

We can show that if all cut formulas are composed exclusively of atomic formulas and \( \forall \) and \( \land \) connectives (resp. \( \exists \) and \( \lor \)), cut elimination is exponential, regardless of the nesting of \( \forall \) (resp. \( \exists \)) quantifiers. For arbitrary cut formulas the complexity depends on the number of alternating \( \forall - \land \) and \( \exists - \lor \) blocks, as was shown in [2]. We discuss two propositions that naturally lead to the result.

Proposition 21. Assume we have in a proof \( \phi \) a cut inference with the cut formula \( \forall \bar{x} B(\bar{x}) \). Then we can eliminate that block of quantifiers simultaneously, replacing the cut with a number of cuts with \( B(\bar{t}) \), for appropriate terms \( \bar{t} \). For the modified proof \( \phi' \) we have that \( |\phi'| < 2 \cdot |\phi| \).

Proof: Let the cut inference be:

\[
\begin{array}{c}
\psi \\
\Gamma \vdash \Delta, \forall \ldots \forall \bar{x} B(\bar{x}) \\
\chi \\
\Gamma, \Pi \vdash \Delta, \Lambda \\
\end{array}
\]

\[
\frac{\psi}{\chi} \text{ cut}
\]

In \( \chi \) we enumerate the \( l \) inferences where an innermost \( \forall \) quantifier of an ancestor of the cut formula is introduced as (for \( 1 \leq i \leq l \))

\[
B(t_{i,1}, \ldots, t_{i,n}), \Pi_i \vdash \Lambda_i
\]

\[
\forall x_n B(t_{i,1}, \ldots, t_{i,n-1}, x_n), \Pi_i \vdash \Lambda_i
\]

\[
\forall : l
\]

Similar to the earlier proof transformation in the \( \forall \)-case, we can obtain proofs \( \psi_i \) of the sequents \( \Gamma \vdash \Delta, B(t_{i,1}, \ldots, t_{i,n}) \). Replacing the enumerated inferences in \( \chi \) by cuts with \( B(t_{i,1}, \ldots, t_{i,n}) \) we obtain the desired result.

Proposition 22. Assume we have in a proof \( \phi \) a cut inference with the cut formula \( \land_{i=1}^n B_i \). Then we can eliminate all those conjunctions simultaneously, replacing that cut with a number of cuts with \( B_i \). For the modified proof \( \phi' \) we have that \( |\phi'| < 2 \cdot |\phi| \).
Proof: The proof is almost identical to the proof of Proposition 21 but instead eliminating cuts where the “innermost” conjunct is introduced.

The crucial insight is that both for the ∀-quantifier and the ∧-connective, the projection to a subformula of the cut formula occurs in the left subproof, leading to new cut inferences in the right subproof χ. Therefore, if all cut formulas are composed exclusively of some formulas $B_i$ and the connectives ∀ and ∧, all connectives can be eliminated in one step.

Lemma 23. Let $\phi$ be a proof of a sequent $\Gamma \vdash \Delta$ with the last inference a cut. Let the cut formula be constructed from formulas $B_1, \ldots, B_n$ by the connectives ∀ and ∧ only (resp. ∃ and ∨). Then we can replace that cut by a number of smaller cuts with cut formulas $B_i$. For the resulting proof $\phi'$ we have $|\phi'| < 2 \cdot |\phi|$.

Proof: Combining the reductions for ∀ and ∧ described in Lemma 21 and Lemma 22 we replace the ∧ or ∀-introductions by cuts. For the resulting proof $\phi'$ we get $|\phi'| < 2 \cdot |\phi|$. The ∃ − ∨-case is handled symmetrically, with roles of the left and right subproofs reversed.

This immediately gives us a double-exponential bound for cut elimination if all cut formulas are composed of ∀, ∧ and atomic formulas. However, we can eliminate the ∀ and ∧ connectives and the remaining atomic cuts in one go, by combining the above lemmas with the technique to eliminate atomic cuts. We state the following theorem:

Theorem 24. Let $\phi$ be a proof of a sequent $\Gamma \vdash \Delta$ with all cut formulas ∀ − ∧ (resp. ∃ − ∨). Then there is a cut-free proof $\phi'$ of the same sequent with $|\phi'| \leq 2^{|\phi|}$.

Proof: In Appendix.

Remark. The result still holds, if we allow negation of atomic formulas in the cut formulas, as elimination of negation can easily be combined with the above cut elimination technique.

The above technique can easily be generalized to arbitrary cut formulas, if we restrict the logical connectives in the cut formulas to ∀, ∧, ∃, ∨ and ¬, with negation only appearing in front of atomic formulas. If we alternatingly eliminate ∀ − ∧-connectives and ∃ − ∨-connectives we get a non-elementary upper bound for cut elimination, where the height of the tower of exponentials depends on the number of alternations of ∀ − ∧ and ∃ − ∨ blocks in the cut formulas. Again the short proofs $\pi_n$ of sequents $S_n$ perfectly exploit this, as the forall-implication structure of the cut formulas $R_i$ corresponds to such alternations, i.e. the number of alternations in $R_i$ is $i$.

Already the elimination of ∀ − ∧ cuts with elementary complexity has applications. In [17] Baaz and Leitsch prove an exponential upper bound for cut elimination in a fragment of $LK$ called $QMON$. At the heart of the proof is a technique for eliminating ∀ − ∧ cuts similar to the one presented above. The rest of their proof relies on special properties of the fragment $QMON$. 

Acknowledgements

This paper describes results from my master’s thesis, written under the supervision of Ulrich Kohlenbach. I am greatly indebted to Ulrich Kohlenbach for many discussions on the subject and for useful comments on drafts of this paper. Furthermore, I am grateful to the referees for many helpful suggestions which led to an improved presentation of the results.

References

Comparing the Succinctness of Monadic Query Languages over Finite Trees

Martin Grohe and Nicole Schweikardt

Laboratory for Foundations of Computer Science
University of Edinburgh, Scotland, UK
{grohe,vschwe}@inf.ed.ac.uk

Abstract. We study the succinctness of monadic second-order logic and a variety of monadic fixed point logics on trees. All these languages are known to have the same expressive power on trees, but some can express the same queries much more succinctly than others. For example, we show that, under some complexity theoretic assumption, monadic second-order logic is non-elementarily more succinct than monadic least fixed point logic, which in turn is non-elementarily more succinct than monadic datalog.

Succinctness of the languages is closely related to the combined and parameterized complexity of query evaluation for these languages.

Keywords: Finite Model Theory, Monadic Second-Order Logic, Fixed Point Logics, μ-Calculus, Monadic Datalog, Tree-like structures, Succinctness

1. Introduction

A central topic in finite model theory has always been a comparison of the expressive power of different logics on finite relational structures. In particular, the expressive power of fragments of monadic second-order logic and various fixed-point logics has already been investigated in some of the earliest papers in finite model theory [Fag75,CH82]. One of the main motivations for such studies was an interest in the expressive power of query languages for relational databases.

In recent years, the focus in database theory has shifted from relational to semi-structured data and in particular data stored as XML-documents. A lot of current research in the database community is concerned with the design and implementation of XML query languages (see, for example, [FSW00,HP00,GK02] or the monograph [ABS99] for a general introduction into semi-structured data and XML). The languages studied in the present paper may be viewed as node-selecting query languages for XML. They all contain the core of the language XPath, which is an important building block of several major XML-related technologies. Recently, monadic datalog has been proposed as a node-selecting query language with a nice balance between expressive power and very good algorithmic properties [GK02,Koc03].

XML-documents are best modelled by trees, or more precisely, finite labelled ordered unranked trees. It turns out that when studying node-selecting query languages for

* Supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD)
Comparing the Succinctness of Monadic Query Languages over Finite Trees

XML-documents, expressive power is not the central issue. Quite to the contrary: Neven and Schwentick [NS02] proposed to take the expressive power of monadic second-order logic (MSO) as a benchmark for node-selecting XML-query languages and, in some sense, suggested that such languages should at least have the expressive power of MSO. However, even languages with the same expressive power may have vastly different complexities. For example, monadic datalog and MSO have the same expressive power over trees [GK02]. However, monadic datalog queries can be evaluated in time linear both in the size of the datalog program and the size of the input tree [GK02], and thus the combined complexity of monadic datalog is in polynomial time, whereas the evaluation of MSO queries is PSPACE complete. The difference becomes even more obvious if we look at parameterized complexity: Unless PTIME $\neq$ NP, there is no algorithm evaluating a monadic second-order query in time $f$ (size of query) $p$ (size of tree) for any elementary function $f$ and polynomial $p$ [FG03]. Similar statements hold for the complexity of the satisfiability problem for monadic datalog and MSO over trees.

The reason for this different behaviour is that even though the languages have the same expressive power on trees, in MSO we can express queries much more succinctly. Indeed, there is no elementary translation from a given MSO-formula into an equivalent monadic datalog program. We also say that MSO is non-elementarily more succinct than monadic datalog. Just to illustrate the connection between succinctness and complexity, let us point out that if there was an elementary translation from MSO to monadic datalog, then there would be an algorithm evaluating a monadic second-order query in time $f$ (size of query) $p$ (size of tree) for an elementary function $f$ and a polynomial $p$.

In this paper, we study the succinctness (in the sense just described) of a variety of fixed point logics on finite trees. Our main results are the following:

1. **MSO is non-elementarily more succinct than monadic least fixed point logic MLFP** (see Theorem 2). Unfortunately, we are only able to prove this result under the odd, but plausible complexity theoretic assumption that for some $i \geq 1$, NP is not contained in DTIME($n^{\log^{(i)}(n)}$), where $\log^{(i)}$ denotes the $i$ times iterated logarithm.

2. **MLFP is non-elementarily more succinct than its 2-variable fragment MLFP$^2$** (see Corollary 3).

3. **MLFP$^2$ is exponentially more succinct than the full modal $\mu$-calculus, that is, the modal $\mu$-calculus with future and past modalities** (see Theorem 3, Example 2, and Theorem 4).

4. **The full modal $\mu$-calculus is at most exponentially more succinct than stratified monadic datalog, and conversely, stratified monadic datalog is at most exponentially more succinct than the full modal $\mu$-calculus** (see Theorem 7 and 8). Furthermore, stratified monadic datalog is at most exponentially more succinct than monadic datalog (see Theorem 6).

The exact relationship between these three languages remains open.

Of course we are not the first to study the succinctness of logics with the same expressive power. Most known results are about modal and temporal logics. The motivation for these results has not come from database theory, but from automated verification and model-checking. The setting, however, is very similar. For example, Kamp’s
well known theorem states that first-order logic and linear time temporal logic have the same expressive power on strings [Kam68], but there is no elementary translation from first-order logic to linear time temporal logic on strings. Even closer to our results, monadic second-order logic and the modal $\mu$-calculus have the same expressive power on (ordered) trees, but again is well-known that there is no elementary translation from the former to the latter. Both of these results can be proved by simple automata theoretic arguments. More refined results are known for various temporal logics [Wil99, AI00, AI01, EVW02]. By and large, however, succinctness has received surprisingly little attention in the finite model theory community. Apart from automata theoretic arguments, almost no good techniques for proving lower bounds on formula sizes are known. A notable exception are Adler and Immerman’s [AI01] nice games for proving such lower bounds. Unfortunately, we found that these games (adapted to fixed point logic) were of little use in our context. So we mainly rely on automata theoretic arguments. An exception is the, complexity theoretically conditioned, result that MSO is non-elementarily more succinct than MLFP. To prove this result, we are building on a technique introduced in [FG03].

The paper is organised as follows: In Section 2 we fix the basic notations used throughout the paper. Section 3 concentrates on the translation from MSO to MLFP. In Section 4 we present our results concerning the two-variable fragment of MLFP and the full modal $\mu$-calculus. In Section 5 we concentrate on monadic datalog, stratified monadic datalog, and their relations to finite automata and to MLFP. Finally, Section 6 concludes the paper by pointing out several open questions.

Due to space limitations, we had to defer detailed proofs of our results to the full version of this paper [GS03].

2. Preliminaries

2.1. Basic Notations. Given a set $\Sigma$ we write $\Sigma^*$ to denote the set of all finite strings over $\Sigma$, and we use $\varepsilon$ to denote the empty string. We use $\mathbb{N}$ to denote the set $\{0, 1, 2, \ldots\}$ of natural numbers. We use $\log_2$ to denote the logarithm with respect to base 2. With a function $f$ that maps natural numbers to real numbers we associate the corresponding function from $\mathbb{N}$ to $\mathbb{N}$ defined by $n \mapsto \lceil f(n) \rceil$. For simplicity we often simply write $f(n)$ instead of $\lceil f(n) \rceil$.

The function $\text{Tower} : \mathbb{N} \to \mathbb{N}$ is inductively defined via $\text{Tower}(0) := 1$ and $\text{Tower}(h+1) = 2^{\text{Tower}(h)}$, for all $h \in \mathbb{N}$. I.e., $\text{Tower}(h)$ is a tower of 2s of height $h$.

We say that a function $f : \mathbb{N} \to \mathbb{N}$ has bound $f(m) \leq \text{Tower}(o(h(m)))$, for some function $h : \mathbb{N} \to \mathbb{N}$, if there is a function $g \in o(h)$ and a $m_0 \in \mathbb{N}$ such that for all $m \geq m_0$ we have $f(m) \leq \text{Tower}(g(m))$. Note that, in particular, every elementary function $f$ has bound $f(m) \leq \text{Tower}(o(m))$. Indeed, for every elementary function $f$ there is an $h \in \mathbb{N}$ such that, for all $n \in \mathbb{N}$, $f(n)$ is less than or equal to the tower of 2s of height $h$ with an $n$ on top.

2.2. Structures. A signature $\tau$ is a finite set of relation symbols and constant symbols. Each relation symbol $R \in \tau$ has a fixed arity $ar(R)$. A $\tau$-structure $\mathcal{A}$ consists of a set $\mathcal{U}_\mathcal{A}$ called the universe of $\mathcal{A}$, an interpretation $c^\mathcal{A} \in \mathcal{U}_\mathcal{A}$ of each constant symbol $c \in \tau$,
and an interpretation $R^A \subseteq (\mathcal{U}^A)^{ar(R)}$ of each relation symbol $R \in \tau$. All structures considered in this paper are assumed to have a finite universe.

The main focus of this paper lies on the class $\text{Trees}$ of finite binary trees. Precisely, finite binary trees are particular structures over the signature

$$\tau_{\text{Trees}} := \{ \text{Root}, \ 1^{\text{st}}\text{Child}, \ 2^{\text{nd}}\text{Child}, \ \text{Has-No-1}^{\text{st}}\text{Child}, \ \text{Has-No-2}^{\text{nd}}\text{Child} \},$$

where $\text{Root}$, $\text{Has-No-1}^{\text{st}}\text{Child}$, $\text{Has-No-2}^{\text{nd}}\text{Child}$ are unary relation symbols and $1^{\text{st}}\text{Child}$, $2^{\text{nd}}\text{Child}$ are binary relation symbols. We define $\text{Trees}$ to be the set of all $\tau_{\text{Trees}}$-structures $T$ that satisfy the following conditions:

1. $\mathcal{U}^T \subseteq \{1, 2\}^*$ and for every string $s_i \in \mathcal{U}^T$ with $i \in \{1, 2\}$ we also have $s \in \mathcal{U}^T$.
2. $\text{Root}^T$ consists of the empty string $\varepsilon$.
3. $1^{\text{st}}\text{Child}^T$ consists of the pairs $(s, s_1)$, for all $s_1 \in \mathcal{U}^T$.
4. $2^{\text{nd}}\text{Child}^T$ consists of the pairs $(s, s_2)$, for all $s_2 \in \mathcal{U}^T$.
5. $\text{Has-No-1}^{\text{st}}\text{Child}^T$ consists of all strings $s \in \mathcal{U}^T$ with $s_1 \not\in \mathcal{U}^T$.
6. $\text{Has-No-2}^{\text{nd}}\text{Child}^T$ consists of all strings $s \in \mathcal{U}^T$ with $s_2 \not\in \mathcal{U}^T$.

For $T \in \text{Trees}$ and $t \in \mathcal{U}^T$ we write $T_t$ to denote the subtree of $T$ with root $t$.

A schema $\sigma$ is a set of unary relation symbols each of which is distinct from $\text{Has-No-1}^{\text{st}}\text{Child}$, $\text{Has-No-2}^{\text{nd}}\text{Child}$, $\text{Root}$. A $\sigma$-labelled tree is a $(\tau_{\text{Trees}} \cup \sigma)$-structure consisting of some $T \in \text{Trees}$ and additional interpretations $P^T \subseteq \mathcal{U}^T$ for all symbols $P \in \sigma$. We sometimes write $\text{label}(t)$ to denote the set $\{ P \in \sigma : t \in P^T \}$ of labels at vertex $t$ in $T$.

We identify a string $w = w_0 \cdots w_{n-1}$ of length $|w| = n \geq 1$ over an alphabet $\Sigma$ with a $\sigma$-labelled tree $T^w$ in the following way: We choose $\sigma$ to consist of a unary relation symbol $P_a$ for each letter $a \in \Sigma$, we choose $T^w$ to be the (unique) element in $\text{Trees}$ with universe $\mathcal{U}^{T^w} = \{ \varepsilon, 1, 11, \ldots, 1^{n-1} \}$, and we choose $P_a^{T^w} := \{ i : w_i = a \}$, for each $a \in \Sigma$. This corresponds to the conventional representation of strings by structures in the sense that $(\mathcal{U}^{T^w}, 1^{\text{st}}\text{Child}, (P_a^{T^w})_{a \in \Sigma})$ is isomorphic to the structure $(\{0, \ldots, n-1\}, \text{Succ}, (P_a^w)_{a \in \Sigma})$ where $\text{Succ}$ denotes the binary successor relation on $\{0, \ldots, n-1\}$ and $P_a^w$ consists of all positions of $w$ that carry the letter $a$. When reasoning about strings in the context of first-order logic, we sometimes also need the linear ordering $<$ on $\{0, \ldots, n-1\}$ (respectively, the transitive closure of the relation $1^{\text{st}}\text{Child}$). In these cases we explicitly write $\text{FO}(<)$ rather than $\text{FO}$ to indicate that the linear ordering is necessary.

XML-documents are usually modelled as ordered unranked trees and not as binary trees. Here ordered refers to the fact that the order of the children of a vertex is given. However, a standard representation of ordered unranked trees as relational structures uses binary relations $1^{\text{st}}\text{Child}$, $\text{Next-Sibling}$ and unary relations $\text{Root}$, $\text{Leaf}$, $\text{Last-Sibling}$ (for details, see [GK02]) and thus essentially represents ordered unranked trees as binary trees. Therefore, all our results also apply to ordered unranked trees.

### 2.3. Logics and Queries

We assume that the reader is familiar with first-order logic, for short: $\text{FO}$, and with monadic second-order logic, for short: $\text{MSO}$ (cf., e.g., the textbooks [EF99,Imm99]). We use $\text{FO}(\tau)$ and $\text{MSO}(\tau)$, respectively, to denote the class of all first-order formulas and monadic second-order formulas, respectively, of signature
\( \tau \). We write \( \varphi(x_1, \ldots, x_k, X_1, \ldots, X_\ell) \) to indicate that the free first-order variables of the formula \( \varphi \) are \( x_1, \ldots, x_k \) and the free set variables are \( X_1, \ldots, X_\ell \). Sometimes we use \( \overline{x} \) and \( \overline{X} \) as abbreviations for sequences \( x_1, \ldots, x_k \) and \( X_1, \ldots, X_\ell \) of variables.

A formula \( \varphi(x) \) of signature \( \tau \) defines the unary query which associates with every \( \tau \)-structure \( A \) the set of elements \( a \in UA \) such that \( A|_A = \varphi(a) \), i.e., \( A \) satisfies \( \varphi \) when interpreting the free occurrences of the variable \( x \) by the element \( a \). A sentence \( \varphi \) of signature \( \tau \) (i.e., a formula that has no free variables) defines the Boolean query that associates the answer “yes” with all \( \tau \)-structures that satisfy \( \varphi \) and the answer “no” with all other \( \tau \)-structures.

Apart from FO and MSO we will also consider monadic least fixed point logic MLFP which is the extension of first-order logic by unary least fixed point operators. We refer the reader to [EF99] for the definition of MLFP (denoted by FO(M-LFP) there).

2.4. Formula Size and Succinctness. In a natural way, we view formulas as finite trees, where leaves correspond to the atoms of the formulas and inner vertices correspond to Boolean connectives, quantifiers, and fixed-point operators. We define the size \( ||\varphi|| \) of a formula \( \varphi \) to be the number of vertices of the tree that corresponds to \( \varphi \).

Note that this measure of formula size is a uniform cost measure in the sense that it accounts just 1 cost unit for each variable and relation symbol appearing in a formula, no matter what its index is. An alternative is to define the size of a formula as the length of a binary encoding of the formula. Such a logarithmic cost measure is, for example, used in [FG03]. Switching between a uniform and a logarithmic measure usually involves a logarithmic factor.

**Definition 1 (Succinctness).** Let \( L_1 \) and \( L_2 \) be logics, let \( F \) be a class of functions from \( \mathbb{N} \) to \( \mathbb{N} \), and let \( C \) be a class of structures.

We say that \( L_1 \) is \( F \)-succinct in \( L_2 \) on \( C \) if there is a function \( f \in F \) such that for every formula \( \varphi_1 \in L_1 \) there is a formula \( \varphi_2 \in L_2 \) of size \( ||\varphi_2|| \leq f(||\varphi_1||) \) which is equivalent to \( \varphi_1 \) on all structures in \( C \). \( \square \)

Intuitively, a logic \( L_1 \) being \( F \)-succinct in a logic \( L_2 \) means that \( F \) gives an upper bound for the size of \( L_1 \)-formulas needed to express all of \( L_2 \). This definition may seem slightly at odds with the common use of the term “succinctness” in statements such as “\( L_2 \) is exponentially more succinct than \( L_1 \)” meaning that there is some \( L_2 \)-formula that is not equivalent to any \( L_1 \)-formula of subexponential size. In our terminology, we would rephrase this last statement as “\( L_1 \) is not \( 2^{o(n)} \)-succinct in \( L_2 \)” (here we interpret subexponential as \( 2^{o(n)} \), but of course this is not the issue). The reason for defining \( F \)-succinctness the way we did is that it makes the formal statements of our results much more convenient. We will continue to use statements such as “\( L_2 \) is exponentially more succinct than \( L_1 \)” in informal discussions.

**Example 1.** MLFP is \( O(m) \)-succinct in MSO on the class of all finite structures, because every formula \( [\text{LFP}_x, X, \varphi(x, X, \overline{y}, \overline{Y})](z) \) is equivalent to \( \forall X (Xz \lor \exists x \neg Xx \land \varphi(x, X, \overline{y}, \overline{Y})) \). \( \square \)
3. From MSO to MLFP

By the standard translation from MSO-logic to tree automata (cf., e.g., [Tho96]) one knows that every MSO-sentence $\Phi$ can be translated into a nondeterministic tree automaton with $\text{Tower}(O(||\Phi||))$ states that accepts exactly those labelled trees that satisfy $\Phi$. This leads to

**Theorem 1 (Folklore).** MSO-sentences are $\text{Tower}(O(m))$-succinct in MLFP on the class of all labelled trees. □

To show that we cannot do essentially better, i.e., that there is no translation from MSO to MLFP of size $\text{Tower}(o(m))$ we need a complexity theoretic assumption that, however, does not seem to be too far-fetched. Let SAT denote the NP-complete satisfiability problem for propositional formulas in conjunctive normal form. Until now, all known deterministic algorithms that solve SAT have worst-case complexity $2^{\Omega(n)}$ (cf., [DGH+02]). Although not answering the P vs. NP question, the exposition of a deterministic algorithm for SAT with worst-case complexity $\leq n^{\lg n}$ would be a surprising and unexpected breakthrough in the SAT-solving community.

In the following, we write $\lg(i)$ to denote the $i$ times iterated logarithm, inductively defined by $\lg(1)(n) := \lg(n)$ and $\lg(i+1)(n) := \lg(\lg(i)(n))$. Moreover, we write $\lg^*$ to denote the “inverse” of the Tower function, that is, the (unique) integer valued function with $\text{Tower}(\lg^*(n)−1) < n \leq \text{Tower}(\lg^*(n))$.

**Theorem 2.** Unless SAT is solvable by a deterministic algorithm that has, for every $i \in \mathbb{N}$, time bound $||\gamma||\lg(i)(n)$ (where $\gamma$ is the input formula and $n$ the number of propositional variables occurring in $\gamma$), MSO is not $\text{Tower}(o(m))$-succinct in MLFP on the class of all finite strings. □

The overall proof idea is to assume that the function $f$ specifies the size of the translation from MSO to MLFP and to exhibit a SAT-solving algorithm which

- constructs a string $w$ that represents the SAT-instance $\gamma$,
- constructs an MSO-formula $\Phi(z)$ of extremely small size that, when evaluated in $w$, specifies a canonical satisfying assignment for $\gamma$ (if $\gamma$ is satisfiable at all),
- tests, for all MLFP-formulas $\Psi(z)$ of size $\leq f(||\Phi||)$, whether $\Psi$ specifies a satisfying assignment for $\gamma$.

Before presenting the proof in detail we provide the necessary notations and lemmas:

It is straightforward to see

**Lemma 1.** There is an algorithm that, given an MLFP-formula $\Psi(z)$, a string $w$, and a position $p$ in $w$, decides in time $|w|^{O(||\Psi||)}$ whether $w \models \Psi(p)$. □

Let us now concentrate on the construction of a string $w$ that represents a SAT-instance $\gamma$ and of an MSO-formula $\Phi(z)$ that specifies a canonical satisfying assignment of $\gamma$ (provided that $\gamma$ is satisfiable at all). Since we want $\Phi$ to be extremely short, we cannot choose $w$ to be the straightforward string-representation of $\gamma$. Instead, we use the following, more complicated, representation of [FG03]:

For all $h \geq 1$ let $\Sigma_h := \{0, 1, <1>, </1>, \ldots, <h>, </h>\}$. The “tags” $<i>$ and $</i>$ represent single letters of the alphabet and are just chosen to improve readability.
For every \( n \geq 1 \) let \( L(n) \) be the length of the binary representation of the number \( n-1 \), i.e., \( L(0) = 0 \), \( L(1) = 1 \), and \( L(n) = \lfloor \log(n-1) \rfloor + 1 \), for all \( n \geq 2 \). By \( \text{bit}(i,n) \) we denote the \( i \)-th bit of the binary representation of \( n \), i.e., \( \text{bit}(i,n) = 1 \) if \( \left\lfloor \frac{n}{2^i} \right\rfloor \) is odd, and \( \text{bit}(i,n) = 0 \) otherwise.

We encode every number \( n \in \mathbb{N} \) by a string \( \mu_h(n) \) over the alphabet \( \Sigma_h \), where \( \mu_h(n) \) is inductively defined as follows: \( \mu_h(0) := \langle \text{cnf} \rangle \), and

\[
\mu_h(n) := \langle h \rangle \mu_h(0) \text{bit}(0,n-1) \cdots \text{bit}(L(n)-1,n-1) \langle/\rangle,
\]

for \( n \geq 1 \). For \( h \geq 2 \) we let \( \mu_h(0) := \langle h \rangle \langle/\rangle \) and

\[
\mu_h(n) := \langle h \rangle \mu_h(0) \text{bit}(0,n-1) \cdots \langle/\rangle \mu_h(L(n)-1) \text{bit}(L(n)-1,n-1) \langle/\rangle,
\]

for \( n \geq 1 \). Here empty spaces and line breaks are just used to improve readability.

To encode a CNF-formula \( \gamma \) by a string we use an alphabet \( \Sigma'_h \) that extends \( \Sigma_h \) by the symbols \( +, -, *, \) and a number of additional tags. Let \( i \in \mathbb{N} \) and let \( X_i \) be a propositional variable. The literal \( X_i \) is encoded by the string

\[
\mu_h(X_i) := \langle \text{lit} \rangle \mu_h(i) \langle/\rangle,
\]

and the literal \( \neg X_i \) is encoded by \( \mu_h(\neg X_i) := \langle \text{lit} \rangle \mu_h(i) \langle/\rangle \).

A clause \( \delta := \lambda_1 \lor \cdots \lor \lambda_r \) of literals is encoded by

\[
\mu_h(\delta) := \langle \text{clause} \rangle \mu_h(\lambda_1) \cdots \mu_h(\lambda_r) \langle/\rangle.
\]

A CNF-formula \( \gamma := \delta_1 \land \cdots \land \delta_m \) is encoded by the string

\[
\mu_h(\gamma) := \langle \text{cnf} \rangle \mu_h(\delta_1) \cdots \mu_h(\delta_m) \langle/\rangle.
\]

We write \( \text{CNF}(n) \) to denote the class of all CNF-formulas the propositional variables of which are among \( X_0, \ldots, X_{n-1} \). To provide the “infrastructure” for specifying a truth assignment, we use the string

\[
\mu_h(X_0, \ldots, X_{n-1}) := \langle \text{ass} \rangle \langle val \rangle \mu_h(0) \ast \langle/\rangle \cdots \mu_h(n) \langle val \rangle \langle/\rangle \langle ass \rangle.
\]

Remark 1. There is a 1–1-correspondence between assignments \( \alpha : \{X_0, \ldots, X_{n-1}\} \rightarrow \{\text{true, false}\} \), on the one hand, and sets \( P \) of positions of \( \mu_h(X_0, \ldots, X_{n-1}) \) that carry the letter \( \ast \), on the other hand: Such a set \( P \) specifies the assignment \( \alpha^P \) that, for each \( i < n \), maps the variable \( X_i \) to the value \( \text{true} \) if the \( \ast \)-position directly after the substring \( \mu_h(i) \) in \( \mu_h(X_0, \ldots, X_{n-1}) \) belongs to \( P \). Conversely, a given assignment \( \alpha \) specifies the set \( P^\alpha \) consisting of exactly those \( \ast \)-positions of \( \mu_h(X_0, \ldots, X_{n-1}) \) that occur directly after a substring \( \mu_h(i) \) where \( \alpha(X_i) = \text{true} \). \( \square \)

Finally, we encode a formula \( \gamma \in \text{CNF}(n) \) by the string

\[
\mu_h(\gamma, \ast) := \mu_h(\gamma) \mu_h(X_0, \ldots, X_{n-1}).
\]

\( \mu_h(\gamma, \ast) \) is the string \( w \) that we will furtheron use as the representative of a SAT-instance \( \gamma \). We use the following result of [FG03]:
Lemma 2.

(a) There is an algorithm that, given $h \in \mathbb{N}$ and $\gamma \in \text{CNF}(n)$, computes (a binary representation of) the string $\mu_h(\gamma, *)$ in time $O(h \cdot (\lg h) \cdot (\lg n)^2 \cdot (||\gamma|| + n))$ (cf., [FG03, Lemma 9]).

The string $\mu_h(\gamma, *)$ has length $|\mu_h(\gamma, *)| = O(h \cdot (\lg n)^2 \cdot (||\gamma|| + n))$.

(b) There is an algorithm that, given $h \in \mathbb{N}$, computes (the binary representation of) a FO($\langle \rangle$)-formula $\varphi_h(Z)$ in time $O(h \cdot \lg h)$, such that for all $n \leq \text{Tower}(h)$, for all $\gamma \in \text{CNF}(n)$, and for all sets $P$ of $*$-positions in the string $\mu_h(\gamma, *)$ we have

$$\mu_h(\gamma, *) \models \varphi_h(P) \iff \alpha^P \text{ is a satisfying assignment for } \gamma$$

(cf., [FG03, Lemma 10]). The formula $\varphi_h(Z)$ has size\(^1\) $||\varphi_h(Z)|| = O(h)$.

Given a CNF($n$)-formula $\gamma$ and its representative $\mu_h(\gamma, *)$, we now specify a canonical satisfying assignment of $\gamma$, provided that $\gamma$ is satisfiable at all. As observed in Remark 1, every assignment $\alpha : \{X_0, \ldots, X_{n-1}\} \rightarrow \{\text{true, false}\}$ corresponds to a set $P^\alpha$ of positions in $\mu_h(\gamma, *)$ that carry the letter $\star$. $P^\alpha$, again, can be identified with the 0-1-string of length $|\mu_h(\gamma, *)|$ that carries the letter 1 exactly at those positions that belong to $P^\alpha$. Now, the lexicographic ordering of these strings gives us a linear ordering on the set of all assignments $\alpha : \{X_0, \ldots, X_{n-1}\} \rightarrow \{\text{true, false}\}$. As the canonical satisfying assignment of $\gamma$ we choose the lexicographically smallest satisfying assignment.

Lemma 3. There is an algorithm that, given $h \in \mathbb{N}$, computes (the binary representation of) an MSO-formula $\Phi_h(z)$ in time $O(h \cdot \lg h)$, such that for all $n \leq \text{Tower}(h)$, for all $\gamma \in \text{CNF}(n)$, and for all positions $p$ of $\mu_h(\gamma, *)$ that carry the letter $\star$, we have

$$\mu_h(\gamma, *) \models \Phi_h(p) \iff \text{the propositional variable corresponding to position } p \text{ is assigned the value true.}$$

The formula $\Phi_h(z)$ has size $||\Phi_h|| = O(h)$.

Finally, we are ready for the Proof of Theorem 2:

Proof of Theorem 2.

Let $f : \mathbb{N} \rightarrow \mathbb{N}$ be a function such that there is, for every MSO-formula $\Phi(z)$, a MLFP-formula $\Psi(z)$ of size $||\Psi|| \leq f(||\Phi||)$ which defines the same query as $\Phi$ on the class of all finite strings (recall that such an $f$ does indeed exist, because MSO and MLFP have the same expressive power over the class of finite strings).

Consider the algorithm displayed in Figure 1, which decides if the input formula $\gamma$ is satisfiable.

The correctness of this algorithm directly follows from Lemma 3 and from the fact that at least one of the formulas $\Psi(z)$ of size $\leq f(||\Phi||)$ defines the same query as $\Phi_h(z)$.

It remains to determine the worst-case running time of the algorithm. Let $\gamma$ be an input CNF-formula for the algorithm, let $n$ be the number of propositional variables of $\gamma$, and let $h := \lg^* n$.

\(^1\) In [FG03], an additional factor $\lg h$ occurs because there a logarithmic cost measure is used for the formula size, whereas here we use a uniform measure (cf., Section 2.4).
Input: a SAT-instance $\gamma$ in CNF

1. Count the number $n$ of propositional variables occurring in $\gamma$, and modify $\gamma$ in such a way that only the propositional variables $X_0, \ldots, X_{n-1}$ occur in it.
2. Compute $h := \lg^* (n)$, i.e., choose $h \in \mathbb{N}$ such that $\text{Tower}(h-1) < n \leq \text{Tower}(h)$.
3. Construct the string $\mu_h(\gamma, \ast)$ that represents $\gamma$ (see Lemma 2(a)).
4. Construct an MSO-formula $\Phi_h(z)$ that has the following property:

   Whenever $p$ is a position in $\mu_h(\gamma, \ast)$ that carries the letter $\ast$, we have

   \[ \mu_h(\gamma, \ast) \models \Phi_h(p) \quad \text{iff} \quad \text{the lexicographically smallest satisfying assignment for } \gamma, \text{the propositional variable corresponding to position } p \text{ is assigned the value true} \]

   (cf., Lemma 3).
5. For all MLFP-formulas $\Psi(z)$ of size $|\Psi| \leq f(|\Phi_h|)$ do:
   (a) Initialise the assignment $\alpha := \emptyset$.
   (b) For all positions $p$ in $\mu_h(\gamma, \ast)$ that carry the letter $\ast$ do
       check whether $\mu_h(\gamma, \ast) \models \Psi(p)$;
       if so, then insert the propositional variable corresponding to $p$ into $\alpha$.
   (c) Check whether $\alpha$ is a satisfying assignment for $\gamma$;
       if so, then STOP with output “$\gamma$ is satisfiable via assignment $\alpha$”.
6. STOP with output “$\gamma$ is not satisfiable”.

**Fig. 1.** A SAT-solving algorithm.

The steps 1–4 of the algorithm will be performed within a number of steps polynomial in $|\gamma|$, and the MSO-formula $\Phi_h(z)$ produced in step 4 will have size $|\Phi_h| \leq c \cdot h$, for a suitable constant $c \in \mathbb{N}$ (cf., Lemma 2(a) and Lemma 3).

The loop in step 5 will be performed for $2^{c_1 \cdot f(|\Phi_h|) - \lg f(|\Phi_h|)}$ times, for a suitable constant $c_1 \in \mathbb{N}$. To see this, note that formulas of length $\leq f(|\Phi_h|)$ use at most $f(|\Phi_h|)$ different first-order variables and at most $f(|\Phi_h|)$ different set variables.

I.e., these formulas can be viewed as strings of length $f(|\Phi_h|)$ over an alphabet of size $c_2 + 2 \cdot f(|\Phi_h|)$, for a suitable constant $c_2 \in \mathbb{N}$. Therefore, the number of such formulas is $\leq (c_2 + 2 \cdot f(|\Phi_h|))^{f(|\Phi_h|)} \leq 2^{c_2 \cdot f(|\Phi_h|) - \lg f(|\Phi_h|)}$.

Each performance of the loop in step 5 will take a number of steps polynomial in

\[ |\mu_h(\gamma, \ast)|^{O(f(|\Phi_h|))} \leq (c_3 \cdot h \cdot (\lg n)^2 \cdot |\gamma|)^{c_4 \cdot f(c \cdot h)}, \]

for suitable constants $c_3, c_4 \in \mathbb{N}$ (cf., Lemma 1 and Lemma 2(a)). Altogether, for suitable constants $c, d \in \mathbb{N}$, the algorithm will perform the steps 1–6 within

\[ |\gamma|^{d \cdot f(c \cdot h) - \lg f(c \cdot h)} \]

steps.

Now let us suppose that $f$ has bound $f(m) \leq \text{Tower}(o(m))$. From Lemma 4 below we then obtain that our SAT-solving algorithm has, for every $i \in \mathbb{N}$, time bound $|\gamma|^{\lg^{(i)}(m)}$. This finally completes the proof of Theorem 2. \hfill \blacksquare
We claim that every nondeterministic finite automaton accepting precisely those strings over alphabet \( \{c, d\} \), satisfies the following

\[ \text{Lemma 4. Let } f : \mathbb{N} \rightarrow \mathbb{N} \text{ be a function with bound } f(m) \leq \text{Tower}(o(m)), \text{ and let } c, d \in \mathbb{N}. \text{ For every } i \in \mathbb{N} \text{ there is an } n_0 \in \mathbb{N} \text{ such that for all } n \geq n_0 \text{ we have} \]

\[ d \cdot f(c \cdot \log^*(n)) \cdot \log (f(c \cdot \log^*(n))) \leq \log^{(i)}(n). \]

\[ \square \]

4. The Two-Variable Fragment of MLFP and the Full Modal \( \mu \)-Calculus

Defining the 2-variable fixed-point logics requires some care: MLFP\(^2\) is the fragment of MLFP consisting of all formulas with just 2 individual variables and no parameters in fixed point operators, i.e., for all subformulas of the form \([\text{LFP}x, X \varphi](y)\), \( x \) is the only free first-order variable of \( \varphi \). This is the monadic fragment of the standard 2-variable least fixed-point logic (cf. \cite{GO99}). Without the restriction on free variables in fixed-point operators, we obtain full MLFP even with just two individual variables (we prove this in the full version of this paper \cite{GS03}).

We first note that MLFP\(^2\), and actually \( \text{FO}^2 \), the two variable fragment of first-order logic, is doubly exponentially more succinct than nondeterministic automata on the class of all finite strings:

\[ \text{Example 2. Let } \sigma := \{L, R, P_1, \ldots, P_n\} \text{ and} \]

\[ \varphi_n := \forall x \Big( Lx \rightarrow \exists y (Ry \bigwedge \bigwedge_{i=1}^{n} (P_ix \leftrightarrow P_iy)) \Big). \]

We claim that every nondeterministic finite automaton accepting precisely those strings over alphabet \( 2^\sigma \) that satisfy \( \varphi \) has at least \( 2^{2^n} \) states. To see this, for every \( S \subseteq 2^{\{1, \ldots, n\}} \), we define strings \( X_n(S) \) and \( Y_n(S) \) such that

- \( L^{X_n(S)} = U^{X_n(S)} \) and \( R^{Y_n(S)} = U^{Y_n(S)} \)
- For all \( s \in S \) there exists an \( x \in U^{X_n(S)} \) and an \( y \in U^{Y_n(S)} \) such that

\[ s = \{ i \mid x \in P_i^{X_n(S)} \} = \{ i \mid y \in P_i^{Y_n(S)} \}. \]

Let \( W_n(S, T) := X_n(S) Y_n(T) \) be the concatenation of \( X_n(S) \) and \( Y_n(T) \). Then \( W_n(S, T) \models \varphi \iff S \subseteq T \). Clearly, a nondeterministic finite automaton accepting precisely those strings \( W_n(S, T) \) with \( S \subseteq T \) needs at least \( 2^{2^n} \) states. \[ \square \]

Let us return to binary trees now. Following Vardi \cite{Var98}, we define the full modal \( \mu \)-calculus \( \text{FL}_\mu \) on binary trees as follows: For each schema \( \sigma \), an \( \text{FL}_\mu \)-formula of schema \( \sigma \) is either:

- \text{true}, \text{false}, \text{P}, or \( \neg \text{P} \), where \( P \in \sigma \cup \{ \text{Root}, \text{Has-No-1st Child}, \text{Has-No-2nd Child} \} \);
- \( \Phi_1 \land \Phi_2 \) or \( \Phi_1 \lor \Phi_2 \), where \( \Phi_1 \) and \( \Phi_2 \) are \( \text{FL}_\mu \)-formulas of schema \( \sigma \);
- \( X \), where \( X \) is a propositional variable;
- \( \langle R \rangle \Phi \) or \( \langle R \rangle \Phi \), where \( R \in \{ \text{1st Child}, \text{2nd Child}, \text{1st Child}^{-1}, \text{2nd Child}^{-1} \} \) and \( \Phi \) is an \( \text{FL}_\mu \)-formula of schema \( \sigma \);
- \( \mu X. \Phi \) or \( \nu X. \Phi \), where \( X \) is a propositional variable and \( \Phi \) is an \( \text{FL}_\mu \)-formula of schema \( \sigma \).

The semantics of \( \text{FL}_\mu \) is defined in the usual way interpreting the binary relations over trees. The following is straightforward:
Proposition 1. $\text{FL}_\mu$ is $O(m)$-succinct in $\text{MLFP}^2$.

Our next result is that there also is a reverse translation from $\text{MLFP}^2$ to $\text{FL}_\mu$ which only incurs an exponential blow-up in size:

Theorem 3. $\text{MLFP}^2$ is $2^{\text{poly}(m)}$-succinct in $\text{FL}_\mu$ on the class of labelled trees. □

Theorem 4 (Vardi [Var98]). For every formula $\Phi$ of the full modal $\mu$-calculus $\text{FL}_\mu$ there is a nondeterministic tree automaton of size $2^{\text{poly}(|\Phi|)}$ that accepts exactly those labelled trees in which $\Phi$ holds at the root. □

As a matter of fact, Vardi [Var98] proved a stronger version of this theorem for infinite trees and parity tree automata. But on finite trees, a parity acceptance condition can always be replaced by a normal acceptance for finite tree automata.

The Theorems 3 and 4 directly imply

Corollary 1. For every $\text{MLFP}^2$-formula $\varphi(x)$ there is a nondeterministic tree automaton of size $2^{2^{\text{poly}(|\varphi|)}}$ that accepts exactly those labelled trees in which $\varphi$ holds at the root. □

5. Monadic Datalog and Stratified Monadic Datalog

We assume that the reader is familiar with $\text{datalog}$, which may be viewed as logic programming without function symbols (cf., e.g., the textbook [AHV95]). A datalog program is $\text{monadic}$ if all its IDB-predicates (i.e., its intensional predicates that appear in the head of some rule of the program) are unary. In this paper we restrict attention to monadic datalog programs that are interpreted over labelled trees. A monadic datalog program of schema $\sigma$ may use as EDB-predicates (i.e., extensional predicates which are determined by the structure the program is interpreted over) the predicates in $\tau_{\text{Trees}}$, the predicates in $\sigma$, and a predicate $\neg P$ for every $P \in \sigma$ which is interpreted as the complement of $P$. We use $\text{IDB}(\mathcal{P})$ to denote the set of IDB-predicates of $\mathcal{P}$, and we write MonDatalog to denote the class of all monadic datalog programs.

More formally, a monadic datalog program $\mathcal{P}$ of schema $\sigma$ is a finite set of rules of the form $X(x) \leftarrow \gamma(x, \overline{y})$, where $\gamma$ is a conjunction of atomic formulas over the signature $\tau_{\text{Trees}} \cup \sigma \cup \{\neg P : P \in \sigma\} \cup \text{IDB}(\mathcal{P})$. Every program has a distinguished goal IDB-predicate that determines the query defined by the program.

We define the size $||\mathcal{P}||$ of $\mathcal{P}$ in the same way as we defined the size of formulas.

In [GK02] it was shown that MonDatalog can define the same unary queries on the class of labelled trees as monadic second-order logic. In the remainder of this section we will compare the succinctness of MonDatalog, $S$-$\text{MonDatalog}$, $\text{FL}_\mu$, $\text{MLFP}$, and a particular kind of tree automaton.

5.1. From MonDatalog to Finite Automata. Several mechanisms have been proposed in the literature for specifying unary queries by finite automata operating on labelled trees (cf., [NS02]). One such mechanism, introduced in [Nev99] and further investigated in [FGK03,Koc03], is the selecting tree automaton:

Definition 2 (STA). Let $\sigma$ be a schema. A selecting $\sigma$-tree automaton ($\sigma$-$\text{STA}$, for short) is a tuple $\mathcal{A} = (Q, 2^\sigma, F, \delta, S)$, where $S \subseteq Q$ is the set of selecting states and
Theorem 5 ([FGK03,GK02]). The class of labelled trees that are definable in monadic-second order logic. It was shown in [FGK03,Nev99] that STAs can define exactly those unary queries on runs \(\rho\). Theorem 6. Using the above proposition, it is not difficult to prove.

\[\Box\text{labelled trees.}\]

2 of size \(O\) size \(\delta:2\) \(\sigma\) the class of all \(P\) S-MonDatalog-program remains open if the exponential size is indeed necessary or if, on the contrary, for every programs can be translated into MonDatalog-programs of at most exponential size. It is not hard to show that this result is asymptotically optimal, that is, that MonDatalog □ trees.

From S-MonDatalog to MonDatalog. In this section we show that S-MonDatalog-programs can be translated into MonDatalog-programs of at most exponential size. It remains open if the exponential size is indeed necessary or if, on the contrary, for every S-MonDatalog-program \(P\) there exists an equivalent MonDatalog-program \(P'\) of size polynomial in \(||P||\).

Lemma 5. For every \(\sigma\)-STA \(A = (Q, 2^\sigma, F, \delta, S)\) there is a MonDatalog-program \(P\) of size \(O(|Q|^3 \cdot |2^\sigma| + |\sigma| \cdot |2^\sigma|)\) that defines the complement of the query defined by \(A\) on the class of all \(\sigma\)-labelled trees.

Using Theorem 5 and Lemma 5 one easily obtains

Proposition 2. For every MonDatalog-program \(P\) there is a MonDatalog-program \(P'\) of size \(2^{O(||P||)}\) that defines the complement of the query defined by \(P\) on the class of labelled trees.

Using the above proposition, it is not difficult to prove

Theorem 6. S-MonDatalog is \(2^{O(m)}\)-succinct in MonDatalog on the class of labelled trees.

5.3. S-MonDatalog vs FL\(_\mu\). From Theorem 4 and Lemma 5 one directly obtains

Theorem 7. FL\(_\mu\) is \(\mathcal{2}^{\text{poly}(m)}\)-succinct in S-MonDatalog on the class of labelled trees.

Conversely, it is not hard to show the following
Theorem 8. S-MonDatalog is $2^{O(m \cdot \log m)}$-succinct in $\text{FL}_\mu$ on the class of labelled trees.

It remains open whether the above bounds are optimal.

5.4. From MLFP to S-MonDatalog. Similarly to Theorem 1 one easily obtains

Theorem 9 (Folklore). MLFP-sentences are $\text{Tower}(O(m))$-succinct in S-MonDatalog on the class of labelled trees.

The aim of this section is to show that there are no essentially smaller translations from MLFP to S-MonDatalog. We will use the following well-known observation:

Proposition 3 (Folklore). There is no function $f : \mathbb{N} \rightarrow \mathbb{N}$ with bound $f(m) \leq \text{Tower}(o(m))$ such that for every FO($<$)-sentence $\varphi$ there is a nondeterministic finite automaton $\mathfrak{A}$ with at most $f(||\varphi||)$ states that accepts exactly those strings that satisfy $\varphi$.

Using Proposition 3 and the results of the Sections 5.1 and 5.2, one obtains the following:

Theorem 10. There is no function $f : \mathbb{N} \rightarrow \mathbb{N}$ with bound $f(m) \leq \text{Tower}(o(m))$ such that for every FO($<$)-sentence $\varphi$ there is a S-MonDatalog-program $\mathcal{P}$ of size $||\mathcal{P}|| \leq f(||\varphi||)$ and a designated goal predicate $X \in \text{IDB}(\mathcal{P})$ such that $(\mathcal{P}, X)$ defines the same Boolean query as $\varphi$ on the class of all finite strings.

Since FO($<$) is included in MLFP, the above theorem directly implies the following:

Corollary 2. MLFP is not $\text{Tower}(o(m))$-succinct in S-MonDatalog on the class of all finite strings.

It remains open if this result remains valid when replacing MLFP with MLFP$^2$. Note, however, that for the proof of Proposition 3 a small number $k$ of first-order variables suffices. I.e., Proposition 3 remains valid when replacing FO($<$) with FO$^k($$<$), and Corollary 2 remains valid when replacing MLFP with MLFP$^k$.

Together with Corollary 1 and Lemma 5, the above Corollary 2 implies

Corollary 3. MLFP is not $\text{Tower}(o(m))$-succinct in MLFP$^2$ on the class of all finite strings.

6. Conclusion

We studied the succinctness of a number of fixed point logics on trees. We believe that the analysis of succinctness, which may be viewed as a refined, “quantitative” analysis of expressive power, is a very interesting topic that deserves much more attention.

Even though we were able to get a good overall picture of the succinctness of monadic fixed point logics on trees, a number of questions remain open. Let us just mention a few of them:

- The exact relationship between monadic datalog, stratified monadic datalog, and the full modal $\mu$-calculus remains unclear. In particular: Is the class of all queries whose complements can be defined by monadic datalog programs polynomially succinct in monadic datalog, or is there an exponential lower bound? (Recall that in Proposition 2 we prove an exponential upper bound.)
Our proof that MSO is not \( \text{Tower}(o(m)) \)-succinct in MLFP relies on a complexity theoretic assumption. Is it possible to prove this result without such an assumption?

We have only considered the 2-variable fragment of MLFP here. What about the \( k \)-variable fragments, for \( k \geq 3 \)? Do they form a strict hierarchy with respect to succinctness?

References


Author Index

Abdulla, Parosh Aziz 1
Aichernig, Bernhard K. 15
Andreoli, Jean-Marc 29
Artemov, Sergei 43

Bodirsky, Manuel 44
Börner, Ferdinand 58
Bouajjani, Ahmed 1, 71
Bradfield, Julian C. 72
Brünnler, Kai 86
Buchberger, Bruno 98
Bulatov, Andrei 58

Chatterjee, Krishnendu 100
Chen, Yijia 114
Chevalier, Yannick 128

Dantchev, Stefan 142
David, René 155
Dawar, Anuj 169
d’Orso, Julien 1
Duparc, Jacques 183

Eiter, Thomas 197
Flum, Jörg 114

Gabbay, Dov 413
Gabbay, Murdoch 513
Gerhardy, Philipp 212
Grohe, Martin 226
Guillaume, Bruno 155

Henzinger, Thomas A. 100
Hitchcock, John M. 241

Iemhoff, Rosalie 255
Igarashi, Atsushi 484

Jeavons, Peter 58
Jurdziński, Marcin 100

Kähler, Detlef 271
Kameyama, Yukiyoshi 484
Kanovich, Max 285

Khoussainov, Bakhadyr 299
Kolaitis, Phokion G. 314
Korovina, Margarita 330
Kristiansen, Lars 345
Krokhin, Andrei 58
Krzeszczakowski, Łukasz 359
Küsters, Ralf 128

Laird, James 372
Leiß, Hans 384
Lutz, Jack H. 241

Makino, Kazuhisa 197
Maksimova, Larisa 397
Mazaré, Laurent 29
Metcalfe, George 413
Meyden, Ron van der 427
Miquel, Alexandre 441
Moser, Georg 455

Nešetřil, Jaroslav 44
Nowak, David 575

Olivetti, Nicola 413

Panttaja, Jonathan 314
Patra, Manas 427
Pitts, Andrew 513

Reus, Bernhard 456
Richerby, David 169
Riis, Søren 142
Rougemont, Michel de 384
Rusinowitch, Michaël 128
Rybina, Tatiana 546

Sakurai, Takafumi 484
Samer, Marko 470
Sato, Masahiko 484
Schröder, Lutz 498
Schweikardt, Nicole 226

Terwijn, Sebastiaan A. 241
Turuani, Mathieu 128
Urban, Christian 513
Vauzeilles, Jacqueline 285
Veith, Helmut 470, 528
Vermeulen, Kess F.M. 530
Vigneron, Laurent 128
Voda, Paul J. 345
Vorobjov, Nicolai 544, 545
Voronkov, Andrei 397, 546
Walukiewicz, Igor 574
Wilke, Thomas 271
Zach, Richard 455
Zhang, Yu 575