Lecture Notes in Computer Science 2471

Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Julian Bradfield (Ed.)

Computer Science Logic

16th International Workshop, CSL 2002
11th Annual Conference of the EACSL
Edinburgh, Scotland, UK, September 22-25, 2002
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editor

Julian Bradfield
Laboratory for Foundations of Computer Science
Division of Informatics, University of Edinburgh
King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
E-mail: csl02@dcs.ed.ac.uk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Computer science logic : 16th international workshop ; proceedings / CSL 2002, Edinburgh, Scotland, UK, September 22 - 25, 2002. Julian Bradfield (ed.). - Berlin ; Heidelberg ; New York ; Hong Kong ; London ; Milan ; Paris ; Tokyo : Springer, 2002
(Annual Conference of the EACSL ... ; 11)
(Lecture notes in computer science ; Vol. 2471)
ISBN 3-540-44240-5

CR Subject Classification (1998): F.4.1, F.4, I.2.3-4, F.3

ISSN 0302-9743
ISBN 3-540-44240-5 Springer-Verlag Berlin Heidelberg New York

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

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2002
Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN: 10871322 06/3142 5 4 3 2 1 0

Preface

The Annual Conference of the European Association for Computer Science Logic, CSL 2002, was held in the Old College of the University of Edinburgh on 22–25 September 2002. The conference series started as a programme of International Workshops on Computer Science Logic, and then in its sixth meeting became the Annual Conference of the EACSL. This conference was the sixteenth meeting and eleventh EACSL conference; it was organized by the Laboratory for Foundations of Computer Science at the University of Edinburgh.

The CSL 2002 Programme Committee considered 111 submissions from 28 countries during a two week electronic discussion; each paper was refereed by at least three reviewers. The Committee selected 37 papers for presentation at the conference and publication in these proceedings.

The Programme Committee invited lectures from Susumu Hayashi, Frank Neven, and Damian Niwiński; the papers provided by the invited speakers appear at the front of this volume.

In addition to the main conference, two tutorials – 'Introduction to Mu-Calculi' (Julian Bradfield) and 'Parametrized Complexity' (Martin Grohe) – were given on the previous day.

I thank the Programme Committee and all the referees for their work in reviewing the papers; the other members of the local organizing team (Dyane Goodchild, Monika Lekuse, and Alex Simpson), as well as the many other LFCS colleagues who helped in various ways, for arranging the event itself; the organizers of CSL 2001, in particular François Laroussinie, for allowing me to inherit the fruits of their labours; and Richard van de Stadt, whose CyberChair system greatly facilitated the handling of submissions and reviews.

Finally, I acknowledge with gratitude the generous support of the U.K.'s Engineering and Physical Sciences Research Council, which sponsored the invited lecturers as well as providing support for students; and the Laboratory for Foundations of Computer Science, which provided both financial support and much time from its staff.

July 2002 Julian Bradfield

Programme Committee

Thorsten Altenkirch *U. Nottingham*
Rajeev Alur *U. Pennsylvania*
Michael Benedikt *Bell Labs*
Julian Bradfield *U. Edinburgh (Chair)*
Anuj Dawar *U. Cambridge*
Yoram Hirshfeld *U. Tel Aviv*
Ulrich Kohlenbach *U. Aarhus*
Johann Makowsky *Technion Haifa*
Dale Miller *Pennsylvania State U.*
Luke Ong *U. Oxford*
Frank Pfenning *Carnegie Mellon U.*
Philippe Schnoebelen *ENS Cachan*
Luc Segoufin *INRIA Rocquencourt*
Alex Simpson *U. Edinburgh*
Thomas Streicher *T.U. Darmstadt*

Referees

Andreas Abel
Natasha Alechina
Jean-Marc Andreoli
Albert Atserias
Jeremy Avigad
Arnon Avron
Matthias Baaz
Roland Backhouse
Christel Baier
Patrick Baillot
Paolo Baldan
Andrej Bauer
Stefano Berardi
Alessandro Berarducci
Josh Berdine
Marc Bezem
Andreas Blass
Gerhard Brewka
Chad E. Brown
Glenn Bruns
Wilfried Buchholz
Martin Bunder
Béatrice Bérard
Cristiano Calcagno
Iliano Cervesato
Kaustuv Chaudhuri
Yifeng Chen
Corina Cîrstea
Hubert Comon
Olivier Danvy
René David
Stéphane Demri
Nachum Dershowitz
Gilles Dowek
Derek Dreyer
Joshua Dunfield
Steve Dunne
Roy Dyckhoff
Martin Erwig
Kousha Etessami
Wenfei Fan
Andrzej Filinski
Jean-Christophe Filliâtre
Arnaud Fleury
Marcus Frick
Carsten Führmann
Bernhard Ganter
Harald Ganzinger
Philipp Gerhardy
Neil Ghani
Alwyn Goodloe
Jean Goubault-Larrecq
William Greenland
Martin Grohe
David Gross-Amblard
Radu Grosu
Stefano Guerrini
Jörgen Gustavsson
Hugo Herbelin
Claudio Hermida
Ian Hodkinson
Wiebe van der Hoek
Martin Hofmann
Joe Hurd
Graham Hutton
Martin Hyland
Radha Jagadeesan
David Janin
Alan Jeffrey
Dick de Jongh
Marcin Jurdziński
Stephan Kahrs
Michael Kaminski
Mathias Kegelmann
Andrew Ker
Sanjeev Khanna
Thomas Kleymann
Beata Konikowska
Jan Krajíček
Andrei Krokhin
Werner Kuich
Oliver Kullmann
Alexander Kurz
Yves Lafont
François Lamarche
Clemens Lautemann
Salvatore La Torre
Daniel Leivant
Paul Blain Levy
Leonid Libkin
John Longley
Tobias Löw
Gavin Lowe
Ian Mackie
Omid Madani
P Madhusudan

Monika Maidl
Nicolas Markey
Ralph Matthes
Conor McBride
Paul-André Melliès
Michael Mendler
Jochen Messner
Eugenio Moggi
Andrzej Murawski
Tom Murphy
Mogens Nielsen
Hans de Nivelle
David Nowak
Peter O'Hearn
Paulo Oliva
Jaap van Oosten
Martin Otto
Catuscia Palamidessi
Prakash Panangaden
Michel Parigot
Brigitte Pientka
Benjamin Pierce
Randy Pollack
Myriam Quatrini
Alex Rabinovich
Uday Reddy
Jason Reed
Laurent Regnier
Horst Reichel
Bernhard Reus
Søren Riis
Eike Ritter
Luca Roversi
Jan Rutten
Vladimiro Sassone
Alexis Saurin
Andrea Schalk
Thomas Schwentick
Helmut Seidl
Peter Selinger
Andrei Serjantov
Sanjit Seshia
Anatol Slissenko
Rick Sommer
Bas Spitters
Robert Stärk
Perdita Stevens
Charles Stewart
Colin Stirling
Gerd Stumme
Aaron Stump
Jean-Marc Talbot
Kazushige Terui
Alwen Tiu
Christian Urban
Tarmo Uustalu
Margus Veanes
Fer-Jan de Vries
Jens Vöge
Uwe Waldmann
David Walker
Kevin Watkins
Andreas Weiermann
Benjamin Werner
Glynn Winskel
Joakim von Wright
Zhe Yang
Richard Zach
Michael Zakharyaschev
Uri Zwick

Local Organizing Committee

Julian Bradfield
Dyane Goodchild
Monika Lekuse
Alex Simpson

Table of Contents

Invited Lectures

Limit-Computable Mathematics and Its Applications 1
Susumu Hayashi and Yohji Akama

Automata, Logic, and XML 2
Frank Neven

μ-Calculus via Games (Extended Abstract) 27
Damian Niwiński

Rewriting and Constructive Mathematics

Bijections between Partitions by Two-Directional Rewriting Techniques 44
Max Kanovich

On Continuous Normalization 59
Klaus Aehlig and Felix Joachimski

Variants of Realizability for Propositional Formulas and the Logic of the Weak Law of Excluded Middle 74
Alexey V. Chernov, Dmitriy P. Skvortsov, Elena Z. Skvortsova, and Nikolai K. Vereshchagin

Compactness and Continuity, Constructively Revisited 89
Douglas Bridges, Hajime Ishihara, and Peter Schuster

Fixpoints and Recursion

Hoare Logics for Recursive Procedures and Unbounded Nondeterminism .. 103
Tobias Nipkow

A Fixpoint Theory for Non-monotonic Parallelism 120
Yifeng Chen

Greibach Normal Form in Algebraically Complete Semirings 135
Zoltán Ésik and Hans Leiß

Linear and Resource Logics

Proofnets and Context Semantics for the Additives 151
Harry G. Mairson and Xavier Rival

A Tag-Frame System of Resource Management for Proof Search in Linear-Logic Programming 167
Joshua S. Hodas, Pablo López, Jeffrey Polakow, Lubomira Stoilova, and Ernesto Pimentel

Resource Tableaux (extended abstract) 183
Didier Galmiche, Daniel Méry, and David Pym

Semantics

Configuration Theories 200
Pietro Cenciarelli

A Logic for Probabilities in Semantics 216
M. Andrew Moshier and Achim Jung

Possible World Semantics for General Storage in Call-By-Value 232
Paul Blain Levy

A Fully Abstract Relational Model of Syntactic Control of Interference 247
Guy McCusker

Temporal Logics and Games

Optimal Complexity Bounds for Positive LTL Games 262
Jerzy Marcinkowski and Tomasz Truderung

The Stuttering Principle Revisited: On the Expressiveness of Nested X and U Operators in the Logic LTL 276
Antonín Kučera and Jan Strejček

Trading Probability for Fairness 292
Marcin Jurdziński, Orna Kupferman, and Thomas A. Henzinger

Probability, Games and Fixpoints

A Logic of Probability with Decidable Model-Checking 306
Danièle Beauquier, Alexander Rabinovich, and Anatol Slissenko

Solving Pushdown Games with a Σ_3 Winning Condition 322
Thierry Cachat, Jacques Duparc, and Wolfgang Thomas

Partial Fixed-Point Logic on Infinite Structures 337
Stephan Kreutzer

On the Variable Hierarchy of the Modal μ-Calculus 352
Dietmar Berwanger, Erich Grädel, and Giacomo Lenzi

Complexity and Proof Complexity

Implicit Computational Complexity for Higher Type Functionals (Extended Abstract) 367
Daniel Leivant

On Generalizations of Semi-terms of Particularly Simple Form 382
Matthias Baaz and Georg Moser

Local Problems, Planar Local Problems and Linear Time 397
Régis Barbanchon and Etienne Grandjean

Equivalence and Isomorphism for Boolean Constraint Satisfaction 412
Elmar Böhler, Edith Hemaspaandra, Steffen Reith, and Heribert Vollmer

Ludics and Linear Logic

Travelling on Designs (Ludics Dynamics) 427
Claudia Faggian

Designs, Disputes and Strategies 442
Claudia Faggian and Martin Hyland

Classical Linear Logic of Implications 458
Masahito Hasegawa

Lambda-Calculi

Higher-Order Positive Set Constraints 473
Jean Goubault-Larrecq

A Proof Theoretical Account of Continuation Passing Style 490
Ichiro Ogata

Duality between Call-by-Name Recursion and Call-by-Value Iteration 506
Yoshihiko Kakutani

Decidability of Bounded Higher-Order Unification 522
Manfred Schmidt-Schauß and Klaus U. Schulz

Open Proofs and Open Terms: A Basis for Interactive Logic 537
Herman Geuvers and Gueorgui I. Jojgov

Logical Relations for Monadic Types 553
Jean Goubault-Larrecq, Slawomir Lasota, and David Nowak

Resolution and Proofs

On the Automatizability of Resolution
and Related Propositional Proof Systems 569
Albert Atserias and María Luisa Bonet

Extraction of Proofs from the Clausal Normal Form Transformation 584
Hans de Nivelle

Resolution Refutations and Propositional Proofs
with Height-Restrictions 599
Arnold Beckmann

Author Index 613

Limit-Computable Mathematics and Its Applications

Susumu Hayashi[1] and Yohji Akama[2]

[1] Kobe University
Rokko-dai Nada Kobe 657-8501, Japan
shayashi@kobe-u.ac.jp
http://www.shayashi.jp
[2] Tohoku University
Sendai Miyagi 980-8578, Japan
akama@math.tohoku.ac.jp
http://www.math.tohoku.ac.jp/~akama

Abstract. Limit-Computable Mathematics (LCM) is a fragment of classical mathematics in which classical principles are restricted so that the existence theorems are realized by limiting recursive functions. LCM is expected to be a right means for "Proof Animation," which was introduced by the first author. In the lecture, some mathematical foundations of LCM will be given together with its relationships to various areas.

LCM is constructive mathematics augmented with some classical principles "executable" by the limiting recursive functions of the computational learning theories. It may be said based on the notion of learning in the same sense that constructive mathematics is based on the notion of computation.

It was introduced to materialize the idea of Proof Animation by the first author, which is a technique to animate formal proofs for validation in the same sense as formal specifications are animated for validation. Proof animation resembles Shapiro's algorithmic debugging of logic programs, which is also based on learning theory.

LCM was conceived through a fact that David Hilbert's original *proof* of his famous finite basis theorem in 1888 is realized by Gold's idea of learning. Hilbert's proof is known to be a "first" non-computational proof of the area.

However, Hilbert's proof gives a limiting recursive process by which the solutions are *learned* (computable in the limit). This is because he used only the laws of excluded middle limited to Σ^0_1-formulas. LCM is a mathematics whose proofs are restricted to this kind of proofs. A remarkable thing is that a wide class of classical proofs of concrete mathematics falls in the scope of LCM.

Some different approaches of mathematical foundations of LCM have been given by the authors and Berardi. Hayashi, Kohlenbach et al. have shown that there is a hierarchy of the laws of excluded middle and their equivalent theorems in mathematics, resembling the hierarchy of reverse mathematics. Relationships of LCM to learning theory, computability theory over real numbers and others have been known. Information including manuscripts on LCM and Proof Animation are available at http://www.shayashi.jp/PALCM/.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, p. 1, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Automata, Logic, and XML

Frank Neven

University of Limburg
frank.neven@luc.ac.be

Abstract. We survey some recent developments in the broad area of automata and logic which are motivated by the advent of XML. In particular, we consider unranked tree automata, tree-walking automata, and automata over infinite alphabets. We focus on their connection with logic and on questions imposed by XML.

1 Introduction

Since Codd [11], databases have been modeled as first-order relational structures and database queries as mappings from relational structures to relational structures. It is, hence, not surprising that there is an intimate connection between database theory and (finite) model theory [58, 60]. As argued by Vianu, finite model theory provides the backbone for database query languages, while in turn, database theory provides a scenario for finite model theory. More precisely, database theory induces a specific measure of relevance to finite model theory questions and provides research issues that, otherwise, were unlikely to have risen independently.

Today's technology trends require us to model data that is no longer tabular. The World Wide Web Consortium has adopted a standard data exchange format for the Web, called Extended Markup Language (XML) [14], in which data is represented as labeled ordered attributed trees rather than as a table. A new data model requires new tools and new techniques. As trees have been studied in depth by theoretical computer scientists [24], it is no surprise that many of their techniques can contribute to foundational XML research. In fact, when browsing recent ICDT and PODS proceedings,[1] it becomes apparent that a new component is already added to the popular logic and databases connection: tree automata theory. Like in the cross-fertilization between logic and databases, XML imposes new challenges on the area of automata and logic, while the latter area can provide new tools and techniques for the benefit of XML research. Indeed, while logic can serve as a source of inspiration for pattern languages or query languages and as a benchmark for expressiveness of such languages, the application of automata to XML can, roughly, be divided into at least four categories:

[1] ICDT and PODS are abbreviations of *International Conference on Database Theory* and *Symposium on the Principles of Database Systems*, respectively. The following links provide more information: `http://alpha.luc.ac.be/ lucp1080/icdt/` and `http://www.acm.org/sigmod/pods/`.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 2–26, 2002.
© Springer-Verlag Berlin Heidelberg 2002

- as a formal model of computation;
- as a means of evaluating query and pattern languages;
- as a formalism for describing schema's; and
- as an algorithmic toolbox.

In this paper, we survey three automata formalisms which are resurrected by recent XML research: unranked tree automata, tree-walking automata, and automata over infinite alphabets. Although none of these automata are new, their application to XML is. The first two formalism ignore attributes and text values of XML documents, and simply take finite labeled (unranked) trees as an abstraction of XML; only the last formalism deals with attributes and text values. For each of the models we discuss their relationship with XML, survey recent results, and demonstrate new research directions.

The current presentation is not meant to be exhaustive and the choice of topics is heavily biased by the author's own research. Furthermore, we only discuss XML research issues which directly motivate the use of the automata presented in this paper. For a more general discussion on database theory and XML, we suggest the survey papers by Abiteboul [1] and Vianu [61] or the book by Abiteboul, Buneman, and Suciu [2]. We do not give many proofs and the purpose of the few ones we discuss is merely to arouse interest and demonstrate underlying ideas. Finally, we mention that automata have been used in database research before: Vardi, for instance, used automata to statically analyze datalog programs [59].

The paper is further organized as follows. In Section 2, we discuss XML. In Section 3, we provide the necessary background definitions concerning trees and logic. In Section 4, we consider unranked tree automata. In brief, unranked trees are trees where every node has a finite but arbitrary number of children. In Section 5, we focus on computation by tree-walking. In Section 6, we consider such automata over infinite alphabets. We conclude in Section 7.

2 Basics of XML

We present a fairly short introduction to XML. In brief, XML is a data-exchange format whose enormous success is due to its flexibility and simplicity: almost any data format can easily be translated to XML in a transparent manner. For the purpose of this paper, the most important observation is that XML documents can be faithfully represented by labeled attributed ordered trees. Detailed information about XML can be found on the web [14] and, for instance, in the O'Reilly XML book [49].

We illustrate XML by means of an example. Consider the XML document in Figure 1 which displays some information about crew members in a spaceship. As for HTML, the building blocks of XML are *elements* delimited by *start-* and *end-tags*. A start-tag of a `crew`-element, for instance, is of the form `<crew>`, whereas the corresponding closing tag, indicating the end of the element, is `</crew>`. So, all text between and including the tags `<crew>` and `</crew>` in Figure 1, constitutes a `crew`-element. Elements can be arbitrarily nested inside other elements:

```
<starship name="Enterprise">
  <crew id="a457">
    <name> Scotty </name>
    <species> Human </species>
    <job> automata </job>
  </crew>
  <crew id="a544">
    <name> Spock </name>
    <species> Vulcan </species>
    <job> logic </job>
  </crew>
</starship>
```

Fig. 1. Example of an XML document

the element `<name> Spock </name>`, for instance, is a subelement of the outer `crew`-element. Elements can also have *attributes*. These are name value pairs separated by the equality sign. The value of an attribute is always atomic. That is, they cannot be nested. The attribute appears in the start-tag of the element it belongs to. For instance, `<starship name="Enterprise">` indicates that the value of the `name` attribute of that particular `starship`-element is `Enterprise`.

An XML document can be viewed as a tree in a natural way: the outermost element is the root and every element has its subelements as children. An attribute of an element is simply an attribute of the corresponding node. The tree in Figure 2, for instance, corresponds to the XML document of Figure 1. There is no unique best way to encode XML documents as trees. Another possibility is to encode attributes as child nodes of the element they belong to. In the present paper we stick to the former encoding.

Usually, we are not interested in documents containing arbitrary elements, but only in documents that satisfy some specific constraints. One way to define such "schema's" is by means of DTDs (Document Type Definitions). DTDs are, basically, extended context-free grammars. These are context-free grammars with regular expressions as right-hand sides. In Figure 3, we give an example of a DTD describing the data type of a spaceship. The DTD specifies that `starship` is the outer most element; that every `crew` element has `name` and

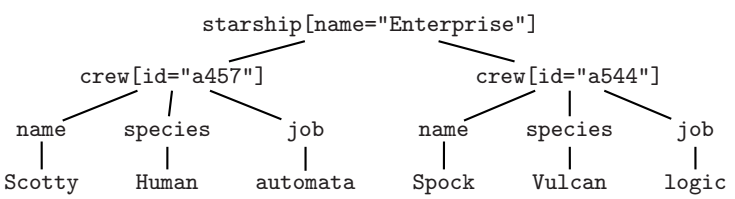


Fig. 2. Tree representation of the XML document in Figure 1

```
<!DOCTYPE starship [
    <!ELEMENT starship (crew)*>
    <!ELEMENT crew     (name,species,(rank | job))>
    <!ELEMENT name     (#PCDATA)>
    <!ELEMENT species  (#PCDATA)>
    <!ELEMENT rank     (#PCDATA)>
    <!ATTLIST starship name   CDATA>
    <!ATTLIST crew     id     CDATA>
]>
```

Fig. 3. A DTD describing the structure of the document of Figure 1

`species` as its first and second subelement, respectively, and `rank` or `job` as its third subelement. So, `|` and `,` denote disjunction and concatenation, respectively. `#PCDATA` indicates that the element has no subelements but consists of text only. `ATTLIST` determines which attribute belongs to which element. The attributes specified in this DTD can only have a single string value. DTDs are not the only means for representing schema's for XML. We briefly come back to this at the end of Section 4.4.

Attributes can also be used to link nodes. For instance, the id `a457` of `Scotty` in Figure 1, can be used in a different place in the document to refer to the latter: for instance,

```
<cabin>
  <number> 988 </number>
  <inhabitant> a457 </inhabitant>
</cabin>
```

Actually, the id-attribute has a special meaning in XML but we do not discuss this as it is not important for the present paper.

As indicated above, XML documents can be faithfully represented by trees. In this respect, inner nodes correspond to elements, while leaf nodes contain in general arbitrary text. In the next sections (with exception of Section 6), we only consider the structure of XML documents and, therefore, will ignore attributes and the text in the leaf nodes. Hence, XML documents are trees over a finite alphabet where the alphabet in question is, for instance, determined by a DTD. However, such trees are unranked: nodes can have an arbitrary number of children (the DTD in Figure 3, for instance, allows an unbounded number of `crew` elements). Although ranked trees, that is, trees where the number of children of each node is bounded by a fixed constant, have been thoroughly investigated during the past 30 years [24, 57], their unranked counterparts have been rather neglected. In Section 4, we recall the definition of unranked tree automata and consider some of their basic properties. First, we introduce the necessary notation in the next section.

3 Trees and Logic

3.1 Trees

For the rest of this paper, we fix a finite alphabet Σ of element names. The set of Σ-trees, denoted by $\mathcal{T}_\Sigma$, is inductively defined as follows:

(*i*) every $\sigma \in \Sigma$ is a Σ-tree;
(*ii*) if $\sigma \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, $n \geq 1$ then $\sigma(t_1, \ldots, t_n)$ is a Σ-tree.

Note that there is no a priory bound on the number of children of a node in a Σ-tree; such trees are therefore *unranked*. For every tree $t \in \mathcal{T}_\Sigma$, the *set of nodes of* t, denoted by $\mathrm{Dom}(t)$, is the subset of $\mathbb{N}^*$ defined as follows: if $t = \sigma(t_1 \cdots t_n)$ with $\sigma \in \Sigma$, $n \geq 0$, and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$, then $\mathrm{Dom}(t) = \{\varepsilon\} \cup \{iu \mid i \in \{1, \ldots, n\}, u \in \mathrm{Dom}(t_i)\}$. Thus, ε represents the root while vj represents the j-th child of v. By $\mathrm{lab}^t(u)$ we denote the label of u in t. In the following, when we say tree we always mean Σ-tree.

Next, we define our formalization of DTDs.

Definition 1. A *DTD* is a tuple (d, s_d) where d is a function that maps Σ-symbols to regular expressions over Σ and $s_d \in \Sigma$ is the start symbol. In the sequel we just say d rather than (d, s_d).

A tree t *satisfies* d iff $\mathrm{lab}^t(\varepsilon) = s_d$ and for every $u \in \mathrm{Dom}(t)$ with n children, $\mathrm{lab}^t(u1) \cdots \mathrm{lab}^t(un) \in d(\mathrm{lab}^t(u))$. Note that if u has no children ε should belong to $d(\mathrm{lab}^t(u))$.

Example 1. As an example consider the following DTD describing the XML document in Figure 1:

$$\begin{aligned}
d(\texttt{starship}) &:= \texttt{crew}^* \\
d(\texttt{crew}) &:= \texttt{name} \cdot \texttt{species} \cdot (\texttt{rank} + \texttt{job}) \\
d(\texttt{name}) &:= \varepsilon \\
d(\texttt{species}) &:= \varepsilon \\
d(\texttt{rank}) &:= \varepsilon \\
d(\texttt{job}) &:= \varepsilon
\end{aligned}$$

Recall that, for the moment, we are only interested in the structure of XML documents. Therefore, `name`, `species`, `rank`, and `job` are mapped to ε. In Section 6, we consider text and attribute values. □

3.2 Logic

We can also view trees as logical structures (in the sense of mathematical logic [18]). We make use of the relational vocabulary $\tau_\Sigma := \{E, <, (O_\sigma)_{\sigma \in \Sigma}\}$ where E and $<$ are binary and all the O_σ are unary relation symbols. The domain of t, viewed as a structure, equals the set of nodes of t, i.e., $\mathrm{Dom}(t)$. Further, E is the edge relation and equals the set of pairs $(v, v \cdot i)$ where $v, v \cdot i \in \mathrm{Dom}(t)$. The relation $<$ specifies the ordering of the children of a node, and equals the

set of pairs $(v \cdot i, v \cdot j)$, where $i < j$ and $v \cdot j \in \mathrm{Dom}(t)$. For each σ, O_σ is the set of nodes that are labeled with a σ.

We consider first-order (FO) and monadic second-order logic (MSO) over these structures. In brief, MSO is FO extended with quantification over set variables. We refer the unfamiliar reader to, e.g., the books by Ebbinghaus and Flum [18], or the chapter by Thomas [57]. In Section 5.3, we also consider transitive closure logic.

Example 2. As an example, consider the MSO formula φ defining the set of trees where every a-labeled node always has a b-labeled descendant:

$$\varphi := \forall x(O_a(x) \rightarrow \exists y(O_b(y) \wedge \mathrm{desc}(x, y))).$$

Here, $\mathrm{desc}(x, y)$ is an abbreviation of the formula

$$\forall X\Big(\big(X(x) \wedge \forall z \forall z'(X(z) \wedge E(z, z') \rightarrow X(z'))\big) \rightarrow X(y)\Big).$$

The formula $\mathrm{desc}(x, y)$ says that any set which contains x and is closed under the edge relation, also contains y. So, it defines the pairs (x, y) where y is a descendant of x. □

4 Unranked Tree Automata

Research on unranked trees in the context of XML was initiated by Brüggemann-Klein, Murata, and Wood [8] based on early work of Pair and Quere [46] and Takahashi [55]. They considered mostly language theoretic properties like non-determinism, two-wayness, tree grammars,.... Since then, quite a number of applications, based on their initial ideas, have risen. We discuss these applications in Section 4.4. In Sections 4.1–4.3, we focus on non-deterministic tree automata, the connection with binary tree automata, expressiveness, and complexity.

4.1 Definition

Definition 2. A *nondeterministic tree automaton (NTA)* is a tuple $B = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a function $Q \times \Sigma \rightarrow 2^{Q^*}$ such that $\delta(q, a)$ is a regular string language over Q^* for every $a \in \Sigma$ and $q \in Q$.

A *run* of B on a tree t is a labeling $\lambda : \mathrm{Dom}(t) \mapsto Q$ such that for every $v \in \mathrm{Dom}(t)$ with n children, $\lambda(v1) \cdots \lambda(vn) \in \delta(\lambda(v), \mathrm{lab}^t(v))$. Note that when v has no children, then the criterion reduces to $\varepsilon \in \delta(\lambda(v), \mathrm{lab}^t(v))$. A run is *accepting* iff $\lambda(\varepsilon) \in F$. A tree is accepted if there is an accepting run. The set of all accepted trees is denoted by $L(B)$. We call a set of trees *regular* when it can by recognized by an NTA.

We illustrate the above definition with an example.

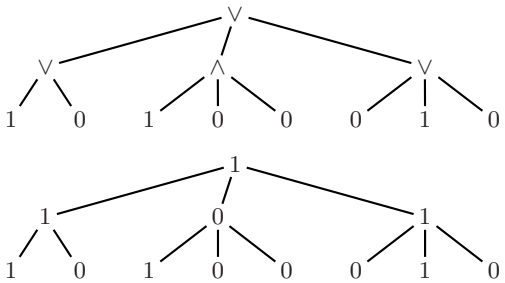


Fig. 4. A tree and an accepting run of the automaton of Example 3

Example 3. (1) Consider the alphabet $\Sigma = \{\wedge, \vee, 0, 1\}$. Suppose for ease of exposition that trees are always of the following form: 0 and 1 only appear at leaves, $\wedge$ and $\vee$ can appear everywhere except at leaves. These are all tree-shaped positive boolean circuits. We next define an automaton accepting exactly the circuits evaluating to 1. Define $B = (Q, \Sigma, \delta, F)$ with $Q = \{0, 1\}$, $F = \{1\}$, and

$$\begin{aligned}
&\delta(0,0) := \delta(1,1) := \{\varepsilon\};\\
&\delta(0,1) := \delta(1,0) := \emptyset;\\
&\delta(0,\wedge) := (0+1)^*0(0+1)^*;\\
&\delta(1,\wedge) := 1^*;\\
&\delta(0,\vee) := 0^*;\ \text{and,}\\
&\delta(1,\vee) := (0+1)^*1(0+1)^*.
\end{aligned}$$

Intuitively, B works as follows: B assigns 0 (1) to 0-labeled (1-labeled) leaves; B assigns a 1 to a $\wedge$-labeled node iff all its children are 1; B assigns a 0 to a $\vee$-labeled node iff all its children are 0. Finally, B accepts when the root is labeled with 1. In Figure 4, we give an example of a tree and an accepting run.

(2) The automaton accepting the DTD of Example 1 is defined as follows: $B = (Q, \Sigma, \delta, F)$ with

$$Q := \{\texttt{starship}, \texttt{crew}, \texttt{name}, \texttt{rank}, \texttt{job}, \texttt{species}\},$$

$F := \{\texttt{starship}\}$, for all $a \in \{\texttt{name}, \texttt{job}, \texttt{species}, \texttt{rank}\}$, $\delta(a, a) = \{\varepsilon\}$, and

$$\begin{aligned}
\delta(\texttt{starship}, \texttt{starship}) &:= \texttt{crew}^*;\\
\delta(\texttt{crew}, \texttt{crew}) &:= \texttt{name} \cdot \texttt{species} \cdot (\texttt{rank} + \texttt{job}).
\end{aligned}$$

The $\delta(q, a)$ that are not mentioned are empty. □

4.2 Connection with Ranked Trees

Before we start to develop a theory of regular unranked trees, it makes sense to reflect upon the relationship with regular binary trees. Unranked trees can be uniformly encoded as binary trees. We just mention one possible encoding. See,

e.g., Figure 5 for an illustration. Intuitively, the first child of a node remains the first child of that node in the encoding, but it is explicitly encoded as a left child. The remaining children are right descendants of the first child. Whenever there is right child but no left child, a # is inserted. Additionally, when there is only a left child, a # is inserted for the right child.

Using the encodings enc and dec of Figure 5 one obtains the following proposition (we represent the transition functions of NTAs by NFAs).

Proposition 1. [54]

1. *For every unranked NTA B there is a binary tree automaton A such that $L(A) = \{enc(t) \mid t \in L(B)\}$. The size of A is polynomial in the size of B.*
2. *For every binary tree automaton A there is an unranked NTA B such that $L(B) = \{dec(t) \mid t \in L(A)\}$. The size of B is polynomial in the size of A.*

The above proposition allows to transfer all closure properties of the class of ranked tree automata to the class of unranked tree automata.

4.3 Expressiveness and Complexity

As enc and dec are MSO definable, Proposition 1 implies that the famous Doner-Thatcher-Wright characterization of ranked tree automata easily carries over to unranked trees [17, 56].

Corollary 1. [43] *A set of trees L is regular iff there is an MSO formula φ such that $L = \{t \mid t \models \varphi\}$.*

Although Proposition 1 provides a tool for transferring results from ranked to unranked trees, it does not deal with issues which are specific for unranked tree automata. The complexity of decision problems for NTAs, for instance, depends on the formalism used to represent the regular string languages $\delta(q, a)$ in the transition function. As there are many ways to represent regular string languages (logical formulas, automata with various forms of control, grammars,

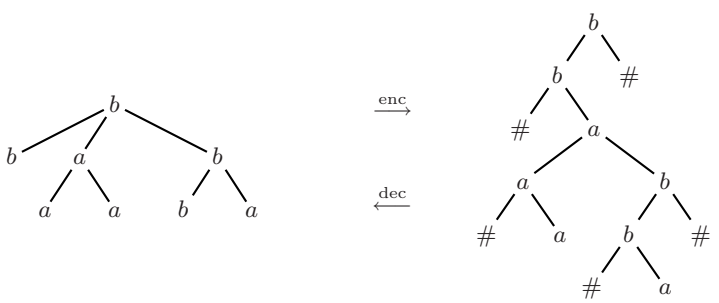


Fig. 5. An unranked tree and its binary encoding

regular expressions), Proposition 1 does not seem to offer immediate help. We dwell a bit further upon this issue.

In the following $\mathcal{M}$ always denotes classes of representations of regular languages. So, NTA($\mathcal{M}$) denotes the set of NTAs where the regular languages $\delta(q, a)$ are represented by elements in $\mathcal{M}$. The translation between binary and unranked automata mentioned in Proposition 1 is polynomial for NTA(NFA)'s. For this reason, the latter class can be seen as the default for unranked tree automata. For the latter class, the complexity of the membership problem is quite tractable. The size of an automaton B is $|Q| + |\Sigma| + \sum_{q,a} |\delta(q, a)| + |F|$. By $|\delta(q, a)|$ we mean the size of the automaton accepting $\delta(q, a)$ not the size of the language.

Proposition 2. *Let $t \in \mathcal{T}_\Sigma$ and $B \in$ NTA(NFA). Testing whether $t \in L(B)$ can be done in time $\mathcal{O}(|t||B|^2)$.*

When using tree automata to obtain upper bounds on the complexity of problems related to XML, one sometimes needs to turn to more succinct formalisms. In [32], for instance, a PSPACE upper bound on the complexity of the typechecking problem for structural recursion is obtained by a reduction to the emptiness problem of NTA(2AFA)'s. Here, 2AFA stands for the class of two-way alternating string automata. In this respect, it, therefore, makes sense to explore various possibilities of $\mathcal{M}$. We mention some initial results.

We consider the following well-known decision problems:

- EMPTINESS($\mathcal{M}$): given an NTA($\mathcal{M}$) B, decide whether $L(B) = \emptyset$;
- CONTAINMENT($\mathcal{M}$): given two NTA($\mathcal{M}$)'s B_1 and B_2, decide whether $L(B_1) \subseteq L(B_2)$;
- EQUIVALENCE($\mathcal{M}$): given two NTA($\mathcal{M}$)'s B_1 and B_2, decide whether $L(B_1) = L(B_2)$;

Proposition 3. [32]

1. EMPTINESS(NFA) *is in* PTIME.
2. EMPTINESS(2AFA) *is in* PSPACE.
3. CONTAINMENT(2AFA) *is in* EXPTIME.
4. EQUIVALENCE(2AFA) *is in* EXPTIME.

Theorem 3 is optimal as even for fixed arity trees, emptiness and containment are PTIME-hard and EXPTIME-hard, respectively [12, 52]. Further, emptiness of 2DFAs is known to be hard for PSPACE.

4.4 Applications

Unranked tree automata can serve XML research in at least four different ways:

1. **as a basis of schema languages and validating of schema's.** Murata was the first to consider tree automata as a schema definition language [35]. In fact, the schema language Relax, a competitor of XML schema [15], is directly inspired upon unranked tree automata. The XDuce type system of

Pierce and Hosoya [27] as well as the specialized DTDs of Papakonstantinou and Vianu [47] correspond precisely to the unranked tree languages [54]. Lee, Mani, and Murata provide a comparison of XML schema languages based on formal language theory [31].

2. **as an evaluation mechanism for pattern languages.** Several researchers defined pattern languages for unranked trees that can be implemented by unranked tree automata: Neumann and Seidl develop a μ-calculus for expressing structural and contextual conditions on forests [37].[2] They show that their formalism can be implemented by push-down forest automata. The latter are special cases of unranked tree automata. Murata defines an extension of path expressions based on regular expressions over unranked trees [36]. Brüggemann-Klein and Wood consider caterpillar expressions [9]. These are regular expressions that in addition to labels can specify movement through the tree. Neven and Schwentick define a guarded fragment ETL of MSO whose combined complexity is much more tractable than that of general MSO [22, 40, 50]. Expressiveness and complexity results on ETL are partly obtained via techniques based on unranked tree automata.
3. **as an algorithmic toolbox.** For instance, Miklau and Suciu used (binary) tree automata to obtain an algorithm for XPath containment [33]. As mentioned above, Martens and Neven obtain upper bounds on the complexity of type checking by a reduction to the emptiness test of unranked tree automata [32]. Unranked tree automata as a toolbox are hardly developed. It would be helpful to have general results on the complexity of unranked tree automata in terms of the complexity of the regular languages representing the transition functions.
4. **as a new paradigm.** Unranked tree automata use regular string languages to deal with unrankedness. The latter simple but effective paradigm found application in several formalisms.
   Neven and Schwentick define query automata [43]. These are two-way deterministic unranked tree automata that can select nodes in the tree. Query automata correspond exactly to the unary queries definable in monadic second-order logic. By a result of Gottlob and Koch they also correspond to the unary queries definable in monadic datalog [25]. In [38], an extension of the Boolean attribute grammars considered in [45] to unranked trees is defined. These also express precisely the unary queries in MSO. A translation of the region algebra, considered by Consens and Milo [13], into these attribute grammars drastically improves the complexity of the optimization problems of the former. We refer the interested reader to [42] for a more detailed overview of pattern languages based on tree automata.

5 Tree - Walking Automata

Next, we focus on computation by tree-walking. This is a well-known paradigm from formal language theory studied in the context of attribute grammars and

[2] A forest is a concatenation of unranked trees.

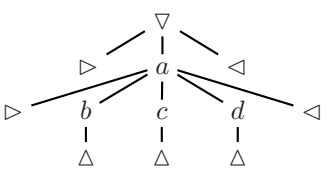
tree-transformations [4, 7, 16]. This paradigm materialized in XML research in various ways. Indeed, a first instance of tree-walking is provided by the caterpillar expressions of Brüggemann-Klein and Wood [9]. Further, Milo, Suciu, and Vianu [34] defined a tree-walking tree-transducer model with pebbles as an abstract model for XML transformations. Segoufin and Vianu considered tree-walking automata in the context of XML streaming [51]. Finally, as argued by Bex, Maneth and Neven [6], stripped down, XSLT is essentially a tree-walking tree-transducer with registers and look-ahead. We embark on the issue of registers and look-ahead in the next section. In the present section, we consider ordinary tree-walking automata.

Admittedly, the application of tree-walking to XML is less direct than that of unranked tree automata. However, we hope that a thorough understanding of the tree-walking paradigm leads to more insight in the operation and expressiveness of languages like XSLT.

5.1 Definition

Before we give the definition of tree-walking automata, let's recall two-way deterministic finite state machines on *strings*: such devices 'walk' in two directions over a string changing state and direction depending on the current state, the current symbol and whether the current position is the left or right-delimiter. An automaton accepts if a final state is reached at some point. Analogously, a *tree*-walking automaton is a finite state device walking a *tree*. Its control is always at one node of the input tree. Based on the label of that node, its state, and its position in the tree (first or last child, root, or leaf), the automaton changes state and moves to one of the neighboring nodes (parent, first child, left or right sibling). The automaton accepts the tree when it enters a final state.

To simplify the definition of two way automata on strings, one usually delimits strings with the start and end symbols $\rhd$ and $\lhd$, respectively. We do the same for trees using the extra symbols $\vartriangle$ and $\vartriangledown$. For instance, if t is the tree $a(bcd)$ then $\mathrm{delim}(t)$ is the tree



Definition 3. A TWA C is a tuple (Q, Σ, q_0, q_F, P) where

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state and $q_F \in Q$ is the final state;
- P is a finite set of rules of the form $(q, \sigma) \to (q', d)$ where $\sigma \in \Sigma$, $q, q' \in Q$, and $d \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \mathrm{stay}\}$

Intuitively, a transitions $(q, \sigma) \to (d, q')$ can only be applied in state q at a node labeled with σ. Further, it changes state to q' and moves in direction d

where $\leftarrow, \uparrow, \rightarrow, \downarrow$, and stay mean *go to left sibling*, *go to parent*, *go to right sibling*, *go to first child*, and *stay put*, respectively. We assume that there is no transition possible from the final state.

Formally, a configuration on a tree t is a tuple $[u, q]$ where $q \in Q$ is the current state and $u \in \mathrm{Dom}(t)$ is the current node. Before we define the transition relation, we define the (partial) move function m_d for every $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, \mathrm{stay}\}$ as follows. For every node u, $m_{\leftarrow}(u)$, $m_{\rightarrow}(u)$, $m_{\uparrow}(u)$, $m_{\downarrow}(u)$, and $m_{\mathrm{stay}}(u)$ equals the left sibling, the right sibling, the parent, the first child of u, and u, respectively (if they exist). Given $\gamma = [u, q]$ and $\gamma' = [u', q']$, we define the one step transition relation $\vdash$ as follows: $\gamma \vdash \gamma'$ iff there is a transition $(q, \sigma) \rightarrow (q', d) \in P$ such that $\mathrm{lab}^t(u) = \sigma$ and $m_d(u) = u'$. By $\vdash^*$ we denote the transitive closure of $\vdash$. Finally, C accepts the input tree t if $[\varepsilon, q_0] \vdash^* [\varepsilon, q_F]$.

We say that C is deterministic if there is at most one rule $(q, \sigma) \rightarrow \alpha$ in P for every $\sigma \in \Sigma$ and $q \in Q$. Denote the class of deterministic TWAs by DTWA.

We illustrate the above definition with an example.

Example 4. We construct an automaton accepting the tree defined by the XPath expression `//a[//b][//c]`. XPath is an XML pattern language employed by, for instance, XSLT [10]. We do not get into the specifics of the syntax. The present pattern selects all trees with an a-labeled node that has both a b and a c-labeled descendant. Let C be the TWA (Q, Σ, q_0, q_F, P) with $Q = \{q_a, q_b, q_c, q_{\mathrm{root}}, q_F\}$ and $q_0 = q_b$. We give the rules in P while explaining the operation of the automaton. First C nondeterministically searches a b using the following transitions: $(q_b, \sigma) \rightarrow (q_b, d)$, for every $\sigma \in \Sigma$ and $d \in \{\leftarrow, \uparrow, \rightarrow, \downarrow, \mathrm{stay}\}$, and $(q_b, b) \rightarrow (q_a, \mathrm{stay})$. Then, C moves up the tree until it finds a suitable a: $(q_a, \sigma) \rightarrow (q_a, \uparrow)$, for every $\sigma \in \Sigma$ and $(q_a, a) \rightarrow (q_c, \downarrow)$. Next, C moves down in search of a c: $(q_c, \sigma) \rightarrow (q_c, d)$, for every $\sigma \in \Sigma$ and $d \in \{\rightarrow, \downarrow\}$, and $(q_c, c) \rightarrow (q_{\mathrm{root}}, \uparrow)$. Finally, C walks to the root and accepts: $(q_{\mathrm{root}}, \sigma) \rightarrow (\sigma, \uparrow)$, for every $\sigma \in \Sigma$ and $(q_{\mathrm{root}}, \triangledown) \rightarrow (q_F, \mathrm{stay})$. □

5.2 Expressiveness

Most of the recent research on TWAs focused on ranked TWAs [19, 21, 41], not on unranked ones as defined here. Ranked TWAs are defined just as unranked ones. The only difference is that there is a fixed n such that only trees of rank n are considered as inputs. In particular, a tree has rank n if every node has n or fewer children. However, as with unranked tree automata, we can transfer results between the ranked and unranked case. Let enc and dec be the encoding and decoding discussed in Section 4.

Proposition 4. *1. For every unranked (D)TWA B there is a ranked (D)TWA C such that $L(C) = \{enc(t) \mid t \in L(B)\}$. The size of C is linear in the size of B.*

2. For every ranked (D)TWA C there is an unranked (D)TWA B such that $L(B) = \{dec(t) \mid t \in L(C)\}$. The size of B is linear in the size of C.

It follows from results in [19, 21] that the language accepted by ranked TWAs is regular. From Proposition 1 and Proposition 4, it follows that unranked TWAs only define regular tree languages. Essentially, the latter is all what is known. Even the most basic questions remain unanswered:

1. Do TWAs capture the regular tree languages?
2. Are DTWAs as expressive as TWAs?
3. Are TWAs closed under complement?

It is believed that the answer to all these questions is negative. Again, using Proposition 1 and Proposition 4, it can be shown that a negative answer on the class of unranked tree implies a negative answer on the class of ranked trees (the converse is obvious).

A negative answer to question one is proved in [41] for ranked TWAs that can visit every subtree only once (and a mild generalization thereof). Engelfriet and Hoogeboom supply two tree languages as possible candidates to separate TWAs from NTAs. Fülöp and Maneth [23] showed that the domains of partial attributed tree transducers correspond to the tree-walking automata in universal acceptance mode. No lower bounds on the expressiveness of TWAs have been obtained.

Denote the class of alternating TWAs by ATWA. The next proposition is a useful characterization of the regular unranked tree languages. It immediately follows from the result by Slutzki [53] and the fact that Proposition 4 also holds for alternation.

Proposition 5. *An unranked tree language is accepted by an ATWA iff it is regular.*

TWAs can also be used as an algorithmic toolbox. An upper bound on the non-circularity test of extended attribute grammars is obtained by a reduction to the emptiness test of TWAs.

Theorem 1. [38] *The emptiness, containment and equivalence problem of TWAs are* EXPTIME*-complete.*

We finish with a remark on robustness. A striking difference between ranked and unranked tree automata is that the former can check the label of the parent of the current node by remembering the child number in the state, moving up, and moving back down to the correct child. Seemingly, unranked automata can not achieve this as the child number is unbounded. For this reason, ranked TWAs can evaluate boolean circuits with a fixed fan-in [41] while unranked TWAs, probably, can not evaluate boolean circuits with an unbounded fan-in. However, a proof of the latter would solve question one above. A solution might be to let a move of an unranked TWA depend both on the label of the current node and the label of its parent. The question remains whether this would be the right model for unranked trees.

5.3 A Logical Characterization

In this section, we present a logical characterization of *ranked* TWAs. We add for every $m > 0$ the unary predicate depth_m to the vocabulary of trees. In all trees, depth_m will contain all vertices the depth of which is a multiple of m.

We characterize tree-walking automata by *transitive closure logic formulas* (TC logic) of a special form. We refer the reader unfamiliar with TC logic to, e.g., [18, 29]. As we only consider TC formulas in normal form, we refrain from defining TC logic in full generality. A TC formula in normal form is an expression of the form

$$\text{TC}[\varphi(x, y)](\varepsilon, \varepsilon),$$

where φ is an FO formula which may make use of the predicate depth_m, for some m, in addition to E, $<$ and the O_σ. Its semantics is defined as follows, for every tree t,

$$t \models \text{TC}[\varphi(x, y)](\varepsilon, \varepsilon),$$

iff the pair $(\varepsilon, \varepsilon)$ is in the transitive closure of the relation

$$\{(u, v) \mid t \models \varphi[u, v]\}.$$

We use *deterministic transitive closure logic formulas* (DTC) in an analogously defined normal form to capture deterministic tree-walking automata. In particular,

$$t \models \text{DTC}[\varphi(x, y)](\varepsilon, \varepsilon),$$

iff the pair $(\varepsilon, \varepsilon)$ is in the transitive closure of the relation

$$\{(u, v) \mid t \models \varphi[u, v] \wedge (\forall z)(\varphi[u, z] \rightarrow z = v)\}.$$

The latter expresses that we disregard vertices u that have multiple φ-successors.

As an example consider the formula

$$\varphi(x, y) := (E(x, y) \wedge O_a(x) \wedge O_a(y)) \vee (\text{leaf}(x) \wedge y = \varepsilon).$$

Here, $\text{leaf}(x)$ is a shorthand expressing that x is a leaf. Then, for all trees t, $t \models \text{DTC}[\varphi(x, y)](\varepsilon, \varepsilon)$ iff there is a path containing only a's from the root to a leaf such that every non-leaf vertex on that path has precisely one a-labeled child. In contrast, $t \models \text{TC}[\varphi(x, y)](\varepsilon, \varepsilon)$ iff there is a path from the root to a leaf carrying only a's.

Theorem 2.

1. *A ranked tree language is accepted by a nondeterministic tree-walking automaton iff it is definable by a TC formula in normal form.*
2. *A ranked tree language is accepted by a deterministic tree-walking automaton iff it is definable by a DTC formula in normal form.*

The simulation in TC-logic is an easy extension of a proof of Potthoff [48] who characterized two-way *string* automata by means of TC formulas in normal form. The latter direction also holds for unranked trees. To show that every TWA can evaluate a TC formula in normal form, we make use of Hanf's Theorem (see, e.g., [18]). This result intuitively says, for graphs of bounded degree, that whether a FO sentence holds depends only on the number of pairwise disjoint spheres of each isomorphism type of some fixed radius. Furthermore, the exact number is only relevant up to a certain fixed threshold, only depending on the formula. As unranked trees do not have bounded degree it is unclear whether the latter result can be extended to unranked trees.

The above result thus implies that any lower bound on (D)TC formulas in normal form is also a lower bound for (non)deterministic tree-walking automata. It is open whether the depth_m predicates are necessary. Unfortunately, proving lower bounds for the above mentioned logics does not seem much easier than the original problem as Ehrenfeucht games for DTC and TC are quite involved [18].

Engelfriet and Hoogeboom showed that tree-walking automata with pebbles correspond exactly to transitive closure logic without restrictions [20]. Hence, when allowing pebbles one can simulate nested TC operators.

6 Tree-Walking and Data-Values

In the previous sections, we primarily focused on the tree structure of XML documents. Our abstraction ignores an important aspect of XML, namely the presence of *data values* attached to leaves of trees or to attributes, and comparison tests performed on them by XML queries. These data values make a big difference – indeed, in some cases the difference between decidability and undecidability (e.g., see [5]). As the connection to logic and automata proved very fruitful in foundational XML research, it is therefore important to extend the automata and logic formalisms to trees with data values.

6.1 Trees and Logic Revisited

We take a radical view when dealing with text. Indeed, we move all text occurring at leaves into the attributes. For instance, the XML document in Figure 1 can be represented as in Figure 6. Although this approach leads to awkward XML documents, it, nevertheless, remains a valid representation.

Next, we add attributes to our Σ-trees. To this end, we assume an infinite domain $\mathbf{D} = \{d_1, d_2, \ldots\}$ and a finite set of attributes A.

```
<starship name="Enterprise">
  <crew id="a457" name="Scotty" species="Human" job="automata"/>
  <crew id="a544" name="Spock" species="Vulcan" job="logic"/>
</starship>
```

Fig. 6. Example of an XML document with all text moved into the attributes

Definition 4. An *attributed Σ-tree* is a pair $(t, (\lambda_a^t)_{a \in A})$, where $t \in \mathcal{T}_\Sigma$ and for each $a \in A$, $\lambda_a^t : \mathrm{Dom}(t) \mapsto \mathbf{D}$ is a function defining the a-attribute of nodes in t.

Of course, in real XML documents, usually, not all element types have the same set of attributes. Obviously, this is just a convenience and not a restriction. Further, XML documents can contain elements with mixed content. For instance, consider the XML document

```
<p>This is <em>not</em> a problem.</p>.
```

Here, we use the special text label `T` and the attribute `text`, to represent the document by the tree. That is,

```
<p>
    <T text="This is"/>
    <em text="not"/>
    <T text="a problem."/>
</p>.
```

In the following, when we say tree we always mean attributed Σ-tree.

For our logics, we make use of the extended vocabulary $\tau_{\Sigma,A} = \{E, <, \prec, (O_\sigma)_{\sigma \in \Sigma}, (\mathrm{val}_a)_{a \in A}\}$. Here, each val_a is a function, from $\mathrm{Dom}(t)$ to $\mathbf{D}$. The logic at hand is based on the logics accompanying the metafinite structures of Grädel and Gurevich [26].

An atomic formula is of the form $E(x, y)$, $x < y$, $x \prec y$, $O_\sigma(x)$, $x = y$, $\mathrm{val}_a(x) = \mathrm{val}_b(y)$ or $\mathrm{val}_a(x) = d$ where $a, b \in A$ and $d \in \mathbf{D}$. Such formulas have the obvious semantics.[3] FO* is obtained by closing the atomic formulas under the boolean connectives and first-order quantification over $\mathrm{Dom}(t)$. As an example consider the FO* sentence $\forall x(\mathrm{val}_a(x) = d \vee \mathrm{val}_a(x) = \mathrm{val}_b(x))$, expressing that the value of every a-attribute is d or is equal to the b-attribute. We stress that no quantification over $\mathbf{D}$ is possible. We get MSO* by extending FO* with set quantification over $\mathrm{Dom}(t)$. To emphasize the difference with the logics of the previous section we use a $*$ and denote the logics by FO* and MSO*, respectively.

6.2 Tree-Walking Automata Extended with Registers

To deal with data values, a tree-walking automaton is equipped with a finite number of registers. An automaton can store data-values in registers and can check whether an attribute value of the current node is equal to the content of some register. This is, essentially, the model of Kaminski and Francez who studied string automata over infinite alphabets [30]. Actually, this is also the way XSLT deals with data values. Indeed, the XSLT counterpart of registers are variables that can be passed between templates. We assume that every attribute of a delimiter $\triangleright, \triangledown, \triangleleft, \vartriangle$ contains $\perp$ where $\perp \notin \mathbf{D}$.

[3] $x \prec y$ says that y is a descendant of x.

Definition 5. A *k-register DTWA* $\mathcal{B}$ is a tuple (Q, q_0, q_F, τ_0, P) where

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state and $q_F \in Q$ is the final state;
- $\tau_0 : \{1, \ldots, k\} \to \mathbf{D} \cup \{\perp\}$ is the initial register assignment; and,
- P is a finite set of rules of the form $(\sigma, q, \xi) \to \alpha$. Here, $\sigma \in \Sigma$, $q \in Q$, and ξ is a Boolean combination of atomic formulas of the form $j = b$ where $j \in \{1, \ldots, k\}$ and $b \in A$. We define α below.

Intuitively, transitions $(\sigma, q, \xi) \to \alpha$ can only be applied in state q at a node carrying a σ that satisfies ξ under the assignment interpreting j by the content of register j and b by the value of the b-attribute. The right-hand side α can be one of the following:

- (q', d) with $q' \in Q$ and $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow, \text{stay}\}$; intuitively, this means *change to state q' and move in direction d*; or,
- (q', i, a) where $q' \in Q$, $i \in \{1, \ldots, k\}$, and $a \in A$; intuitively, this means *change to state q', and replace the content of register i by the value of attribute a.*

We assume that no transition is possible from the final state. Further, we assume that the automaton never moves off the input tree.

Given a tree t, a *configuration of $\mathcal{B}$ on t* is a tuple $[u, q, \tau]$ where $u \in \mathrm{Dom}(t)$, $q \in Q$, and $\tau : \{1, \ldots, k\} \to \mathbf{D}$. That is, u is the current node, q the current state, and τ the register content. The *initial* configuration is $\gamma_0 := [\varepsilon, q_0, \tau_0]$. A configuration $[u, q_F, \tau]$ is *accepting*. A rule (σ, p, ξ) applies to a configuration $[u, q, \tau]$ iff $\mathrm{lab}^t(u) = \sigma$, $p = q$ and ξ holds under the interpretation induced by τ where in addition each $a \in A$ is interpreted by $\mathrm{val}^t_a(u)$. We assume that automata are deterministic: if (σ, q, ξ_1) and (σ, q, ξ_2) appear as left-hand sides then there is never a configuration such that both ξ_1 and ξ_2 apply.

Given $\gamma = [u, q, \tau]$ and $\gamma' = [u', q', \tau']$, we define the one step transition relation $\vdash$ as follows: $\gamma \vdash \gamma'$ iff there is a transition $(\sigma, q, \xi) \to \alpha$ that applies to γ and if α is of the form (p, d) then $p = q'$, $m_d(u) = u'$, and $\tau = \tau'$; otherwise, if α is of the form (q, i, a) then $p = q'$, $u = u'$, $\tau'(i) = \mathrm{val}^t_a(u)$, and $\tau'(j) = \tau(j)$ for all $j \neq i$. By $\vdash^*$ we denote the transitive closure of $\vdash$.

Finally, $\mathcal{B}$ accepts the input tree t if $\gamma_0 \vdash^* \gamma$ for some accepting configuration γ.

Example 5. Consider the tree-walking automaton that checks whether there is a node with the same a-attribute as the root. We assume $\Sigma = \{\sigma\}$ and $A = \{a\}$. The automaton starts by putting the value of the a-attribute of the root in the first register; subsequently, it makes a depth-first traversal of the tree; it accepts when it encounters a node with the same a-value as in the first register. Define $\mathcal{B} = (Q, q_0, Q_F, \tau_0, P)$ as the one-register automaton where $Q = \{q_0, q_{\text{down}}, q_{\text{up}}, q_F\}$, $Q_F = \{q_F\}$, $\tau(1) = \perp$ and P contains the following

rules:

$$
\begin{array}{ll}
(q_0, \triangledown, \text{true}) & \rightarrow (q_0, \downarrow) \\
(q_0, \triangleright, \text{true}) & \rightarrow (q_0, \rightarrow) \\
(q_0, \sigma, \text{true}) & \rightarrow (q_{\text{down}}, 1, a) \\
 & \\
(q_{\text{down}}, \sigma, 1 = a) & \rightarrow (q_F, \text{stay}) \\
(q_{\text{down}}, \sigma, \neg(1 = a)) & \rightarrow (q_{\text{down}}, \downarrow) \\
(q_{\text{down}}, \triangleright, \text{true}) & \rightarrow (q_{\text{down}}, \downarrow) \\
(q_{\text{down}}, \vartriangle, \text{true}) & \rightarrow (q_{\text{up}}, \uparrow) \\
(q_{\text{down}}, \triangleleft, \text{true}) & \rightarrow (q_{\text{up}}, \uparrow) \\
 & \\
(q_{\text{up}}, \sigma, \text{true}) & \rightarrow (q_{\text{down}}, \rightarrow) \\
(q_{\text{up}}, \triangledown, \text{true}) & \rightarrow (q_{\text{up}}, \text{stay})
\end{array}
$$

□

6.3 Expressiveness

The expressiveness of k-register DTWAs behaves in a strange way: on the one hand, they can compute properties not in MSO*, while on the other hand they cannot even compute all FO*-definable properties. This is a bit awkward as, in database theory, first-order logic is generally accepted as the minimum expressiveness a query language should have, while MSO, due to its correspondence with various automata, stands for regularity and robustness.

Interestingly, the inexpressibility proof makes use of communication complexity [28]. The latter technique is inspired by a proof of Abiteboul, Herr, and Van den Bussche [3] separating the temporal query languages ETL from TS-FO. In particular, they show that every query in ETL on a special sort of databases can be evaluated by a communication protocol with a constant number of messages, whereas this is not the case for TS-FO. To simulate k-register DTWAs we need a more powerful protocol where the number of messages depends on the number of different data values in the input, but the idea is essentially the same. We sketch the argument below.

Theorem 3. [44]

1. *MSO* cannot define all properties computable by k-register DTWAs; and,*
2. *k-register DTWAs cannot compute all properties definable in FO*.*

Proof. (Sketch of (2)) We can already separate k-register DTWAs and FO* on strings as opposed to trees. In communication complexity the input string is divided in a pre-determined manner between two parties (generally referred to as I and II) that can send messages to each other according to a given protocol. A language is accepted by a protocol if for each string both parties can decide after execution of the protocol whether the string belongs to the language. Both parties have unlimited computation power on their part of the string. The protocol only restricts the way in which the parties communicate, typically by restricting the form and number of messages.

We consider strings of the form $f\#g$ where f and g encode sets of sets of **D**-symbols in a suitable way. For instance, the string $\$\sigma\sigma\$\sigma\$$ where the attribute values of the σ-symbols are a, b, and c, respectively, encodes the set $\{\{a, b\}, \{c\}\}$. The language

$$L := \{f\#g \mid f \text{ and } g \text{ represent the same set of sets}\}$$

is definable in FO*. To show that the language is not accepted by a k-register DTWA, we note that each such automaton working on strings of the form $f\#g$ can be simulated by a protocol in the following way: I is given f while II is given g. The first party simulates the automaton until this computation tries to cross the delimiter # to the right. At this point, it sends the present state q and the data values $d_1, \ldots, d_k$ currently in its registers. Hence, II gets full information about the configuration of the automaton (as the position of the symbol # is fixed). Then II sends in turn the current configuration to I. This process continues until one of the parties detects a final state. What kind of protocol can simulate such behavior? First, we need a message for every configuration. Suppose we restrict to at most N different data values in the strings $f\#g$. Then $M := |Q| \cdot N^k$ different messages are needed. Here, k is the number of registers and Q is the state set. Call a sequence of messages a dialogue. We only need to consider dialogues up to length M (as every message can only be sent once in every direction). Hence, there are only different **D**-symbols. The latter value is exponential in N. However, there are 2^{2^N} sets of sets of N different **D**-symbols. So for large enough N there must be different strings $f\#f$ and $g\#g$ with $f \neq g$ accepted by the protocol via the same dialogue. But, this means that $f\#g$ is also accepted. Hence, no such protocol can define L, which implies that no k-register DTWA accepts L. □

Actually, Neven, Schwentick, and Vianu [44] do not consider register automata on trees but on strings. However, as the separation results hold for strings they definitely hold for trees. In addition, the authors studied automata with various control mechanisms: non-determinism, alternation, one-way and two-way. In fact, the communication complexity technique sketched above can be extended to show that even *alternating* k-register automata cannot compute all FO*-definable properties. As an alternative to registers, the authors consider pebbles for dealing with data values. Every automaton is equipped with a finite number of pebbles whose use is restricted by a stack discipline. That is, pebble i can only be lifted when pebble $i + 1$ is not placed. Further, the automaton can test equality by comparing the attribute values of the pebbled symbols. It turns out that pebble automata behave much better than register automata: their expressiveness lies between FO* and MSO*, so they are neither too strong nor too weak.

6.4 Subcomputations and Relational Storage

In [39], we consider an extension of the register based model which is closer to XSLT. To be precise, DTWAs are extended in two ways: (i) registers can store

arbitrary relations over $\mathbf{D}$ (as opposed to single $\mathbf{D}$-values); (*ii*) subcomputations can be started. A transition rule is of the form $(\sigma, q, \xi) \to \alpha$, where $\sigma \in \Sigma$, $q \in Q$, and ξ is an FO formula over the relational storage. So, if for instance, the relational storage contains one set X_1 and ξ is the formula $\forall x \forall y (X_1(x) \wedge X_1(y) \to x \neq y)$, then the above transition will be applied if the current symbol is σ, the current state is q and X_1 contains at most one value. The right-hand side α can determine three kinds of actions:

1. a move: $\alpha = (q', d)$ where d is a direction and q' a state;
2. a change of the relational storage: $\alpha = (q', \psi, i)$ where q' is a state, ψ an FO formula over the relational storage and the attribute values of the current node, and i is the number of a register. The intended meaning is that the content of register i is replaced by the relation defined by ψ;
3. a subcomputation: $\alpha = (q', atp(\varphi(x, y), p), i)$, where q', p are states, i is the number of a register and $\varphi(x, y)$ is an FO($\exists$) formula over the tree (extended with some other predicates). The logic FO($\exists$) functions as an abstraction of XPath. Supppose the current node is u. Intuitively, register i is replaced by the result of $\mathrm{atp}(\varphi(x, y), p)$; the latter, starts ℓ subcomputations at the nodes $\{u_1, \ldots, u_\ell\} = \{v \mid t \models \varphi(u, v)\} \subseteq \mathrm{Dom}(t)$; these computations are started in state p and with the current relational store; when they end in a final state, the contents of the first register is returned; the content of register i is then the union of the results of all subcomputations. The main thread then resumes computation at u.

We do not define this model formally but only illustrate it by means of an example.

Example 6. Assume $\Sigma = \{\sigma, \delta\}$ and $A = \{a\}$. We define an automaton that accepts a tree if for every δ-labeled node all its leaf-descendants have the same a-attribute. By leaf-descendants we do not mean nodes labeled with $\triangle$ but the parents of those nodes. We define a 1-register automaton where the register X_1 is a set. Let $Q = \{q_0, q_1, q_2, q_3, q_4, q_F\}$, and $\tau_0(1) = \emptyset$. P consists of the following rules:

$$(\triangledown, q_0, \mathrm{true}) \to (q_1, \mathrm{atp}(\varphi_1, q_2), 1) \tag{1}$$
$$(\triangledown, q_1, \mathrm{true}) \to (q_F, \mathrm{stay}) \tag{2}$$
$$(\delta, q_2, \mathrm{true}) \to (q_3, \mathrm{atp}(\varphi_2, q_4), 1) \tag{3}$$
$$(\delta, q_3, \xi) \to (q_F, \mathrm{stay}) \tag{4}$$
$$(\delta, q_4, \mathrm{true}) \to (q_F, x = a, 1) \tag{5}$$
$$(\sigma, q_4, \mathrm{true}) \to (q_F, x = a, 1) \tag{6}$$

where $\varphi_1 \equiv x \prec y \wedge O_\delta(y)$, $\varphi_2 \equiv \exists y_1 (x \prec y \wedge E(y, y_1) \wedge O_\triangle(y_1)$, and $\xi \equiv \exists x X_1(x) \wedge \forall x \forall y (X_1(x) \wedge X_1(y) \to x \neq y)$.

The automaton works as follows: (1) a subcomputation is initiated that selects all δ-labeled descendants of the root; (2) when all subcomputations return, that is, state q_1 is reached, the tree is accepted; (3) every δ-labeled node selects

all leaves (recall that we work with delimited trees); (4) when the returned set is a singleton, the subcomputation accepts (otherwise, the subcomputation gets stuck and the main computation rejects); as (5) and (6) make sure that every leaf returns the value of its a attribute, the computations initiated by (3) accept in (4) iff every leaf has the same a-attribute. Note that $x = a$ is the formula that defines the set containing the value of the a-attribute of the current node. □

Allthough the present model seems quite powerful, the additions still do not suffice to capture FO*:

Theorem 4. *DTWA automata extended with look-ahead and relational storage do not capture FO*.*

Again, the proof of the latter theorem is based on communication complexity. However, the protocol is no longer memory-less, as is the case in the proof of Theorem 3: both parties need a stack to process incoming messages.

The weakness of register automata is that when they leave a node, they usually cannot relocate that node. In strong contrast, if there is a fixed attribute such that for every node the value of that attribute is unique among all nodes in a tree, that is, unique ids are available, then we show that various restrictions capture natural complexity classes:

Theorem 5. *In the presence of unique ids,*

1. *DTWAs extended with single-valued registers capture* LOGSPACE*;*
2. *DTWAs extended with single-valued registers and subcomputations capture* PTIME*;*
3. *DTWAs extended with relational storage capture* PSPACE*; and,*
4. *DTWAs extended with relational storage and subcomputations capture* EXPTIME*.*

Actually, the above characterizations are not obtained for the mentioned standard complexity classes but for a Turing Machine model directly operating on attributed trees. It can be shown that the latter and the standard model recognize the same class of tree languages. Although the proofs of the above results are combinations of known techniques in complexity, finite model theory, and formal languages, they provide a quite complete picture of the expressiveness of query languages based on tree-walking. The most surprising might be that DTWAs extended with single-valued registers and subcomputations, which is the abstraction of XSLT defined in [6], captures in fact precisely PTIME.

7 Discussion

We considered three automata models that regained interest by the advent of XML. Our main focus was on their connection with logic and on questions motivated by XML. We hope to have convinced the reader that XML poses new challenges on the automata and logic connection. In fact, the application of automata theory in XML research has only just started.

Indeed, although unranked tree automata have found already many applications, apart from the work of Brüggemann-Klein, Murata, and Wood [8], no systematic study has been undertaken. Especially the development of their algorithmic properties deserves much more attention.

Not much is known about tree-walking automata and not that many techniques are available. Solving the questions in Section 5.2 would be an interesting starting point. However, as these questions are open for quite some time, they appear to be difficult. The connection with TC-logic made in Theorem 2 learns that the TWA–NTA problem is an "easier" instance of the open question whether (full) unary TC logic and MSO on trees are equally expressive. It is even more of a mystery, whether, and in which way, the addition of pebbles to the formalism, as in [34], increases the expressiveness. The core of XPath, for instance, can easily be expressed by DTWAs with pebbles. It is unclear how many are needed (if they are needed at all).

Tree-walking automata with registers can serve as an abstraction of transformation languages like XSLT. Characterizing their expressiveness is, hence, meaningful. However, as an algorithmic toolbox, register automata are worthless as almost all decision problems are undecidable [44]. Nevertheless, in the context of streaming [51] or typechecking [5] in the presence of datavalues, it would be interesting to find the most expressive formalism for which emptiness would remain decidable. The inexpressibility results in [44] and [39] are obtained via communication complexity. As illustrated in the proof of Theorem 3, such a proof consists of two parts: (1) showing that your formalism can be simulated by a protocol; (2) no protocol can express your property. In both case, step (2) is rather straightforward, while (1) is the most involved one (especially in [39]). It would be interesting to come up with general criteria from which a simulation lemma can be derived automatically, rather than proving the simulation lemma by hand for every new model.

Acknowledgment

I thank Thomas Schwentick and Stijn Vansummeren for their comments on a previous version of this paper.

References

[1] S. Abiteboul. Semistructured data: from practice to theory. In *Proc. 16th IEEE Symposium on Logic in Computer Science (LICS 2001)*, pages 379–386, 2001. 3

[2] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999. 3

[3] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal connectives versus explicit timestamps to query temporal databases. *Journal of Computer and System Sciences*, 58(1):54–68, 1999. 19

[4] A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Inform. and Control*, 19:439–475, 1971. 12

[5] N. Alon, T. Milo, F. Neven, D. Suciu, and V. Vianu. XML with data values: Typechecking revisited. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 560–572, 2001. 16, 23

[6] G. J. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. *Information Systems*, 27(1):21–39, 2002. 12, 22

[7] R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000. 12

[8] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets: Version 1, April 3, 2001. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001. 7, 23

[9] A. Brüggemann-Klein and D. Wood. Caterpillars: A context specification technique. *Markup Languages*, 2(1):81–106, 2000. 11, 12

[10] J. Clark. XML Path Language (XPath). `http://www.w3.org/TR/xpath`. 13

[11] E. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970. 2

[12] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. 10

[13] M. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *Journal of Computer and System Sciences*, 3:272–288, 1998. 11

[14] World Wide Web Consortium. Extensible Markup Language (XML). `http://www.w3.org/XML/`. 2, 3

[15] World Wide Web Consortium. XML schema. `http://www.w3.org/XML/Schema`. 10

[16] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars: Definition, Systems and Bibliography*, volume 323 of *Lecture Notes in Computer Science*. Springer, 1988. 12

[17] J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970. 9

[18] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995. 6, 7, 15, 16

[19] J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhumäki, H. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999. 13, 14

[20] J. Engelfriet and H. J. Hoogeboom. Private communication. 2002. 16

[21] J. Engelfriet, H. J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999. 13, 14

[22] M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *Proc. 17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, 2002. 11

[23] Z. Fülöp and S. Maneth. Domains of partial attributed tree transducers. *Information Processing Letters*, 73(5–6):175–180, 2002. 14

[24] F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 1, pages 1–68. Springer, 1997. 2, 5

[25] G. Gottlob and C. Koch. Monadic datalog and the expresive power of languages for web information extraction. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 17–28. ACM Press, 2002. 11

[26] E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26–81, 1998. 17

[27] H. Hosoya and B. C. Pierce. Regular expression pattern matching for XML. In *Proceedings of 28th Symposium on Principles of Programming Languages (POPL 2001)*, pages 67–80. ACM Press, 2001. 11

[28] J. Hromkovic. *Communication Complexity and Parallel Computing*. Texts in Theoretical Computer Science - An EATCS Series. Springer-Verlag, 2000. 19

[29] N. Immerman. *Descriptive Complexity*. Springer, 1998. 15

[30] M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994. 17

[31] D. Lee, M. Mani, and M. Murata. Reasoning about XML schema languages using formal language theor. Technical report, IBM Almaden Research Center, 2000. Log# 95071. 11

[32] W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. Manuscript. 10, 11

[33] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 65–76, 2002. 11

[34] T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. In *Proceedings of the Nineteenth ACM Symposium on Principles of Database Systems*, pages 11–22. ACM Press, 2000. 12, 23

[35] M. Murata. Data model for document transformation and assembly. In E. V. Munson, K. Nicholas, and D. Wood, editors, *Proceedings of the workshop on Principles of Digital Document Processing*, volume 1481 of *Lecture Notes in Computer Science*, pages 140–152, 1998. 10

[36] M. Murata. Extended path expressions for xml. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 126–137. ACM Press, 2001. 11

[37] A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, pages 134–145. Springer, 1998. 11

[38] F. Neven. Extensions of attribute grammars for structured document queries. In R. Connor and A. Mendelzon, editors, *Research Issues in Structured and Semistructured Database Programming (DBPL'99)*, volume 1949 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2000. 11, 14

[39] F. Neven. On the power of walking for querying tree-structured data. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 77–84. ACM Press, 2002. 20, 23

[40] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proc. 19th Symposium on Principles of Database Systems (PODS 2000)*, pages 145–156, 2000. 11

[41] F. Neven and T. Schwentick. On the power of tree-walking automata. In U. Montanari, J. D. P. Rolim, and E. Welzl, editors, *International Colloquium on Automata, Languages and Programming (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2000. 13, 14

[42] F. Neven and T. Schwentick. Automata- and logic-based pattern languages for tree-structured data. Unpublished, 2001. 11

[43] F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002. 9, 11

[44] F. Neven, T. Schwentick, and V. Vianu. Towards regular languages over infinite alphabets. In J. Sgall, A. Pultr, and P. Kolman, editors, *Mathematical Foundations of Computer Science (MFCS 2001)*, volume 2136 of *Lecture Notes in Computer Science*, pages 560–572. Springer, 2001. 19, 20, 23

[45] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1), 2002. 11

[46] C. Pair and A. Quere. Définition et etude des bilangages réguliers. *Information and Control*, 13(6):565–593, 1968. 7

[47] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 35–46. ACM Press, 2001. 11

[48] A. Potthoff. *Logische Klassifizierung regulärer Baumsprachen*. Doctor's thesis, Institut für Informatik u. Prakt. Math., Universit"at Kiel, 1994. 16

[49] E. T. Ray. *Learning XML*. O'Reilly, 2001. 3

[50] T. Schwentick. On diving in trees. In *Proceedings of 25th Mathematical Foundations of Computer Science (MFCS 2000)*, pages 660–669, 2000. 11

[51] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proc. 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 53–64. ACM Press, 2002. 12, 23

[52] H. Seidl. Deciding equivalence of finite tree automata. *SIAM Journal on Computing*, 19(3):424–437, 1990. 10

[53] G. Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41(2–3) :305–318, 1985. 14

[54] D. Suciu. Typechecking for semistructured data. In *Proceedings of the 8th Workshop on Data Bases and Programming Languages (DBPL 2001)*, 2001. 9, 11

[55] M. Takahashi. Generalizations of regular sets and their application to a study of context-free languages. *Information and Control*, 27(1):1–36, 1975. 7

[56] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968. 9

[57] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7, pages 389–456. Springer, 1997. 5, 7

[58] J. Van den Bussche. Applications of Alfred Tarski's ideas in database theory. In L. Fribourg, editor, *Computer Science Logic (CSL 2001)*, volume 2142 of *Lecture Notes in Computer Science*, pages 20–37. Springer, 2001. 2

[59] M. Y. Vardi. Automata theory for database theoreticians. In *Proceedings of the Eighth ACM Symposium on Principles of Database Systems*, pages 83–92. ACM Press, 1989. 3

[60] V. Vianu. Databases and finite-model theory. In N. Immerman and Ph. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *AMS DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 97–148. American Mathematical Society, 1997. 2

[61] V. Vianu. A web odyssey: From Codd to XML. In *Proc. 20th Symposium on Principles of Database Systems (PODS 2001)*, pages 1–15, 2001. 3

μ-Calculus via Games (Extended Abstract)

Damian Niwiński*

Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland
niwinski@mimuw.edu.pl

1 Introduction

In this survey I would like to present some connections between the μ-calculus and games, more specifically games of possibly infinite duration played on labeled graphs. A fundamental connection was established by Emerson and Jutla [12] and subsequently developed by several authors [13, 36, 42, 1]. Essentially, the result is that any formula of the μ-calculus expresses the existence of a strategy in a certain game. The idea of such a correspondence can be traced back to Büchi and McNaughton who observed a similar property of monadic second order arithmetic (see [10]).

The contribution is mainly conceptual: infinite games (specifically, parity games) can help us to understand the μ-calculus formulas, and eventually design more efficient algorithms for related algorithmic problems as, e.g., model-checking.

For a reader not well familiar with the μ-calculus, infinite games provide a good introductory example since they exhibit fixed points that are not necessarily least fixed points, and the μ-calculus arose from an observation that some properties can be expressed by fixed points, but not the least fixed points. Indeed, the expressive power of the μ-calculus results from the alternation of mutually dependent least and greatest fixed point operators. However the same feature makes the μ-calculus difficult to understand by a human. In fact, in contrast to first-order or temporal logic, the μ-calculus did not emerge by a formalization of the natural language.

Some insight to the μ-calculus has been previously given by reducing it to Rabin automata [37] (which, in fact have the same expressive power [30]). The close connection between automata and games was known already to Büchi, and the alternating automata of Muller and Schupp [27] are already games in disguise. When compared to automata, the combinatorial structure of parity games appears simpler and, in some sense, more symmetric, achieving perhaps the best trade-off between expressive power and simplicity.

For a future development, one may expect that a game-theoretic perspective will bring some new ideas which, in particular, will be helpful for solving the

* Supported by Polish KBN grant No. 7 T11C 027 20

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 27–44, 2002.
© Springer-Verlag Berlin Heidelberg 2002

open problem of the complexity of the μ-calculus model-checking. Some steps in this direction have been already made [33, 18, 39, 32]. In a broader perspective, one can even think of a more general theory subsuming the μ-calculus. There seem to be analogies between fixed-point theorems of various kinds, but this direction remains to be investigated.

In this note I first recall the concept of general infinite games, and then the parity games relevant for the μ-calculus, drawing the reader's attention to the fixed point equations for the winning sets. The μ-calculus is subsequently presented in the style of [31, 3], that is as an algebraic system rather than a modal logic, which is slightly more general and allows a particularly simple presentation of the model-checking games, where the pairs of operations $\Diamond, \Box$ and $\vee, \wedge$ are treated in uniform way. Perhaps more importantly, the algebraic perspective reveals an algebraic aspect of the games: the set of winning strategies in a game designed for a term t turns out to be itself an interpretation of this term in a suitable poweralgebra of strategies. In the last section I briefly review the results on several aspects of complexity.

2 Parity Games

A general *infinite game* is a game with perfect information of possibly infinite duration played by two players, whom we call *Eva* and *Adam*, to hint a connection to existential and universal quantifiers, respectively.

The game is specified by a set of *positions* Pos partitioned into two disjoint subsets Pos_e and Pos_a of positions of Eva and Adam, respectively, the move relation $Mov \subseteq Pos \times Pos$, a ranking function $rank : Pos \to \omega$, and a set $Win_e \subseteq \omega^\omega$. (Throughout the paper ω denotes the set of natural numbers.) A play can be viewed as moving a token along the edges of the (directed) graph $\langle Pos, Mov\rangle$. A token can be moved from a given position v to any w such that $(v, w) \in Mov$. The actual choice of w is made by Eva or by Adam, depending on whether $v \in Pos_e$ or $v \in Pos_a$. If there is no such w, the player who has to play, looses; in this case we call v a deadlock position.

A result of a play (which we can identify with the play itself) is a sequence $v_0, v_1, \ldots$, such that, for each $i > 0$, $(v_{i-1}, v_i) \in Mov$, and this sequence is either infinite, or ends in a deadlock position. If the play is infinite, the win depends on the sequence of ranks: $rank(v_0)\, rank(v_1)\, rank(v_2) \ldots$. If this sequence belongs to Win_e, Eva wins the game, otherwise Adam is the winner.

A strategy for Eva is a mapping that tells Eva her next move in the play, depending on the current history. That is, a strategy maps $\alpha \in (Pos)^* Pos_e$ to $p \in Pos$ such that $(last(\alpha), p) \in Mov$.

If we fix an initial position v then we need not require a strategy be defined for all histories in $(Pos)^* Pos_e$ but only those that can be reached if Eva actually plays according to the strategy. Thus a strategy from v can be viewed as a tree. We find it convenient to present it as a labeled tree, i.e., a mapping s :

dom $s \to Pos$, where dom $s \subseteq \omega^*$ is a set of finite strings over natural numbers, say[1], closed under initial segments. The following properties are required of s.

- $s(\varepsilon) = v$ (since s is a strategy from v).
- If $s(\alpha) \in Pos_a$ then, for each p such that $(s(\alpha), p) \in Mov$, α has a successor α' in dom s labeled $s(\alpha') = p$ (and no other successors).
- If $s(\alpha) \in Pos_e$ then α has exactly one[2] successor α' in dom s such that $(s(\alpha), s(\alpha')) \in Mov$.

Clearly, the labeled paths in the tree s correspond to some possible plays of the game. (Note that any two different paths are labeled in different way.) Observe that by definition no finite play consistent with a path in s is lost by Eva. A strategy s is *winning* for Eva if, additionally, any infinite path $(\varepsilon = \alpha_0, \alpha_1, \alpha_2, \ldots)$ in dom s satisfies the winning condition, that is, $(rank(s(\alpha_0))\, rank(s(\alpha_1))\, rank(s(\alpha_2)) \ldots)$ belongs to Win_e. In other words, a strategy is winning if Eva wins any play played according to the strategy.

We say that v is a *winning position* of Eva if there exists a winning strategy for Eva from this position. (Note that v itself need not be a position of Eva.)

The concepts of strategy and winning position of Adam are defined analogously.

Let V_e and V_a be the sets of winning positions of Eva and Adam respectively. Clearly $V_e \cap V_a = \varnothing$. It would be plausible to think that $V_e \cup V_a = Pos$, but it is not always the case. If it happens, we say that a game is *determinate*. It follows from Axiom of Choice that there exist games that are not determinate [14] (see [28, 15]). But the realm of determinate games is large: the celebrated theorem by D. A. Martin says that any game with a Borel winning set $W_e \subseteq \omega^\omega$ is determinate [23] (see also [28]).

In reality, a strategy need not always depend on the whole history of the game played so far. An important case is if it depends only on the current position. We call a strategy of Eva s *positional* (or memoryless) if, whenever $s(\alpha) = s(\beta) \in Pos_e$, then we have also $s(\alpha') = s(\beta')$ for the successors of α and β, respectively.

We call a game *positionally determinate* if it is determinate, and, moreover, the winner has always a positional strategy.

Before we focus on parity games which are most relevant for the μ-calculus, we would like to consider a more general situation that allows to present the winning sets by fixed points. For $(a_n)_n$ and $(b_n)_n$ elements of ω^ω, we let $(a_n)_n \sim (b_n)_n$ if there exist i, j such that the sequences $(a_n)_{n \geq i}$ and $(b_n)_{n \geq j}$ are identical. We call a game *eventuality game* if the winning sets W_e and $W_a = \omega^\omega - W_e$ are saturated by relation $\sim$. (That is, if $(a_n)_n$ is winning for Eva then so are all the sequences equivalent to it; similarly for Adam.) Note that in an eventuality

[1] If the set Pos is uncountable, a larger cardinal should be used.

[2] The uniqueness requirement is not very essential; by relaxing it, we get non-deterministic strategies.

game the win of an infinite play is not altered if we change a finite segment of the play. In particular, if Eva has a winning strategy s from a position v then any position occurring as $s(\alpha)$ is also winning. Also, if at position v Eva has a strategy to reach some winning position w then the position v is also winning. Analogous properties hold for Adam by symmetry.

Now we observe that the set of winning positions of Eva in an eventuality game can be viewed as a fixed point. For, let us adopt a notation from modal logic

$$\Diamond X = \{v : \exists w,\ w \in X \ \wedge\ Mov(v,w)\}$$
$$\Box X = \{v : \forall w,\ Mov(v,w) \Rightarrow w \in X\}$$

and let for brevity $Pos_e = E$ and $Pos_a = A$. Then the set V_e of all positions winning for Eva satisfies the equation

$$X = (E \cap \Diamond X) \cup (A \cap \Box X). \tag{1}$$

Indeed, let $v \in V_e$. If v is a position of Eva then a winning strategy tells Eva a move $(v,w) \in Mov$ to a position which, by eventuality, is again winning. Hence $v \in \Diamond V_e$. A similar argument shows that if v is a position of Adam then $v \in \Box V_e$. Conversely, if $v \in (E \cap \Diamond V_e)$ then Eva has a move (v,w) to a position w from where she already has a winning strategy. By eventuality, Eva has also a winning strategy from v. A similar argument shows that $A \cap \Box V_e \subseteq V_e$.

In other words, the set V_e is a fixed point of an operator

$$Eva(X) = (E \cap \Diamond X) \cup (A \cap \Box X)\,.$$

Note that by the Knaster–Tarski Theorem, this operator has a least fixed point $\mu X.Eva(X)$ and a greatest fixed point $\nu X.Eva(X)$, and hence V_e lies somewhere between them.

The reader should be warned here that a (positional) strategy: *stay in* V_e is no guarantee to win.[3]

On the other hand, we can observe that Eva has always a positional strategy from the positions belonging to the least fixed point of $Eva(X)$. (But note that this set may be empty!) Indeed, we have

$$\mu X.Eva(X) = \bigcup_{\xi<\eta} Eva^{\xi}(\varnothing)$$

for some ordinal η, where $Eva^{\xi+1}(\varnothing) = Eva(Eva^{\xi}(\varnothing))$, and $Eva^{\gamma}(\varnothing) = \bigcup_{\xi<\gamma} Eva^{\xi}(\varnothing)$ for a limit ordinal γ. Now, let the *order* of a position $v \in \mu X.Eva(X)$ (not to be confused with $rank(v)$) be the least ordinal ξ such that

[3] It is easy to see that this strategy is winning only if the set of rank labellings of all the winning plays is closed in the standard (Baire) topology on ω^{ω}. Related questions about maximal nondeterministic strategies are investigated in [5].

$v \in Eva^{\xi}(\varnothing)$. Then it is easy to see that if v is a position of Eva then there exists a position w in $\mu X.Eva(X)$ of a strictly lower order such that $Mov(v, w)$. Similarly, if v is position of Adam then any w such that $Mov(v, w)$ has strictly lower order. Hence a strategy *decrease the order* is well defined and winning for Eva for all positions in $\mu X.Eva(X)$.

Now, in a similar mode, the set V_a of winning positions of Adam is a fixed point of an operator

$$Adam(X) = (A \cap \Diamond X) \cup (E \cap \Box X)\,.$$

It can be further observed that the complement of $Eva(X)$ satisfies

$$\overline{Eva(X)} = Adam(\overline{X})$$

and hence

$$X = Eva(X) \Longrightarrow \overline{X} = Adam(\overline{X})\,. \tag{2}$$

That is, if X is a fixed point of *Eva* then its complement is a fixed point of *Adam*.

Nevertheless, an eventuality game need not always be determinate. In fact, it is not difficult to adapt the Gale and Stewart diagonal construction [14] (see [15]) to obtain an indeterminate game over the binary tree[4].

But the situation is not always so puzzling. In particular, if $Win_e = \omega^\omega$ (that is, *any* infinite play is won by Eva) then it is easy to see that the winning set of Eva is the *greatest* fixed point of the equation (1) and a positional strategy "stay in V_e " is winning in this case. Indeed, if we have *any* L such that $L = (E \cap \Diamond L) \cup (A \cap \Box L)$ and a position v in L, then Eva can preserve this situation after her move, and also any move of Adam remains in L (or perhaps Adam is stuck). Hence $\nu X.Eva(X) \subseteq V_e$, but since V_e is a fixed point, we get equality.

By (2), the complement $\overline{V_e}$ is the *least* fixed point of $Adam(X)$ and by remark above, Adam has a winning strategy on this set. Hence we get determinacy in this very simple case.

The class of *parity games* is a subclass of general games obtained by assuming that the image $rank(Pos)$ of the function $rank : Pos \to \omega$ is *finite* and Win_e consists of those sequences $(a_0, a_1, \ldots)$ in $(rank(Pos))^\omega$ that the maximal a_i appearing infinitely often is even. That is, Eva wins an infinite play $(v_0, v_1, \ldots)$ if $\limsup_{i\to\infty} rank(v_i)$ is even, otherwise Adam is the winner. Clearly this condition is Borel and hence parity games are always determinate by Davis' theorem. But the parity games turn out to enjoy the positional determinacy: the winner has always a positional strategy which, moreover, need not depend on an initial position of the play.

[4] In the construction of the winning set W ([15], page 94), instead of requiring that $X_\alpha \notin \{Y_\beta : \beta < \alpha\}$, we require that $X_\alpha \notin \bigcup_{\beta<\alpha}[Y_\beta]_\sim$, and similarly for the Y_α's. There is no problem since the sets in question are countable.

That is, for any parity game we have $V_e \cup V_a = Pos$, and there exist two functions, say $s_e : V_e \to Pos$ and $s_a : V_a \to Pos$, which induce the winning strategies for Eva and Adam whenever some winning strategies exist.

The property of positional determinacy of parity games was discovered independently by Emerson and Jutla [12], Mostowski [25], and McNaughton [24] (the last with restriction to finite graphs). These authors were partly motivated by searching for a simpler proof of the Rabin Tree Theorem (see [38]). Proof via determinacy of certain games was given previously by Gurevich and Harrington [16] and, independently, by A. A. Muchnik [26]. McNaughton [24] analyzed a general class of winning conditions presented by a family of sets $\mathcal{F} \subseteq \mathcal{P}(rank(Pos))$ (assuming $rank(Pos)$ finite), where Eva wins an infinite play iff the set of ranks occurring infinitely often belongs to $\mathcal{F}$. Clearly a parity condition can be also presented in this way; McNaughton [24] (who presented this condition differently) showed that this is the only winning condition of the analyzed class that always guarantee the positional determinacy.

It might be an interesting direction of research to investigate what class of Borel conditions lead to the positional determinacy and, in particular, if it is somehow possible to extend the concept of parity condition to the case when $rank(Pos)$ is infinite.

What especially concerns us here is that the sets of winning positions in parity games can be presented and (at least in principle) computed as fixed points. Clearly, the parity games are eventuality games and, by considerations above the sets of wining positions of Eva and Adam are fixed points of the operators Eva and $Adam$, respectively. It turns out that these sets can be characterized by means of mutually dependent fixed point operators μ and ν. (This kind of expressions gives rise to a μ-calculus which we formalize in the next section.)

If we let E_i (resp. A_i) be the set of positions of Eva (resp. Adam) with $rank(v) = i$ then, assuming $rank(Pos) = \{0, 1, 2, \ldots, m\}$, the set of winning positions of Eva is precisely

$$V_e = \theta X_m. \ldots . \nu X_2. \mu X_1. \nu X_0. \bigcup_i (E_i \cap \Diamond X_i) \cup \bigcup_i (A_i \cap \Box X_i) \tag{3}$$

where θ stands for μ or ν depending on whether m is odd or even. The reader may easily verify that this set is a fixed point of the equation (1). The characterization (3) was established by Emerson and Jutla [12] and perhaps this result has inspired their currently common definition of parity games which we use also here. (The formulations in [25, 24] were different.) Another, very transparent proof of this property was given later by Walukiewicz [41], by a refinement of the argument of ordinal orders that we have used above for ω^ω games (which, by the way, are parity games with all ranks equal 0). Note that the equation (3) immediately implies the determinacy of parity games (although not yet the positional determinacy). Indeed, the winning set for Adam is characterized by a symmetric formula which turns out to define the complement of (3).

A proof of the positional determinacy of parity games based on the Boolean μ-calculus was given by Arnold [1] (see also [4]).

3 The μ-Calculus

Let *Sig* be a signature, i.e., a finite set of function symbols, each $f \in Sig$ given with arity $ar(f)$. In view of subsequent applications, it is convenient to extend *Sig* by a disjoint copy of dual symbols; let $Sig_\sim = Sig \cup \{\tilde{f} : f \in Sig\}$, with $ar(\tilde{f}) = ar(f)$. We will write f^* to denote f or $\tilde{f}$. We fix a countable set of variables *Var* and define the set of *fixed-point terms* (with dualities) as the least set of expressions $\mathcal{F}_{Sig_\sim}$ such that

- $Var \subseteq \mathcal{F}_{Sig_\sim}$,
- if $f^* \in Sig_\sim$ with $ar(f^*) = k$ and $t_1, \ldots, t_k \in \mathcal{F}_{Sig_\sim}$ then $f(t_1, \ldots, t_k) \in \mathcal{F}_{Sig_\sim}$,
- if $x \in Var$ and $t \in \mathcal{F}_{Sig_\sim}$ then $\mu x.t \in \mathcal{F}_{Sig_\sim}$ and $\nu x.t \in \mathcal{F}_{Sig_\sim}$.

Fixed-point terms can be interpreted by an abstract interpretation

$$\mathcal{D} = \langle\langle D, \leq_\mathcal{D}\rangle, \{f^{*\mathcal{D}} : f^* \in Sig_\sim\}\rangle$$

where $\langle\langle D, \leq_\mathcal{D}\rangle$ is a complete lattice and, for $f^* \in Sig$ of arity k, $f^{*\mathcal{D}}$ is a k-ary operation on D monotonic in all variables. Given a valuation of variables $v : Var \to D$, the interpretation proceeds by induction on the structure of term t:

$$\begin{aligned}
[\![x]\!]^v_\mathcal{D} &= v(x) \\
[\![f^*(t_1, \ldots, t_k)]\!]^v_\mathcal{D} &= f^{*\mathcal{D}}([\![t_1]\!]^v_\mathcal{D}, \ldots, [\![t_k]\!]^v_\mathcal{D}) \\
[\![\mu x.t]\!]^v_\mathcal{D} &= \bigwedge\{d : [\![t]\!]^{v[d/x]}_\mathcal{D} \leq_\mathcal{D} d\} \\
[\![\nu x.t]\!]^v_\mathcal{D} &= \bigwedge\{d : d \leq_\mathcal{D} [\![t]\!]^{v[d/x]}_\mathcal{D}\}
\end{aligned}$$

where $v[d/x]$ is a valuation that sends x to d and otherwise coincides with v.

The concept of a free variable is defined as usual (see, e.g., [4]). If a term t is *closed* (without free variables), we omit valuation and write simply $[\![t]\!]_\mathcal{D}$.

Note that we do not *a priori* require any connection between the interpretation of f and $\tilde{f}$, but in concrete realizations these operations often appear coupled by some duality connections.

A useful class of concrete interpretations arises by applying the powerset functor to model-theoretic structures.

A *semi-algebra* [31] over *Sig* can be presented by $\mathcal{B} = \langle B, \{f^\mathcal{B} \mid f \in Sig\}\rangle$, where B is the universe of $\mathcal{B}$ and, for each $f \in Sig$ of arity $ar(f) = k$, $f^\mathcal{B}$ is a relation over B of arity $k+1$, that is, $f^\mathcal{B} \subseteq B^{k+1}$. We call $f^\mathcal{B}$ a k-ary *basic operation* from B^k to B (although it need not be a function), and write $b \doteq f^\mathcal{B}(a_1, \ldots, a_k)$ to mean $(a_1, \ldots, a_k, b) \in f^\mathcal{B}$.

Any semi-algebra over *Sig* naturally induces an interpretation of the fixed-point terms in $\mathcal{F}_{Sig_\sim}$, which we call the *powerset algebra* of $\mathcal{B}$,

$$\wp\mathcal{B} = \langle\langle \mathcal{P}(B), \subseteq\rangle\{f^{\wp\mathcal{B}} \mid f \in Sig\} \cup \{\tilde{f}^{\wp\mathcal{B}} \mid f \in Sig\}\rangle.$$

Here $\langle \mathcal{P}(B), \subseteq \rangle$ is the complete lattice of all subsets of B ordered by the subset ordering and, for each $f \in Sig$ of arity k, and for $L_1, \ldots, L_k \subseteq B$,

$$f^{\wp\mathcal{B}}(L_1, \ldots, L_k) = \{b : \exists a_1 \in L_1, \ldots, \exists a_k \in L_k, f^{\mathcal{B}}(a_1, \ldots, a_k) \doteq b\},$$
$$\tilde{f}^{\wp\mathcal{B}}(L_1, \ldots, L_k) = \{b : \forall a_1, \ldots, a_k \in B,\ f^{\mathcal{B}}(a_1, \ldots, a_k) \doteq b \Rightarrow a_i \in L_i \text{ for some } i\}.$$

Note that we have

$$\tilde{f}^{\wp\mathcal{B}}(L_1, \ldots, L_k) = \overline{f^{\wp\mathcal{B}}(\overline{L_1}, \ldots, \overline{L_k})}$$

where $\overline{X} = B - X$.

An important example is that of the Kozen's *modal μ-calculus* ([20], see also a recent survey [8]). The syntax and semantics of this calculus are well known, so we recall them here only very briefly. In the usual setting, the formulas are constructed from variables $x \in Var$, propositions $p \in Prop$, and their duals $\bar{p}$, by means of logical connectives $\wedge$ and $\vee$, the fixed-point operators μx and νx, and the modalities $\langle a \rangle$ and $[a]$, for a ranging over some set of *actions* Act. Semantics is provided by a *Kripke structure*, or a labeled transition system, given by a set S of worlds (or states), and an interpretation of each proposition p by a set $[\![p]\!] \subseteq S$, and each action a by a relation $[\![a]\!] \subseteq S \times S$. In our framework, this structure can be presented as a semi-algebra, say $\mathcal{S}$, over signature $Sig = Prop \cup Act \cup \{eq\}$, where the propositions are considered of arity 0, the actions of arity 1, and eq is a special symbol of arity 2 (added in order to handle the logical connectives $\vee$ and $\wedge$). The universe of $\mathcal{S}$ is S. For $p \in Prop$ and $s \in S$, we let $p^{\mathcal{S}} \doteq s$ if and only if $s \in [\![p]\!]$. For $a \in Act$ and $s_1, s_2 \in S$, we let $s_1 \doteq a^{\mathcal{S}}(s_2)$ if and only if $(s_1, s_2) \in [\![a]\!]$. Finally, we let $eq^{\mathcal{S}}(s_1, s_2) \doteq s_3$ if and only if $s_1 = s_2 = s_3$. Then the classical semantics of the modal μ-calculus coincides with the one given by the following translation ρ of the formulas into fixed-point terms.

$$\begin{array}{ll}
\rho : x \mapsto x & \\
\quad : p \mapsto p & : \bar{p} \mapsto \tilde{p} \\
\quad : (\varphi \wedge \psi) \mapsto eq(\rho(\varphi), \rho(\psi)) & : (\varphi \vee \psi) \mapsto \tilde{eq}(\rho(\varphi), \rho(\psi)) \\
\quad : \langle a \rangle \varphi \mapsto a(\rho(\varphi)) & : [a]\varphi \mapsto \tilde{a}(\rho(\varphi)) \\
\quad : \mu x.\varphi \mapsto \mu x.\rho(\varphi) & : \nu x.\varphi \mapsto \nu x.\rho(\varphi)
\end{array}$$

The converse transformation is also easy to provide (see [4]).

Substantially, the framework of semi-algebras is not much more general than that of the modal μ-calculus. However, it is not more complex either, and we believe that an algebraic perspective may give some insight into the connections between various realizations of the μ-calculus.

4 Game Semantics for μ-Calculus

Let $\mathcal{B}$ be a semi-algebra over Sig and t a closed fixed-point term in $\mathcal{F}_{Sig_\sim}$. We are going to define a parity game such that winning this game by Eva from a position (somehow encoding an element) $b_{start} \in B$ is equivalent to $b_{start} \in [\![t]\!]_{\mathcal{D}}$.

To this end it will be convenient to make a *proviso* that variables occurring in t are indexed in some special way. Without loss of generality we may assume that each variable is bound only once and thus can be qualified as a μ- or ν-variable. We assume that each variable appears in the form $x_{i,j}$, where i is even for ν-variables and odd for μ-variables. Furthermore we require that the first indices decrease top–down. More precisely, we assume that if a variable $x_{k,\ell}$ appears *free* in the scope of $\theta x_{i,j}\ldots$ (where θ stands for μ or ν) then $k \geq i$. Note that the inequality must be strict for variables bound by different fixed-point operators. For example, a term $\mu x.\nu y.f(x, y, \mu z.\nu w.f(x, z, w))$ can be presented $\mu x_{11}.\nu x_{01}.f(x_{11}, x_{01}, \mu x_{12}.\nu x_{02}.f(x_{11}, x_{12}, x_{02}))$. (Note that, e.g., x_{11} appears free in the scope of μx_{12} but x_{01} does not.)

Let $Sub(t)$ be the set of all subterms of t.

We call a term of the form $f^*(\tau_1, \ldots, \tau_k)$ *guarded*. It will be further convenient to have an operation which reduces a term $\tau \in Sub(t)$ to its "outermost guarded" form, or, if it is not possible, to one of two special symbols $\bot$ or $\top$. It can be defined in the following way. Consider a graph of subterms of t where, for each subterm $\theta x.\sigma$, we draw an edge from $\theta x.\sigma$ to σ and also from x to $\theta x.\sigma$ if x is a subterm of t. Then, for each $\tau \in Sub(t)$, one of the following possibilities occurs.

- There is a path from τ to a unique guarded term σ; in this case we let $red(\tau) = \sigma$. (The path is trivial if τ itself is guarded.)
- The path originating in τ has a loop involving a unique edge of the form $(x, \theta x.\sigma)$; in this case we let $red(\tau) = \bot$ if $\theta = \mu$ and $red(\tau) = \top$ if $\theta = \nu$.

For example, $red(\mu x.\nu y.f(x, y)) = f(x, y)$ and $red(\mu x.\nu y.\mu z.y) = \top$.

We additionally fix two symbols *head* and *arg*.

We are ready to define our game

$$\mathcal{G}(\mathcal{B}, t) = \langle Pos_e, Pos_a, Mov, rank\rangle\,.$$

The set of positions $Pos = Pos_e \cup Pos_a$ will be a disjoint union of two sets:

- the set of *head positions* $B \times Sub(t) \times \{head\}$, and
- the set of *argument positions*, that is a subset of $B^* \times Sub(t) \times \{arg\}$ defined by
$$\{\langle\langle a_1, \ldots, a_k\rangle, \tau, arg\rangle : red(\tau) \text{ is } f^*(\ldots) \text{ and } ar(f^*) = k\ \}\,.$$

(We allow an empty sequence $\langle\rangle$ if $k = 0$.) A head position $\langle b, \tau, head\rangle$ belongs to Eva (i.e., to Pos_e) if $red(\tau)$ is of the form $f(\ldots)$ or $\bot$. The remaining head positions belong to Adam. For the argument positions, the qualification is dual. That is, an argument position $\langle\langle a_1, \ldots, a_k\rangle, \tau, arg\rangle$ belongs to Eva if and only if $red(\tau)$ is of the form $\tilde{f}(\ldots)$.

The *Mov* relation consists of edges defined as follows. Suppose $red(\tau) = f^*(\tau_1, \ldots, \tau_k)$ and $b \doteq f^{\mathcal{B}}(a_1, \ldots, a_k)$. Then

$$(\langle b, \tau, head\rangle, \langle\langle a_1, \ldots, a_k\rangle, \tau, arg\rangle) \in Mov$$
$$(\langle\langle a_1, \ldots, a_k\rangle, \tau, arg\rangle, \langle a_i, \tau_i, head\rangle) \in Mov$$

for $i = 1, \ldots, k$.

To complete the definition of the game, we need to define the function *rank*. For a head position $p = \langle b, x_{i,j}, head\rangle$ we let $rank(p) = i$, and we let $rank(p) = 0$ for all remaining positions.

A fundamental connection between the μ-calculus and parity games is the following.

Theorem 1 *Eva has a winning strategy in the game $\mathcal{G}(\mathcal{B}, t)$ from a position $\langle b_{start}, t, head\rangle$ if and only if $b_{start} \in [\![t]\!]_{\wp\mathcal{B}}$.*

This result was essentially established by Emerson and Jutla [12], although the games considered there characterize a more general problem of the *satisfiability* of the μ-calculus formulas rather than problem precisely corresponding to our computing $[\![t]\!]_{\wp\mathcal{B}}$, which is the celebrated *model checking* problem. Games for model-checking appeared subsequently in [13] in less direct way (via automata); an explicit construction has been given by C. Stirling [36]. The proof in [4] exploits a representation of the problem within the Boolean μ-calculus. It is possible to give a direct proof of Theorem 1, by induction on the structure of t.

We will give in the next section (an idea of) a yet another argument, emphasizing an algebraic structure of the powerset of strategies.

5 An Algebraic View of Strategies

Again, let $\mathcal{B}$ be a semi-algebra over *Sig*. We will now define the concept of pre-strategy which is, roughly speaking, a "hint" for Eva in the game of the previous section, although it does not depend on a particular fixed-point term. We first define a set $X_{\mathcal{B}}$, very similar to the set of positions in the game $\mathcal{G}(\mathcal{B}, t)$, but with terms in $Sub(t)$ replaced by symbols in $Sig_{\sim} \cup \{\top\}$. Let

$$X_{\mathcal{B}} = B \times (Sig_{\sim} \cup \{\top\}) \times \{head\} \cup \bigcup_{f^* \in Sig_{\sim}} \{\langle\langle a_1, \ldots, a_k\rangle, f^*, arg\rangle : a_1, \ldots, a_k \in B\}.$$

A tree $s : \mathrm{dom}\ s \to X_{\mathcal{B}}$ is a *pre-strategy* if the following conditions hold.

- $s(\varepsilon) = \langle b, f^*, head\rangle$, for some b, f^*.
- If $s(\alpha) = \langle b, f, head\rangle$ then there exist $a_1, \ldots, a_k \in B$ such that $b \doteq f^{\mathcal{B}}(a_1, \ldots, a_k)$ and α has a unique[5] successor α' in dom s labeled $s(\alpha') = \langle\langle a_1, \ldots, a_k\rangle, f, arg\rangle$.
- If $s(\alpha) = \langle b, \tilde{f}, head\rangle$ then, for all $a_1, \ldots, a_k \in B$ such that $b \doteq f^{\mathcal{B}}(a_1, \ldots, a_k)$, α has a successor α' in dom s labeled $s(\alpha') = \langle\langle a_1, \ldots, a_k\rangle, \tilde{f}, arg\rangle$.
- If $s(\alpha) = \langle\langle a_1, \ldots, a_k\rangle, f, arg\rangle$ then, for any $i = 1, \ldots, k$, α has a successor αi in dom s labeled $s(\alpha i) = \langle a_i, g_i^*, head\rangle$, for some $g_i^* \in Sig_{\sim} \cup \{\top\}$.

[5] Again, uniqueness is not essential, c.f. footnote 2.

- If $s(\alpha) = \langle\langle a_1, \ldots, a_k\rangle, \tilde{f}, arg\rangle$ then, for some i, α has a (unique) successor αi labeled $s(\alpha i) = \langle a_i, g_i^*, head\rangle$, for some $g_i^* \in Sig_\sim \cup \{\top\}$.
- If $s(\alpha) = \langle b, \top, head\rangle$, for some b, then α is a leaf.

We will write $s = s_b$ to indicate that $s(\varepsilon) = \langle b, f^*, head\rangle$, for some f^* (clearly, there can be many such s_b's).

Let $S_\mathcal{B}$ be the set of all pre-strategies. The powerset $\mathcal{P}(S_\mathcal{B})$ can be organized into an interpretation of the fixed-point terms in $\mathcal{F}_{Sig_\sim}$,

$$\wp\mathcal{S}_\mathcal{B} = \langle\langle \mathcal{P}(S_\mathcal{B}), \subseteq\rangle, \{f^{\wp\mathcal{S}_\mathcal{B}} : f \in Sig\} \cup \{\tilde{f}^{\wp\mathcal{S}_\mathcal{B}} : f \in Sig\}\rangle.$$

Here, for $L_1, \ldots, L_k \subseteq S_\mathcal{B}$,

- $f^{\wp\mathcal{S}_\mathcal{B}}(L_1, \ldots, L_k)$ consists of all pre-strategies s such that $s(\varepsilon) = \langle b, f, head\rangle$, for some b, the unique successor of ε (say, 1) is labeled $\langle\langle a_1, \ldots, a_k\rangle, f, arg\rangle$ (for some $a_1, \ldots, a_k \in B$ such that $b \doteq f^\mathcal{B}(a_1, \ldots, a_k)$), and the subtrees rooted in the k successors of 1, that are $11, \ldots, 1k$, are some pre-strategies $s_{a_1} \in L_1, \ldots, s_{a_k} \in L_k$, respectively. More formally,

$$\begin{aligned} \text{dom } s &= \{\varepsilon, 1\} \cup \bigcup_{i=1,\ldots,k} 1i \,\text{dom } s_{a_i} \\ s(\varepsilon) &= \langle b, f, head\rangle \\ s(1) &= \langle\langle a_1, \ldots, a_k\rangle, f, arg\rangle \\ s(1iw) &= s_{a_i}(w) \text{ for } w \in \text{dom } s_{a_i},\ i = 1, \ldots, k. \end{aligned}$$

- Similarly, $\tilde{f}^{\wp\mathcal{S}_\mathcal{B}}(L_1, \ldots, L_k)$ consists of all pre-strategies s such that $s(\varepsilon) = \langle b, \tilde{f}, head\rangle$, for some b, and, for each $a_1, \ldots, a_k$ satisfying $b \doteq f^\mathcal{B}(a_1, \ldots, a_k)$, there is a successor of ε, say ξ, labeled $s(\xi) = \langle\langle a_1, \ldots, a_k\rangle, f, arg\rangle$, and this ξ has a unique successor ξi, for some $i \in \{1, \ldots, k\}$, which is a root of some pre-strategy s_{a_i} in L_i.

Now, let π be a projection which maps a set of pre-strategies $L \subseteq S_\mathcal{B}$ to a subset $\{b : \exists s_b \in L\}$ of B. The interesting feature stemming directly from the definition of $\wp\mathcal{S}_\mathcal{B}$ is that this mapping is a homomorphism with respect to the interpretation of $f^* \in Sig_\sim$, that is,

$$\begin{aligned} \pi(f^{\wp\mathcal{S}_\mathcal{B}}(L_1, \ldots, L_k)) &= f^{\wp\mathcal{B}}(\pi(L_1), \ldots, \pi(L_k)) \\ \pi(\tilde{f}^{\wp\mathcal{S}_\mathcal{B}}(L_1, \ldots, L_k)) &= \tilde{f}^{\wp\mathcal{B}}(\pi(L_1), \ldots, \pi(L_k)). \end{aligned}$$

Moreover, π clearly preserves arbitrary joints and meets, i.e., for any $\mathcal{X} \subseteq \mathcal{P}(S_\mathcal{B})$,

$$\begin{aligned} \pi(\bigcup \mathcal{X}) &= \bigcup\{\pi(X) : X \in \mathcal{X}\} \\ \pi(\bigcap \mathcal{X}) &= \bigcap\{\pi(X) : X \in \mathcal{X}\}. \end{aligned}$$

These two conditions imply that π is a homomorphism of the fixed-point interpretations $\wp\mathcal{S}_\mathcal{B}$ and $\wp\mathcal{B}$, i.e., preserves all fixed-point terms (see, e.g., [3]).

Proposition 2 *For any closed term $t \in \mathcal{F}_{Sig_\sim}$, $\pi([\![t]\!]_{\wp\mathcal{S}_\mathcal{B}}) = [\![t]\!]_{\wp\mathcal{B}}$. In particular, $b \in [\![t]\!]_{\wp\mathcal{B}}$ if and only if there exists a pre-strategy s_b in $[\![t]\!]_{\wp\mathcal{S}_\mathcal{B}}$.*

We will infer Theorem 1 from the above proposition if we can show that a pre-strategy s_b in $[\![t]\!]_{\wp\mathcal{S}_\mathcal{B}}$ can be used by Eva as a strategy in the game $\mathcal{G}(\mathcal{B}, t)$, from the position $\langle b, t, head\rangle$. This can be achieved as follows.

Let us fix a closed term t. It is easy to see that any strategy s for Eva in the game $\mathcal{G}(\mathcal{B}, t)$ can be modified to a pre-strategy s' just by extracting the head function symbol of each current term. More specifically, dom s' = dom s, and s coincides with s' except for that whenever the second component of $s(v)$ is τ with $red(\tau) = f^*$, then the value of the analogous component of $s'(v)$ is f^*, and if $red(\tau) = \top$ then this value is $\top$. (The case of $red(\tau) = \bot$ will not occur since such a position would be losing for Eva, contradicting our definition of a strategy.) In this case we say that the pre-strategy s' *induces* the strategy s. Then Theorem 1 relies on the following more technical fact.

Proposition 3 *$[\![t]\!]_{\wp\mathcal{S}_\mathcal{B}}$ consists of those pre-strategies in $S_\mathcal{B}$ that induce winning strategies in the game $\mathcal{G}(\mathcal{B}, t)$.*

Sketch of proof. To use induction on structure of t, we need to extend the concept of our game to terms with free variables. For a term $t = t(z_1, \ldots, z_m)$ and a valuation $v : Var \to \mathcal{P}(B)$, the game $\mathcal{G}(\mathcal{B}, t, v)$ is defined similarly as for closed terms, but there are new head positions of the form $(b, z_i, head)$. Such a position is always deadlock and it belongs to Eva iff $b \notin v(z_i)$. (Thus Eva wins iff $b \in v(z_i)$.) The concept of inducing a strategy s by a pre-strategy s' is slightly modified: We now only require that dom $s \subseteq$ dom s' and the coincidence of s and s' holds for all $v \in$ dom s, except possibly if $s(v) = \langle b, z_i, head\rangle$, when we only require $s'(v) = \langle b, \varphi, head\rangle$, for some $\varphi \in Sig_\sim \cup \{\top\}$.

Now the claim which is to be established by induction, is the following.

> For any valuation $val : Var \to \mathcal{P}(S_\mathcal{B})$, $[\![t]\!]^{val}_{\wp\mathcal{S}_\mathcal{B}}$ consists of those pre-strategies in $S_\mathcal{B}$ that induce winning strategies in the game $\mathcal{G}(\mathcal{B}, t, \pi \circ val)$.

The proof can proceed along standard lines, similarly as, e.g., the proof of the equivalence of fixed-point terms and automata (see, e.g., [4]). Here we will only sketch the induction step for $t = \mu x.\tau(x, z_1, \ldots, z_m)$. Let K be the set of pre-strategies inducing the winning strategies in the game $\mathcal{G}(\mathcal{B}, t, \pi \circ val)$, let us call them *good* for t and val, for short. By Knaster–Tarski Theorem, it is enough to show two things.

(i) $[\![\tau]\!]^{val[K/x]}_{\wp\mathcal{S}_\mathcal{B}} \subseteq K$.

(ii) If $[\![\tau]\!]^{val[L/x]}_{\wp\mathcal{S}_\mathcal{B}} \subseteq L$ then $K \subseteq L$.

To show (i) it is enough to modify a pre-strategy good for τ and $val[K/x]$ to a pre-strategy good for $\mu x.\tau$ and val.

To show (ii), consider a strategy s good for $\mu x.\tau$ and val. Note that if the induced strategy never calls the variable x (as the value of the second component) then s is also good for τ and $val[L/x]$ and hence, by assumption, $s \in L$. In general

the set of occurrences of x need not be empty, however, it is well-founded by the parity condition (since x is the outermost μ-variable in t). Then the proof proceeds by the Principle of Tree Induction. □

6 Remarks on Complexity

Definability in arithmetic. Theorem 1 gives us an upper bound on the macro-scale of complexity, which can be relevant for infinite structures. If we can interpret our semi-algebra within the standard model of arithmetic (expressing the basic operations by first-order formulas) then the sets of integers definable by fixed-point terms remain on the level Δ^1_2, although by a direct translation we would obtain an arbitrary nesting of second-order quantifiers here. Indeed, a Δ^1_2 formula is sufficient to express the existence of a winning strategy in a parity game. However, this fact has been essentially known in the inductive definability theory, as well as the fact that the $\Sigma^1_1 \cup \Pi^1_1$ level can indeed be achieved (see Theorems 3.2 and 3.10 in [17]).

Relation to alternation depth. The *alternating depth* measures the number of alternations of μ and ν in a term. The hierarchy can be conveniently presented by organizing the set $\mathcal{F}_{Sig_\sim}$ in classes Σ_n, Π_n, where

- $\Sigma_0 = \Pi_0$ are terms without fixed-point operators,
- Σ_{n+1} is the closure of Π_n under composition and operator μ,
- Π_{n+1} is the closure of Σ_n under composition and operator ν.

The reader may notice a direct connection between the alternation depth and our special indexing of variables in the construction of games.

It is perhaps worth to note that an idea of the parity condition historically preceded parity games themselves (although it was not called that way). In particular, Wagner [40] showed that the sets of strings $L_n \subseteq \{0, 1, 2, \ldots, n\}^\omega$ satisfying the parity condition are hard for deterministic Rabin automata with n pairs (see [38]), and the author used [29] the sets M_n of binary trees with all branches in L_n as a first example of an interpretation of the μ-calculus with a strict alternation–depth hierarchy.

For the modal μ-calculus, the problem if there exist formulas whose alternation depth cannot be reduced was open for more than a decade.

It would nicely fit in our story if we could say that it was eventually solved by showing the hardness of the game formulas (3) (formalized in the signature of the modal μ-calculus), but it was not quite the case. Indeed, Bradfield [7] solved the problem in 1996 by a sophisticated reduction of some hierarchy in arithmetic. A closely related result was shown independently by Lenzi [21]. But the hardness of game formulas was eventually established by Bradfield [7]. Another proof of this result was given later by Arnold [2], by an ingenious application of the Banach Fixed-Point Theorem.

In any case, the game formulas (3) seem to be a simplest example of formulas with irreducible alternation depth.

Relation to model-checking problem. In our setting, the problem can be stated as follows.

Problem: Model checking.
Given: A finite semi-algebra $\mathcal{B}$, and a closed fixed-point term t.
Compute: $[\![t]\!]_{\wp\mathcal{B}}$.

By Theorem 1 and the positional determinacy of parity games, the above problem is polynomial time reducible to the following.

Problem: Parity game solving.
Given: A finite parity game $\langle Pos_a, Pos_e, Mov, rank\rangle$.
Compute: The partition of Pos to the sets of winning positions $V_e \cup V_a$.

The converse reduction is easy via formulas (3), so the problems are polynomial time equivalent. The game solving problem can be stated more ambitiously.

Problem: Constructive game solving.
Given: A finite parity game $\langle Pos_a, Pos_e, Mov, rank\rangle$.
Compute: (Some) positional winning strategies for both players.

This formulation can be viewed as an advantage gained from the game concept, since a strategy can give us an evidence why an element b is, or is not, in the interpretation $[\![t]\!]_{\wp\mathcal{B}}$.

Given a positional strategy for Eva (say, a "candidate" for a winning strategy), it is easy to check in polynomial time if this strategy is winning [12]. This observation places the decision problem: *is a position v winning for Eva ?* in the class NP. Since by determinacy any position is winning for one of the players, the problem is also in co-NP.

It is worth to note that no polynomial time algorithm is known to verify if a candidate partition $V'_e \cup V'_a$ is indeed a desired partition to the winning sets. Also, there is no evidence that knowing exactly all winning positions can help us to find a winning strategy. (As we have already noted, a simple strategy: *stay in V_e* is usually not winning.)

Deterministic algorithms that have been proposed for the game solving, in some sense carry the problem back to the μ-calculus, as they all boil down to computing some fixed points.

In fact, from an algorithmic point of view, the problem is better presented in a yet another polynomially equivalent form (see, e.g., [4]).

Problem: Evaluation of vectorial Boolean fixed-point terms.
Given:

$$\theta \begin{bmatrix} x_1^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix} . \cdots . \nu \begin{bmatrix} x_1^{(2)} \\ \vdots \\ x_n^{(2)} \end{bmatrix} . \mu \begin{bmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix} . \nu \begin{bmatrix} x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} . \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

Compute: The value of this fixed point in $\{0,1\}^n$.

Here $f_1, \ldots, f_n$ are monotonic Boolean functions in $n \cdot k$ variables $x_j^{(i)}$.

We can see that the above problem depends on two parameters: n and k which in the game formulation correspond to the number of positions and the number of different ranks, respectively. A standard method of computing the fixed point by subsequent iterations will evaluate this expression in space $\mathcal{O}(n \cdot k)$ and time $n^k \cdot n^{\mathcal{O}(1)}$ (where the polynomial factor $n^{\mathcal{O}(1)}$ is responsible for the computation of functions f_i). This can be improved to $(n/k+1)^k \cdot n^{\mathcal{O}(1)}$ [11] (which implies a bound of $2^n \cdot n^{\mathcal{O}(1)}$). A significant improvement in the iteration algorithm was made by Browne *et al.* [9], decreasing the exponent by half, which was subsequently still improved by Seidl [35], who achieved time $(n/k+1)^{k/2} \cdot n^{\mathcal{O}(1)}$. Both algorithms however use exponential space.

Jurdziński [19] proposed a new algorithm matching the best previous time bound and using only space $\mathcal{O}(n \cdot k)$. The algorithm relies on the game version of the problem and boils down to a computation of a single least fixed point (but in a larger state space).

Some authors approached the problem by making a connection with other games. Stirling [36] reports an observation by Mark Jerrum on a reduction of parity games to simple stochastic games. Puri [33] found a polynomial time reduction of parity games to mean payoff games. Jurdziński independently discovered a similar reduction and used it to show [18] that the problem *is a position v winning for Eva ?* is not only in NP, but even in UP (unambiguous nondeterministic polynomial time), and hence by symmetry also in UP $\cap$ co-UP.

Vöge and Jurdziński [39] embarked on a discretization of a Hoffman and Karp's *strategy–improvement algorithm* originally proposed for stochastic games (in 1966) and using the arithmetic of real numbers. The resulted discrete algorithm for parity games provides a deep insight to the problem, but the complexity of the algorithm is unknown.

Petersson and Vorobyov [32] used a reduction to discounted mean payoff games to obtain a randomized algorithm working in subexponential time in the case if k grows faster then $\sqrt{n}$.

One may expect that further investigations of the game solving problem will not only lead to better algorithms for the μ-calculus model-checking, but perhaps will also open new paths in the μ-calculus itself.

Acknowledgments

I wish to thank Marcin Jurdziński for enlightening remarks on game solving algorithms. Special thanks go to André Arnold for very helpful remarks on the preliminary version of this note. I also thank Igor Walukiewicz for last minute comments.

References

[1] A. Arnold. A selection property of the boolean μ-calculus and some of its applications. *RAIRO-Theoretical Informatics and Applications*, 31:371–384, 1997. 27, 32

[2] A. Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *RAIRO-Theoretical Informatics and Applications*, 33:329–339, 1999. 39

[3] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In M. Nivat and A. Podelski, editors, *Tree automata and Languages*, pages 159–188. Elsevier, 1992. 28, 37

[4] A. Arnold and D. Niwiński. Rudiments of μ-Calculus. Elsevier Science, Studies in Logic and the Foundations of Mathematics, 146, North–Holland, Amsterdam, 2001. 32, 33, 34, 36, 38, 40

[5] J. Bernet, D. Janin and I. Walukiewicz. Permissive strategies: from parity games to safety games. Theoretical Informatics and Applications (RAIRO), to appear. 30

[6] J. C. Bradfield. The modal mu-calculus alternation hierarchy is strict. *Theoretical Computer Science*, 195:133–153, 1997.

[7] J. C. Bradfield. Simplifying the modal mu-calculus alternation hierarchy. In M. Morvan, C. Meinel, and D. Krob, editors, *Proc. STACS '98*, pages 39–49. Lect. Notes Comput. Sci. 1373, 1998. 39

[8] J. Bradfield and C. Stirling. Modal logics and mu-calculi: an introduction. In J. Bergstra, A. Ponse and S. Smolka, editors, *Handbook of Process Algebra*, pages 293-332. Elsevier, North-Holland, 2001. 34

[9] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178:237–255, 1997. 41

[10] J. R. Büchi. Using determinacy to eliminate quantifiers. In M. Karpinski, editor, *Fundamentals of Computation Theory*, volume 56, pages 367–378. Lect. Notes Comput. Sci., 1977. 27

[11] R. Cleaveland, M. Klein and B. Steffen. Faster model checking for the modal mu-calculus. In G.v. Bochmann and D. Probst, editors, *Computer-Aided Verification (CAV '92)*, volume 663, pages 410–422, Lect. Notes Comput. Sci., 1992. 41

[12] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings 32th Annual IEEE Symp. on Foundations of Comput. Sci.*, pages 368–377. IEEE Computer Society Press, 1991. 27, 32, 36, 40

[13] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model-checking for fragments of the μ-calculus. In C. Courcoubetis, editor, *Computer Aided Verification*, pages 385–396. Lect. Notes Comput. Sci. 697, 1993. 27, 36

[14] D. Gale and F. M. Stewart. Infinite games with perfect information. Ann. of Math. Studies 28 (Contribution to the Theory of Games II), pages 245–266, Princeton, 1953. 29, 31

[15] Y. Gurevich. The logic in computer science column. *Bull. EATCS*, 38, pages 93–100, 1989. 29, 31

[16] Y. Gurevich and L. Harrington. Trees, automata and games. In *Proc. 14th ACM Symp. on the Theory of Computing*, pages 60–65, 1982. 32

[17] P. G. Hinman. Recursion-Theoretic Hierarchies. Perspective in Mathematical Logic, Springer-Verlag, 1978. 39

[18] M. Jurdziński. Deciding the winner in parity games is in UP $\cap$ co-UP. *Information Processing Letters*, 68:119–124, 1998. 28, 41

[19] M. Jurdziński. Small progress measures for solving parity games. In *Proc. 17th Symp. Theoretical Aspects of Computer Science*, volume 1770, pages 290–301. Lect. Notes Comput. Sci., 2000. 41

[20] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983. 34

[21] G. Lenzi. A hierarchy theorem for the mu-calculus. In F. Meyer auf der Heide and B. Monien, editors, *Proc. ICALP '96*, pages 87–109. Lect. Notes Comput. Sci. 1099, 1996. 39

[22] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[23] D. A. Martin. Borel determinacy. Annals of Math. 102, pages 363–371, 1975. 29

[24] R. McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65:149–184, 1993. 32

[25] A. W. Mostowski. Games with forbidden positions. Technical Report Technical Report 78, Instytut Matematyki, University of Gdansk, 1991. 32

[26] A. A. Muchnik. Games on infinite trees and automata with dead-ends: a new proof of the decidability of the monadic theory of two successors (in Russian). *Semiotics and Information*, 24:17–40, 1984. 32

[27] D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987. 27

[28] J. Mycielski. Games with perfect information. In R. J. Aumann and S. Hart, editors, *Handbook of Game Theory with Economic Applications*, volume 1, pages 41–70. North-Holland, 1992. 29

[29] D. Niwiński. On fixed point clones. In L. Kott, editor, *Proc. 13th ICALP*, pages 464–473. Lect. Notes Comput. Sci. 226, 1986. 39

[30] D. Niwiński. Fixed points vs. infinite generation. In *Proc. 3rd IEEE Symp. on Logic in Comput. Sci.*, pages 402–409, 1988. 27

[31] D. Niwiński. Fixed points characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, 189:1–69, 1997. 28, 33

[32] V. Petersson and S. Vorobyov. A randomized subexponential algorithm for parity games. *Nordic Journal of Computing*, 8:324–345, 2001. 28, 41

[33] A. Puri. Theory of Hybrid Systems and Discrete Event Systems. PhD thesis, College of Engineering, University of California, Berkeley, 1995. 28, 41

[34] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Soc*, 141:1–35, 1969.

[35] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59:303–308, 1996. 41

[36] C. Stirling. Local model checking games. In *Proc. Concur'95*, volume 962, pages 1-11, Lect. Notes Comput. Sci., 1995. 27, 36, 41

[37] R. S. Streett and E. A. Emerson. An automaton theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81:249–264, 1989. 27

[38] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997. 32, 39

[39] J. Vöge and M. Jurdziński. A discrete strategy improvement algorithm for solving parity games. In *Proc. CAV 2000*, volume 1855, pages 202-215, Lect. Notes Comput. Sci., 2000. 28, 41

[40] K. Wagner. Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen. *J. Inf. Process. Cybern. EIK*, 13:473–487, 1977. 39

[41] I. Walukiewicz. Monadic second-order logic on tree-like structures. In C. Puech and R. Reischuk, editors, *Proc. STACS '96*, pages 401–414. Lect. Notes Comput. Sci. 1046, 1996. 32

[42] I. Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001. 27

Bijections between Partitions by Two-Directional Rewriting Techniques

Max Kanovich

Department of Computer and Information Science, University of Pennsylvania
200 South 33rd Street, Philadelphia, PA 19104
kanovich@saul.cis.upenn.edu

Abstract. One basic activity in combinatorics is to establish combinatorial identities by so-called 'bijective proofs,' which consist in constructing explicit bijections between two types of the combinatorial objects under consideration. We show how such bijective proofs can be established, and how the bijections are computed by means of multiset rewriting, for a variety of combinatorial problems involving partitions. In particular, we fully characterizes all equinumerous partition ideals with 'disjointly supported' complements. As a corollary, a new proof, the 'bijective' one, is given for all equinumerous classes of the partition ideals of order 1 from the classical book "The Theory of Partitions" by G.Andrews. Establishing the required bijections involves novel two-directional reductions in the sense that forward and backward application of rewrite rules head for two different normal forms (representing the two combinatorial types). It is well-known that non-overlapping multiset rules are confluent. As for termination, it generally fails even for multiset rewriting systems that satisfy certain natural invariant balance conditions. The main technical development of the paper, which is important for establishing that the mapping yielding the combinatorial bijection is functional, is that the 'restricted' two-directional strong normalization holds for the multiset rewriting systems in question.

Keywords: multiset rewriting, Church-Rosser property, confluence, termination, strong normalization, combinatorics, integer partitions, partition identities.

1 Motivating Examples and Summary

The aim of the paper is to demonstrate the possibility of using rewriting techniques (*two-directional* in the sense that forward and backward application of rewrite rules head for two different normal forms) for the purpose of establishing *explicit bijections* between combinatorial objects of two different types (represented by the normal forms).

The starting point of one of the most intrigue combinatorics - the theory of integer partitions [1, 10, 11], is Euler's Partition Theorem: *Whatever positive integer n we take, the number of partitions of n into odd parts equals the number of partitions of n into distinct parts.*

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 44–58, 2002.
© Springer-Verlag Berlin Heidelberg 2002

A *partition* of n is a *multiset* M consisting of positive integers $m_1, m_2, \ldots, m_k$ whose sum is n. Each m_i is called a *part* of the partition. E.g., $\{3, 3, 1, 1, 1, 1\}$ is a partition of 10 with six odd parts, and $\{6, 4\}$ is a partition of 10 into distinct parts.

One approach to the proof of the combinatorial theorems like Euler's Partition Theorem is to count *separately* the number of partitions of n into odd parts and show that it is the same as the number of partitions of the n into distinct parts (by means of generating functions, see [1]).

Another approach to the problem is to find an explicit bijection h that associates with every partition into odd parts a partition into distinct parts, and vice versa. Garsia, Milne, Remmel, and Gordon have developed a unified method for constructing bijections for a large class of partition identities, based on a sophisticated 'ping-pong-ing' back and forth between two specific sets [4, 9, 5, 10]. In all these cases, the explicitness of the bijections produced by their complicated machinery remains debatable. Later, O'Hara [7] came up with a 'straightforward' algorithm, which works as follows. We are given two lists of pairwise disjoint multisets of positive integers $\mathcal{A} = A_1, A_2, \ldots, A_i, \ldots$, and $\mathcal{B} = B_1, B_2, \ldots, B_i, \ldots$, such that $\sum_{a \in A_i} a = \sum_{b \in B_i} b$ for all i. Now given the partition M which contains none of the B_i's, repeat the following until no A_i is contained in M: "Replace some A_i in M by its matched B_i."

"The key to this algorithm is that the mapping that it produces is independent of the order in which the A_i's are chosen - but this requires a good deal of effort to prove" (cited from Wilf's [11]).

Example 1. ([11]) Partitions into odd parts are multisets that do not contain any of the even parts collected in $\mathcal{B}_1 = \{2\}, \{4\}, \{6\}, \ldots$. Partitions into distinct parts are multisets that do not have any of the following list of 'repetition diseases': $\mathcal{A}_1 = \{1, 1\}, \{2, 2\}, \{3, 3\}, \ldots$. Each of the following replacement rules is intended to cure an $\mathcal{A}$-disease (but contaminates with a $\mathcal{B}$-disease):

$$\gamma_1 : \{1, 1\} \to \{2\},\ \gamma_2 : \{2, 2\} \to \{4\},\ \gamma_3 : \{3, 3\} \to \{6\},\ \gamma_4 : \{4, 4\} \to \{8\},\ \ldots \tag{1}$$

From the rewriting point of view. these rules: $\{i, i\} \to \{2i\}$, as well as the reversed rules: $\{2i\} \to \{i, i\}$, are non-overlapping multiset rules, and, hence, both system (1) and system of the reversed rules, say $(1)^{-1}$, are *confluent*. Since every rule from (1) contracts the number of parts in a given partition of n, each of the reduction sequences performed by (1), as well as the reduction sequences performed by $(1)^{-1}$, must *terminate* at most in n steps. Thus we obviously get a *total bijection* between "odd"-normal forms s and "distinct"--normal forms t: if the (unique!) (1)-normal form of s is t then the (unique!) $(1)^{-1}$-normal form of t must be s.
E.g., $\{3, 3, 1, 1, 1, 1\} \xrightarrow{\gamma_3} \{6, 1, 1, 1, 1\} \xrightarrow{\gamma_1} \{6, 2, 1, 1\} \xrightarrow{\gamma_1} \{6, 2, 2\} \xrightarrow{\gamma_2} \{6, 4\}$.

But this observation: "If both rewriting systems Γ and Γ^{-1} are confluent and terminating, then you get a total bijection between Γ-normal forms and Γ^{-1}-normal forms", does not apply to more general cases in which termination is more subtle: **it does not hold in general** but for normal forms.

We illustrate this with a 'Church-Rosser' translation (2) from the (unique) representation of n in base 4 into its (unique) binary form:

Example 2. The number of partitions of n, in which each part of the form 2^k, if any, is a power of 4, and furthermore, each power of 4 may occur at most thrice, is equal to the number of partitions of the n, in which each power of 2 may occur at most once.

The former partitions, say $\mathcal{B}$-normal forms, are multisets that do not contain any of the list $\mathcal{B}_2$: $\{2\}, \{1,1,1,1\}, \{8\}, \{4,4,4,4\}, \{32\}, \{16,16,16,16\}, \ldots$. The latter partitions, say $\mathcal{A}$-normal forms, are multisets that do not have any of the list $\mathcal{A}_2$: $\{1,1\}, \{2,2\}, \{4,4\}, \{8,8\}, \{16,16\}, \{32,32\}, \ldots$.

A total bijection between $\mathcal{B}$-normal forms and $\mathcal{A}$-normal forms is expected to be provided by the following 'well-balanced' rules:

$$\begin{array}{c} \gamma_0 : \{1,1\} \to \{2\}, \quad \gamma_1 : \{2,2\} \to \{1,1,1,1\}, \quad \gamma_2 : \{4,4\} \to \{8\}, \\ \gamma_3 : \{8,8\} \to \{4,4,4,4\}, \quad \gamma_4 : \{16,16\} \to \{32\}, \quad \ldots \end{array} \tag{2}$$

Both systems (2) and $(2)^{-1}$ are obviously confluent.

The 'balance conditions' - that $\sum_{a \in A_i} a = \sum_{b \in B_i} b$ for all i, provide that every reduction sequence that started from a partition of n may contain only partitions of the same n. Since the number of partitions of n is finite, the *termination problem* seems to be trivial, as well.

But each of the systems (2) and $(2)^{-1}$ is **not** even *weakly normalizing.*

E.g., the partition $\{2,2,2\}$ always generates an infinite 'loop':

$$\{2,2,2\} \xrightarrow{\gamma_1} \underline{\{2,1,1,1,1\}} \xrightarrow{\gamma_0} \{2,2,1,1\} \xrightarrow{\gamma_1} \{1,1,1,1,1,1\} \xrightarrow{\gamma_0} \underline{\{2,1,1,1,1\}} \xrightarrow{\cdots} \cdots.$$

In spite of this negative fact, we prove *general theorems* (Theorems 10 and 15), which guarantee that both systems (2) and $(2)^{-1}$ are *strongly normalizing* in the following restricted but desired sense:

> Any partition of n, which is the *correct* representation of the n in base 4, e.g., $\{4,4,4,1,1,1\}$, always converges to the *correct* binary form, e.g., $\{8,4,2,1\}$, and vice versa.

Generally, the classes $\mathcal{C}$ of partitions considered in the literature have the 'local' property that if M is a partition in $\mathcal{C}$ and one part is removed from M to form a new partition M', then M' is also in $\mathcal{C}$ [1]. Such a class $\mathcal{C}$ is called a *partition ideal* [1], or an *order ideal*, of the lattice $\mathcal{P}$ of finite multisets of positive integers, ordered by $\subseteq$. Andrews [1] has introduced a hierarchy of *partition ideals of order k*. The partitions mentioned above in Examples 1 and 2 form partition ideals of order 1 (the partition ideals of order 1 are just the *ideals* of $\mathcal{P}$).

In Corollary 28 we give a *new proof*, the 'bijective' one, for the Andrews' theorem [1, Theorem 8.4] that fully characterizes the equinumerous partition ideals of order 1. The bijective proof found here allows us to get a broader understanding of the essence of the Andrews' criterion.

Example 3. This extreme partition identity is taken from [9]: *The number of partitions of n such that their parts congruent to 1 or 4 mod 5 do not differ by 2 is equal to the number of partitions of the n such that their parts congruent to 1 or 4 mod 5 do not differ by 8.*

The former partitions, say $\mathcal{B}$-normal forms, are multisets that do not contain any of the list $\mathcal{B}_3$: $\{4,6\},\{9,11\},\{14,16\},\ldots$.

The latter partitions, say $\mathcal{A}$-normal forms, are multisets that do not have any of the list $\mathcal{A}_3$: $\{1,9\},\{6,14\},\{11,19\},\ldots$.
The 'bijective proof' is expected to be provided by the 'well-balanced' rules:

$$\gamma_1 : \{1,9\} \to \{4,6\},\ \ \gamma_2 : \{6,14\} \to \{9,11\},\ \ \gamma_3 : \{11,19\} \to \{14,16\},\ \ \ldots. \quad (3)$$

Being non-overlapping, both systems (3) and $(3)^{-1}$ are obviously *confluent*. As for *termination*, it is a good exercise to prove that both (3) and $(3)^{-1}$ are strongly normalizing, in spite of the "chaotic" reduction sequences like the following one (wherein one and the same $\mathcal{A}$-disease $\{6,14\}$ appears "persistently" two times):

$$\{1,\underline{6,14},19\} \xrightarrow{\gamma_2} \{1,9,11,19\} \xrightarrow{\gamma_1} \{4,6,11,19\} \xrightarrow{\gamma_3} \{4,\underline{6,14},16\} \xrightarrow{\gamma_2} \{4,9,11,16\}.$$

In terms of [1], we are dealing here with two partition ideals of order 3 and of order 9, resp., so they are not within reach of the Andrews' characterization [1, Theorem 8.4] of partition ideals of order 1.

We cover much more general cases with Theorem 22, which **fully characterizes** all equinumerous partition ideals with 'disjointly supported' complements.

The scheme is as follows.

Let Γ be a set of reduction rules: $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, $\ldots$, $\gamma_i : A_i \to B_i$, $\ldots$. The reversed rules $\gamma_i^{-1} : B_i \to A_i$ form Γ^{-1}. The $\mathcal{A}$-*normal forms* are the Γ-irreducible forms, i.e. the forms that do not contain any of the following list: $\mathcal{A} = A_1, A_2, \ldots, A_i, \ldots$, and the $\mathcal{B}$-*normal forms* are the Γ^{-1}-irreducible forms, i.e. the forms that do not contain any of the following list: $\mathcal{B} = B_1, B_2, \ldots, B_i, \ldots$.

An intended bijection h between $\mathcal{B}$-normal forms and $\mathcal{A}$-normal forms is defined as follows:

$$h(M) := \widetilde{M}, \quad \text{if } \widetilde{M} \text{ is an } \mathcal{A}\text{-normal form, and } M \text{ is } \Gamma\text{-reducible to } \widetilde{M}.$$

Definition 4. *We will say that Γ is* $\mathcal{B}$-terminating *if every sequence of Γ-reductions must terminate in a finite number of steps, whenever it started from a $\mathcal{B}$-normal form.*

Proposition 5. *Let both Γ and Γ^{-1} be confluent, Γ be $\mathcal{B}$-terminating, and Γ^{-1} be $\mathcal{A}$-terminating. Then the above h is a well-defined total bijection between $\mathcal{B}$-normal forms and $\mathcal{A}$-normal forms.*

Comment 1. As a matter of fact, we need a strong "stratified" version of the conclusion of Proposition 5 to supply a bijection between two sets of *partitions*

of a fixed n, and to show thereby that the two sets of partitions of the n are equinumerous:

For any fixed n, the above h should be a total bijection between $\mathcal{B}$-normal partitions of the n and $\mathcal{A}$-normal partitions of the same n.

The most natural way to guarantee this property is to invoke the following 'balance conditions' - that $\sum_{a \in A_i} a = \sum_{b \in B_i} b$, for all i. In this 'balance case' only partitions of one and the same n may appear within any sequence of Γ-reductions. □

Section 2 contains the main technical development of the paper.

It is well-known that the non-overlapping multiset rewrite systems Γ are confluent. We discover here a *new phenomenon* that just the same non-overlapping conditions provide, in addition, the *'restricted' loop-freeness*, namely, *there is no repetitions in the intermediate multisets within every sequence of Γ-reductions, whenever it started from a $\mathcal{B}$-normal form* (Theorem 10).

Now, by combining the general non-overlapping conditions with the natural 'balance conditions' caused by the nature of a particular combinatorics problem (see Comments 1 and 2), we obtain the desired *strong $\mathcal{B}$-normalization* (Theorems 15 and 17), which is important for establishing that the mapping h yielding the combinatorial bijection is functional (Theorems 16 and 18).
Our approach is easily generalized to other combinatorial objects.

Example 6. (cf. Euler's Partition Theorem) *The number of factorizations of n into nonsquare integer factors greater than* 1 *is equal to the number of factorizations of the n into distinct integer factors greater than* 1.

The former factorizations are multisets that do not have any of the list $\mathcal{B}_6 = \{4\}, \{9\}, \{16\}, \ldots$. The latter factorizations are multisets that do not have any of the list: $\mathcal{A}_6 = \{2,2\}, \{3,3\}, \{4,4\}, \ldots$. A total bijection between these two kinds of factorizations is provided by the reduction rules:

$$\gamma_1 : \{2,2\} \to \{4\},\ \gamma_2 : \{3,3\} \to \{9\},\ \gamma_3 : \{4,4\} \to \{16\},\ \gamma_4 : \{5,5\} \to \{25\},\ \ldots$$

Comment 2. In the case of factorizations, we need a strong "stratified" version of the conclusion of Proposition 5 to supply a bijection between two sets of factorizations of a fixed n, and to show thereby that the two sets of factorizations of the n are equinumerous:

For any fixed n, the above h from Proposition 5 should be a total bijection between $\mathcal{B}$-normal factorizations of the n and $\mathcal{A}$-normal factorizations of the same n.

The most natural way to guarantee this property is to invoke the following 'product balance conditions' - that $\prod_{a \in A_i} a = \prod_{b \in B_i} b$, for all i, which provides that only factorizations of one and the same n may appear within any sequence of Γ-reductions. □

Example 7. (cf. Euler's Partition Theorem) *The number of rooted forests of n vertices such that the trees are all different equals the number of rooted forests with no even tree [11].*

Furthermore, one can find a `polytime` *bijection h that associates with every rooted forest with no even tree a forest whose trees are all different, and vice versa.*

(According to [11], if we take two copies of the same rooted tree T and join their two roots together by a new edge, with the new root being one of the original roots, then the resulting tree is an *even tree* G_T.)

The desired bijection is provided by system Γ that consists of all rules γ_T of the form $\gamma_T : \{T, T\} \to \{G_T\}$ where T is a rooted tree.

2 Termination

Given a multiset M of the form $M = X \cup A \cup Y$, a rewriting rule $\gamma : A \to B$, replaces A with B, resulting in $M' = X \cup B \cup Y$. This fact is abbreviated as: $M \xrightarrow{\gamma} M'$, or $M' \xleftarrow{\gamma} M$, or $M' = \gamma(M)$. The latter functional notation is correct because of the following fundamental lemma:

Lemma 8. *Assume that $M \xrightarrow{\gamma} M'$ and $M \xrightarrow{\gamma} M''$. Then $M' = M''$.*

Proof. It follows from the fact that the multiset rewriting rules are rules modulo associativity and commutativity. □

Definition 9. *Let Γ be a set of rules: $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, ..., $\gamma_i : A_i \to B_i$, ..., and let $\mathcal{B} := B_1, B_2, B_3, \ldots, B_i, \ldots$.*
We say that Γ is $\mathcal{B}$-loop-free if whatever sequence of Γ-reductions

$$K_0 \xrightarrow{\alpha_1} K_1 \xrightarrow{\alpha_2} K_2 \xrightarrow{\alpha_3} K_3 \xrightarrow{\alpha_4} \cdots$$

that started from a $\mathcal{B}$-normal form K_0 we take, all these K_0, K_1, K_2, K_3,..., are different.

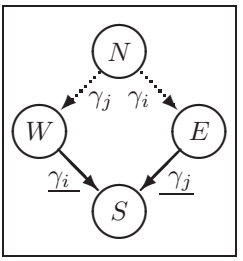
Theorem 10 ((Loop-Freeness)). *Let Γ be a set of multiset rewriting rules of the form $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, ..., $\gamma_i : A_i \to B_i$, ..., such that $B_i \cap B_j = \emptyset$ for any $i \neq j$. Then Γ is $\mathcal{B}$-loop-free.*

Comment 3. Thus the non-overlapping conditions - that $B_i \cap B_j = \emptyset$ for any $i \neq j$, provide that

(a) the reversed Γ^{-1} is confluent, and, in addition,
(b) Γ is $\mathcal{B}$-loop-free (Theorem 10).

The merely confluence of Γ^{-1} cannot guarantee the desired $\mathcal{B}$-loop-freeness.
Let Γ consist of two rules: $\gamma : \{2\} \to \{1, 1\}$, $\gamma' : \{1, 1\} \to \{1, 1\}$.
Here $\mathcal{A} = \{2\}$, $\{1, 1\}$, and $\mathcal{B} = \{1, 1\}$, $\{1, 1\}$.
Notwithstanding that both Γ and Γ^{-1} are confluent and have the 'balance property', Γ is not $\mathcal{B}$-loop-free: e.g., for the $\mathcal{B}$-normal $\{2, 1\}$:
$\{2, 1\} \xrightarrow{\gamma} \{1, 1, 1\} \xrightarrow{\gamma'} \{1, 1, 1\} \xrightarrow{\gamma'} \{1, 1, 1\} \xrightarrow{\gamma'} \cdots$ □

The proof of Theorem 10 is based on the following chain of lemmas.



Lemma 11 ((“strict” strong confluence)).
Let: $W \xrightarrow{\gamma_i} S \xleftarrow{\gamma_j} E$, *where rewriting rules* γ_i *and* γ_j *are of the form* $\gamma_i : A_i \to B_i$, *and* $\gamma_j : A_j \to B_j$, *with* B_i *and* B_j *being disjoint multisets. Then there is an* N *such that:* $W \xleftarrow{\gamma_j} N \xrightarrow{\gamma_i} E$.

Proof. The fact that the reversed rules $\gamma_i^{-1} : B_i \to A_i$ and $\gamma_j^{-1} : B_j \to A_j$ are applicable to S means that S, W, and E are to be of the form $S = X \cup B_i \cup B_j$, $W = X \cup A_i \cup B_j$, and $E = X \cup B_i \cup A_j$.
Taking N as: $X \cup A_i \cup A_j$, we provide the desired: $W \xleftarrow{\gamma_j} N \xrightarrow{\gamma_i} E$. □

Lemma 12. *Suppose that*

$$M_0 \xrightarrow{\alpha_1} M_1 \xrightarrow{\alpha_2} M_2 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_m} M_m \xrightarrow{\alpha_{m+1}} \cdots \xrightarrow{\alpha_k} M_k \xleftarrow{\beta} N_0, \tag{4}$$

and β *differs from each of the* $\alpha_1, \alpha_2, \ldots, \alpha_k$. *Then one can find* $N_1, N_2, \ldots, N_k$, *so that*

$$N_0 \xleftarrow{\alpha_k} N_1 \xleftarrow{\alpha_{k-1}} N_2 \xleftarrow{\alpha_{k-2}} \cdots \xleftarrow{\alpha_{k-m}} N_{m+1} \xleftarrow{\alpha_{k-m-1}} \cdots \xleftarrow{\alpha_1} N_k \xrightarrow{\beta} M_0. \tag{5}$$

Proof. By repeatedly applying Lemma 11, we construct the desired sequence $N_1, N_2, \ldots, N_k$. The case where $k = 3$ is shown in Figure 1. □

Lemma 13. *Let* $N_0 \xrightarrow{\beta} M_0$, *Then every loop* $M_0 \xrightarrow{\alpha_1\alpha_2\ldots\alpha_k} M_0$ *causes, for some permutation* π, *a loop of the form* $N_0 \xrightarrow{\alpha_{\pi(1)}\alpha_{\pi(2)}\ldots\alpha_{\pi(k)}} N_0$.

Proof. Assume a computation of the form (4), with $M_k = M_0$. There are two cases to be considered.

(1) Suppose that rule β differs from each of the rules $\alpha_1, \alpha_2, \ldots, \alpha_k$.
Then Lemma 12 provides us with a computation of the form (5).
According to Lemma 8, $N_k = \beta^{-1}(M_0) = \beta^{-1}(M_k) = N_0$.

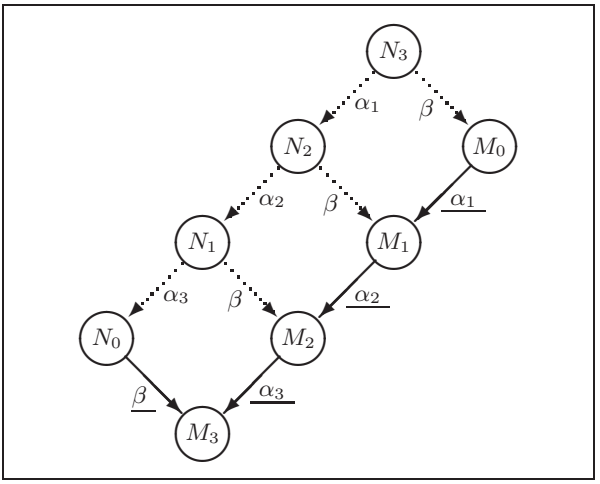
(2) Suppose for some m that $\beta = \alpha_m$, but β differs from each of the last rules $\alpha_{m+1}, \alpha_{m+2}, \ldots, \alpha_k$.
Since $M_m \xrightarrow{\alpha_{m+1}} M_{m+1} \xrightarrow{\alpha_{m+2}} M_{m+2} \cdots \xrightarrow{\alpha_k} M_k \xleftarrow{\beta} N_0$, Lemma 12 provides us with $N_1, N_2, \ldots, N_{k-m}$, such that

$$N_0 \xleftarrow{\alpha_k} N_1 \xleftarrow{\alpha_{k-1}} N_2 \xleftarrow{\alpha_{k-2}} \cdots \xleftarrow{\alpha_{m+1}} N_{k-m} \xrightarrow{\beta} M_m.$$

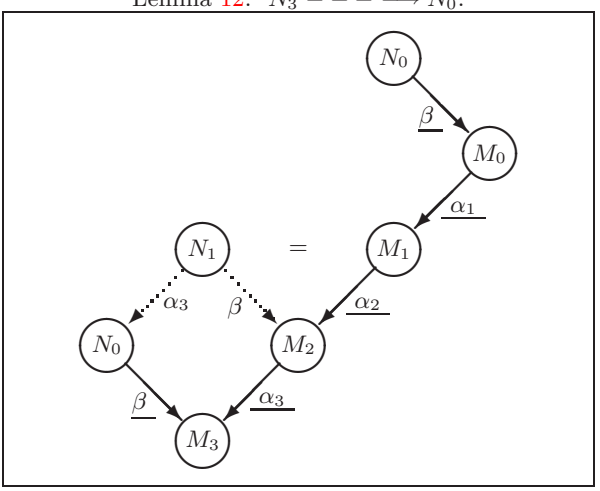
According to Lemma 8, $M_{m-1} = \alpha_m^{-1}(M_m) = \beta^{-1}(M_m) = N_{k-m}$.
Now we construct the desired loop as follows:

$$N_0 \xrightarrow{\beta} M_0 \xrightarrow{\alpha_1} M_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_{m-1}} M_{m-1} = N_{k-m} \xrightarrow{\alpha_{m+1}} \cdots \xrightarrow{\alpha_{k-1}} N_1 \xrightarrow{\alpha_k} N_0.$$

The case where $k = 3$, $m = 2$, is shown in Figure 1. □



Lemma 12: $N_3 \overset{\alpha_1\alpha_2\alpha_3}{- - - \longrightarrow} N_0$.



Lemma 13: $N_0 \overset{\alpha_2\alpha_1\alpha_3}{- - - \longrightarrow} N_0$ where $\beta = \alpha_2$.

Fig. 1. A lower loop: $M_0 \Longrightarrow M_0$, causes an upper loop: $N_0 \Longrightarrow N_0$

Lemma 14. *Let N_0 be Γ-reducible to M_0. Then every loop*

$$M_0 \overset{\alpha_1\alpha_2\ldots\alpha_k}{- - - - - - \longrightarrow} M_0$$

causes, for some permutation π, a loop of the form

$$N_0 \overset{\alpha_{\pi(1)}\alpha_{\pi(2)}\ldots\alpha_{\pi(k)}}{- - - - - - \longrightarrow} N_0.$$

Proof. By repeatedly applying Lemma 13. □

Proof of Theorem 10. Let $K_0 \stackrel{\beta_1}{\to} K_1 \stackrel{\beta_2}{\to} K_2 \stackrel{\beta_3}{\to} K_3 \stackrel{\beta_4}{\to} \cdots$ be a sequence of Γ-reductions, and K_0 be a $\mathcal{B}$-normal form.

According to Lemma 14, had any repetition happened within this sequence of the multisets, it would have produced a non-trivial loop of the form

$$K_0 \stackrel{\alpha_1\alpha_2\ldots\alpha_k}{- - - \longrightarrow} K_0,$$

communicating thereby some $B_{k'}$ into K_0 by means of rule α_k, which contradicts to the fact that K_0 has no $\mathcal{B}$-diseases. □

Theorem 15 ((Strong Normalization)). *Let Γ be a set of multiset rewriting rules of the form $\gamma_1 : A_1 \to B_1,\ \gamma_2 : A_2 \to B_2, \ldots,\ \gamma_i : A_i \to B_i, \ldots,$ such that all B_i's are multisets of positive integers. If Γ is $\mathcal{B}$-loop-free, and, in addition, the 'balance conditions': $\sum_{a\in A_i} a = \sum_{b\in B_i} b$, hold for all i, then Γ is* $\mathcal{B}$-terminating.

Proof. Let $K_0 \stackrel{\beta_1}{\to} K_1 \stackrel{\beta_2}{\to} K_2 \stackrel{\beta_3}{\to} K_3 \stackrel{\beta_4}{\to} \cdots$ be a sequence of Γ-reductions, and K_0 be a $\mathcal{B}$-normal form. Multiset K_0 can be conceived of as a partition of n, where $n = \sum_{k\in K_0}$. The 'balance property' yields that each of the K_1, K_2, K_3, ... is a partition of one and the same n. Since Γ is $\mathcal{B}$-loop-free, all these K_1, K_2, K_3, ... must be different. Hence, the length of the sequence cannot exceed $p(n)$, the number of distinct partitions of n. □

Comment 4. If we allow 0's, "$\mathcal{B}$-loop-freeness + balance" not necessarily implies $\mathcal{B}$-termination. E.g., let Γ consist of one rule $\gamma : \{0\} \to \{0,0\}$. Here $\mathcal{A} = \{0\}$, and $\mathcal{B} = \{0,0\}$. Notwithstanding that Γ is $\mathcal{B}$-loop-free and has the 'balance property', an infinite sequence of reductions happens even if we started with the $\mathcal{B}$-normal form $\{0,1\}$: $\{0,1\} \xrightarrow{\gamma} \{0,0,1\} \xrightarrow{\gamma} \{0,0,0,1\} \xrightarrow{\gamma} \{0,0,0,0,1\} \xrightarrow{\gamma} \ldots$

Theorem 16. *Let Γ be a set of multiset rewriting rules of the form $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2, \ldots,\ \gamma_i : A_i \to B_i, \ldots,$ such that for any $i \neq j$, A_i and A_j are disjoint multisets of positive integers, and B_i and B_j are disjoint multisets of positive integers, and, in addition, the 'balance conditions': $\sum_{a\in A_i} a = \sum_{b\in B_i} b$, hold for all i.*
Define h as follows:

$$h(M) := \widetilde{M}, \quad \text{if } \widetilde{M} \text{ is an } \mathcal{A}\text{-normal form, and } M \text{ is } \Gamma\text{-reducible to } \widetilde{M}.$$

Then for every n, the h is a well-defined total bijection between $\mathcal{B}$-normal partitions of the n and $\mathcal{A}$-normal partitions of the same n.

Proof. Both Γ and Γ^{-1} are obviously confluent. By Theorems 10 and 15, Γ is $\mathcal{B}$-terminating, and Γ^{-1} is $\mathcal{A}$-terminating. See Proposition 5 and Comment 1 for the further details. □

Theorem 17 ((Strong Normalization)). *Let Γ be a set of multiset rewriting rules of the form $\gamma_1 : A_1 \to B_1,\ \gamma_2 : A_2 \to B_2, \ldots,\ \gamma_i : A_i \to B_i, \ldots,$ such that all B_i's are multisets of integers ≥ 2. If Γ is $\mathcal{B}$-loop-free, and, in addition, the 'balance conditions': $\prod_{a\in A_i} a = \prod_{b\in B_i} b$, hold for all i, then Γ is* $\mathcal{B}$-terminating.

Proof. Let $K_0 \overset{\beta_1}{\to} K_1 \overset{\beta_2}{\to} K_2 \overset{\beta_3}{\to} K_3 \overset{\beta_4}{\to} \cdots$ be a sequence of Γ-reductions, and K_0 be a $\mathcal{B}$-normal form. Multiset K_0 can be conceived of as a factorization of n, where $n = \prod_{k \in K_0}$. The 'balance property' yields that each of the K_1, K_2, K_3, ... is a factorization of one and the same n. Since Γ is $\mathcal{B}$-loop-free, all these K_1, K_2, K_3, ... must be different. Hence, the length of the sequence cannot exceed $\widehat{p}(n)$, the number of factorizations of n into integers ≥ 2. □

Theorem 18. *Let Γ be a set of multiset rewriting rules of the form $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, ..., $\gamma_i : A_i \to B_i$, ..., such that for any $i \neq j$, A_i and A_j are disjoint multisets of integers ≥ 2, and B_i and B_j are disjoint multisets of integers ≥ 2, and, in addition, the 'balance conditions': $\prod_{a \in A_i} a = \prod_{b \in B_i} b$, hold for all i.*
Define h as follows:

$$h(M) := \widetilde{M}, \quad \text{if } \widetilde{M} \text{ is an } \mathcal{A}\text{-normal form, and } M \text{ is } \Gamma\text{-reducible to } \widetilde{M}.$$

Then for every n, the h is a well-defined total bijection between $\mathcal{B}$-normal factorizations of the n and $\mathcal{A}$-normal factorizations of the same n.

Proof. Both Γ and Γ^{-1} are obviously confluent. By Theorems 10 and 17, Γ is $\mathcal{B}$-terminating, and Γ^{-1} is $\mathcal{A}$-terminating. See Proposition 5 and Comment 2 for the further details. □

3 Partition Ideals

Definition 19. *[1] Two classes of partitions $\mathcal{C}_1$ and $\mathcal{C}_2$ are* equivalent*: $\mathcal{C}_1 \sim \mathcal{C}_2$, if $p(\mathcal{C}_1, n) = p(\mathcal{C}_2, n)$ for all n. Here $p(\mathcal{C}, n)$ denotes the number of partitions of the n that belong to a given class $\mathcal{C}$.*

Generally, the classes $\mathcal{C}$ of partitions considered in the literature have the 'local' property that if M is a partition in $\mathcal{C}$ and one part is removed from M to form a new partition M', then M' is also in $\mathcal{C}$ [1].

Let $\mathcal{P}$ be the lattice $\mathcal{P}$ of finite multisets of positive integers, ordered by $\subseteq$.

We let a sort of the "norm" by the following: $\|M\| := \sum_{m \in M} m$.

Definition 20. *A class $\mathcal{C} \subseteq \mathcal{P}$ is an* order ideal*, or a* partition ideal *in terms of [1], if for any M and M' from $\mathcal{P}$ such that $M' \subseteq M \in \mathcal{C}$, necessarily $M' \in \mathcal{C}$. Dually, a class $\mathcal{F} \subseteq \mathcal{P}$ is an* order filter *if $M' \in \mathcal{F}$, whenever $M \in \mathcal{F}$ and $M \subseteq M'$. It is readily seen that $\mathcal{C}$ is a partition ideal if and only if its complement $\overline{\mathcal{C}}$ is an order filter.*

As for the fundamental problem stated in [1]:

Fully characterize the equivalence classes of partition ideals.

we give a full characterization for a wide class of partition ideals having certain similarities in their lattice structure.

Definition 21. *M is* minimal *in an order filter $\mathcal{F} \subseteq \mathcal{P}$, if $M \in \mathcal{F}$ and no $M' \in \mathcal{F}$ for a proper submultiset M' of M. The* support *of $\mathcal{F}$, i.e. the set of all its minimal elements, is denoted by $\mu_{\mathcal{F}}$. We say that the support $\mu_{\mathcal{F}}$ is* disjoint, *if $M \cap M' = \emptyset$ for any distinct M and M' in $\mu_{\mathcal{F}}$.*

Theorem 22. *Let $\mathcal{C}$ and $\mathcal{C}'$ be partition ideals such that the support of the order filter $\overline{\mathcal{C}}$,*

$$\mu_{\overline{\mathcal{C}}} = \{A_1, A_2, A_3, \ldots\},$$

is disjoint, and the support of the order filter $\overline{\mathcal{C}'}$,

$$\mu_{\overline{\mathcal{C}'}} = \{B_1, B_2, B_3, \ldots\},$$

is disjoint.
Then $\mathcal{C} \sim \mathcal{C}'$ if and only if the two sequences of integers $\|A_1\|$, $\|A_2\|$, $\|A_3\|$, ... and $\|B_1\|$, $\|B_2\|$, $\|B_3\|$, ... are merely reorderings of each other.

Proof.

(a) Let the two sequences of $\|A_1\|$, $\|A_2\|$, $\|A_3\|$, ... and $\|B_1\|$, $\|B_2\|$, $\|B_3\|$, ... be merely reorderings of each other.
Without loss of generality, we can assume that the two lists $\mathcal{A} := A_1, A_2, \ldots, A_i, \ldots$, and $\mathcal{B} := B_1, B_2, \ldots, B_i, \ldots$, have been already sorted so that $\|A_i\| = \|B_i\|$ for all i.
Letting Γ be a set of reduction rules: $\gamma_1 : A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, ..., $\gamma_i : A_i \to B_i$, ..., by Theorem 16 we construct a function h such that, for every n, the h is a total bijection between $\mathcal{B}$-normal partitions of the n and $\mathcal{A}$-normal partitions of the same n.
Taking into account that:

- $M \in \mathcal{C} \iff M \notin \overline{\mathcal{C}} \iff A_i \not\subseteq M$ for every $i \iff M$ is a $\mathcal{A}$-normal form, and
- $M \in \mathcal{C}' \iff M \notin \overline{\mathcal{C}'} \iff B_i \not\subseteq M$ for every $i \iff M$ is a $\mathcal{B}$-normal form,

we can conclude that $\mathcal{C} \sim \mathcal{C}'$.

(b) Suppose that $\mathcal{C} \sim \mathcal{C}'$, and thereby $\overline{\mathcal{C}} \sim \overline{\mathcal{C}'}$.
Let the two lists $\mathcal{A} := A_1, A_2, \ldots, A_i, \ldots$, and $\mathcal{B} := B_1, B_2, \ldots, B_i, \ldots$, be sorted in ascending order of the integers $\|A_i\|$'s and $\|B_i\|$'s, and a_n be the number of A_i's such that $\|A_i\| = n$, and b_n be the number of B_i's such that $\|B_i\| = n$.
Assume that k is the least positive integer such that $a_k \neq b_k$.
Take i_0 to be the largest index such that $\|A_{i_0}\| = \|B_{i_0}\| < k$, and define $\mathcal{F}_k$ to be the order filter generated by $\mathcal{A}_k := A_1, A_2, \ldots, A_{i_0}$, and $\mathcal{F}'_k$ to be the order filter generated by $\mathcal{B}_k := B_1, B_2, \ldots, B_{i_0}$.

Lemma 23.

$$\{M \in \overline{\mathcal{C}} \mid \|M\| = k\} \;=\; \{M \in \mathcal{F}_k \mid \|M\| = k\} \;\cup\; \{A_i \mid \|A_i\| = k\},$$

and the above union is disjoint.

Proof.
(i) If M belongs to the union then $A_i \subseteq M$ for some i, and, hence, $M \in \overline{\mathcal{C}}$.
(ii) Suppose that $\|M\| = k$, and $M \in \overline{\mathcal{C}}$. Then $A_i \subseteq M$ for some i.
If $i \leq i_0$ then $M \in \mathcal{F}_k$. For $i > i_0$, we have $\|A_i\| \geq k$, which together with $A_i \subseteq M$ and $\|M\| = k$ yields that $M = A_i$.
(iii) The above union is disjoint: Let both M be some A_i with $\|A_i\| = k$, and M belong to $\mathcal{F}_k$. The effect is that there is an A_j such that $\|A_j\| < k$, and $A_j \subseteq M = A_i$, which contradicts to the minimality of A_i.
Lemma 23 shows that $p(\overline{\mathcal{C}}, k) = p(\mathcal{F}_k, k) + a_k$, and $a_k = p(\overline{\mathcal{C}}, k) - p(\mathcal{F}_k, k)$. Similarly, $b_k = p(\overline{\mathcal{C}'}, k) - p(\mathcal{F}'_k, k)$.
According to the previous item (a), $\overline{\mathcal{F}_k} \sim \overline{\mathcal{F}'_k}$, and, hence, $\mathcal{F}_k \sim \mathcal{F}'_k$.
Therefore, $a_k = p(\overline{\mathcal{C}}, k) - p(\mathcal{F}_k, k) = p(\overline{\mathcal{C}'}, k) - p(\mathcal{F}'_k, k) = b_k$, which is a contradiction.
Thus, $a_n = b_n$, for all n, and thereby $\|A_i\| = \|B_i\|$ for all i. □

Theorem 22 provides a new proof, the 'bijective' one, for the Andrews' theorem [1, Theorem 8.4] that fully characterizes the equivalent partition ideals of order 1.

Let us recall the results from [1] we are dealing with.

Definition 24. *[1] Any partition M is represented as a sequence $\{f_i\}_{i=1}^{\infty}$ where f_i is the number of occurrences of i in M.*

Definition 25. *[1] A partition ideal $\mathcal{C}$ has order k if k is the least positive integer such that whenever $\{f_i\}_{i=1}^{\infty} \notin \mathcal{C}$, then there exists m such that $\{f'_i\}_{i=1}^{\infty} \notin \mathcal{C}$ where*

$$f'_i = \begin{cases} f_i, \text{ for } i = m, m+1, \ldots, m+k-1, \\ 0, \quad \text{otherwise.} \end{cases}$$

E.g., Example 3 gives two partition ideals of order 3 and of order 9, respectively.

Proposition 26. [1] *A partition ideal $\mathcal{C}$ has order 1 if and only if*

$$\mathcal{C} = \{ \ \{f_i\}_{i=1}^{\infty} \mid f_i \leq d_i, \text{ for all } i \ \}$$

where $d_j := \sup_{\{f_j\} \in \mathcal{C}} f_j$.

Theorem 27. [1, Theorem 8.4.] *Let $\mathcal{C}$ and $\mathcal{C}'$ be partition ideals of order 1 with $d_j = \sup_{\{f_j\} \in \mathcal{C}} f_j$ and $d'_j = \sup_{\{f_j\} \in \mathcal{C}'} f_j$.*
Then $\mathcal{C} \sim \mathcal{C}'$ if and only if the two sequences of positive integers $\{j(d_j + 1)\}_{j=1,\ d_j < \infty}^{\infty}$ and $\{j(d'_j + 1)\}_{j=1,\ d'_j < \infty}^{\infty}$ are merely reorderings of each other.

The proof proposed in [1] relies heavily upon "a very usable representation of the generating function for $p(\mathcal{C}, n)$ whenever $\mathcal{C}$ is a partition ideal of order 1". The above numbers $\{j(d_j + 1)\}$ and $\{j(d'_j + 1)\}$ have appeared there by pure technical reasons.

Corollary 28. *Theorem 27 follows directly from Theorem 22.*

Proof. For each $d_j < \infty$, define A_j to be the multiset that consists of exactly d_j+1 copies of the number j, and for each $d'_j < \infty$, define B_j to be the multiset that consists of exactly d'_j+1 copies of the number j:

$$A_j := \{\underbrace{j, j, \ldots, j}_{d_j+1 \ times}\}, \quad \text{and} \quad B_j := \{\underbrace{j, j, \ldots, j}_{d'_j+1 \ times}\}.$$

Notice that A_j's are disjoint and $\|A_j\| = j(d_j + 1)$, and B_j's are disjoint and $\|B_j\| = j(d'_j + 1)$.

Proposition 26 shows that $M \notin \mathcal{C} \Longleftrightarrow A_j \subseteq M$ for some j, which means that the order filter $\overline{\mathcal{C}}$ is exactly generated by the A_j's. In other terms, our A_j's form the disjoint support of $\overline{\mathcal{C}}$.

Similarly, our B_j's form the disjoint support of $\overline{\mathcal{C}'}$.

It remains to apply Theorem 22. □

Thus, the Andrews' theorem [1, Theorem 8.4] has two proofs: the corresponding partition identities has been proven through the use of generating functions [1], but the bijective proof found here allows us to get a broader understanding of the result.

It should be pointed out that the class of partition ideals with 'disjointly supported' complements is "orthogonal" to the Andrews' hierarchy by 'order k'. Indeed, our class includes all partition ideals of order 1, and many others of 'unbounded/infinite order' (see Example 3). The indirect evidence of the size of this class is that it seems problematic to find a usable general representation of the generating functions for the whole variety of the partition ideals with 'disjointly supported' complements, which would provide a uniform proof of Theorem 22.

4 Complexity

In practical cases (e.g., Euler's Partition Theorem) the reduction sequences converge very fast, which provides `polytime` bijections h between $\mathcal{B}$-normal forms and $\mathcal{A}$-normal forms. Under reasonable hypotheses on the complexity of the lists $\mathcal{A}$ and $\mathcal{B}$, the two-directional rewriting machinery guarantees a `sub-exponential time`, at the very worst:

Corollary 29. *Let Γ be a set of multiset rewriting rules of the form $\gamma_1{:}A_1 \to B_1$, $\gamma_2 : A_2 \to B_2$, ..., $\gamma_i : A_i \to B_i$, ..., such that A_i's and B_i's are recognizable in* `polynomial time`*, and for any $i \neq j$, A_i and A_j are disjoint multisets of positive integers, and B_i and B_j are disjoint multisets of positive integers, and, in addition, the 'balance conditions': $\sum_{a \in A_i} a = \sum_{b \in B_i} b$, hold for all i.*

Then for every n, Theorem 16 yields a total bijection h between $\mathcal{B}$-normal partitions of the n and $\mathcal{A}$-normal partitions of the same n, which runs at most in `sub-exponential time`*.*

Proof. It follows from Theorem 15, since the asymptotic growth of $p(n)$, the number of partitions of the integer n, is sub-exponential [6]:

$$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{2n/3}} \qquad (n \to \infty).$$

5 Concluding Remarks

The *novelty* of our approach to the combinatorics is in the use of rewriting techniques (*two-directional* in the sense that forward and backward application of rewrite rules head for two different normal forms) for the purpose of establishing *explicit bijections* between combinatorial objects of two different types (represented by the normal forms). For a variety of combinatorial problems involving partitions, we have shown how such bijective proofs can be established, and how the bijections are computed by means of multiset rewriting systems.

Although the non-overlapping multiset rules are obviously *confluent*, the *termination problem* for the interesting combinatorial objects is more subtle: it generally *fails* even for multiset rewriting systems that satisfy certain natural balance conditions. We have proved the *'restricted' two-directional strong normalization* for the multiset rewriting systems under consideration, which guarantees the desired combinatorial bijections.
As for the fundamental problem stated in [1]:

Fully characterize the equivalence classes of partition ideals.

we have fully characterized a *new* wide class of partition ideals, namely, we have *fully characterized* all equinumerous partition ideals with 'disjointly supported' complements.

As a corollary, a *new proof*, the 'bijective' one, has been given for all equinumerous classes of the partition ideals of order 1 from the classical book "The Theory of Partitions" by G.Andrews. As compared to the proof through the use of generating functions, the bijective proof found here allows us to get a broader understanding of the essence of the result.

We have stated here some results on factorizations and forests to show that the ideas underlying our approach are most likely applicable to many other combinatorial problems. In a forthcoming paper we will discuss the challenges of the 'overlapping' multiset rewriting systems, and the corresponding 'bijective' proofs of the Rogers-Ramanujan-like identities.

Acknowledgments

I owe special thanks to Herb Wilf for his inspiring introduction to the world of integer partitions and for valuable discussions on how techniques from one field can sometimes be useful in another. I am greatly indebted to Peter Freyd and Andre Scedrov for their helpful comments on the paper.

Since the paper used techniques from the rewriting world to sort out specific mathematical problems studied completely independently in the world of combinatorics, selecting and organizing the material was a real challenge that time. I am very much obliged to the referees for their insightful comments and fruitful recommendations, which have allowed me to improve the exposition of this paper.

References

[1] George E. Andrews. The Theory of Partitions, Cambridge University Press, 1998 44, 45, 46, 47, 53, 55, 56, 57

[2] Franz Baader and Tobias Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.

[3] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J.van Leeuwen, ed., Handbook of Theoretical Computer Science, volume B, pp 243–320. Elsevier, 1990.

[4] A. M. Garsia and S. C. Milne, Method for constructing bijections for classical partition identities. Proc. Nat. Acad. Sci. U. S. A. **78** (1981), no. 4(1), 2026–2028. 45

[5] Basil Gordon, Sieve-equivalence and explicit bijections, J. Combin. Theory Ser. A **34** (1983), no. 1, 90–93. 45

[6] G. H. Hardy and S. Ramanujan, Asymptotic formulae in combinatory analysis, Proc. London Math. Soc. **17** (1918), 175–115. 57

[7] Kathleen M. O'Hara, Bijections for partition identities. J. Combin. Theory Ser. A **49** (1988), no. 1, 13–25. 45

[8] J. W. Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, ed., Handbook of Logic in Computer Science, volume 2, pp 1–116. Oxford University Press, New York, 1992.

[9] Jeffrey B. Remmel, Bijective proofs of some classical partition identities, J. Combin. Theory Ser. A **33** (1982), 273-286. 45, 47

[10] Herbert S. Wilf, Sieve equivalence in generalized partition theory, J. Combin. Theory Ser. A **34** (1983), 80-89. 44, 45

[11] Herbert S. Wilf, Lectures on Integer Partitions, University of Victoria, Victoria, B. C., Canada, 2000, (`<http://cis.upenn.edu/~wilf>`). 44, 45, 48, 49

On Continuous Normalization

Klaus Aehlig* and Felix Joachimski

Mathematisches Institut, Ludwig-Maximilians-Universität München
Theresienstrasse 39, 80333 München, Germany
{aehlig,joachski}@mathematik.uni-muenchen.de

Abstract. This work aims at explaining the syntactical properties of continuous normalization, as introduced in proof theory by Mints, and further studied by Ruckert, Buchholz and Schwichtenberg.
In an extension of the untyped coinductive λ-calculus by void constructors (so-called repetition rules), a primitive recursive normalization function is defined. Compared with other formulations of continuous normalization, this definition is much simpler and therefore suitable for analysis in a coalgebraic setting. It is shown to be continuous w.r.t. the natural topology on non-wellfounded terms with the identity as modulus of continuity. The number of repetition rules is locally related to the number of β-reductions necessary to reach the normal form (as represented by the Böhm tree) and the number of applications appearing in this normal form.

1 Introduction

Continuous normalization has been introduced by Mints [Min78, KMS75] in order to separate cut-elimination for semiformal systems from their ordinal analysis. By introducing a logical rule of *repetition* $(\mathcal{R})\,\frac{\Gamma\vdash A}{\Gamma\vdash A}$, it is possible to describe manipulations of infinitary derivations in a finitary manner, turning the cut-elimination operator into a primitive recursive function (see [Buc91] for a concise exposition). As Mints observed, this cut-elimination operator can also be applied to non-wellfounded derivations, resulting in a continuous function on derivation trees. More recently, Schwichtenberg [Sch98] has transferred these ideas from sequent calculi to natural deduction systems and a λ-calculus with an infinitary branching rule, building on [Buc91] and previous work by Ruckert [Ruc85].

$\mathcal{R}$ = "please wait". The central idea of continuous normalization is well-known to every user of modern computer operation systems: Whenever the result of a computation cannot immediately be displayed, a procedure should at least give regular life signs (like "please wait...") to be considered productive. In the terminology of coalgebra, in particular Coquand's conception of guarded recursion [Coq94], a partial function computing non-wellfounded objects can always be rendered total by adding a void constructor $\mathcal{R}$ (for repetition) in the

* Supported by the "Graduiertenkolleg Logik in der Informatik" of the Deutsche Forschungsgemeinschaft

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 59–74, 2002.
© Springer-Verlag Berlin Heidelberg 2002

clauses where productivity cannot be guaranteed. This corresponds to the above mentioned repetition rule.

Normalization for diverging terms. In our example of a normalization function for λ-terms, this amounts to returning $\mathcal{R}$, whenever the head constructor of the normal form cannot immediately be read off from the argument. So instead of non-wellfounded normal forms (Böhm trees), normal forms in an extended (co)grammar are computed. While for instance the head constructor of the normal form of λxr must be λ, the head constructor of an application rs depends on whether r is an abstraction, a variable, or again an application. The continuous normalization function therefore outputs $\mathcal{R}$, before further analysing r to find out whether s will be used for a substitution (if r were an abstraction) or has to be processed next (if r is a variable). As a consequence, the result of the diverging term $(\lambda x.xx)\lambda x.xx$ is an infinite sequence of repetition rules.

Coinductive λ-calculus. According to Mints' observation, the so obtained normalization procedure can be used to normalize non-wellfounded derivations, or in our case, terms of the coinductive λ-calculus Λ^{co} (see, e.g., [KKSdV97, Joa01a]). One of the interesting features of such a calculus is the possibility to directly define a fixpoint $Y_r^{\mathrm{co}} := rY_r^{\mathrm{co}}$.

Explaining $\mathcal{R}$. It has to be ensured that not too many $\mathcal{R}$ are produced in order to retain correctness of the normalization function for those terms that actually have a finitary normal form. Our analysis will reveal that every $\mathcal{R}$ corresponds to either a β-reduction or to an application in the Böhm tree of the term. To make this precise, another constructor β is introduced, which justifies a previous $\mathcal{R}$ in the normal form and stands for one reduction step in the normalization process. For example, normalization of the fixpoint combinator $Y := \lambda f.(\lambda x.f(xx))\lambda x.f(xx)$ applied to $K := \lambda y\lambda xy$ results in

$$(YK)^\beta = \mathcal{R}\beta\mathcal{R}\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\beta\mathcal{R}\beta\lambda x\ldots$$

while for the fixpoint combinator $\Theta := (\lambda x, f.f(xxf))\lambda x, f.f(xxf)$ we obtain

$$(\Theta K)^\beta = \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x.\ \mathcal{R}\mathcal{R}\beta\beta\mathcal{R}\beta\lambda x\ldots$$

So, apart from computing the non-wellfounded normal form $\lambda x\lambda x\lambda x\ldots$,[1] the result yields insight into the normalization behaviour of its argument and thus permits a more detailed analysis: It shows that unfolding ΘK requires 3 reductions

$$\begin{aligned}\Theta K &= (\lambda x, f.f(xxf))\lambda x, f.f(xxf))K\\ &\to (\lambda f.f(\Theta f))K\\ &\to K(\Theta K)\\ &\to \lambda x.\Theta K\end{aligned}$$

[1] Note that both terms have undefined Böhm trees

to obtain the next λ, while $Y_K := (\lambda x.K(xx))\lambda x.K(xx)$ only needs two:

$$(\lambda x.K(xx))\lambda x.K(xx) \to KY_K \to \lambda x Y_K.$$

Not very surprising, the directly defined fixpoint Y_K^{co} is the most efficient: $Y_K^{\mathrm{co}} = KY_K^{\mathrm{co}} \to \lambda x Y_K^{\mathrm{co}}$. In fact,

$$\begin{array}{rl} \Theta^\beta = & \mathcal{R}\beta\lambda x.\mathcal{R}(x(\mathcal{R}\mathcal{R}\beta\beta\mathcal{R}(x(\mathcal{R}\mathcal{R}\beta\beta\mathcal{R}(x(\ldots & \text{while} \\ Y^\beta = & \lambda x.\mathcal{R}\beta\mathcal{R}(x(\ \ \mathcal{R}\beta\ \ \mathcal{R}(x(\ \ \mathcal{R}\beta\ \ \mathcal{R}(x(\ldots \\ (Y_x^{\mathrm{co}})^\beta = & \qquad\quad \mathcal{R}(x(\qquad\ \mathcal{R}(x(\qquad\ \mathcal{R}(x(\ldots \end{array}$$

Analysis. Compared to Schwichtenberg's presentation of continuous normalization for the λ-calculus, the algorithm is simpler and therefore better suited for a precise analysis.

- The function is proved continuous w.r.t. the natural topology on non-well-founded terms that arises from the notion of equality up to k observations. The modulus of continuity is the identity.
- Lower and upper bounds for the number of $\mathcal{R}$ and β are exhibited.
- Each β is shown to correspond exactly to one reduction in the leftmost-outermost normalization strategy.

Outline of the contents. Section 2 recalls the definition of the coinductive λ-calculus with de Bruijn indices and introduces the extension by $\mathcal{R}$ and β. Section 3 presents the normalization algorithm, proves the modulus of continuity and correctness for finite normal forms. Section 4 establishes a coinductive characterization of the set of normal forms which arise by continuous normalization. This is used in section 5 to make the above remarks on the number of $\mathcal{R}$ and β precise. The appendix contains an example implementation in `Haskell`.

Acknowledgements. We are grateful to Andreas Abel for helpful comments on previous drafts of this work. Wilfried Buchholz gave valuable remarks on the history of continuous normalization.

2 The Coinductive λ-Calculus with $\mathcal{R}$ and β

The coinductive λ-calculus Λ^{co} arises by a coinductive interpretation of the defining grammar of the usual λ-calculus. Since this construction includes infinitary λ-terms, a variable's property of being new w.r.t. a given term is no longer decidable. Terms like r_0 with $r_n := x_n r_{n+1}$ (where $x_0, x_1, x_2, \ldots$ is an enumeration of all variables) may even contain all variables. It is thus reasonable to retreat

into a de Bruijn discipline [Bru72] in handling free and bound variables in order to keep constructions like substitution primitive recursive.

2.1. Terms. Terms of the inductive and coinductive λ-calculus and their extensions by $\mathcal{R}$ and β are given by[2]

$$\begin{array}{lllllll} \Lambda & \ni r, s ::= & k & \mid & rs & \mid & \lambda r \\ \Lambda^{\mathrm{co}} & \ni r, s ::=^{\mathrm{co}} & k & \mid & rs & \mid & \lambda r \\ \Lambda_{\mathcal{R}} & \ni r, s ::= & k & \mid & rs & \mid & \lambda r \mid \mathcal{R}r \mid \beta r \\ \Lambda^{\mathrm{co}}_{\mathcal{R}} & \ni r, s ::=^{\mathrm{co}} & k & \mid & rs & \mid & \lambda r \mid \mathcal{R}r \mid \beta r \end{array}$$

Notation. x, y, k, l, m, n range over natural numbers. Obviously $\Lambda^{\mathrm{co}}_{\mathcal{R}} \supset \Lambda^{\mathrm{co}} \supset \Lambda \subset \Lambda_{\mathcal{R}} \subset \Lambda^{\mathrm{co}}_{\mathcal{R}}$. The notorious dot notation is applied as follows: A dot stands for a pair of parentheses that open at the dot and close as far right as syntactically possible. For instance $\lambda\lambda\lambda.2\,0.1\,0$ stands for $\lambda\lambda\lambda((2\,0)(1\,0))$, i.e., the combinator S. $\boldsymbol{r}^{\,n}$ (the superscript n will be omitted whenever reasonable) denotes a possibly empty list of terms $r_1, \ldots, r_n$. ε stands for the empty list. A comma is used for pre- and postfixing terms as well as appending lists. $\mathbf{0}^{\,n}$ stands for the list $0, \ldots, 0$ (n zeros).

Examples. By the guarded[3] recursive definition $Y^{\mathrm{co}}_r := rY^{\mathrm{co}}_r$ a direct implementation of a fixpoint for any term r is admissible in Λ^{co}. Less reasonable terms are $r := rr$ or $r := \lambda r$. In $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ we may define

$$\bot_n ::=^{\mathrm{co}} \mathcal{R}\bot_{n+1} \mid \beta\bot_{n-1} \mid \lambda\bot_n.$$

In section 4 we will see that each $\bot_0$ arises exactly as a normal form of a term with undefined Böhm tree.

2.2. Observational equality. Define the equivalence relation $\simeq_k$ on $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ inductively by

$$\frac{}{r \simeq_0 r'} \qquad \frac{}{x \simeq_l x} \qquad \frac{r, s \simeq_k r', s'}{rs \simeq_{k+1} r's'} \qquad \frac{r \simeq_k r'}{\lambda r, \mathcal{R}r, \beta r \simeq_{k+1} \lambda r', \mathcal{R}r', \beta r'}$$

Here we used the abbreviation $\boldsymbol{r}^{\,n} \simeq_k \boldsymbol{s}^{\,n} \;:\Leftrightarrow\; r_1 \simeq_k s_1 \;\wedge\; \ldots \;\wedge r_n \simeq_k s_n$.

Remarks. Obviously $r \simeq_{k+n} r'$ implies $r \simeq_k r'$ (weakening). $\simeq_k$-equivalence classes define the open sets of a topology on terms in coinductive calculi which is also induced by the metric $\mathrm{d}(r, s) := \frac{1}{1+\sup_k r \simeq_k s}$.

Equality on non-wellfounded terms is given by the bisimulation

$$r = s \;:\Longleftrightarrow\; \forall k. r \simeq_k s.$$

[2] We write $::=^{\mathrm{co}}$ to signify that the grammar should be interpreted coinductively, i.e., instead of the least we choose the greatest fixpoint of the underlying Set-endofunctor. For instance, Λ^{co} is the set of — not necessarily wellfounded — unary and binary branching trees with leaves labeled by natural numbers.

[3] In the sense of [Coq94, Gim95].

2.3. Lifting. Define by guarded[4] recursion $(-)\uparrow_n : \Lambda^{\mathrm{co}}_{\mathcal{R}} \longrightarrow \Lambda^{\mathrm{co}}_{\mathcal{R}}$.

$$\begin{aligned} (rs)\uparrow_n &:= r\uparrow_n s\uparrow_n & (\lambda r)\uparrow_n &:= \lambda.r\uparrow_{n+1} \\ (\mathcal{R}r)\uparrow_n &:= \mathcal{R}.r\uparrow_n & (\beta r)\uparrow_n &:= \beta.r\uparrow_n \\ k\uparrow_n &:= \begin{cases} k & \text{if } k < n \\ k+1 & \text{otherwise} \end{cases} & r\uparrow &:= r\uparrow_0 \end{aligned}$$

Remark. Lifting is continuous: $r \simeq_k s$ implies $r\uparrow_l \simeq_k s\uparrow_l$.

2.4. Substitution. Define by guarded recursion $-[-] : \Lambda^{\mathrm{co}}_{\mathcal{R}} \times \Lambda^{\mathrm{co}}_{\mathcal{R}} \longrightarrow \Lambda^{\mathrm{co}}_{\mathcal{R}}$.

$$\begin{aligned} (rs)[t]_n &:= r[t]_n\, s[t]_n & (\lambda r)[t]_n &:= \lambda.r[t\uparrow]_{n+1} \\ (\mathcal{R}r)[t]_n &:= \mathcal{R}.r[t]_n & (\beta r)[t]_n &:= \beta.r[t]_n \\ k[t]_n &:= \begin{cases} k & \text{if } k < n \\ t & \text{if } k = n \\ k-1 & \text{otherwise,} \end{cases} & r[t] &:= r[t]_0 \end{aligned}$$

This notion of substitution is tailored for β-reduction, only. A general substitution for non-wellfounded term systems which satisfies the usual monadic laws can for instance be found in [FPT99]. For an adaptation to Λ^{co} see [Joa01b].

Proposition 1 (Continuity). $r, s \simeq_k r', s' \implies r[s]_l \simeq_k r'[s']_l$.

2.5. β-reduction in Λ. The reduction relation $\to$ is only needed and defined on the inductive calculus Λ. It is the compatible closure of elementary β-reduction $(\lambda r)s \mapsto r[s]$, i.e., defined inductively by

$$\frac{}{(\lambda r)s \to r[s]} \qquad \frac{r \to r'}{\lambda r, rs, sr \to \lambda r', r's, sr'}$$

(with pointwise reading of $\boldsymbol{r} \to \boldsymbol{s}$). $\to^*$ is the reflexive transitive closure of $\to$.

Remark. If used in Λ^{co}, the reduction $\to$ is no longer confluent: With $r := 0r$ and $s := ((\lambda 0)0)s$ we have

$$r \;{}^*\!\not\leftarrow s \leftarrow (\lambda r)((\lambda 0)0) \to (\lambda r)0 \to r \not\to^* s.$$

For a more precise analysis and positive confluence results see [Joa01a].

2.6. Normal forms. The set of *normal forms*

$$\mathrm{NF} \ni r, s ::=^{\mathrm{co}} x\boldsymbol{r} \mid \lambda r \mid \mathcal{R}r \mid \beta r$$

contains only terms without redexes. Note that $\mathrm{NF} \cap \Lambda$ is the usual set of normal forms of Λ. However, there are some normal terms in $\Lambda^{\mathrm{co}}_{\mathcal{R}}$ which are not captured by this cogrammar, such as $(\mathcal{R}0)0$.[5]

[4] To justify this definition, the framework of Coquand/Gimenez is sufficient. For the following definitions, we assume a more liberal metatheory such as [TT97].

[5] The complete cogrammar for *all* normal forms is $r, s ::=^{\mathrm{co}} x\boldsymbol{r} \mid \lambda r \mid (\mathcal{R}r)\boldsymbol{s} \mid (\beta r)\boldsymbol{s}$.

3 Continuous Normalization

This section defines the primitive recursive normalization function $()^\beta$. The result of r^β can be understood as the normal form of r, enriched by information on the reduction sequence that was used to reach it.

The definition of r^β takes recourse to an auxiliary function $r@\boldsymbol{s}$, which intuitively should compute the normal form of $r\boldsymbol{s}$.[6]

3.1. Definition. We define $r@\boldsymbol{s} \in \mathrm{NF}$ (with $r, \boldsymbol{s} \in \Lambda_{\mathcal{R}}^{\mathrm{co}}$) by guarded recursion, using the abbreviation $r^\beta := r@\varepsilon$.

$$\begin{aligned} (\lambda r)@(s, \boldsymbol{s}\,) &:= \beta.r[s]@\boldsymbol{s} & (\lambda r)@\varepsilon &:= \lambda r^\beta \\ x@\boldsymbol{s} &:= x\boldsymbol{s}^{\,\beta} & (\mathcal{R}r)@\boldsymbol{s} &:= \mathcal{R}.r@\boldsymbol{s} \\ (rs)@\boldsymbol{s} &:= \mathcal{R}.r@(s, \boldsymbol{s}\,) & (\beta r)@\boldsymbol{s} &:= \beta.r@\boldsymbol{s} \end{aligned}$$

The normalization function outputs $\mathcal{R}$ whenever it faces an application rs, because it cannot foresee what to do with the argument s. When it next encounters an abstraction $r = \lambda r'$, the s will be used for the substitution $r'[s]$, so the $\mathcal{R}$ is justified ex post. If on the other hand a variable $r = x$ should follow then the s will be further normalized to produce the normal form of xs. The $\mathcal{R}$ produced in the step $(rs)^\beta = \mathcal{R}.r@s$ thus accounts for the application in the normal form xs^β.

Example. $(SKK)^\beta = \mathcal{R}\mathcal{R}\beta\beta\lambda\mathcal{R}\mathcal{R}\beta\beta 0$ with $S := \lambda\lambda\lambda.2\,0.1\,0$, $K := \lambda\lambda 1$. Note that $SKK \to^2 \lambda.K0.K0 \to^2 \lambda 0$.

Remarks. It is easy to see that the β-constructor is not necessary to ensure well-definedness, but it guarantees that the modulus of continuity is the identity.

For $r \in \Lambda$, the last two clauses are not needed to compute r^β.

Finally, we note a simple property: $(r\boldsymbol{s}^{\,n})^\beta = \mathcal{R}^n.r@\boldsymbol{s}$.

3.2. Continuity. The following lemma establishes that $()^\beta$ is continuous with modulus of continuity $k \mapsto k$.

Lemma 2. $r \simeq_k r' \;\wedge\; \boldsymbol{s} \simeq_k \boldsymbol{s'} \implies r@\boldsymbol{s} \simeq_k r'@\boldsymbol{s'}$.

Proof. The claim implies in particular that $r^\beta \simeq_k r'^\beta$ for $r \simeq_k r'$. It is proved by induction on k. Of course, the case $k = 0$ is trivial. **Case** $x@\varepsilon$. Trivial. **Case** $x@(\boldsymbol{s}, s)$ with $\boldsymbol{s}, s \simeq_k \boldsymbol{s'}, s'$. By induction hypothesis $x\boldsymbol{s} \simeq_k x\boldsymbol{s'}$, so $x\boldsymbol{s}s \simeq_{k+1}$

[6] It should be noted that our main concern is the conituity of the normalization function and the additional constructors needed for this purpose. The idea of collecting arguments of an application in a list and hence having $r@\boldsymbol{s}$ as a main normalization device is suggested by the way we analyze terms and was, for example, also used by Berarducci and Böhm [BB01, p.27] in their non-continuous normalization algorithm.

$x\boldsymbol{s'}s'$ and by weakening $xss \simeq_k x\boldsymbol{s'}s'$. **Case** $(\lambda r)@(s, \boldsymbol{s}\,)$ with $\lambda r \simeq_{k+1} \lambda r'$ and $s, \boldsymbol{s} \simeq_{k+1} s', \boldsymbol{s'}$.

$$\begin{array}{rll} s, \boldsymbol{s} \simeq_k & s', \boldsymbol{s'} & \text{by weakening} \\ r[s] \simeq_k & r'[s'] & \text{by proposition 2.4} \\ r[s]@\boldsymbol{s} \simeq_k & r'[s']@\boldsymbol{s'} & \text{by induction hypothesis} \\ \beta.r[s]@\boldsymbol{s} \simeq_{k+1} & \beta.r'[s']@\boldsymbol{s'} & \\ (\lambda r)@(s, \boldsymbol{s}\,) \simeq_{k+1} & (\lambda r')@(s', \boldsymbol{s'}). & \end{array}$$

Case $(rs)@\boldsymbol{s}$ with $\boldsymbol{s} \simeq_{k+1} \boldsymbol{s'}$ and $rs \simeq_{k+1} r's'$, i.e., $r, s \simeq_k r', s'$. By weakening $\boldsymbol{s} \simeq_k \boldsymbol{s'}$, so the induction hypothesis yields $r@(s, \boldsymbol{s}\,) \simeq_k r'@(s'\boldsymbol{s'})$. It follows that

$$r@(s, \boldsymbol{s}\,) = \mathcal{R}.r@(s, \boldsymbol{s}\,) \simeq_{k+1} \mathcal{R}.r'@(s', \boldsymbol{s'}) = r'@(s', \boldsymbol{s'}).$$

All other cases are simple applications of the induction hypothesis. □

3.3. Normalization. The next goal is to verify that the result computed by r^β is actually the normal form of r, if it is finite. Since r^β might contain some $\mathcal{R}$ and β, we have to eliminate them to prove this correctness property.

Definition. For $r \in \Lambda_\mathcal{R}$ define recursively $r^* \in \Lambda$.

$$x^* := x, \qquad (rs)^* := r^*s^*, \qquad (\lambda r)^* := \lambda r^*, \qquad (\mathcal{R}r)^* := (\beta r)^* := r^*.$$

Proposition 3. $r \in \Lambda \ \wedge \ r^\beta \in \Lambda_\mathcal{R} \implies r \to^* r^{\beta *}$.

Proof. Induction on *the size* of r^β. Cases on r. **Case** $x\boldsymbol{r}$. We compute $(x\boldsymbol{r}\,)^\beta = \mathcal{R}^n.x\boldsymbol{r}^{\,\beta}$. Thus

$$\begin{array}{rll} x\boldsymbol{r}^{\,n} \to^* & x\boldsymbol{r}^{\,\beta *} & \text{by IH} \\ = & (\mathcal{R}^n.x\boldsymbol{r}^{\,\beta})^* & \\ = & (x\boldsymbol{r}\,)^{\beta *}. & \end{array}$$

Case $(\lambda r)s\boldsymbol{s}^{\,n}$. Note $((\lambda r)s\boldsymbol{s}^{\,n})^\beta = \mathcal{R}^n.(\lambda r)@(s, \boldsymbol{s}\,) = \mathcal{R}^{n+1}.r[s]@\boldsymbol{s}$. Thus

$$\begin{array}{rll} (\lambda r)s\boldsymbol{s}^{\,n} \to & r[s]\boldsymbol{s} & \\ \to^* & (r[s]@\boldsymbol{s}\,)^* & \text{by IH} \\ = & (\mathcal{R}^{n+1}\beta.r[s]@\boldsymbol{s}\,)^* & \\ = & ((\lambda r)s\boldsymbol{s}\,)^{\beta *}. & \end{array}$$

The case λr is a simple application of the induction hypothesis. □

4 Continuous Normal Forms

NF still contains many terms that are not in the image of $()^\beta$, e.g., $\beta 7$. The objective for this section is to find a precise characterization of the set CNF :=

$\{r \in \Lambda^{\mathrm{co}}_{\mathcal{R}} \mid \exists s \in \Lambda^{\mathrm{co}}.s^{\beta} = r\}$ of terms that arise as normal forms under the process of continuous normalization.

4.1. Well-formedness. Intuitively, a term is well-formed, if each $\mathcal{R}$ is justified either by a β or an application. More precisely, we define $\boldsymbol{s} \vdash r$, where each of the $\boldsymbol{s}$ is accounted for by a $\mathcal{R}$. The coinductive rules for $\boldsymbol{s} \vdash r$ are

$$(\mathcal{R}_s)\ \frac{s, \boldsymbol{s} \vdash r}{\boldsymbol{s} \vdash \mathcal{R}r} \qquad (v)\ \frac{\vdash \boldsymbol{r}}{\boldsymbol{r} \vdash x\boldsymbol{r}} \qquad (\lambda)\ \frac{\vdash r}{\vdash \lambda r} \qquad (\beta)\ \frac{\boldsymbol{s} \vdash r}{s, \boldsymbol{s} \vdash \beta r}$$

r is *well-formed* if $\vdash r$. In this case obviously $r \in \mathrm{NF}$.

Notation. In $\boldsymbol{s} \vdash r$ the list $\boldsymbol{s}$ is named the *context*. The terms $\boldsymbol{r}$ in rule (v) are called the *Eigenterms* of this rule.

Example. $\mathbf{0}^{\,n} \vdash \bot_n$.

Lemma 4. $\boldsymbol{s} \vdash r@\boldsymbol{s}$.

Proof. Guarded induction along the definition of @.[7]

Case $x@\boldsymbol{s}$. By induction hypothesis $\vdash \boldsymbol{s}^{\,\beta}$. Thus $\boldsymbol{s} \vdash x@\boldsymbol{s}$ by (v).

Case $(\lambda r)@(s, \boldsymbol{s}\,)$. By induction hypothesis $\boldsymbol{s} \vdash r[s]@\boldsymbol{s}$, so by rule (β): $s, \boldsymbol{s} \vdash \beta.r[s]@\boldsymbol{s} = (\lambda r)@(s, \boldsymbol{s}\,)$.

Case $(rs)@\boldsymbol{s}$. By induction hypothesis: $s, \boldsymbol{s} \vdash r@(s, \boldsymbol{s}\,)$, so by rule $(\mathcal{R}_s)$: $\boldsymbol{s} \vdash \mathcal{R}.r@(s, \boldsymbol{s}\,) = (rs)@\boldsymbol{s}$.

Case $(\lambda r)@\varepsilon$. By induction hypothesis: $\vdash r^{\beta}$, so by (λ): $\vdash \lambda r^{\beta} = (\lambda r)@\varepsilon$. $\square$

Thus every CNF term is well-formed. For the converse implication a further analysis of derivations of $\boldsymbol{s} \vdash r$ is necessary. In particular, each $\mathcal{R}$ should be attributed to either a β or an application. The corresponding notion is that of

4.2. Well-explained terms. Define coinductively

$$(\mathcal{R}_s)\ \frac{s, \boldsymbol{s} \vDash_0 r \qquad \vDash_0 s}{\boldsymbol{s} \vDash_0 \mathcal{R}r} \qquad (v)\ \frac{}{\boldsymbol{r} \vDash_0 x\boldsymbol{r}} \qquad (\lambda)\ \frac{\vDash_0 r}{\vDash_0 \lambda r}$$

$$(\mathcal{R})\ \frac{\boldsymbol{s} \vDash_{n+1} r}{\boldsymbol{s} \vDash_n \mathcal{R}r} \qquad (\beta)\ \frac{\boldsymbol{s} \vDash_n r}{\boldsymbol{s} \vDash_{n+1} \beta r}$$

r is *well-explained* if $\vDash_0 r$.

In a well-explained term we have a precise explanation for each $\mathcal{R}$: either it is introduced by $(\mathcal{R})$, in which case it will be canceled later by a β; or it is introduced by $(\mathcal{R}_s)$ and the well-explained term s will occur in an application.

[7] This means that the non-wellfounded derivation of $\boldsymbol{s} \vdash r@\boldsymbol{s}$ is constructed coiteratively by guarded recursion.

Example. A derivation of $Y^\beta = \lambda\phi$ with $\phi := \mathcal{R}\beta\mathcal{R}.0\phi$ is shown on the right.

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{}{\phi \vDash_0 0\phi}\,(v) \qquad \dfrac{\dfrac{\vdots}{\vDash_1 \beta\mathcal{R}.0\phi}\,(\beta)}{\vDash_0 \phi}\,(\mathcal{R})}{\vDash_0 \mathcal{R}.0\phi}\,(\mathcal{R}_\phi)}{\vDash_1 \beta\mathcal{R}.0\phi}\,(\beta)}{\vDash_0 \phi}\,(\mathcal{R})}{\vDash_0 \lambda\phi}\,(\lambda)
\end{array}
$$

Proposition 5. $\vdash \boldsymbol{s} \wedge \boldsymbol{s} \vDash_n r \implies \mathbf{0}^n, \boldsymbol{s} \vdash r$

Proof. Guarded induction on $\boldsymbol{r} \vDash_n r$.[8]

Case $(\mathcal{R}_s)$: $\boldsymbol{s} \vDash_0 \mathcal{R}r$ has been derived from $s, \boldsymbol{s} \vDash_0 r$ and $\vDash_0 s$. By induction hypothesis $\vdash s$, so, again by induction hypothesis $s, \boldsymbol{s} \vdash r$ and rule $(\mathcal{R}_s)$ yields $\boldsymbol{s} \vdash \mathcal{R}r$.

Case (v): $\boldsymbol{s} \vDash_0 x\boldsymbol{s}$. By assumption $\vdash \boldsymbol{s}$, so rule (v) proves $\boldsymbol{s} \vdash x\boldsymbol{s}$.

Case $(\mathcal{R})$: $\boldsymbol{s} \vDash_n \mathcal{R}r$ has been concluded from $\boldsymbol{s} \vDash_{n+1} r$. By induction hypothesis $0, \mathbf{0}^n, \boldsymbol{s} \vdash r$, so by rule $(\mathcal{R}_0)$ $\mathbf{0}, \boldsymbol{s} \vdash \mathcal{R}r$.

Case (β): $\boldsymbol{s} \vDash_{n+1} \beta r$ from $\boldsymbol{s} \vDash_n r$. By induction hypothesis $\mathbf{0}^n, \boldsymbol{s} \vdash r$, so rule (β) shows $0, \mathbf{0}, \boldsymbol{s} \vdash \beta r$.

Case (λ). Simple application of the induction hypothesis. □

Definition. We write $\boldsymbol{r} \mid \boldsymbol{s} \vdash r$ if $\boldsymbol{r}, \boldsymbol{s} \vdash r$ and $\boldsymbol{s}$ are Eigenterms of the derivation and $\boldsymbol{r}$ are not. We write $\boldsymbol{r} \mid \boldsymbol{s} \vdash^* r$ if all contexts of the derivation of $\boldsymbol{r}, \boldsymbol{s} \vdash r$ can be consistently split that way.

Theorem 6 (Clairvoyance).
If $\boldsymbol{r} \vdash r$ then there is a partition $\boldsymbol{r} = (\boldsymbol{s}, \boldsymbol{t})$ such that $\boldsymbol{s} \mid \boldsymbol{t} \vdash^* r$.

Proof. Using classical logic,[9] each term either is or is not an Eigenterm of the derivation. Noting that every term on the right of an Eigenterm in any context of the derivation has to be an Eigenterm as well (in fact, even of the same variable rule), we get the claim. □

Example. $\mathbf{0}^n \mid \vdash^* \bot_n$.

Lemma 7. $\boldsymbol{r}^n \mid \boldsymbol{s} \vdash^* r \implies \boldsymbol{s} \vDash_n r$.

Proof. Guarded induction on $\boldsymbol{r}^n \mid \boldsymbol{s} \vdash^* r$.

Case $(\mathcal{R}_s)$ with $\boldsymbol{s}^n \mid \boldsymbol{t} \vdash \mathcal{R}r$ from $s, \boldsymbol{s} \mid \boldsymbol{t} \vdash^* r$. By induction hypothesis $\boldsymbol{t} \vDash_{n+1} r$, so rule $(\mathcal{R})$ yields $\boldsymbol{t} \vDash_n \mathcal{R}r$.

Case $(\mathcal{R}_s)$ with $\mid \boldsymbol{s} \vdash^* \mathcal{R}r$ from $\mid s, \boldsymbol{s} \vdash^* r$. As s is an Eigenterm of a rule (v) which contains $\vdash^* s$ as a premise, we can apply the induction hypothesis to obtain $\vDash_0 s$. Also $s, \boldsymbol{s} \vDash_0 r$ by induction hypothesis, so rule $(\mathcal{R}_s)$ proves $\boldsymbol{s} \vDash_0 \mathcal{R}r$.

Case (β): $r, \boldsymbol{r}^n \mid \boldsymbol{s} \vdash^* \beta r$ from $\boldsymbol{r} \mid \boldsymbol{s} \vdash^* r$. By induction hypothesis $\boldsymbol{r} \vDash_n r$, so rule (β) yields $\boldsymbol{r} \vDash_{n+1} \beta r$.

The two remaining cases are simple. □

[8] See previous footnote. The formulation "guarded induction on ..." provides the additional information that for finite derivations of $\vDash_n$ a well-founded derivation of $\vdash$ is obtained.

[9] Note that a constructive proof of this theorem cannot exist, because its underlying algorithm would allow to decide the halting problem. Moreover, this is the only place in the article where non-constructive reasoning is invoked.

Example. $\vDash_n \bot_n$ using just $(\mathcal{R}), (\beta), (\lambda)$.

Corollary 8 (Equivalence). $\vdash r \iff \vDash_0 r$.

4.3. Completeness. With the equivalent reformulation of $\vdash$ by $\vDash_0$ at hand, we can now show that every well-formed term arises as a continuous normal form. The witness of this existence is computed by the function T.

Definition. Define a term $T(\boldsymbol{s} \vDash_n r) \in \Lambda^{\mathrm{co}}$ by guarded recursion on the derivation of $\boldsymbol{s} \vDash_n r$. We use the notation of the definition of $\vDash_n$.

$$\begin{array}{lll}
(\mathcal{R}_s) & T(\boldsymbol{s} \vDash_0 \mathcal{R}r) & := T(s, \boldsymbol{s} \vDash_0 r)T(\vDash_0 s) \\
(v) & T(\boldsymbol{r} \vDash_0 x\boldsymbol{r}) & := x \\
(\lambda) & T(\vDash_0 \lambda r) & := \lambda T(\vDash_0 r) \\
(\beta) & T(\boldsymbol{s} \vDash_{n+1} \beta r) & := \lambda.T(\boldsymbol{s} \vDash_n r)\uparrow \\
(\mathcal{R}) & T(\boldsymbol{s} \vDash_n \mathcal{R}r) & := T(\boldsymbol{s} \vDash_{n+1} r)0
\end{array}$$

Example. For the derivation of $\vDash_0 Y^\beta$ in subsection 4.2 we obtain $T(\vDash_0 Y^\beta) = \lambda.\lambda(1.\lambda(2.\lambda(\ldots)2)1)0$

Proposition 9 (Completeness). $\boldsymbol{t}^\beta = \boldsymbol{s} \Longrightarrow T(\boldsymbol{s} \vDash_n r)@(\boldsymbol{0}^n, \boldsymbol{t}) = r$.

Corollary 10. $T(\vDash_0 r)^\beta = r$.

In particular, each $\bot_n$ is a normal form of a term with undefined Böhm tree.

Proof (of the Proposition). Guarded induction on $r \vDash_n \boldsymbol{s}$.[10] **Case** (v): $\boldsymbol{s} \vDash_0 x\boldsymbol{s}$. By the premise $\boldsymbol{t}^\beta = \boldsymbol{s}$, so $T(\boldsymbol{s} \vDash_0 x\boldsymbol{s})@\boldsymbol{t} = x\boldsymbol{t}^\beta = x\boldsymbol{s}$. **Case** $(\mathcal{R}_s)$: $\boldsymbol{s} \vDash_0 \mathcal{R}r$ from $s, \boldsymbol{s} \vDash_0 r$ and $\vDash_0 s$. By induction hypothesis $T(\vDash_0 s)^\beta = s$. Thus with $t := T(\vDash_0 s)$, by induction hypothesis, $T(s, \boldsymbol{s} \vDash_0 r)@(t, \boldsymbol{t}) = r$, hence

$$T(\boldsymbol{s} \vDash_0 \mathcal{R}r)@\boldsymbol{t} = (T(s, \boldsymbol{s} \vDash_0 r)t)@\boldsymbol{t} = \mathcal{R}.T(s, \boldsymbol{s} \vDash_0 r)@(t, \boldsymbol{t}) = \mathcal{R}r.$$

Case $(\mathcal{R})$: $\boldsymbol{s} \vDash_n \mathcal{R}r$ from $\boldsymbol{s} \vDash_{n+1} \mathcal{R}$. With $0^\beta = 0$ we get

$$\begin{aligned}
T(\boldsymbol{s} \vDash_n \mathcal{R}r)@(\boldsymbol{0}^n, \boldsymbol{t}) &= (T(\boldsymbol{s} \vDash_{n+1} r)0)@(\boldsymbol{0}^n, \boldsymbol{t}) \\
&= \mathcal{R}.T(\boldsymbol{s} \vDash_{n+1} r)@(\boldsymbol{0}^{n+1}, \boldsymbol{t}) \\
&= \mathcal{R}r && \text{by IH.}
\end{aligned}$$

Case (β): $\boldsymbol{s} \vDash_{n+1} \beta r$ from $\boldsymbol{s} \vDash_n r$.

$$\begin{aligned}
T(\boldsymbol{s} \vDash_{n+1} \beta r)@(0, \boldsymbol{0}^n, \boldsymbol{t}) &= (\lambda.T(\boldsymbol{s} \vDash_n r)\uparrow)@(0, \boldsymbol{0}^n, \boldsymbol{t}) \\
&= \beta.(T(\boldsymbol{s} \vDash_n r)\uparrow)[0]@(\boldsymbol{0}^n, \boldsymbol{t}) \\
&= \beta.T(\boldsymbol{s} \vDash_n r)@(\boldsymbol{0}^n, \boldsymbol{t}) \\
&= \beta r && \text{by IH.}
\end{aligned}$$

Case (λ): $\vDash_0 \lambda r$ from $\vDash_0 r$. Trivial. □

[10] More precisely, the asserted equality amounts to showing $\simeq_k$ for all k and this is done by induction on k. The formulation chosen here is hopefully more reminiscent of traditional inductive equality proofs.

5 Analysis

Using well-formedness we can now provide a precise analysis of each $\mathcal{R}$ and β in the normalization process.

5.1. Bounds for $\mathcal{R}$. The first goal is to show that each $\mathcal{R}$ produced during of continuous normalization corresponds to either a β-reduction or an application in the normal form. In order to prove this for possibly non-wellfounded normal forms, we need the concept of paths.

Definition (Paths). A *path* is a list of natural numbers, i.e., $\zeta ::= \varepsilon \mid k{\cdot}\zeta$. The set of paths of a term r is given inductively by

$$\frac{}{\varepsilon \in r} \qquad \frac{\zeta \in r_k}{k \cdot \zeta \in x\boldsymbol{r}} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \lambda r} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \mathcal{R}r} \qquad \frac{\zeta \in r}{k \cdot \zeta \in \beta r}$$

ζ *is complete in* r (written $\zeta \in_c r$) iff $\varepsilon \in r$ is used only for abstractions λs and variable eliminations $x\boldsymbol{r}$. In other words, at the end of a complete path there are no pending terms to be applied.

(Only) for valid paths $\zeta \in r$ we define the number $\mathcal{R}_\zeta r$ of $\mathcal{R}$s, the number $\beta_\zeta r$ of βs and the number $A_\zeta r$ of applications in the path by

$$\begin{array}{llllll}
\mathcal{R}_{k\cdot\zeta}\lambda r & := \mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\lambda r & := \beta_\zeta r & A_{k\cdot\zeta}\lambda r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}\mathcal{R}r & := 1{+}\mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\mathcal{R}r & := \beta_\zeta r & A_{k\cdot\zeta}\mathcal{R}r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}\beta r & := \mathcal{R}_\zeta r & \beta_{k\cdot\zeta}\beta r & := 1{+}\beta_\zeta r & A_{k\cdot\zeta}\mathcal{R}r & := A_\zeta r \\
\mathcal{R}_{k\cdot\zeta}(x\boldsymbol{r}) & := \mathcal{R}_\zeta r_k & \beta_{k\cdot\zeta}(x\boldsymbol{r}) & := \beta_\zeta r_k & A_{k\cdot\zeta}(x\boldsymbol{r}^{\,n}) & := n{+}A_\zeta r_k \\
\mathcal{R}_\varepsilon r & := 0 & \beta_\varepsilon r & := 0 & A_\varepsilon(x\boldsymbol{r}^{\,n}) & := n \\
 & & & & A_\varepsilon r & := 0 \text{ otherwise}
\end{array}$$

Lemma 11. $\boldsymbol{r}^{\,n} \vdash s \implies n + \mathcal{R}_\zeta s \geq \beta_\zeta s + A_\zeta s$ *with equality for* $\zeta \in_c s$.

Proof. Induction on $\zeta \in s$.
Case ε. If s is not of the form $x\boldsymbol{r}$ then the claim is trivial, because $n + \mathcal{R}_\varepsilon s = n \geq 0 = \beta_\varepsilon s + A_\varepsilon s$. For a variable elimination $\boldsymbol{r}^{\,n} \vdash x\boldsymbol{r}$ we compute

$$n + \mathcal{R}_\varepsilon(x\boldsymbol{r}) = n + 0 = n = \beta_\varepsilon(x\boldsymbol{r}) + A_\varepsilon(x\boldsymbol{r}).$$

Case $l \cdot \zeta$. Subcase ($\mathcal{R}_s$): $\boldsymbol{r} \vdash \mathcal{R}r$ from $s, \boldsymbol{r} \vdash r$.

$$\begin{aligned}
n + \mathcal{R}_{l\cdot\zeta}\mathcal{R}r &= n + 1 + \mathcal{R}_\zeta r \\
&\geq \beta_\zeta r + A_\zeta r && \text{by IH} \\
&= \beta_{l\cdot\zeta}\mathcal{R}r + A_{l\cdot\zeta}\mathcal{R}r.
\end{aligned}$$

Subcase (v): $\boldsymbol{r}^{\,n} \vdash x\boldsymbol{r}$ from $\vdash \boldsymbol{r}$.

$$\begin{aligned}
n + \mathcal{R}_{l\cdot\zeta}(x\boldsymbol{r}) &= n + \mathcal{R}_\zeta r_l \\
&\geq n + \beta_\zeta r_l + A_\zeta r_l && \text{by IH} \\
&= \beta_{l\cdot\zeta}(x\boldsymbol{r}) + A_{l\cdot\zeta}(x\boldsymbol{r}^{\,n}).
\end{aligned}$$

Subcase (β): $r, \boldsymbol{r}^{\,n} \vdash \beta r$ from $\boldsymbol{r} \vdash r$.

$$\begin{aligned} n{+}1 + \mathcal{R}_{l\cdot\varsigma}\beta r &= 1 + n + \mathcal{R}_\varsigma r \\ &\geq 1 + \beta_\varsigma r + A_\varsigma r && \text{by IH} \\ &= \beta_{l\cdot\varsigma}\beta r + A_{l\cdot\varsigma}\beta r. \end{aligned}$$

Subcase (λ). Simple application of the induction hypothesis. □

As an instance of this lemma we obtain for well-formed s that the number of $\mathcal{R}$s on each complete path is precisely the number of βs plus the number of applications.

5.2. Bounds for β. Next we show that each β produced during continuous normalization corresponds to exactly one β-reduction step in the leftmost-outermost normalization strategy, as given inductively by

$$(v)\ \frac{\boldsymbol{r} \rightsquigarrow_{\boldsymbol{n}} \boldsymbol{s}}{x\boldsymbol{r} \rightsquigarrow_{\Sigma\,\boldsymbol{n}} x\boldsymbol{s}} \qquad (\lambda)\ \frac{r \rightsquigarrow_n s}{\lambda r \rightsquigarrow_n \lambda s} \qquad (\beta)\ \frac{r[s]\boldsymbol{s} \rightsquigarrow_n t}{(\lambda r)s\boldsymbol{s} \rightsquigarrow_{n+1} t} \qquad (r)\ \frac{}{r \rightsquigarrow_0 r}$$

($\boldsymbol{r} \rightsquigarrow_{\boldsymbol{n}} \boldsymbol{s}$ is read pointwise) The index counts the number of β-reduction steps performed.

Remark. By standardization, $r \to^* s \in \mathrm{NF} \cap \Lambda$ implies that there exists an n with $r \rightsquigarrow_n s$.

Definition. Let $\rhd_{\mathcal{R}}$ and $\rhd_\beta$ be the compatible closures of $\mathcal{R}r \rhd_{\mathcal{R}} r$ and $\beta r \rhd_\beta r$. $\rhd_{\mathcal{R}}^n$ stands for n steps of $\rhd_{\mathcal{R}}$. Let $\rhd_n^k$ contain reduction sequences with k $\rhd_{\mathcal{R}}$ and n $\rhd_\beta$-reductions (mixed ad libitum). Furthermore, we set $\rhd_n := \rhd_n^n$.

Lemma 12. $r \rightsquigarrow_n s \implies r^\beta \rhd_n s^\beta$.

Proof. Induction on $\rightsquigarrow_n$. **Case** $(\lambda r)s\boldsymbol{s}^{\,k} \rightsquigarrow_{n+1} t$.

$$\begin{aligned} ((\lambda r)s\boldsymbol{s}\,)^\beta &= \mathcal{R}^k\mathcal{R}.(\lambda r)@(s, \boldsymbol{s}\,) \\ &= \mathcal{R}^k\mathcal{R}\beta.r[s]@\boldsymbol{s} \\ &\rhd_{\mathcal{R}} \mathcal{R}^k\beta.r[s]@\boldsymbol{s} \\ &\rhd_\beta \mathcal{R}^k.r[s]@\boldsymbol{s} \\ &= (r[s]\boldsymbol{s}\,)^\beta \\ &\rhd_n t^\beta && \text{by IH.} \end{aligned}$$

Case $x\boldsymbol{r} \rightsquigarrow_{\Sigma\,\boldsymbol{n}} x\boldsymbol{s}$.

$$\begin{aligned} (x\boldsymbol{r}^{\,k})^\beta &= \mathcal{R}^k.x\boldsymbol{r}^{\,\beta} \\ &\rhd_{\Sigma\,\boldsymbol{n}} \mathcal{R}^k.x\boldsymbol{s}^{\,\beta} && \text{by IH} \\ &= (x\boldsymbol{s}\,)^\beta. \end{aligned}$$

Cases $(\lambda), (r)$. Simple. □

5.3. Weakly normalizing terms. If a term r actually has a finite normal form, then we can slightly strengthen the claim of the last lemma.

Definition. For $r \in \Lambda \cap \mathrm{NF}$ we define recursively

$$|x\boldsymbol{r}^{\,n}| := n + \Sigma\,|\boldsymbol{r}\,|, \qquad |\lambda r| := |r|.$$

Proposition 13. $r \in \Lambda \cap \mathrm{NF} \implies r^\beta \rhd_{\mathcal{R}}^{|r|} r.$

Proof. Trivial by remark 3.1. □

Corollary 14. $r \rightsquigarrow_n s \in \mathrm{NF} \cap \Lambda \implies r^\beta \rhd_n^{n+|s|} s.$

Hence, for every weakly normalizing term r, the normal form is computed by r^β and we have precise information on the number of steps in the standard reduction sequence leading to it.

6 Conclusions

By adopting a coinductive viewpoint on non-wellfounded terms it has become possible to simplify the formulation of continuous normalization. In particular, it is not necessary to introduce explicit substitution operators, because substitution is itself an admissible function in the coinductive λ-calculus. Yet it would be interesting to make the connection to previous presentations precise by adding normalization rules for an explicit substitution operator.

An extension to infinitary term systems with term formers corresponding to the infinitary ω-rule along the lines of [Sch98] is conceivable. Previous work on confluence of non-wellfounded infinitely branching term systems with permutative reductions [Joa01a] suggests that such a study is feasible, although the details concerning continuous normalization have not been worked out yet.

References

[BB01] A. Beraducci and Corrado Böhm. General recursion on second order term algebras. In *Rewriting Techniques and Applications*, volume 2051 of *Lecuture notes in Computer Science*, pages 15–30. Springer, 2001. 64

[Bru72] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972. 62

[Buc91] Wilfried Buchholz. Notation systems for infinitary derivations. *Archive for Mathematical Logic*, 30:277–296, 1991. 59

[Coq94] Thierry Coquand. Infinite objects in type theory. In H. Barendregt and T. Nipkow, editors, *Proc. 1st Types 1993 (Nijmengen)*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1994. Available from `http://www.cs.chalmers.se/ coquand/`. 59, 62

[FPT99] Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *Proc. 14th LICS 1999 (Trento)*, pages 193–202. IEEE Computer Science Press, 1999. Available from `http://www.dcs.ed.ac.uk/home/gdp/publications/`. 63

[Gim95] Eduardo Gimenez. Codifying guarded definitions with recursive schemes. In J. Smith, B. Nordström, and P. Dybjer, editors, *Proc. Types'94 (Bastad)*, volume 996 of *Lecture Notes in Computer Science*, pages 35–59. Springer, 1995. Available from `http://pauillac.inria.fr/~gimenez/papers.html`. 62

[Joa01a] Felix Joachimski. Confluence of the coinductive lambda-calculus. Submitted to *Theoretical Computer Science*, available from `http://www.mathematik.uni-muenchen.de/~joachski`, September 2001. 60, 63, 71

[Joa01b] Felix Joachimski. *Reduction Properties of ΠIE-Systems*. PhD thesis, LMU München, 2001. Available from `http://www.mathematik.uni-muenchen.de/~joachski`. 63

[KKSdV97] Richard Kennaway, Jan-Willem Klop, Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theoretical Computer Science*, 175(1):93–125, 1997. Available from `http://www.sys.uea.ac.uk/~jrk/`. 60

[KMS75] G. Kreisel, G. E. Mints, and S. G. Simpson. The use of abstract language in elementary metamathematics: Some pedagogic examples. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 38–131. Springer, 1975. 59

[Min78] Grigori E. Mints. Finite investigations of transfinite derivations. *Journal of Soviet Mathematics*, 10:548–596, 1978. Translated from: Zap. Nauchn. Semin. LOMI 49 (1975). Cited after Grigori Mints. *Selected papers in Proof Theory*. Studies in Proof Theory. Bibliopolis, 1992. 59

[Ruc85] Martin Ruckert. *Church–Rosser Theorem und Normalisierung für Termkalküle mit unendlichen Termen unter Einschluß permutativer Reduktionen*. PhD thesis, Mathematisches Institut der LMU München, 1985. 59

[Sch98] Helmut Schwichtenberg. Finite notations for infinite terms. *Annals of Pure and Applied Logic*, 94:201–222, 1998. Available from `http://www.mathematik.uni-muenchen.de/~schwicht`. 59, 71

[TT97] Alastair Telford and David Turner. Ensuring Streams Flow. In Michael Johnson, editor, *Proc. 6th AMAST 1997 (Sydney)*, volume 1349 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1997. Available from `http:// www.cs.ukc.ac.uk/people/staff/ajt/`. 63

Appendix: Implementation in `Haskell`

```
data Term = Var Integer | App Term Term | Lam Term |
            Rep Term | Bet Term deriving Show

lift :: Term -> Integer -> Term
lift (Var k)   n = Var (k + if k < n then 0 else 1)
lift (App r s) n = App (lift r n) (lift s n)
lift (Lam r)   n = Lam (lift r (n + 1))
lift (Rep r)   n = Rep (lift r n)
lift (Bet r)   n = Bet (lift r n)

subst :: Term -> Term -> Integer -> Term
```

```
subst (Var k)    s n | k == n = s
subst (Var k)    s n = Var (k - if k < n then 0 else 1)
subst (App r r') s n = App (subst r s n) (subst r' s n)
subst (Lam r)    s n = Lam (subst r (lift s 0) (n + 1))
subst (Rep r)    s n = Rep (subst r s n)
subst (Bet r)    s n = Bet (subst r s n)

beta :: Term -> Term
beta r = app r []

app :: Term -> [Term] -> Term
app (Lam r) (s:l) = Bet (app (subst r s 0) l)
app (Lam r)  []   = Lam (beta r)
app (Rep r)   l   = Rep (app r l)
app (Bet r)   l   = Bet (app r l)
app (Var k)   l   = foldl App (Var k) (map beta l)
app (App r s) l   = Rep (app r (s:l))
```

Examples.

```
s = Lam (Lam (Lam (Var 2 `App` (Var 0) `App` (Var 1 `App` (Var 0)))))
k = Lam (Lam (Var 1))
y = Lam (Lam (Var 1 `App` (Var 0 `App` (Var 0))) `App`
        (Lam (Var 1 `App` (Var 0 `App` (Var 0)))))
yco r = r `App` (yco r)
theta = Lam (Lam (Var 0 `App` (Var 1 `App` (Var 1) `App` (Var 0)))) `App`
        Lam (Lam (Var 0 `App` (Var 1 `App` (Var 1) `App` (Var 0))))
church n = Lam (Lam (iterate (App (Var 1)) (Var 0) !! n))
```

Here are some test runs of the program:

```
Main> beta (s `App` k `App` k)
  Rep (Rep (Bet (Bet (Lam (Rep (Rep (Bet (Bet (Var 0)))))))))
Main> beta (yco k)
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Bet (Lam (Rep (Bet (Lam
                (Rep (Bet (Lam (Rep (Bet (Lam (Rep {Interrupted!}
Main> beta (y `App` k)
  Rep (Bet (Rep (Bet (Rep (Bet (Lam (Rep (Bet (Rep (Bet (Lam
           (Rep (Bet (Rep (Bet (Lam (Rep (Bet (Rep {Interrupted!}
Main> beta (theta `App` k)
  Rep (Rep (Bet (Bet (Rep (Bet
      (Lam (Rep (Rep (Bet (Bet (Rep (Bet (Lam
           (Rep (Rep (Bet (Bet (Rep (Bet (Lam {Interrupted!}
Main> beta (church 2 `App` (church 3))
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Rep (Bet (Bet (Rep (App (Var 1)
      (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
      (App (Var 1) (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep
      (Bet (Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (App (Var 1)
      (Var 0))) [..])
Main> beta (church 3 `App` (church 2))
  Rep (Bet (Lam (Rep (Bet (Lam (Rep (Rep (Bet (Bet (Rep (Rep (Bet
      (Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
```

```
(App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep (Rep (Bet
(Bet (Rep (App (Var 1) (Rep (App (Var 1) (Rep (Rep (Bet (Bet (Rep
(App (Var 1) (Rep (App (Var 1) (Var 0))) [..] )
```

Variants of Realizability for Propositional Formulas and the Logic of the Weak Law of Excluded Middle

Alexey V. Chernov[1], Dmitriy P. Skvortsov[2], Elena Z. Skvortsova[3], and Nikolai K. Vereshchagin[1]

[1] Dept. of Mathematical Logic and Theory of Algorithms, Moscow State University
Leninskie Gory, Moscow, 119992, Russia
{chernov,ver}@mccme.ru

[2] All-Russian Institute of Technical and Scientific Information
ul. Usievicha 20a, Moscow, Russia
skvortsov@viniti.ru

[3] All-Russian Multisubject School, Moscow State University
Leninskie Gory, Moscow, 119992, Russia

Abstract. It is unknown, whether the logic of propositional formulas that are realizable in the sense of Kleene has a finite or recursive axiomatization. In this paper another approach to realizability of propositional formulas is studied. This approach is based on the following informal idea: a formula is realizable if it has a "simple" realization for each substitution. More precisely, logical connectives are interpreted as operations on sets of natural numbers and a formula is interpreted as a combined operation; if some sets are substituted for variables, then elements of the result are called realizations. A realization (a natural number) is simple if it has low Kolmogorov complexity, and a formula is called realizable if it has at least one simple realization whatever sets are substituted. Similar definitions may be formulated in arithmetical terms. A few "realizabilities" of this kind are considered and it is proved that all of them give the same finitely axiomatizable logic, namely, the logic of the weak law of excluded middle.

Keywords: realizability; Kolmogorov complexity; superintuitionistic logics.

1 Introduction

1.1 Preliminary Notes

Kolmogorov in [5] proposed a constructive semantics for the propositional intuitionistic calculus, the so-called "calculus of problems". The main idea is the following. Let us fix a set of "elementary" problems and interpret the propositional connectives ($\vee$, $\wedge$, $\rightarrow$) as natural operations on this set ("to solve one of the problems", "to solve both problems", "to solve the second problem if any

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 74–88, 2002.
© Springer-Verlag Berlin Heidelberg 2002

solution of the first problem is known"). Thus, substituting some problems for propositional variables in a formula, we get a combined problem. And if a formula is intuitionistically deducible, then the combined problem assigned to the formula has a common solution for all possible substitutions.

Kolmogorov did not define exactly what a "problem" is, he only gave some examples. Afterwards there were several attempts to construct a formal semantics for intuitionism based on Kolmogorov's ideas, for instance, Kleene realizability (see [4, §82]) and Medvedev's logic of finite problems (see [8, 9, 10]). However the intuitionistic propositional calculus turned out to be incomplete with respect to these interpretations. Moreover, the logic of finite problems has no finite axiomatization (see [7]), and it is unknown, whether it has a recursive axiomatization. For the logic of Kleene realizability both questions are open.

In this paper we consider a few new interpretations of the following kind. Let us fix some complexity measure on problems. We say that a formula is realizable if the complexity of the combined problem is bounded by some fixed function of the complexities of the substituted elementary problems. Changing the class of elementary problems, the complexity measure, and the bounding function, we can get various "realizabilities". We consider several definitions of this kind; they lead to the same set of realizable formulas.

1.2 Definitions and Results

Propositional formulas consist of variables p, q (with indices), the constant $\bot$ ("false") and the connectives $\vee$, $\wedge$, $\rightarrow$. The common abbreviations $(\Phi \leftrightarrow \Psi) \rightleftharpoons (\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$, $\neg\Psi \rightleftharpoons (\Psi \rightarrow \bot)$, $\top \rightleftharpoons (\bot \rightarrow \bot)$ also will be used. *Positive* formulas are formulas that do not contain the constant $\bot$ (and the connective $\neg$). Propositional formulas will usually be denoted by capital Greek letters Φ, Ψ (and arithmetical formulas will be denoted by ϕ, ψ).

$\mathfrak{Int}$ denotes the intuitionistic propositional calculus (with *modus ponens* and substitution). A *superintuitionistic logic* is a set of propositional formulas that is closed under deduction in $\mathfrak{Int}$. We write $L \vdash \Phi$ if the formula Φ belongs to the logic L. If L is a logic and Γ is a set of formulas, then the least superintuitionistic logic containing the set $(L \cup \Gamma)$ is denoted by $(L + \Gamma)$. The set of all positive formulas of a logic L is called the *positive fragment* of L and is denoted by L_Π. We say that a logic L *has the intuitionistic positive fragment* if $L_\Pi = \mathfrak{Int}_\Pi$. In the sequel, we need the so-called Jankov logic (or the logic of the weak law of excluded middle); it is the superintuitionistic logic $\mathfrak{J} = \mathfrak{Int} + \{\neg p \vee \neg\neg p\}$, which was considered by Jankov in [3].

A *problem* is an arbitrary set of natural numbers, a *solution* of the problem is any element of this set. We thus identify a problem with the set of its solutions encoded by natural numbers.

Let us define operations on problems corresponding to the logical connectives. To this end we fix some effective enumeration U of all partial computable functions from $\mathbb{N}$ to $\mathbb{N}$. We assume that U has the following property: for every computable partial function $V(e, x)$ there is a total computable function $f(e)$ such that $V(e, x) = U(f(e), x)$ for all e, x (this provides s-m-n-theorem). For

brevity we write $e(x)$ instead of $U(e,x)$. As $e(x)$ specifies a computable function of x for any fixed e, we often say that e is a program for this function. Let us also fix an effective enumeration of all pairs $\langle x,y\rangle$ and of all sequences $\langle x_1,\dots,x_k\rangle$.

Definition 1. *Let* $X,Y\subseteq\mathbb{N}$.

$$\begin{aligned} X\wedge Y &\rightleftharpoons \{\langle x,y\rangle \mid x\in X, y\in Y\};\\ X\vee Y &\rightleftharpoons \{\langle 0,x\rangle \mid x\in X\}\cup\{\langle 1,y\rangle \mid y\in Y\};\\ X\to Y &\rightleftharpoons \{e\in\mathbb{N} \mid \forall x\in X \quad e(x)\in Y\};\\ \bot &\rightleftharpoons \emptyset,\\ \neg X &\rightleftharpoons X\to\bot \rightleftharpoons X\to\emptyset. \end{aligned}$$

The set $\Phi(X_1,\dots,X_n)$ is defined by induction for any formula $\Phi(X_1,\dots,X_n)$ and for any sets $X_1,\dots,X_n$. This set is said to be the result of substituting the sets $X_1,\dots,X_n$ for the variables $p_1,\dots,p_n$ in the formula Φ.

Now we define Kleene realizability for propositional formulas. It is convenient to do this using *a set of realizations of a closed arithmetical formula.* Suppose the formula ϕ is atomic, then the set $R(\phi)$ of its realizations is the set $\{0\}$ if the formula ϕ is true and $\emptyset$ otherwise. Let $R(\phi\circ\psi)\rightleftharpoons R(\phi)\circ R(\psi)$, where $\circ$ is $\vee$, $\wedge$ or $\to$; $R(\forall x\phi(x))\rightleftharpoons\{e \mid \forall k\in\mathbb{N} \quad e(k)\in R(\phi(k))\}$, and $R(\exists x\phi(x))\rightleftharpoons\{\langle a,k\rangle \mid a\in R(\phi(k))\}$. We say that a number e *realizes* a closed formula ϕ if $e\in R(\phi)$ (this definition is equivalent to the definition of realizability from [4, §82]); a number r realizes a formula $\phi(x_1,\dots,x_n)$ with free variables if $r(\langle k_1,\dots,k_n\rangle)\in R(\phi(k_1,\dots,k_n))$ for all $k_1,\dots,k_n\in\mathbb{N}$. An arithmetical formula is called *realizable* if it has a realization.

A propositional formula $\Phi(p_1,\dots,p_n)$ is called *realizable* if the arithmetical formula $\Phi(\phi_1,\dots,\phi_n)$ is realizable for all arithmetical formulas $\phi_1,\dots,\phi_n$ (possibly with free variables). The set of all realizable propositional formulas will be denoted by $\mathfrak{R}$.

If for all closed arithmetical formulas $\phi_1,\dots,\phi_n$ a realization of $\Phi(\phi_1,\dots,\phi_n)$ can be found effectively, then Φ is called *effectively realizable.* If there is an r that realizes $\Phi(\phi_1,\dots,\phi_n)$ for all closed $\phi_1,\dots,\phi_n$ then Φ is called *constantly (or uniformly) realizable.* The set of effectively realizable formulas is denoted by $\mathfrak{R}_{\mathrm{eff}}$ and the set of constantly realizable formulas is denoted by $\mathfrak{R}_{\mathrm{const}}$.

It is easy to see that $\mathfrak{R}$, $\mathfrak{R}_{\mathrm{eff}}$, $\mathfrak{R}_{\mathrm{const}}$ are superintuitionistic logics (it follows from Nelson's theorem in [4]). Obviously, $\mathfrak{R}_{\mathrm{const}}\subseteq\mathfrak{R}_{\mathrm{eff}}\subseteq\mathfrak{R}$. In [13] Rose showed that $\mathfrak{R}_{\mathrm{const}}\neq\mathfrak{Int}$. The natural question is whether these logics can be described axiomatically (with a finite or enumerable set of axioms). Unfortunately, the answer is unknown. But an interesting property was discovered by Medvedev in [9] (Medvedev's original proof contains an error; in [11] Plisko gave a correct proof).

Theorem 1 (Medvedev, 1963; Plisko, 1973). *The logics* $\mathfrak{R}$, $\mathfrak{R}_{\mathrm{eff}}$, $\mathfrak{R}_{\mathrm{const}}$ *have the intuitionistic positive fragment.*

All the new notions of realizability defined in this paper will have the following form: we say $\Phi(p_1,\dots,p_n)$ is realizable if the complexity of the set

$\Phi(A_1, \ldots, A_n)$ is related somehow to that of the sets $A_1, \ldots, A_m$. Depending on the class of sets $A_1, \ldots, A_m$ allowed for substitution and on the complexity measure in question we obtain several versions.

In the first bunch of new realizabilities, substituted sets are arithmetical ones and complexity of A is measured by the level of A in the arithmetical hierarchy. Let us reformulate Kleene's first realizability in this vein. We say that a family of sets $A(k) \subset \mathbb{N}$, $k \in \mathbb{N}$, is *arithmetical* if there is an arithmetical formula $\phi(x, y)$ such that $A(k) = \{m \mid \phi(m, k)\}$.

Proposition 1. *A propositional formula $\Phi(p_1, \ldots, p_n)$ is realizable iff for arbitrary arithmetical families of sets $A_1(x), \ldots, A_n(x)$ there exists a number r (a realization) such that $r(k) \in \Phi(A_1(k), \ldots, A_n(k))$ for any $k \in \mathbb{N}$.*

Let us present the weakest[1] non-trivial definition of this kind.

Definition 2. *A propositional formula $\Phi(p_1, \ldots, p_n)$ is* weakly realizable *(belongs to the set $\mathfrak{R}_{\mathrm{w}}$) if for some $i > 0$ and for arbitrary arithmetical families $A_1(x), \ldots, A_n(x)$ there is an arithmetical family $B(x)$ such that $B(x)$ (more formally, an arithmetical formula that specifies $B(x)$) belongs to the class Σ_i of the arithmetical hierarchy and for all $k \in \mathbb{N}$ the set $B(k)$ is finite and intersects with $\Phi(A_1(k), \ldots, A_n(k))$.*

The crucial difference of this definition with Kleene's is that we do not require $B(k)$ to be a singleton. Note that any realizable formula is weakly realizable: we take $i = 1$ and $B(k)$ consisting of a single element for any k. The strongest non-trivial definition of this kind is as follows.

Definition 3. *A propositional formula $\Phi(p_1, \ldots, p_n)$ belongs to the set $\mathfrak{R}_{O(1)}$ if there is a number C such that for any arithmetical sets $A_1, \ldots, A_n$ there exists a natural number $r \leq C$ such that $r \in \Phi(A_1, \ldots, A_n)$.*

Note that the definition of $\mathfrak{R}_{O(1)}$ is similar to that of $\mathfrak{R}_{\mathrm{const}}$. There are other options to define realizabilities of this kind, but the corresponding logics are intermediate between $\mathfrak{R}_{O(1)}$ and $\mathfrak{R}_{\mathrm{w}}$, and we shall prove that $\mathfrak{R}_{O(1)}$ and $\mathfrak{R}_{\mathrm{w}}$ are equal.

The definitions immediately imply that $\mathfrak{R}_{\mathrm{w}}$ and $\mathfrak{R}_{O(1)}$ are superintuitionistic logics, $\mathfrak{R} \subseteq \mathfrak{R}_{\mathrm{w}}$, $\mathfrak{R}_{\mathrm{const}} \subseteq \mathfrak{R}_{O(1)}$, and $\mathfrak{R}_{O(1)} \subseteq \mathfrak{R}_{\mathrm{w}}$.

Theorem 2. $\mathfrak{R}_{\mathrm{w}} = \mathfrak{R}_{O(1)} = \mathfrak{J}$.

The second approach was proposed by A. Shen (see [14]). First, we will substitute arbitrary sets, not only arithmetical ones (this idea for predicate formulas was considered by Plisko in [12]). Second, the complexity of a set is defined as the minimum Kolmogorov complexity of its elements.

Informally, the *Kolmogorov complexity* $K(x)$ of a number x is the length of the shortest description of x. Formally, we fix any computable partial function F

[1] The word "weakest" means that the number of realizable formulas is maximal.

such that for every computable partial function G there is a constant c such that $\forall e \exists e'(\ell(e') \leq \ell(e)+c,\ F(e') = G(e))$, where $\ell(e)$ is the length of the binary representation of the number e. It is easy to prove that such functions exist, see [6]. Then we put $K(x) \rightleftharpoons \min\{\ell(e) \mid F(e) = x\}$.

We state a few important properties of Kolmogorov complexity (they are proved in the monograph [6]):

1. $\exists c \forall x\, K(x) \leq \ell(x) + c$;
2. for any partial computable function f $\quad \exists c \forall x\, K(f(x)) \leq K(x) + c$;
3. $\forall x, y \quad K(\langle x, y\rangle) \leq K(x) + K(y) + O(\log(K(x) + K(y)))$;
4. the set $\{\langle x, n\rangle \mid K(x) < n\}$ is recursively enumerable;
5. the set $\{x \mid K(x) < n\}$ contains at most $2^n - 1$ elements.

Let the *Kolmogorov complexity of a set* X be $K(X) \rightleftharpoons \min\{K(x) \mid x \in X\}$ (and $K(\emptyset) = \infty$).

It can be easily proved (by induction, using the properties 2, 3) that

$$K(\Phi(X_1, \ldots, X_n)) \leq K(\bigwedge_{X_i \neq \emptyset} X_i) + O(1) \leq \sum_{X_i \neq \emptyset} K(X_i) + O(\log \sum_{X_i \neq \emptyset} K(X_i))$$

for any propositional formula $\Phi(p_1, \ldots, p_n)$ and for any sets $X_1, \ldots, X_n$ such that $\Phi(X_1, \ldots, X_n) \neq \emptyset$.

Definition 4.

$$L_{O(1)} = \{\Phi \mid K(\Phi(X_1, \ldots, X_n)) = O(1)\}$$
$$L_{o(\Sigma)} = \{\Phi \mid K(\Phi(X_1, \ldots, X_n)) = o(\sum_{X_i \neq \emptyset} K(X_i))\}$$

It follows from the definition that $L_{O(1)}$ and $L_{o(\Sigma)}$ are superintuitionistic logics, $L_{O(1)} \subseteq L_{o(\Sigma)}$, $L_{O(1)} \subseteq \mathfrak{R}_{O(1)}$.

Theorem 3. $L_{O(1)} = L_{o(\Sigma)} = \mathfrak{J}$.

In the third approach, we substitute only finite sets for variables and measure the complexity of finite sets as follows. Fix some computable enumeration of all finite sets of natural numbers. Let the complexity $\tilde{K}(X)$ of a finite set X be the Kolmogorov complexity of its number in this enumeration. Note that $\tilde{K}(\emptyset)$ is finite in contrast to $K(\emptyset) = \infty$. Note also that a set $\Phi(X_1, \ldots, X_n)$ can be infinite even for finite $X_1, \ldots, X_n$, therefore the complexity of $\Phi(X_1, \ldots, X_n)$ must be measured as earlier.

Definition 5. *A propositional formula $\Phi(p_1, \ldots, p_n)$ belongs to the set $\tilde{L}_{o(\Sigma)}$ if $K(\Phi(X_1, \ldots, X_n)) = o(\tilde{K}(X_1) + \ldots + \tilde{K}(X_n))$ for all finite sets $X_1, \ldots, X_n$.*

In contrast to the previous cases, it is not obvious that $\tilde{L}_{o(\Sigma)}$ is a superintuitionistic logic. Nevertheless the following theorem is true.

Theorem 4. $\tilde{L}_{o(\Sigma)} = \mathfrak{J}$.

Let us represent relations between the described logics on two diagrams. On the first one, we represent relations that are clear immediately from the definitions ($A \longrightarrow B$ denotes $A \subseteq B$).

$$\begin{tikzcd}
\mathfrak{Int} \arrow[r] \arrow[d] & \mathfrak{R}_{\mathrm{const}} \arrow[r] \arrow[d] & \mathfrak{R}_{\mathrm{eff}} \arrow[r] & \mathfrak{R} \arrow[d] \\
L_{O(1)} \arrow[r] \arrow[d] & \mathfrak{R}_{O(1)} \arrow[r] & \dots \arrow[r] & \mathfrak{R}_{\mathrm{w}} \\
L_{o(\Sigma)} \arrow[r] & \tilde{L}_{o(\Sigma)} & &
\end{tikzcd}$$

Our results significantly simplify this scheme, showing that many inclusions here are actually equalities:

$$\mathfrak{Int} \subset \mathfrak{R}_{\mathrm{const}} \subseteq \mathfrak{R}_{\mathrm{eff}} \subseteq \mathfrak{R} \subset \mathfrak{R}_{\mathrm{w}} = \mathfrak{R}_{O(1)} = L_{O(1)} = L_{o(\Sigma)} = \tilde{L}_{o(\Sigma)} = \mathfrak{J} \ .$$

The rest of the paper is organized as follows. The proofs of Theorems 2, 3, 4 (and Plisko's proof of the Theorem 1) are based on Medvedev's characterization of logics with the intuitionistic positive fragment. In the next section we formulate this and some other logical results, which will be used. In the Appendix A we prove Medvedev's theorem, because no proof has been published yet and we need a stronger formulation than Medvedev's original one. Section 3 is devoted to properties of the weak realizabilities and contains proofs of Theorems 2, 3, 4. The proofs use one technical lemma; its proof is given in Appendix B.

2 Logics with the Intuitionistic Positive Fragment

Medvedev in [8] proposed a convenient criterion characterizing whether all positive formulas of a given superintuitionistic logic are deducible in $\mathfrak{Int}$. Using this criterion, Medvedev proved that the logic of finite problems has the intuitionistic positive fragment. We will use it for logics of weak realizabilities.

Definition 6 (Medvedev, 1962). A critical implication J *is a positive formula that has the form*[2]

$$J = \bigwedge_{i=1}^{k} ((P_i \to Q_i) \to Q_i) \to R \ ,$$

where P_i are conjunctions of variables, Q_i and R are disjunctions of variables, for all i, the formulas P_i and Q_i have no common variables and none of P_i, Q_i, R is empty.

It can be easily checked that critical implications are not deducible in $\mathfrak{Int}$.

[2] We keep Medvedev's notation for critical implications and their subformulas.

Theorem 5 (Medvedev, 1962). *Let Φ be an arbitrary positive formula such that $\mathfrak{Int} \nvdash \Phi$. Then there exists a critical implication J such that $(\mathfrak{Int} + \Phi) \vdash J$.*

We need a stronger statement. For every $n > 0$ fix the weakest[3] critical implication J_n in the variables $p_1, \ldots, p_n$:

$$J_n = \bigwedge_{\emptyset \neq E \subset \{1,\ldots,n\}} ((\bigwedge_{j \notin E} p_j \to \bigvee_{i \in E} p_i) \to \bigvee_{i \in E} p_i) \to \bigvee_{i=1}^{n} p_i. \tag{1}$$

Theorem 6. *Let $\Phi(q_1, \ldots, q_m)$ be a positive formula such that $\mathfrak{Int} \nvdash \Phi$. Then $\mathfrak{Int} \vdash (\Phi^* \to J_n)$ for some $n > 0$, where Φ^* is the result of substituting some formulas of the form $\vee(\wedge p_i)$ (p_i are variables of J_n) for the variables $q_1, \ldots, q_m$ in Φ.*

This theorem is proved in Appendix A.

To prove the main results we need another criterion, which was proved by Jankov in [3].

Theorem 7 (Jankov, 1968). *A superintuitionistic logic L has the intuitionistic positive fragment iff $L \subseteq \mathfrak{J}$, where $\mathfrak{J} = \mathfrak{Int} + \{\neg p \vee \neg\neg p\}$.*

In other words, the logic $\mathfrak{J}$ is the greatest logic with the intuitionistic positive fragment. This criterion is convenient for axiomatically specified logics (note that the logic $\mathfrak{J}$ is decidable). Conversely, Medvedev's criterion is more convenient for semantically specified logics (as logics of realizability). To prove our results we use both criteria. First, using Medvedev's criterion, we prove that a logic L (one of the weak realizability logics) has the intuitionistic positive fragment. Then, using Jankov's criterion, we prove that $L \subseteq \mathfrak{J}$.

3 Weak Realizabilities

In this section we prove our main results.

To prove that $\mathfrak{R}_{\mathrm{w}}$ and $\mathfrak{R}_{O(1)}$ are closed under substitution we note that substituting arithmetical sets $A_1, \ldots, A_n$ in a propositional formula $\Phi(\Psi_1, \ldots, \Psi_k)$ is equivalent to substituting the arithmetical sets $\Psi_i(A_1, \ldots, A_n)$ in Φ. The closure under *modus ponens* is obvious: applying all "possible realizations" of $\Psi \to \Phi$ to all "possible realizations" of Ψ, we get a set of "possible realizations" of Φ.

Let us consider $L_{O(1)}$, $L_{o(\Sigma)}$, $\tilde{L}_{o(\Sigma)}$. Obviously, $L_{O(1)} \subseteq L_{o(\Sigma)}$. Since for nonempty finite sets $K(X) \leq \tilde{K}(X) + O(1)$, we get $L_{o(\Sigma)} \subseteq \tilde{L}_{o(\Sigma)}$. The inclusion $L_{O(1)} \subseteq \mathfrak{R}_{O(1)}$ follows from the property 5 of Kolmogorov complexity. Each formula from $\mathfrak{Int}$ has a realization, which does not depend on substituted sets, and therefore $\mathfrak{Int} \subseteq L_{O(1)}$.

It holds $K(\Phi) \leq K(\Psi \to \Phi) + K(\Psi) + O(\log(K(\Psi)))$ (it follows from the properties 3, 2), hence $L_{O(1)}$, $L_{o(\Sigma)}$, $\tilde{L}_{o(\Sigma)}$ are closed under *modus ponens*. The closure

[3] For any other critical implication J with the same variables we have $\mathfrak{Int} \vdash J \to J_n$.

of $L_{O(1)}$, $L_{o(\Sigma)}$ under substitution follows from the bound $K(\Phi(X_1,\dots,X_n)) \le K(\bigwedge_{X_i\neq\emptyset} X_i) + O(1) \le \sum_{X_i\neq\emptyset} K(X_i) + O(\log \sum_{X_i\neq\emptyset} K(X_i))$. Thus $L_{O(1)}$, $L_{o(\Sigma)}$ are superintuitionistic logics. We cannot prove this for $\tilde{L}_{o(\Sigma)}$ immediately: if a formula contains an implication, then the corresponding set may be infinite even for finite substituted sets, and therefore the closure under substitution is not so obvious. However $\tilde{L}_{o(\Sigma)}$ is closed under a restricted substitutions. More specifically, we can substitute formulas without implications, as it holds that $\tilde{K}(\bigwedge Y_i) \le \sum \tilde{K}(Y_i) + O(\log \sum \tilde{K}(Y_i))$. The last bound can be proved by induction, using the trivial inequalities $\tilde{K}(\bot) = O(1)$, $\tilde{K}(X\wedge Y) \le \tilde{K}(X)+\tilde{K}(Y)+O(\log(\tilde{K}(X)))$, $\tilde{K}(X\vee Y) \le \tilde{K}(X)+\tilde{K}(Y)+O(\log(\tilde{K}(X)))$.

Now we proceed to relations between the Jankov logic $\mathfrak{J}$ and weak realizabilities. It follows from Jankov's criterion (Theorem 7) and Theorem 1 that the logic of (Kleene) realizability $\mathfrak{R}$ is a subset of $\mathfrak{J}$. It can be easily checked that $\neg p \vee \neg\neg p \notin \mathfrak{R}$, and therefore $\mathfrak{R} \neq \mathfrak{J}$.

Lemma 1.

1. $\neg p \vee \neg\neg p \in L_{O(1)}$;
2. $p \vee \neg p \notin L_{o(\Sigma)}$;
3. $p \vee \neg p \notin \mathfrak{R}_{\mathrm{w}}$.

Proof.

1. If $X = \emptyset$, then $\neg X = \mathbb{N}$; if $X \neq \emptyset$, then $\neg X = \emptyset$ and $\neg\neg X = \mathbb{N}$. Hence $K(\neg X \vee \neg\neg X) \le \max\{K(\langle 0,0\rangle), K(\langle 1,0\rangle)\} = O(1)$.
2. Let $X = \{x \mid K(x) = n\}$. Then $K(X) = n$, $\neg X = \emptyset$, and $K(X \vee \neg X) = K(\{0\}\times X) = n + O(1) \neq o(n)$.
3. Let us fix an arbitrary number $i > 0$ and any arithmetical enumeration $B_1(x), B_2(x), \dots$ of all families of sets from Σ_i. Let us take the arithmetical family of sets $D(x) = \{r \mid \langle 0,r\rangle \notin B_x(x)\}$.
   Assume that a family $B_k(x) \in \Sigma_i$ weakly realizes $D(x) \vee \neg D(x)$, i. e., for any m the set $B_k(m)$ is finite and $B_k(m) \cap (D(m) \vee \neg D(m)) \neq \emptyset$. Consider the set $D(k)$. Since $B_k(k)$ is finite, the set $D(k)$ is not empty, and the set $\neg D(k)$ is empty. Then $D(k) \vee \neg D(k) = \{\langle 0,r\rangle \mid r \in D(k)\}$, and $B_k(k) \cap (D(k) \vee \neg D(k)) = \{\langle 0,r\rangle \mid r \in D(k) \text{ and } \langle 0,r\rangle \in B_k(k)\} = \emptyset$. Thus B_k does not weakly realize $D \vee \neg D$, and this contradiction proves that the formula $p \vee \neg p$ does not belong to $\mathfrak{R}_{\mathrm{w}}$.

□

This lemma implies, in particular, that $\mathfrak{Int} \neq L_{O(1)}$, and $L_{o(\Sigma)}$, $\mathfrak{R}_{\mathrm{w}}$ are strictly contained in the set of classically true formulas. In addition, the logic $L_{O(1)}$ (and therefore $L_{o(\Sigma)}$, $\tilde{L}_{o(\Sigma)}$, $\mathfrak{R}_{O(1)}$, $\mathfrak{R}_{\mathrm{w}}$) includes the Jankov logic $\mathfrak{J}$.

Thus, to prove Theorems 2 and 3 it is sufficient to prove that $L_{o(\Sigma)}$ and $\mathfrak{R}_{\mathrm{w}}$ are contained in $\mathfrak{J}$, i. e., have the intuitionistic positive fragment. By Theorem 6 we must prove that no critical implication belongs to these logics.

The proof is based on the following lemma. We say that a program q *enumerates* a set A if $A = \{q(i) \mid i \in \mathbb{N}\}$, and a program q *co-enumerates* a set A if q enumerates the complement of A.

Lemma 2. *Given natural numbers m, n and a program q that enumerates a set $M \subset \mathbb{N}$ of cardinality not greater than 2^m we can effectively construct programs $a_1, \ldots, a_n$ that co-enumerate non-empty sets $A_1, \ldots, A_n$ respectively such that M and $J_n(A_1, \ldots, A_n)$[4] are disjoint; in addition, any element of $A_1, \ldots, A_n$ is not greater than $C2^{Cm}$ where C depends on n only.*

The proof is given in Appendix B.

Theorem 8. *The logics $\mathfrak{R}_{\mathrm{w}}$ and $L_{o(\Sigma)}$ have the intuitionistic positive fragment.*

Proof.

1. By Theorem 6, it is sufficient to prove that for every n the critical implication J_n does not belong to $\mathfrak{R}_{\mathrm{w}}$. Fix an i. Let $D(x)$ be an arithmetical family of sets such that for all k the set $D(k)$ is finite and for every family $B(x) \in \Sigma_i$ there exists k such that $B(k)$ is infinite or $D(k) = B(k)$ (for example, $D(k) = B_k(k)$ for finite $B_k(k)$ and is empty otherwise). Applying Lemma 2 to the set $M = D(k)$ and $m = \lceil \log_2 |D(k)| \rceil$, we get sets $A_1(k), \ldots, A_n(k)$. It is clear that the relation $x \in A_j(k)$ is arithmetical for all $j \leq n$. By construction, for every k the set $D(k)$ is disjoint with $J_n(A_1(k), \ldots, A_n(k))$. Therefore for any family $B(x) \in \Sigma_i$ there exists $k \in \mathbb{N}$ such that the set $B(k)$ is infinite or $B(k)$ and $J_n(A_1(k), \ldots, A_n(k))$ are disjoint.

2. It is sufficient to prove that for every n the critical implication J_n does not belong to $L_{o(\Sigma)}$. Applying Lemma 2 to the set of numbers with Kolmogorov complexity less than m, we get finite non-empty sets $A_1^m, \ldots, A_n^m$ such that $K(A_i^m) \leq Cm + O(1)$, but $K(J_n(A_1^m, \ldots, A_n^m)) \geq m$. □

It remains to prove Theorem 4.

Proof (of Theorem 4). We know that $\mathfrak{J} \subseteq \tilde{L}_{o(\Sigma)}$. To prove that $\tilde{L}_{o(\Sigma)} \subseteq \mathfrak{J}$ we need a stronger version of Theorem 7. In [3], actually the following is proved. Suppose a formula $\Phi(q_1, \ldots, q_k)$ is not deducible in $\mathfrak{J}$. Then there is a positive formula Ψ that is not deducible in $\mathfrak{Int}$ and a formula Φ^* that is a result of substituting new variables and the constant $\perp$ for $q_1, \ldots, q_k$ in Φ such that $\mathfrak{Int} \vdash (\Phi^* \to \Psi)$.

Since in Theorem 6 only substitutions of the form $\vee(\wedge p_i)$ are used and the set $\tilde{L}_{o(\Sigma)}$ is closed under such substitutions and *modus ponens*, it is sufficient to prove that $J_n \notin \tilde{L}_{o(\Sigma)}$ for all n.

Let us take the sets $A_1^m, \ldots, A_n^m$ constructed in the second part of the previous theorem's proof. Lemma 2 says that $A_i^m \subseteq \{1, \ldots, C2^{Cm}\}$ and we have programs that co-enumerate these sets. If we know the exact cardinalities of A_i^m, then we know the cardinalities of $\{1, \ldots, C2^{Cm}\} \setminus A_i^m$ and can effectively find all elements of $\{1, \ldots, C2^{Cm}\} \setminus A_i^m$; hence we can find A_i^m (their numbers in the enumeration of all finite sets). Thus we have $\tilde{K}(A_i^m) \leq Cm + O(1)$. This completes the proof, as $K(J_n(A_1^m, \ldots, A_n^m)) \geq m$. □

[4] Recall that $J_n(p_1, \ldots, p_n)$ is the weakest critical implication in variables $p_1, \ldots, p_n$ defined by (1).

Acknowledgments

The new approach to realizability considered in this paper is proposed by Alexander Shen. The authors are grateful to Alexander Shen and Andrej A. Muchnik for useful discussions.

The authors were partially supported by the Russian Foundation for Basic Research grants 01-01-01028 and 01-01-00505.

References

[1] M. C. Fitting. *Intuitionistic Logic, Modal Theory and Forcing*. North-Holland, Amsterdam, 1969. 84

[2] V. A. Jankov. O svjazi mezhdu vyvodimost'ju v intuitsionistskom ischislenii vyskazyvanij i konechnymi implicativnymi strukturami. *Doklady AN SSSR*, v. 151, N. 6, 1963, pp. 1293–1294. 84

[3] V. A. Jankov. Ob ischislenii slabogo zakona iskluchennogo tret'jego. *Izvestija AN SSSR*, ser. matem., v. 32, N. 5, 1968, pp. 1044–1051. 75, 80, 82

[4] S. K. Kleene. *Introduction to metamathematics*. New York, 1952. 75, 76, 85

[5] A. Kolmogoroff. Zur Deutung der intuitionistishen Logik. *Mathematische Zeitschrift*, Bd. 35, H. 1, S. 57–65. 74

[6] M. Li, P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. New York, Springer-Verlag, 1997. 78

[7] L. L. Maksimova, D. P. Skvortsov, V. B. Shehtman. Nevozmozhnost' konechnoj aksiomatizatsii logiki finitnyh zadach Medvedeva. *Doklady AN SSSR*, v. 245, N. 5, 1979, pp. 1051–1054. 75

[8] Yu. T. Medvedev. Finitnye zadachi. *Doklady AN SSSR*, v. 142, N. 5, 1962, pp. 1015–1018. 75, 79

[9] Yu. T. Medvedev. Interpretatsija logicheskih formul posredstvom finitnyh zadach i eyo svjaz' s teoriej realizuemosti. *Doklady AN SSSR*, v. 148, N. 4, 1963, pp. 771–774. 75, 76

[10] Yu. T. Medvedev. Ob interpretatsii logicheskih formul posredstvom finitnyh zadach. *Doklady AN SSSR*, v. 169, N. 1, 1966, pp. 20–24. 75

[11] V. E. Plisko. O realizuemyh predikatnyh formulah. *Doklady AN SSSR*, v. 212, N. 3, 1973, pp. 553–556. 76

[12] V. E. Plisko. Nekotorye varianty ponjatija realizuemosti dlja predikatnyh formul. *Izvestija AN SSSR*, ser. matem., v. 42, N. 3, 1978, pp. 636–653. 77

[13] G. F. Rose. Propositional calculus and realizability. *Transactions of the American Mathematical Society*, v. 75, N. 1, 1953, pp. 1–19. 76

[14] A. Shen, N. Vereshchagin. Logical operations and Kolmogorov complexity. *Theoretical Computer Science*, v. 271, 2002, p. 125–129. 77

A Proof of Medvedev's Theorem

The proof is divided into three lemmas. We begin with some notation. Let F be a finite Kripke frame. The Heyting algebra of this frame is denoted by $H(F)$ (the maximal and minimal elements of $H(F)$ are denoted by $\mathbf{1}$ and $\mathbf{0}$ respectively), and the logic of propositional formulas that are valid in $H(F)$ is denoted by $L(F)$

(about Kripke semantics see monograph [1]). Let $\sigma(F)$ be the Kripke frame consisting of all proper subsets of the set F ordered by inclusion. Every such frame is isomorphic to one of the frames $\sigma_n = \sigma(\{1, \ldots, n\})$.

Lemma 3. *If Φ is a positive formula and $\mathfrak{Int} \nvdash \Phi$, then for some n it holds that $\Phi \notin L(\sigma_n)$.*

The proof is omitted.

The idea of the next definition and lemma is taken from Jankov's paper [2].

Definition 7. *Let variables q_a correspond to elements a of an algebra $H(F)$. We say that $X_\Pi(F)$ is a* positive characteristic formula *of the frame F, if $X_\Pi(F) = Y_F \to q_\omega$, where $\omega \rightleftharpoons F \setminus \{0_F\}$ is the greatest non-identity element of $H(F)$, Y_F is the conjunction of all formulas of the forms*

$$q_a \wedge q_b \leftrightarrow q_{a \cap b},$$
$$q_a \vee q_b \leftrightarrow q_{a \cup b},$$
$$(q_a \to q_b) \leftrightarrow q_{a \to b},$$

where $a, b \in H(F)$.

Lemma 4. *Let Φ be a positive formula, F a finite Kripke frame. Then $\Phi \notin L(F) \Leftrightarrow (\mathfrak{Int}{+}\Phi) \vdash X_\Pi(F)$. And if $\Phi \notin L(F)$, then there exist $a_1, \ldots, a_k \in H(F)$ such that $\mathfrak{Int} \vdash \Phi(q_{a_1}, \ldots, q_{a_k}) \to X_\Pi(F)$.*

The proof is similar to one in [2].

Lemma 5. *For any n it holds $(\mathfrak{Int} + X_\Pi(\sigma_n)) \vdash J_n$.*

Moreover, $\mathfrak{Int} \vdash (X^*_\Pi(\sigma_n) \to J_n)$, where $X^*_\Pi(\sigma_n)$ is the result of substituting the constant $\top$ for the variable q_1 and $\bigvee_{E \in a}(\bigwedge_{i \in E} p_i)$ for other variables q_a in $X_\Pi(\sigma_n)$ ($\bigwedge_{i=1}^n p_i$ is substituted for $q_\emptyset$), where $p_1, \ldots, p_n$ are variables of J_n.

Proof. Let F denote the frame σ_n. For any $E \subseteq \{1, \ldots, n\}$ let P_E be the formula $P_E = \bigwedge_{i \in E} p_i$, $P_\emptyset = \top$. For any $a \in H(F)$ put $Q_a = \bigvee_{E \in a} P_E$ (in particular, $Q_F = P_\emptyset = \top$, $Q_\omega = \bigvee_{i=1}^n P_{\{i\}} = \bigvee_{i=1}^n p_i$), $Q_\emptyset = P_{\{1,\ldots,n\}}$. It is easy to see that $\mathfrak{Int} \vdash (P_E \wedge P_{E'} \leftrightarrow P_{E \cup E'})$, $\mathfrak{Int} \vdash (P_E \to P_{E'})$ for $E' \subseteq E$, $\mathfrak{Int} \vdash (Q_\emptyset \to Q_a)$ for all $a \in H(F)$.

Let X^* be the result of substituting formulas Q_a for variables q_a in $X_\Pi(F)$. Then $X^* = Y^* \to \bigvee\limits_{i=1}^{n} p_i$, where Y^* is the conjunction of the formulas (for all $a, b \in H(F)$):

$$(Q_a \wedge Q_b) \leftrightarrow Q_{a \cap b} \tag{2}$$
$$(Q_a \vee Q_b) \leftrightarrow Q_{a \cup b} \tag{3}$$
$$(Q_a \to Q_b) \leftrightarrow Q_{a \to b} \tag{4}$$

We must prove that $\mathfrak{Int} \vdash (X^* \to J_n)$. It is sufficient to prove that the premise of J_n implies the premise of X^*, i. e., $\mathfrak{Int} \vdash (\bigwedge\limits_{\emptyset \neq E \subset \{1,\ldots,n\}} Z_E \to C)$ for all C from the conjunction Y^*, where $Z_E = (\bigwedge\limits_{j \notin E} p_j \to \bigvee\limits_{i \in E} p_i) \to \bigvee\limits_{i \in E} p_i$.

It can be easily checked that formulas (2) and (3) are deducible in $\mathfrak{Int}$. Let us consider formulas (4). Put $a_E = \{E'' \mid E \subseteq E'' \subset \{1,\dots,n\}\}$ and $b_E = \{E'' \subset \{1,\dots,n\} \mid E'' \cap E \neq \emptyset\}$, where $\emptyset \neq E \subset \{1,\dots,n\}$. Then $Q_{a_E} = P_E = \bigwedge\limits_{i\in E} p_i$ and $Q_{b_E} = \bigvee\limits_{i\in E} p_i$. Every a from $H(F)$ (except $\mathbf{1}$ and $\mathbf{0}$) can be represented as a union of a_E, and every b (except $\mathbf{1}$) can be represented as an intersection of $b_{E'}$. Hence, it is sufficient to deduce formulas (4) for $a = a_E$, $b = b_{E'}$. (The remaining cases with formulas containing $\mathbf{1}$ and $\mathbf{0}$ as a and b are simple.)

If $E \cap E' \neq \emptyset$, then $\mathfrak{Int} \vdash (Q_{a_E} \to Q_{b_{E'}})$ and $a_E \subseteq b_{E'}$ (if $E'' \supseteq E$, then $E'' \cap E' \supseteq E'' \cap E' \cap E = E' \cap E \neq \emptyset$), that is $a_E \to b_{E'} = \mathbf{1}$ and $(Q_{a_E \to b_{E'}}) = \top$.

Let $E \cap E'$ is empty. Then we claim that $a_E \to b_{E'} = b_{E'}$. Indeed, suppose $E'' \notin b_{E'}$, i. e., $E' \cap E'' = \emptyset$. Then $(E \cup E'') \cap E' = \emptyset$, therefore $E'' \subseteq E \cup E'' \in (a_E \setminus b_{E'})$, and $E'' \notin (a_E \to b_{E'})$. Since $E \subseteq \{1,\dots,n\} \setminus E'$, we have that $Z_{E'}$ implies (in $\mathfrak{Int}$) $(\bigwedge\limits_{j\in E} p_j \to \bigvee\limits_{i\in E'} p_i) \to \bigvee\limits_{i\in E'} p_i$. Thus $\mathfrak{Int} \vdash (Z_{E'} \to [(Q_{a_E} \to Q_{b_{E'}}) \leftrightarrow Q_{b_{E'}}])$. □

To complete the proof of Theorem 6 we show that we can avoid substituting the constant $\top$. Indeed, in Lemma 5 it is substituted for the variable $q_{\mathbf{1}}$ only. Using Lemma 3, let us choose n such that $\Phi \notin L(\sigma_{n-1})$. We can consider the frame σ_{n-1} as a subframe of σ_n, then $a_1,\dots,a_k \in H(\sigma_n)$ in Lemma 4 are subsets of the subframe σ_{n-1}, and therefore they are not equal to $\mathbf{1}$.

B Proof of Lemma 2

We start with $A_1,\dots,A_n$ equal to the set of all natural numbers less than $K_1,\dots,K_n$, respectively. The numbers $K_1,\dots,K_n$ will be specified later. Then we run an algorithm $\mathfrak{A}$ that removes certain elements from those sets; the algorithm is given m,n,q. By definition $J_n(A_1,\dots,A_n)$ is equal to $(\bigwedge\limits_{i=1}^{I}((P_i \to Q_i) \to Q_i)) \to R$. For brevity, we omit arguments in formulas P_i, Q_i, we assume that variables $t_1,\dots,t_n$ are replaced by $A_1,\dots,A_n$. We will assume also that $R = \{\langle j,a\rangle \mid a \in A_j\}$. First we will define auxiliary programs f_{ilr} for $i = 1,\dots,I$, $l = 1,\dots,2^m$, $r = 1,2,\dots$. We want to define them so that for every $e \in M$ there be $l = l(e)$, $r = r(e)$ such that

$$f_{ilr} \in (P_i \to Q_i)$$

for all $i \le I$ and $e(\langle f_{1lr},\dots,f_{Ilr}\rangle) \notin R$. The result of the program f_{ilr} on the input s will be computed by the same algorithm $\mathfrak{A}$. Using the Recursion theorem (see, e.g. [4, § 66, Theorem XXVII]), we may assume that the algorithm knows all the programs f_{ilr}. Indeed, the result of the program f_{ilr} on an input s is computed by the algorithm $\mathfrak{A}$ given s and i,l,r,m,n,q. Thus we can find the program f_{ilr} given i,l,r,m,n,q and the program of the algorithm $\mathfrak{A}$. As the algorithm knows i,l,r,m,n,q, to find f_{ilr} it needs only its own program. The Recursion theorem just states (in one of its formulations) that we may assume that the algorithm knows its own program.

The algorithm $\mathfrak{A}$ works as follows. We first partition every set A_j in 2^m sets $A_{j1}, \ldots, A_{j2^m}$ of equal size. Then we enumerate the graph of the universal function $U(s,x)$. Without loss of generality we may assume that exactly one new value $U(s,x)$ appears on any step of that enumeration. After step t in the enumeration of the graph of U, the algorithm performs the following 5 steps denoted by $5t+1$, $5t+2$, $5t+3$, $5t+4$, $5t+5$. Let M^t stand for the part of M that has appeared on steps $1,\ldots,t$ in the enumeration of U; in the similar way we define U^t and $s^t(x) = U^t(s,x)$.

Step $5t+1$. If on the step t in the enumeration of U a new element e was enumerated into the set M then we let $l(e)$ to be the first number $l = 1,\ldots,2^m$ different from $l(e')$ for those $e' \in M$ that have appeared before e. Let also $r(e) = t$. Later the value $r(e)$ can increase but the value $l(e)$ will not change. The program e is declared *refuted*. Later, we may again declare it non-refuted. At the start all programs are declared non-refuted.

Step $5t+2$. If on the step t in the enumeration of U we find out that for some $e \in M^t$ the value $e^t(\langle f_{1l(e)r(e)},\ldots,f_{Il(e)r(e)}\rangle)$ is defined and is equal to some $\langle j,a\rangle$ then we remove a from A_j (if it is there). Thus the sets A_j will decrease and we will denote by A_j^t that part of A_j that is obtained after this step. We define A_{jl}^t, P_i^t, Q_i^t, R^t in the similar way.

Step $5t+3$. Assume that for some $i \le I$ and some s we have $s^t \in P_i^t \to Q_i^t$. For every t the sets $A_1^t,\ldots,A_n^t$ will be non-empty, therefore the sets P_i^t will be non-empty too. Thus there are only finitely many such programs s. For all such i,s and all $r \le t$, $l \le 2^m$ we define the value of f_{ilr} on s as follows. Let P_{il}^t stand for the set P_i with A_j replaced by A_{jl}^t and $s(P_{il}^t)$ for the set of all the results of the program s on tuples in P_{il}^t. We will define the initial cardinalities $A_1,\ldots,A_n$ in such a way that for all t the following inequalities are true:

$$\begin{aligned} &|A_{jl}^t| > n|A_{j+1}| \quad \text{for all } j < n, \\ &|A_{nl}^t| > 0 \end{aligned} \tag{5}$$

for all $l \le 2^m$. This implies that there is $\langle j,a\rangle \in s(P_{il}^t) \subset Q_i^t$ such that

> for all $k < j$ there are two tuples in P_{il}^t, differing in kth coordinate and mapped by s to $\langle j,a\rangle$.

Indeed, assume that there is no such $\langle j,a\rangle$. Then pick for every $\langle j,a\rangle \in s(P_{il}^t)$ some $k < j$ such that all the tuples in P_{il}^t mapped by s to $\langle j,a\rangle$ have the same kth coordinate. The number of such tuples is at most

$$|P_{il}^t|/|A_{kl}^t| < |P_{il}^t|/(n|A_j^t|).$$

Therefore the number of tuples in P_{il}^t mapped by s to $\{j\} \times A_j^t$ is less than $|P_{il}^t|/n$. However every tuple in P_{il}^t is mapped by s to $Q_i^t = \bigcup_j \{j\} \times A_j^t$ (the union is over all those j for which A_j is a part of Q_i). Therefore P_{il}^t has less than $n|P_{il}^t|/n$ elements, which is a contradiction.

The value of f_{ilr} on s is defined as the first $\langle j,a\rangle \in s(P_{il}^t)$ having the above property. The set of all tuples in P_{il}^t mapped by s to $\langle j,a\rangle$ is called *the base of* f_{ilr} on s.

Step $5t+4$. For all refuted $e \in M^t$ and all $i \leq I$ we make the following. If after step $5t+2$ the program $f_{il(e)r(e)}$ has become *incorrect*, that is, for some s it holds $s^t \in P_i^t \to Q_i^t$ but $f_{il(e)r(e)}(s) \notin Q_i^t$ then we change $r(e)$ and let $r(e) = t$. The program e is declared non-refuted. Note that if the program $f_{il(e)r(e)}$ has become incorrect that for some s, on steps $5t'+2$ with $t' \leq t$, we have removed $f_{il(e)r(e)}(s)$ and for every tuple from the base of $f_{il(e)r(e)}$ on s we have removed at least one component (otherwise $s^t \notin P_i^t \to Q_i^t$). Therefore such event cannot happen often compared to removing elements (we will specify this later).

Step $5t+5$. If on the step $5t+2$, due to removal of a from A_j, for some s, i, l, r the base of f_{ilr} on s decreases or the value $f_{ilr}(s)$ is removed then we declare f_{ilr} *suspicious* (as the chances that later it will become incorrect increase). For all non-refuted $e \in M^t$ and all $i \leq I$ such that $f_{il(e)r(e)}$ is declared suspicious we change the value of $r(e)$ and let $r(e) = t$. Note that all f_{ilt} are not suspicious as we started to define their values only on step $5t+3$. Hence before every step $5t'+2$ for all non-refuted program $e \in M^{t'}$ and all $i \leq I$ the programs $f_{il(e)r(e)}$ are not suspicious. We do not change, on this step, $r(e)$ for refuted programs e, even if $f_{il(e)r(e)}$ was declared suspicious.

It remains to show that we can define the initial cardinalities of $A_1, \ldots, A_n$ so that for all t the inequalities (5) are true. Assume that this is proven. Then let t be a step after which the set $A_1^t, \ldots, A_n^t$ and M^t remains stable. We have to prove that the sets M and $J_n(A_1^t, \ldots, A_n^t)$ do not intersect. Let $e \in M = M^t$. The value $r(e)$ does not change after step t in the enumeration of U. After each step $5t'+3$ for $t' \geq t$ for all s we have

$$s^{t'} \in (P_i^t \to Q_i^t) \quad \Longrightarrow \quad f_{il(e)r(e)}(s) \text{ is defined and belongs to } Q_i^t.$$

Hence

$$f_{il(e)r(e)} \in (P_i^t \to Q_i^t) \to Q_i^t.$$

Assume that $e(\langle f_{1l(e)r(e)}, \ldots, f_{Il(e)r(e)} \rangle)$ is defined and belongs to R^t. Then on a step t' in the enumeration of U we find out that this is the case and remove $e(\langle f_{1l(e)r(e)}, \ldots, f_{Il(e)r(e)} \rangle)$ from R on step $5t'+2$, which is a contradiction.

To prove that the inequalities (5) are true for an appropriate choice of initial cardinalities of $A_1, \ldots, A_n$ we need to upper bound the total number of removals of elements on steps $5t+2$. Let N_k stand for the number of steps $5t+2$ such that an element $\langle k, a \rangle$ was removed on that step. Such steps are called *steps of rank* k. We will prove that

$$N_k \leq 2^{m+1}(N_1 + \ldots + N_{k-1}) + 2^{m+1}.$$

Note that after each removal the number of refuted program is incremented by 1. Some of those programs may become later non-refuted. Let K_0 stand for the number of triples $\langle e, t_1, t_2 \rangle$ such that program e was declared refuted on step $5t_1+2$ and later, for the first time, it was declared non-refuted on step $5t_2+4$. Obviously,

$$N_1 + \ldots + N_n \leq K_0 + 2^m.$$

Let us upper bound K_0. For every of those triples $\langle e, t_1, t_2\rangle$ there is s, i such that, on steps between $5t_1 + 2$ and $5t_2 + 2$, we remove $f_{il(e)r(e)}(s)$ and remove some component from all tuples in the base of $f_{il(e)r(e)}$ on s. Fix i, s for all those triples. If some of those removals happens on step $5t + 2$ we say that this step is *connected* with the triple $\langle e, t_1, t_2\rangle$, and if that was a removal of the second type (that is, a removal of a component from a base), we say that this step is *strongly connected* with the triple $\langle e, t_1, t_2\rangle$. Divide triples $\langle e, t_1, t_2\rangle$ into three categories: (1) those connected with at least on step of rank strictly less than k, (2) those connected only to steps of rank k or greater and strongly connected to at least on step of rank strictly greater than k, and (3) those connected only to steps of rank k or greater and strongly connected only to steps of rank k.

The number of triples of the first type is at most $2^m(N_1+\ldots+N_{k-1})$. Indeed, for every different triples $\langle e, t_1, t_2\rangle$ and $\langle e, t'_1, t'_2\rangle$ with the same first component the intervals $[t_1, t_2]$ and $[t'_1, t'_2]$ are disjoint. If a step $5t + 2$ is connected to the triple $\langle e, t_1, t_2\rangle$ then $t_1 \le t \le t_2$ therefore it is connected to no other triple $\langle e, t'_1, t'_2\rangle$. Hence the total number of triples connected to any step is at most $|M| \le 2^m$.

The number of triple of the second type is at most $N_{k+1} + \ldots + N_n$. Indeed, any step is strongly connected with at most one triple: for all different e_1, e_2 and all j the sets $A_{jl(e_1)}$ and $A_{jl(e_2)}$ are disjoint, hence on every step we cannot remove some component both from a tuple in a base of $f_{i'l(e_1)r(e_1)}$ and from a tuple in a base of $f_{i''l(e_2)r(e_2)}$.

The number of triples of the third type is at most $N_k/2$. To show this it suffices to prove that every triple $\langle e, t_1, t_2\rangle$ of the third type is strongly connected to at least two steps of rank k. Assume that $f_{il(e)r(e)}(s)$ is equal to $\langle j, a\rangle$. All the removals of components of tuples from the base of $f_{il(e)r(e)}$ on s were done on steps of rank k. The definition of a critical implication implies that $j \ne k$. Since $f_{il(e)r(e)}(s)$ was removed on a step of rank k or greater we conclude that $k < j$. Thus the base of $f_{il(e)r(e)}$ on s has two tuples with different k-coordinates, which cannot be removed on the same step of rank k.

So we have proven that

$$N_1 + \ldots + N_n \le 2^m(N_1 + \ldots + N_{k-1}) + N_k/2 + N_{k+1} + \ldots + N_n + 2^m,$$

therefore

$$N_k \le 2^{m+1}(N_1 + \ldots + N_{k-1}) + 2^{m+1},$$

hence $N_k \le 2^{m+1}(2^{m+1} + 1)^{k-1} < 2^{(m+2)n}$. For the last inequality in (5), it is sufficient to let $|A_n| = 2^{(m+2)(n+1)}$. For other inequalities in (5) we need $|A_{k-1}|2^{-m} > n|A_k|+N_{k-1}$ for all $k = n, \ldots, 2$. The second term in the right hand side of this inequality is less than the first one. Therefore it suffices to let $|A_{k-1}| = n2^{m+1}|A_k|$. Finally we obtain the bound $|A_k| = 2^{(m+2)(n+1)}(n2^{m+1})^{n-k}$.

Compactness and Continuity, Constructively Revisited

Douglas Bridges[1], Hajime Ishihara[2], and Peter Schuster[3]

[1] Department of Mathematics & Statistics, University of Canterbury
Private Bag 4800, Christchurch, New Zealand
d.bridges@math.canterbury.ac.nz

[2] School of Information Science, Japan Advanced Institute of Science and Technology
Tatsunokuchi, Ishikawa 923-1292, Japan
ishihara@jaist.ac.jp

[3] Mathematisches Institut, Ludwig-Maximilians-Universität München
Theresienstraße 39, 80333 München, Germany
pschust@mathematik.uni-muenchen.de

Abstract. In this paper, the relationships between various classical compactness properties, including the constructively acceptable one of total boundedness and completeness, are examined using intuitionistic logic. For instance, although every metric space clearly is totally bounded whenever it possesses the Heine-Borel property that every open cover admits of a finite subcover, we show that one cannot expect a constructive proof that any such space is also complete. Even the Bolzano-Weierstraß principle, that every sequence in a compact metric space has a convergent subsequence, is brought under our scrutiny; although that principle is essentially nonconstructive, we produce a reasonable, classically equivalent modification of it that is constructively valid. To this end, we require each sequence under consideration to satisfy uniformly a classically trivial approximate pigeonhole principle—that if infinitely many elements of the sequence are close to a finite set of points, then infinitely many of those elements are close to one of these points—whose constructive failure for arbitrary sequences is then detected as the obstacle to any constructive relevance of the traditional Bolzano-Weierstraß principle.

2000 MSC (AMS): Primary 03F60; Secondary 26E40, 54E45
Keywords: Compact Metric Spaces, Uniform Continuity, Constructive Analysis

1 Introduction

We consider the relations between various notions associated with compactness. What is distinctive about our study is that we work constructively—that is, using intuitionistic logic,[1] which enables us to distinguish between certain weak forms

[1] We also assume the principle of dependent choice, which is widely thought to be constructive, and known to imply that of countable choice. According to Bishop,

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 89–102, 2002.
© Springer-Verlag Berlin Heidelberg 2002

of the law of excluded middle (LEM), and to determine where such weaker laws suffice, and in some cases are needed, to establish equivalences that traditionally are proved using the full form of LEM.

Among the most important weak forms of LEM is the **limited principle of omniscience (LPO)**, which says that

$$\forall \mathbf{a} \in \{0,1\}^{\mathbf{N}} \ (\mathbf{a} = \mathbf{0} \vee \mathbf{a} \neq \mathbf{0}) \,,$$

where $\mathbf{0}$ denotes the zero sequence and $\mathbf{a} \neq \mathbf{0}$ means that some term of the sequence $\mathbf{a}$ equals 1. Note that, in deducing LPO from a certain statement, one may restrict one's attention without loss of generality to *increasing* binary sequences.

LPO is equivalent to the the decidability of the equality[2]

$$\forall x \in \mathbf{R} \ (x = 0 \vee x \neq 0)$$

on the real numbers $\mathbf{R}$, where $x \neq 0$ means $|x| > 0$, and therefore also to the decidability of the equality on an arbitrary metric space (X, ρ), where $x \neq y$ is understood as $\rho(x, y) > 0$.

Being clearly related to the decidability of the halting problem, LPO is essentially nonconstructive [9][3], as is the aforementioned decidability of the equality on $\mathbf{R}$—which may equivalently be expressed as the **law of trichotomy**

$$\forall x \in \mathbf{R} \ (x < 0 \vee x = 0 \vee x > 0) \ ,$$

simply because $x \neq 0$ amounts to $x < 0 \vee x > 0$ for any $x \in \mathbf{R}$.

However, a practicable constructive substitute for this genuinely classical property of $\mathbf{R}$ consists in the **approximate splitting principle**

$$\forall \alpha, \beta \in \mathbf{R} \ (\alpha < \beta \Rightarrow \forall x \in \mathbf{R} \, (x < \beta \vee x > \alpha)) \ ,$$

which can easily be verified by approximating the real numbers under consideration sufficiently closely by rational numbers. Note that, in the conclusion of the approximate splitting principle, the two cases of the disjunction overlap.

'[countable, or even dependent] choice is implied by the very meaning of existence' [3]; more specifically, these particular choice principles arise naturally from the Brouwer-Heyting-Kolmogorov interpretation of universal existential quantifiers. (A fairly unrestricted form of the axiom of choice, on the other hand, implies the law of excluded middle [12]; whence the former is at least as constructively inacceptable as the latter.) We refer to [16] for more on the role of those putatively constructive choice principles, especially within elementary analysis, and to [15] for Richman's alternative strategy to do constructive mathematics without countable choice.

[2] To infer this from LPO, one needs WCC, a very weak form of countable choice that holds classically without *any* choice [10].

[3] LPO is even provably false in either nonclassical standard model of constructive mathematics, in recursive and intuitionistic mathematics ([9], Chapters 3 and 5). It is just LPO, in the form of the law of trichotomy, that in classical mathematics allows to define discontinuous functions, entities which are completely foreign to both the intuitionistic and recursive setting.

For additional background information about constructive mathematics see [1], [2], [3], [9], [17], [18].

We say that a metric space (X, ρ) has

- the **Heine-Borel property (HB)** if every open cover has a finite[4] subcover;
- the **Lebesgue covering property (LCP)** if to each open cover $\mathcal{U}$ of X there corresponds a positive **Lebesgue number** r such that each open ball of radius r is contained in some set in $\mathcal{U}$;
- the **uniform continuity property (UC)** if each (pointwise) continuous function from X to $\mathbf{R}$ is uniformly continuous;
- the **Heine-Borel-Lebesgue property (HBL)** if each open cover with a Lebesgue number has a finite subcover;
- the **approximate Heine-Borel property (aHB)** if for each open cover $(U_i)_{i\in I}$ of X, and each $\varepsilon > 0$, there exists a finite subset J of I such that

$$X = \bigcup_{i\in J}\left(\bigcup_{x\in U_i} B(x,\varepsilon)\right);$$

- the **pseudo-Heine-Borel property (pHB)**[5] if every sequence of closed subsets with the finite intersection property[6] has a nonempty intersection.

The classically well-known facts that **HB** implies **LCP**, and that **LCP** implies **UC**, are easily seen to obtain constructively. In particular, **HB** is equivalent to the conjunction of **LCP** and **HBL**. Classically, **LCP** and **UC** are even equivalent, whereas **HB** is strictly stronger than **UC**: the set $\mathbf{N}$ of natural numbers clearly possesses the latter but not the former property. For details see Section 3.3 of [4].

Needless to say, **HB** and **pHB** are classically equivalent to each other. We will realise later on that **pHB** implies a constructive weakening of **HB**, but also that there is no hope for a constructive proof of the converse.

Each of the properties **HB, HBL,** and **aHB** has a 'countable' version which applies to countable covers. For example, (X, ρ) has the **countable Heine-Borel-Lebesgue property** if each countable open cover with a Lebesgue number has a finite subcover. We denote the countable version of property **P** by $\mathbf{P_N}$. For a separable metric space X, the properties $\mathbf{HBL_N}$ and $\mathbf{aHB_N}$ are equivalent to each other and to X being totally bounded (see below).

We also need the important constructive properties of locatedness and semidetachability. A subset S of a metric space X is said to be

[4] Throughout this article, we mean by a **finite set** one that consists of finitely many elements, moreover, 'finitely many' is tacitly understood as embodying 'at least one'. (We thus deviate, for the sake of simplicity, from the use of 'finite' in many constructive contexts where, in addition, the equality on the set in question is required to be decidable, and where finite sets in our sense are usually named 'finitely enumerable' or 'subfinite', and sometimes allowed to be empty.)

[5] The name of this property is taken from [13].

[6] Recall that a family $\mathcal{F}$ of sets has the **finite intersection property** if every intersection of finitely many sets in $\mathcal{F}$ is nonempty.

– **located** (in X) if the distance

$$\rho(x, S) = \inf \{\rho(x, y) : y \in S\}$$

exists for each $x \in X$.

– **semidetachable**[7] (in X) if

$$\forall x \in X \ (\neg (x \in \sim S) \Rightarrow x \in S),$$

where

$$\sim S = \{x \in X : \forall s \in S \ (x \neq s)\}$$

is the **complement** of S. (Recall that $x \neq s$ means $\rho(x, s) > 0$.)

For instance, S is located whenever it is **totally bounded:** in other words, for every $\varepsilon > 0$ there is a finite ε–approximation to S—that is, a finite subset S_ε of S such that S is covered by the open balls of radius ε and with centre in S_ε; see [3], pages 94–95. To prove this, one needs the **constructive least-upper-bound principle**, which says that any nonempty subset T of $\mathbf{R}$ that is bounded above possesses a supremum provided that, for all $\alpha < \beta$, either $t < \beta$ for every $t \in T$ or $t > \alpha$ for some $t \in T$. Note that every totally bounded set of real numbers satisfies the hypotheses of this principle.[8] A criterion for semidetachability will be given below.

We denote by $\widehat{X}$ the completion of the metric space X. One way of constructing $\widehat{X}$ is to take the elements of $\widehat{X}$ to be sequences $\mathbf{x} = (x_n)$ in X that are **regular** in the sense that

$$\rho(x_m, x_n) \leqslant \tfrac{1}{m} + \tfrac{1}{n} \quad (m, n \geqslant 1),$$

and to define two such sequences $\mathbf{x}$ and $\mathbf{y} = (y_n)$ to be **equal** if

$$\rho(x_n, y_n) \leqslant \tfrac{2}{n} \quad (n \geqslant 1).$$

Moreover, the metric ρ on X is extended to $\widehat{X}$ by setting

$$\rho(\mathbf{x}, \mathbf{y}) = \lim_{n \to \infty} \rho(x_n, y_n),$$

so that X is isometrically embedded into $\widehat{X}$ by way of constant sequences. For further details see [3], Chapter 4, Section 3.

Of course, X equals $\widehat{X}$ if and only if X is **complete:** that is, every Cauchy sequence converges in X. According to *loc.cit.*, Lemma (3.8),[9] every nonempty complete located subset of the metric space X is **reflective:** that is, for every $x \in X$ there is $s \in S$ so that if $x \neq s$, then $x \in \sim S$. A reflective subset clearly is

[7] This notion was coined in [6].

[8] According to our general supposition, finite sets are inhabited, and so is $S_1 \subset S$.

[9] Like many constructions of sequences in this paper, this result does not need full countable choice: as demonstrated in [10], WCC suffices (cf. footnote 2).

semidetachable; to see this, one needs to observe that the inequality on X, just as that on $\mathbf{R}$, is **tight**—that is, $x = y$ holds if (and only if) $x \neq y$ is impossible.

One could likewise define a real number to be a regular sequence of rational numbers, and then demonstrate, for instance, the least-upper-bound principle (cf. [3], Chapter 2, Section 2 and Lemma (4.3)). We prefer to leave the notion of a real number somewhat unspecified, and to work instead with the axiom system presented in [5] that collects together all the constructively reasonable properties of $\mathbf{R}$, including the constructive least-upper-bound principle and the approximate splitting principle mentioned above.

2 Some Important Connections

In this section we show that HB implying completeness amounts to LPO, and establish a number of relations involving the various compactness notions introduced above. Throughout, (X, ρ) will be a metric space, and $\widehat{X}$ its completion.

Proposition 1. *If X satisfies **UC** and is semidetachable in $\widehat{X}$, then X is complete.*

Proof. Let $\xi \in \widehat{X}$, and suppose that $\xi \in \sim X$. Then $f(x) = 1/\rho(x, \xi)$ defines a continuous mapping of X into $\mathbf{R}$ and so is uniformly continuous. Choose $\delta > 0$ such that if $x, y \in X$ and $\rho(x, y) < \delta$, then $|f(x) - f(y)| < 1$.

Since X is dense in $\widehat{X}$ we can find a point x of X such that $\rho(\xi, x) < \delta/2$. In view of $\xi \in \sim X$, there also is a positive integer n with $\rho(\xi, x) > 1/n$. For this n, we can again find a point y of X with $\rho(\xi, y) < 1/(n+1)$. Then

$$0 < \rho(\xi, y) < \tfrac{1}{n+1} < \tfrac{1}{n} < \rho(\xi, x) < \delta/2;$$

whence $\rho(x, y) < \delta$ and $f(y) > n + 1 > n > f(x)$, so that $|f(x) - f(y)| > 1$, which contradicts our choice of δ. Hence $\neg(\xi \in \sim X)$ and therefore, as X is semidetachable in $\widehat{X}$, $\xi \in X$. Thus $\widehat{X} = X$ and X is complete. Q.E.D.

Proposition 2. *If LPO holds, then every metric space satisfying **HB** is complete.*

Proof. Assuming LPO, let X satisfy **HB**, and consider any ξ in $\widehat{X}$. For each $x \in X$, let $U_x = B(x, 1)$ or $U_x = B\left(x, \frac{1}{2}\rho(\xi, x)\right)$, depending on whether $x = \xi$ or $x \neq \xi$, respectively, which we can decide by way of LPO. Then $(U_x)_{x \in X}$ is an open cover of X, from which we can extract a finite subcover, say $\{U_{x_1}, \ldots, U_{x_n}\}$. Again by LPO, either $\xi = x_k$ for some k or else $\xi \neq x_k$ for all k. In the former case we conclude that $\xi = x_k \in X$ for some k, and therefore that $\widehat{X} = X$; so X is complete. In the latter case, which eventually will turn out impossible, we have

$$U_{x_k} = B\left(x_k, \tfrac{1}{2}\rho(\xi, x_k)\right) \quad (1 \leqslant k \leqslant n)$$

and

$$0 < r = \tfrac{1}{2} \min_{1 \leqslant k \leqslant n} \rho(\xi, x_k).$$

Let x be any point of X. Since $\{U_{x_1}, \ldots, U_{x_n}\}$ covers X, there is k such that $x \in U_{x_k}$, i.e. $\rho(x, x_k) < \frac{1}{2}\rho(\xi, x_k)$. We now see that

$$\rho(\xi, x) \geqslant \rho(\xi, x_k) - \rho(x, x_k) > \tfrac{1}{2}\rho(\xi, x_k) \geqslant r.$$

Hence ξ is bounded away from X, which is absurd as X is dense in $\widehat{X}$. Q.E.D.

Proposition 3. *If every metric space satisfying **HB** is semidetachable in its completion, then LPO holds.*

Proof. Assume that every metric space satisfying **HB** is semidetachable in its completion, and consider the space

$$X = \{0\} \cup \left\{\tfrac{1}{n} : n \in \mathbf{N}^+\right\}$$

with the usual metric. If $\mathcal{U}$ is an open cover of X, then we can find $U \in \mathcal{U}$ and $r > 0$ such that $B(0, r) \subset U$. Choosing N such that $1/N < r$, and then sets $U_1, \ldots, U_N$ in $\mathcal{U}$ such that $1/k \in U_k$ $(1 \leqslant k \leqslant N)$, from $\mathcal{U}$ we obtain a finite subcover $\{U, U_1, \ldots, U_N\}$ of X. Thus X satisfies **HB** and is therefore semidetachable in $\widehat{X}$. Now consider an increasing binary sequence $(a_n)_{n=1}^{\infty}$. Define a sequence (ξ_n) in X such that if $a_n = 0$, then $\xi_n = 1/n$, and if $a_n = 1$, then $\xi_n = 1/m$ for the first $m \leqslant n$ so that $a_m = 1$. Then (ξ_n) is a Cauchy sequence in $\mathbf{R}$ and so converges to a limit $\xi \in \mathbf{R}$. If $\xi \in \sim X$, then it is clear that

$$\neg\left(\forall n\,(a_n = 0) \vee \exists n\,(a_n = 1)\right),$$

which is absurd. Since X is semidetachable in $\widehat{X}$, it follows that $\xi \in X$; whence either $\xi = 0$, and so $a_n = 0$ for all n, or else $\xi = 1/N$ for some N, and therefore $a_N = 1$. In other words, LPO holds. Q.E.D.

Alternatively, we could have completed the foregoing proof as follows. Since, as we recalled above, **HB** implies **UC** through **LCP**, if every space satisfying **HB** is semidetachable in its completion, then, by Proposition 1, **HB** implies completeness. But then the point ξ constructed above belongs to X, and the end of the proof goes through as before.

Up to now, the principal achievement of our investigations is the following consequence of the foregoing results.

Corollary 1. *LPO is equivalent to the statement that every metric space satisfying **HB** is complete.*

In particular, although **HB** implies total boundedness, one half of the constructively reasonable classical equivalent to **HB**, one cannot expect also the other half, completeness, to constructively follow from **HB**.

Moreover, in the absence of LPO there is no hope to constructively prove that **HB** implies **pHB**. In [14], namely, **pHB** was constructively shown to coincide with the sequential compactness of X, i.e. the unrestricted Bolzano-Weierstraß

principle, and therefore to imply the completeness (in fact, also the total boundedness) of X. So if **pHB** was to follow from **HB**, then one could deduce the completeness of X from **HB**, a step for which LPO turned out indispensable before.

We state the following complementary results without their proofs.

Proposition 4. *If X is totally bounded, then it satisfies $\boldsymbol{HBL}_{\mathbf{N}}$.*

Proposition 5. *If X satisfies $\boldsymbol{HBL}_{\mathbf{N}}$, then it satisfies $\boldsymbol{aHB}_{\mathbf{N}}$.*

Proposition 6. *If X is separable and satisfies $\boldsymbol{aHB}_{\mathbf{N}}$, then it is totally bounded.*

Now we prove two lemmas that will enable us to find conditions under which **pHB** implies an open cover property that, with classical logic, is equivalent to the full form of $\mathbf{HB}_{\mathbf{N}}$.

Lemma 1. *Suppose that X be a separable metric space with the property **pHB**. Let $f_1, \ldots, f_\nu$ be continuous mappings of X into $\mathbf{R}$, and let $\alpha < \beta$. Then*

either for each $x \in X$ there exists k such that $f_k(x) < \beta$
or there exists $x \in X$ such that $f_k(x) > \alpha$ for each k.

Proof. Let (x_n) be a dense sequence in X, and set $\varepsilon = \frac{1}{3}(\beta - \alpha)$. By virtue of the approximate splitting principle, we can choose an increasing binary sequence (λ_n) such that

$$\lambda_n = 0 \Rightarrow \forall j \leqslant n\, \exists k\, (f_k(x_j) < \beta - \varepsilon)\,,$$
$$\lambda_n = 1 \Rightarrow \exists j \leqslant n\, \forall k\, (f_k(x_j) > \alpha + \varepsilon)\,.$$

We may assume that $\lambda_1 = 0$. If $\lambda_n = 0$, set $F_n = X$; if $\lambda_n = 1$, set

$$F_n = \{x \in X : \forall k\, (f_k(x) \geqslant \alpha + \varepsilon)\}\,.$$

Then (F_n) is a decreasing sequence of nonempty closed subsets of X; so, by **pHB**, there exists a point $y \in \bigcap_{n=1}^{\infty} F_n$. Again by the approximate splitting principle, either $f_k(y) > \alpha$ for each k, in which case the proof is complete, or there exists k such that $f_k(y) < \alpha + \varepsilon$. In the latter case, if $\lambda_n = 1$, then $y \in F_n$ and so $f_k(y) \geqslant \alpha + \varepsilon$, a contradiction. Hence $\lambda_n = 0$ for all n. Given $x \in X$, and using the continuity of each of the functions f_j, we now choose n such that $|f_j(x) - f_j(x_n)| < \varepsilon$ for each j $(1 \leqslant j \leqslant \nu)$. Then, since $\lambda_n = 0$, there exists k such that $f_k(x_n) < \beta - \varepsilon$ and therefore $f_k(x) < \beta$. Q.E.D.

Lemma 2. *Under the hypotheses of the preceding lemma,*

either for each $x \in X$ there exists k such that $f_k(x) > \alpha$
or there exists $x \in X$ such that $f_k(x) < \beta$ for each k.

Proof. Apply the preceding lemma with f_k replaced by $-f_k$, and α, β replaced, respectively, by $-\beta, -\alpha$. Q.E.D.

An open subset U of a metric space is **coherent** if $-(\sim U) = U$, where, for any subset S of X, the **metric complement** of S

$$-S = \{x \in X : \exists \delta > 0 \, \forall s \in S \, \rho(x, s) \geqslant \delta\},$$

subsumes all points of X that are bounded away from S. It is shown in [11] that, constructively, an open set is coherent if and only if it is a metric complement.

As, classically, every nonempty subset is located, and every open subset is coherent, in the following proposition we require only classically trivial conditions from the members of the cover under consideration.

Proposition 7. *Let X be a separable metric space with the property **pHB**, and let $(U_n)_{n=1}^{\infty}$ be a (countable) open cover of X such that U_n is coherent, $U_n \subset U_{n+1}$, and $\sim U_n$ is located for each n. Then $X \subset U_n$ for some n.*

Proof. For convenience, write $F_n = \sim U_n$. As each F_n is assumed to be located, $\rho(x, F_n)$ exists for every n. Using Lemma 2, choose an increasing binary sequence (λ_n) such that

$$\lambda_n = 0 \;\Rightarrow\; \exists x \in X \, \forall k \leqslant n \, \left(\rho(x, F_k) < \tfrac{1}{n}\right),$$
$$\lambda_n = 1 \;\Rightarrow\; \forall x \in X \, \exists k \leqslant n \, \left(\rho(x, F_k) > \tfrac{1}{4n}\right).$$

Since, by hypothesis, each U_n is coherent, it suffices to find some ν with $\lambda_\nu = 1$: as then each $x \in X$ is bounded away from $F_k = \sim U_k$ for some $k \leqslant \nu$, and thus belongs to $-(\sim U_k) = U_k$ for this k, we get $X \subset U_\nu$. In particular, we may assume that $\lambda_1 = 0$.

For each n, if $\lambda_n = 0$, set

$$G_n = \left\{x \in X : \rho(x, F_n) \leqslant \tfrac{1}{n+1}\right\};$$

if $\lambda_n = 1$, set $G_n = G_m$ for the last $m \leqslant n$ with $\lambda_m = 0$. Then (G_n) is a decreasing sequence of nonempty closed subsets of X, so there exists $\xi \in \bigcap_{n=1}^{\infty} G_n$, according to **pHB**. Pick N such that $\xi \in U_N$, and an integer $\nu \geqslant N$ such that $B(\xi, 1/\nu) \subset U_N \subset U_\nu$; in particular, $\rho(\xi, F_\nu) \geqslant 1/\nu$. Then $\lambda_\nu = 1$, and so $X \subset U_\nu$. Indeed, if $\lambda_\nu = 0$, then $\rho(\xi, F_\nu) \leqslant 1/(\nu + 1)$ (because $\xi \in G_\nu$), a contradiction. Q.E.D.

It is tempting to hope that the restricted Heine-Borel property in Proposition 7 might hold constructively at least when $X = [0, 1]$. However, although one can deduce, from Brouwer's fan theorem and the principle of continuous choice, that every open cover of a compact metric space has a finite subcover ([9], Chapter 5, Theorem (1.4), Theorem (3.5)), in the recursive model of constructive mathematics $[0, 1]$ can be covered, thanks to the presence of Specker sequences, by a sequence of bounded open intervals any finite collection of which has total length less than $1/2$ ([9], page 60, Theorem (4.1)).

3 Almost Sequential Compactness

In this section we introduce almost sequential compactness, a classical equivalent of sequential compactness. Among other things, we show that almost sequential compactness obtains for metric spaces that are compact—that is, totally bounded and complete—and that one precisely needs LPO for the step from almost sequential compactness to sequential compactness.

Since the sequential compactness of even such a simple space as $\{0,1\}$ entails LPO, constructive mathematicians have tended to ignore sequential compactness altogether. Also, as we have indicated above, HB is not appropriate for constructive purposes, whereas total boundedness plus completeness has proved a good constructive choice from the classically equivalent definitions of compactness. It is nevertheless reasonable to seek sequential compactness properties that hold constructively for totally bounded and complete metric spaces like $[0,1]$ and that are classically equivalent to the usual sequential compactness property. One such was presented in [7]; another one is introduced by the following definitions.

A sequence (x_n) in a metric space X is said to be **discriminating** whenever for each $\varepsilon > 0$ there exists $\delta > 0$ with the following property: if Y is a finite subset of X and $\rho(x_n, Y) < \delta$ for infinitely many n, then there exists $\xi \in Y$ such that $\rho(x_n, \xi) < \varepsilon$ for infinitely many n. Note that if Y is finite, then $\rho(x, Y) = \min_{y \in Y} \rho(x, y)$ exists as the minimum of a finite set of real numbers.

We shall see in a moment that, constructively, every Cauchy sequence is discriminating, whereas, for instance, the sequences $1, 2, 3, \ldots$ and $+1, -1, +1, \ldots$ are discriminating but, of course, not Cauchy. In the presence of LPO, on the other hand, every sequence is discriminating (see below).

We say that X is **almost sequentially compact** if every discriminating sequence in X has a convergent subsequence. Clearly, almost sequential compactness follows from sequential compactness. Unlike the latter, which constructively is too strong to admit of any example more substantial than a singleton space, the former turns out to have constructive content, as follows.

It is easily seen that $\{0,1\}$ is almost sequentially compact. More generally, we now show that every compact metric space is almost sequentially compact.

Lemma 3. *If X is totally bounded, then each discriminating sequence in X possesses a Cauchy subsequence.*

Proof. Let (x_n) be a discriminating sequence in X. We successively choose positive integers $n_0 = 1 < n_1 < \ldots$ and compact sets $X_0 = X \supset X_1 \supset \cdots$ such that $x_n \in X_k$ for infinitely many n, including n_k, and such that $\operatorname{diam}(X_k) < 2^{-k}$ for each $k \geqslant 1$. To this end, assume that we have found n_k and X_k. For $\varepsilon = 2^{-k-3}$, let $\delta > 0$ be as in the definition of '(x_n) is discriminating', and let $\{\xi_1, \ldots, \xi_m\}$ be a δ–approximation to X_k. Then there exists ν such that $\rho(x_n, \xi_\nu) < 2^{-k-3}$ for infinitely many n; so we can pick $n_{k+1} > n_k$ such that $\rho(x_{n_{k+1}}, \xi_\nu) < 2^{-k-3}$. Now, X_k is compact: that is, totally bounded and complete. According to [9] (Chapter 2, Theorem 4.7), there exists a compact subset X_{k+1} of X_k such that

$$B\left(\xi_\nu, 2^{-k-3}\right) \subset X_{k+1} \subset B\left(\xi_\nu, 2^{-k-2}\right);$$

in particular, X_{k+1} contains infinitely many terms x_n, including $x_{n_{k+1}}$, and has diameter less than 2^{-k-1}. This completes the inductive construction.

For all $j \geqslant k$ we have

$$\rho(x_{n_j}, x_{n_k}) \leqslant \operatorname{diam}(X_k) < 2^{-k},$$

so $(x_{n_k})_{k=1}^{\infty}$ is a Cauchy sequence in X. Q.E.D.

Proposition 8. *A metric space that is compact, in the sense of being totally bounded and complete, is almost sequentially compact.*

Proof. Let (x_n) be a discriminating sequence in a metric space X. By Lemma 3, (x_n) has a Cauchy subsequence whenever X is totally bounded. If also X is complete, this subsequence of (x_n) converges in X. Q.E.D.

It is noteworthy that none of the hypotheses of Proposition 8 is completely redundant. First, observe that every bounded nonempty open interval is totally bounded but not almost sequentially compact, because otherwise it would be complete (Proposition 9 below), which it is not. Secondly, every closed subset X of $\mathbf{R}$ that contains $\mathbf{N}$ (for instance, $X = \mathbf{R}$) is complete, whereas it fails to be almost sequentially compact. To see the latter, observe that the sequence of positive integers clearly possesses no convergent subsequence, but still is discriminating by virtue of *ex falso quodlibet*, because it is impossible, for any $\delta > 0$ and any finite subset Y of X, that $\rho(n, Y) < \delta$ for infinitely many n.

As compared with compactness in the sense of total boundedness plus completeness, a feature of almost sequential compactness is that, just as for completeness, every closed subset of an almost sequentially compact metric space is almost sequentially compact, too, whereas only *located* closed subsets inherit total boundedness from the ambient space.

Let us now aim at some partial converses of Proposition 8.

Lemma 4. *Every Cauchy sequence is discriminating.*

Proof. Let (x_n) be a Cauchy sequence in X. For any $\varepsilon > 0$, let $\delta > 0$ with $2\delta \leqslant \varepsilon/2$, and pick N such that $\rho(x_m, x_n) < \delta$ for all $m, n \geqslant N$. Now if Y is a finite subset of X such that $\rho(x_n, Y) < \delta$ for infinitely many n, then there exist $m \geqslant N$ and $\xi \in Y$ such that $\rho(x_m, \xi) < \delta$; whence $\rho(x_n, \xi) < 2\delta \leqslant \varepsilon$ for all $n \geqslant m$. Q.E.D.

In $\mathbf{R}$, and likewise any subset of $\mathbf{R}$ containing $+1$ and -1, the sequence $(-1)^n$ is discriminating but not a Cauchy sequence. Indeed, given $\varepsilon > 0$, set $\delta = \varepsilon$; if Y is a finite subset of $\mathbf{R}$ such that $\rho((-1)^N, Y) < \delta$—that is, $\rho((-1)^N, \xi) < \delta$ for some $\xi \in Y$—already for a single N, then $\rho((-1)^n, \xi) < \varepsilon$ for this ξ, and for all even n or for all odd n, depending on whether N is even or odd, respectively.

Proposition 9. *An almost sequentially compact space is complete.*

Proof. Let (x_n) be a Cauchy sequence in a metric space X. By Lemma 4, (x_n) is a discriminating sequence. So if X is almost sequentially compact, then (x_n) has a convergent subsequence; whence (x_n)—being a Cauchy sequence—is itself convergent. Q.E.D.

From this and the foregoing proposition, we can deduce the following.

Corollary 2. *For any totally bounded metric space, almost sequential compactness is equivalent to completeness.*

In particular, compactness is equivalent, for an arbitrary metric space, to almost sequential compactness plus total boundedness.

In the sequel, we need to impose the condition

(*) *for all positive α, β with $\alpha < \beta$,*

either $\rho(x, Y) < \beta$ for every $x \in X$
or $\rho(x, Y) > \alpha$ for some $x \in X$

some times on a finite $Y \subset X$ (Recall that $\rho(x, Y)$ exists for any such Y.)

Proposition 10. *Let X be almost sequentially compact, and suppose that there exists a finite subset Y of X satisfying (*). Then X is bounded.*

Proof. By (*), we can choose an increasing binary sequence (λ_n) such that

$$\begin{aligned} \lambda_n = 0 \;&\Rightarrow\; \exists x \in X\ (\rho(x, Y) > n)\,, \\ \lambda_n = 1 \;&\Rightarrow\; \forall x \in X\ (\rho(x, Y) < n + 1)\,. \end{aligned}$$

Now it suffices to find some n with $\lambda_n = 1$: indeed, for any $x, z \in X$, we have $\rho(x, z) \leqslant \rho(x, y) + \rho(z, y)$ for all $y \in Y$, and thus $\rho(x, z) \leqslant \rho(x, Y) + \rho(z, Y)$; whence $\operatorname{diam}(X) \leqslant 2(n + 1)$ if only $\lambda_n = 1$. In particular, we may assume that $\lambda_1 = 0$.

For each n, if $\lambda_n = 0$, pick $x_n \in X$ such that $\rho(x_n, Y) > n$; if $\lambda_n = 1$, set $x_n = x_{n-1}$. To prove that the sequence (x_n) is discriminating, let Z be a finite subset of X, and $\varepsilon > 0$. We shall see that $\delta = \varepsilon$ works. To this end, suppose that $\rho(x_n, Z) < \varepsilon$ for infinitely many n. Pick a positive integer

$$N > \varepsilon + \sup_{y \in Y,\, z \in Z} \rho(y, z).$$

If $\lambda_N = 0$, then for all $n \geqslant N$, $y \in Y$, and $z \in Z$,

$$\rho(x_n, z) \geqslant \rho(x_n, y) - \rho(y, z) \geqslant \rho(x_n, y) - (N - \varepsilon)\,;$$

so

$$\rho(x_n, z) \geqslant \rho(x_n, Y) - (N - \varepsilon) > N - (N - \varepsilon) = \varepsilon$$

and therefore $\rho(x_n, Z) \geqslant \varepsilon$, a contradiction. Hence $\lambda_N = 1$, and $x_n = x_N$ for all $n \geqslant N$; since $\rho(x_n, Z) < \varepsilon$ for infinitely many n, there exists $\xi \in Z$ such that $\rho(x_n, \xi) < \varepsilon$ for all $n \geqslant N$.

As X is almost sequentially compact, there exists a subsequence $(x_{n_k})_{k=1}^{\infty}$ of (x_n) that converges to a limit $x_\infty \in X$. Choosing a positive integer $\nu > \rho(x_\infty, Y)$, and then $K > \nu$ such that $\rho(x_{n_k}, Y) < \nu$ for all $k \geqslant K$, we have that $\lambda_{n_K} = 1$ (because if $\lambda_{n_K} = 0$, then $\rho(x_{n_K}, Y) > n_K \geqslant K > \nu$, a contradiction); whence $\operatorname{diam}(X) \leqslant 2(n_K + 1)$ as above, so that X is bounded. Q.E.D.

Corollary 3. *If X is an almost sequentially compact metric space, and Y a finite subset of X satisfying condition (*), then $\sup_{x \in X} \rho(x, Y)$ exists.*

Proof. By Proposition 10,

$$T = \{\rho(x, Y) : x \in X\}$$

is a bounded subset of $\mathbf{R}$. Taken with the constructive least-upper-bound principle, our hypotheses ensure that $\sup T$ exists. Q.E.D.

Proposition 11. *The following are equivalent conditions on an almost sequentially compact metric space X.*

(i) *X is totally bounded.*
(ii) *$\sup_{x \in X} \rho(x, Y)$ exists for each finite subset Y of X.*
(iii) *Every finite subset Y of X satisfies (*).*

Proof. If X is totally bounded, then for each finite subset Y of X the function $x \mapsto \rho(x, Y)$, being uniformly continuous on X, has totally bounded range, and therefore possesses a supremum. Hence (i) implies (ii). It is easily shown that (ii) implies (iii); the reverse implication is an immediate consequence of Corollary 3. To complete the proof, it remains to show that (iii) implies (i). Assuming (iii) and given $x_0 \in X$ and $\vartheta > 0$, we can successively choose an increasing binary sequence $(\lambda_n)_{n=1}^{\infty}$ and a sequence $(x_n)_{n=0}^{\infty}$ in X, beginning with the given x_0, such that

$$\begin{aligned} \lambda_n = 0 \;&\Rightarrow\; \exists x_n \in X \; (\rho\,(x_n, \{x_0, \ldots, x_{n-1}\}) > \vartheta)\;, \\ \lambda_n = 1 \;&\Rightarrow\; \forall x \in X \; (\rho\,(x, \{x_0, \ldots, x_{n-1}\}) < 2\vartheta) \text{ and } x_n = x_{n-1}. \end{aligned}$$

Our goal is now to find some n with $\lambda_n = 1$, because $\{x_0, \ldots, x_{n-1}\}$ is a finite 2ϑ–approximation to X for any such n, and thus X totally bounded.

We show first that the sequence (x_n) is discriminating. Given $\varepsilon > 0$, any choice of $\delta > 0$ with $\delta \leqslant \varepsilon$ and $2\delta \leqslant \vartheta$ will suffice. To see this, let Y be a finite subset of X, and suppose that there exists a subsequence $(x_{n_k})_{k=1}^{\infty}$ of (x_n) such that $\rho(x_{n_k}, Y) < \delta$ for each k. Since Y is finite, there exist $y \in Y$ and j, k such that $j > k, \rho(x_{n_j}, y) < \delta$, and $\rho(x_{n_k}, y) < \delta$; whence $\rho(x_{n_j}, x_{n_k}) < 2\delta \leqslant \vartheta$. This implies that $\lambda_{n_j} = 1$ (only for the moment, under the assumption that such a finite subset Y be present!); so $x_n = x_{n_j}$, and therefore $\rho(x_n, y) < \delta \leqslant \varepsilon$, for all $n \geqslant n_j$. Thus (x_n) is discriminating and therefore has a convergent subsequence $(x_{n_k})_{k=1}^{\infty}$. Now choose M such that $\rho(x_{n_j}, x_{n_k}) < \vartheta$ for all $j, k \geqslant M$; then $\lambda_{n_{M+1}} = 1$ as required. Indeed, if $\lambda_{n_{M+1}} = 0$, then $\rho(x_{n_{M+1}}, x_{n_M}) > \vartheta$, a contradiction. Q.E.D.

Corollary 4. *An almost sequentially compact metric space X is compact, i.e. complete and totally bounded, provided that every finite subset Y of X satisfies condition (*).*

Finally, we investigate how the constructively reasonable property of almost sequential compactness differs from the stronger, and constructively irrelevant one, of sequential compactness. To this end, let us state first the following completely straightforward characterisation of LPO as a kind of pigeonhole principle.[10]

Lemma 5. *LPO is equivalent to the statement that, for every sequence in a union of finitely many sets, one of these sets contains infinitely many elements of the sequence.*

Proposition 12. *LPO implies that, in an arbitrary metric space X, every sequence is discriminating. Conversely, if every sequence in $\{0,1\}$ is discriminating, then LPO obtains.*

Proof. Suppose LPO; let (x_n) be a sequence in X and $\varepsilon > 0$. For $\delta = \varepsilon$, if $Y \subset X$ is finite such that $x_n \in \bigcup_{y \in Y} B(y, \delta)$ for infinitely many n, then Lemma 5 produces $\xi \in Y$ with $x_n \in B(\xi, \varepsilon)$ for infinitely many of the aforementioned n; whence (x_n) is discriminating.

For the converse, let $\mathbf{a} = (a_n)$ be an increasing binary sequence, assume that it is discriminating in $\{0,1\}$, and let $\varepsilon = \frac{1}{2}$. As $\rho(a_n, Y) < \delta$ for $Y = \{0,1\}$, arbitrary $\delta > 0$, and all n, there is $\eta \in \{0,1\}$ with $\rho(a_n, \eta) < \varepsilon$—that is, $a_n = \eta$—for infinitely many n. In particular, $\mathbf{a} = \mathbf{0}$ or $\mathbf{a} \neq \mathbf{0}$, depending on whether $\eta = 0$ or $\eta = 1$, respectively. Q.E.D.

Corollary 5. *LPO is equivalent to the statement that every almost sequentially compact metric space is sequentially compact.*

Proof. Given LPO, every sequence is discriminating according to Proposition 12; whence in an almost sequentially compact metric space every sequence possesses a convergent subsequence. Conversely, note that $\{0,1\}$ is almost sequentially compact; if it is also sequentially compact, then LPO obtains. Q.E.D.

In particular, almost sequential compactness is classically equivalent to sequential compactness.

In retrospect, it is now clear what the reason is that the classical Bolzano-Weierstraß principle fails to be of any constructive value. On the one hand, each metric space that is compact, in the sense of being totally bounded and complete, is almost sequentially compact—and every discriminating sequence in it has a convergent subsequence (Proposition 8). On the other hand, we cannot expect that, in the absence of LPO, every almost sequentially compact metric space is sequentially compact (Corollary 5), let alone that every sequence in a compact metric space is discriminating (Proposition 12).

[10] For a similar result, see [8].

Acknowledgements

The authors wish to express their gratitude to the anonymous referees for some suggestions that helped to improve the presentation of this paper, to Rudolf Taschner for an essential hint, and, especially, to Lilla Hartyáni for her most generous hospitality.

References

[1] Michael J. Beeson, *Foundations of Constructive Mathematics*, Ergebn. Math. Grenzgeb. Math. (3) **6**, Springer, Heidelberg, 1985. 91

[2] Errett Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967. 91

[3] Errett Bishop and Douglas Bridges, *Constructive Analysis*, Grundlehr. math. Wiss. **279**, Springer, Heidelberg, 1985. 90, 91, 92, 93

[4] Douglas S. Bridges, *Foundations of Real and Abstract Analysis,* Graduate Texts Math. **174**, Springer, New York, 1998. 91

[5] Douglas S. Bridges, 'Constructive mathematics: a foundation for computable analysis', Theoret. Comput. Sci. **219**, 95–109, 1999. 93

[6] Douglas S. Bridges, 'Prime and maximal ideals in constructive ring theory', Communic. Algebra **29**, 2787–2803, 2001. 92

[7] Douglas Bridges, Hajime Ishihara, and Peter Schuster, 'Sequential compactness in constructive analysis', Österreich. Akad. Wiss. Math.-Natur. Kl. Sitzungsber. II **208**, 159–163, 1999. 97

[8] Douglas Bridges and Ayan Mahalanobis, 'Bounded variation implies regulated: a constructive proof', J. Symb. Logic **66**, 1695–1700, 2001. 101

[9] Douglas Bridges and Fred Richman, *Varieties of Constructive Mathematics*, London Math. Soc. Lect. Notes Math. **97**, Cambridge University Press, 1987. 90, 91, 96, 97

[10] Douglas Bridges, Fred Richman, and Peter Schuster, 'A weak countable choice principle', Proc. Amer. Math. Soc. **128**(9), 2749–2752, 2000. 90, 92

[11] Douglas Bridges, Fred Richman, and Wang Yuchuan, 'Sets, complements and boundaries', Proc. Koninklijke Nederlandse Akad. Wetenschappen (Indag. Math., N. S.) **7**(4), 425–445, 1996. 96

[12] Nicolas D. Goodman and John Myhill, 'Choice implies excluded middle', Zeit. Math. Logik Grundlag. Math. **24**, 461, 1978. 90

[13] Hajime Ishihara, 'An omniscience principle, the König lemma and the Hahn-Banach theorem'. Zeit. Math. Logik Grundlag. Math. **36**, 237–240, 1990. 91

[14] Hajime Ishihara and Peter Schuster, 'Constructive compactness continued'. Preprint, University of Munich, 2001. 94

[15] Fred Richman, 'Constructive mathematics without choice'. In: Peter Schuster, Ulrich Berger, and Horst Osswald, eds., *Reuniting the Antipodes. Constructive and Nonstandard Views of the Continuum.* Proc. 1999 Venice Symposion. Synthese Library **306**, 199–205. Kluwer, Dordrecht, 2001. 90

[16] Peter M. Schuster, 'Unique existence, approximate solutions, and countable choice', Theoret. Comput. Sci., to appear. 90

[17] Rudolf Taschner, *Lehrgang der konstruktiven Mathematik* (three volumes), Manz and Hölder-Pichler-Tempsky, Wien, 1993, 1994, 1995 . 91

[18] Anne S. Troelstra and Dirk van Dalen, *Constructivism in Mathematics* (two volumes), North-Holland, Amsterdam, 1988. 91

Hoare Logics for Recursive Procedures and Unbounded Nondeterminism

Tobias Nipkow

Fakultät für Informatik, Technische Universität München
http://www.in.tum.de/~nipkow/

Abstract. This paper presents sound and complete Hoare logics for partial and total correctness of recursive parameterless procedures in the context of unbounded nondeterminism. For total correctness, the literature so far has either restricted recursive procedures to be deterministic or has studied unbounded nondeterminism only in conjunction with loops rather than procedures. We consider both single procedures and systems of mutually recursive procedures. All proofs have been checked with the theorem prover Isabelle/HOL.

1 Introduction

Hoare logic has been studied extensively since its inception [8], both for its theoretical interest and its practical relevance. Strangely enough, procedures have not been treated with adequate attention: many proof systems involving procedures are unsound, incomplete, or ignore completeness altogether (see [2, 21] for details). In particular the combination of procedures with (unbounded) nondeterminism was an open issue altogether.

Let us briefly review the history of Hoare logics for deterministic languages with procedures. The system proposed by Hoare [9] was later shown to be sound and complete by Olderog [21]. Apt [2] presents sound and complete systems both for partial correctness (following Gorelick [6]) and for total correctness (following and completing Sokołowski [25]). The one for total correctness is later found to be unsound by America and de Boer [1], who modify the system and give new soundness and completeness proofs. A new twist is added by Kleymann (né Schreiber) [24] who uses a little known consequence rule due to Morris [14] to subsume the three adaption rules by America and de Boer. In particular, he formalizes his work in the theorem prover LEGO [23]: this is the first time that a new Hoare logic is first proved sound and complete in a theorem prover.

We continue our earlier work on Hoare logic in Isabelle/HOL [17, 18] while taking advantage of Kleymann's technical advances. The main contribution of our paper is to simplify some aspects of Kleymann's proof system and, more importantly, to provide the first Hoare logics for partial and for total correctness of recursive procedures in the context of unbounded nondeterminism, both for single procedures and mutually recursive procedures. At this point we connect with the work by Apt [3] and Apt and Plotkin [4] on unbounded nondeterminism.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 103–119, 2002.
© Springer-Verlag Berlin Heidelberg 2002

The main differences are that they use ordinals and we use well-founded relations, and that they do not consider procedures, thus avoiding the difficulties explained below which are at the heart of many unsound and incorrect proof systems in the literature.

1.1 The Problem with Procedures

Consider the following parameterless procedure which calls itself recursively:

```
proc = if i=0 then skip else i := i-1; CALL; i := i+1
```

A classic example of the subtle problems associated with reasoning about procedures is the proof that `i` is invariant: {`i=N`} `CALL` {`i=N`}. This is done by induction: we assume {`i=N`} `CALL` {`i=N`} and have to prove {`i=N`} `body` {`i=N`}, where `body` is the body of the procedure. The case `i=0` is trivial. Otherwise we have to show {`i=N`}`i:=i-1;CALL;i:=i+1`{`i=N`}, which can be reduced to {`i=N-1`} `CALL` {`i=N-1`}. But how can we deduce{`i=N-1`} `CALL` {`i=N-1`} from the induction hypothesis {`i=N`} `CALL` {`i=N`}? Clearly, we have to instantiate `N` in the induction hypothesis — after all `N` is arbitrary as it does not occur in the program. The problems with procedures are largely due to unsound or incomplete adaption rules. We follow the solution of Morris and Kleymann and adjust the value of auxiliary variables like `N` with the help of the consequence rule. In §4.4 we show how this example is handled with our rules.

1.2 The Extensional Approach

In modelling the assertion language, we follow the *extensional* approach where assertions are identified with functions from states to propositions. That is, we model only the semantics but not the syntax of assertions. This is common practice in the theorem proving literature (with the exception of [11], but they do not consider completeness) and can also be found in standard sematics texts [16]. Because our underlying logic is higher order, *expressiveness*, i.e. whether the assertion language is strong enough to express all intermediate predicates that may arise in a proof, is not much of an issue. Thus our completeness results do not automatically carry over to other logical systems, say first order arithmetic. The advantage of the extensional approach is that it separates reasoning about programs from expressiveness considerations — the latter can then be conducted in isolation for each assertion language. We discuss this further in §6.

1.3 Isabelle/HOL

Isabelle/HOL [19] is an interactive theorem prover for HOL, higher-order logic. The whole paper is generated directly from the Isabelle input files, which include the text as comments. That is, if you see a lemma or theorem, you can be sure its proof has been checked by Isabelle. Most of the syntax of HOL will be familiar

to anybody with some background in functional programming and logic. We just highlight some of the nonstandard notation.

The space of total functions is denoted by the infix $\Rightarrow$. Other type constructors, e.g. *set*, are written postfix, i.e. follow their argument as in *state set*.

The syntax $[\![P;\ Q]\!] \Longrightarrow R$ should be read as an inference rule with the two premises P and Q and the conclusion R. Logically it is just a shorthand for $P \Longrightarrow Q \Longrightarrow R$. Note that semicolon will also denote sequential composition of programs, which should cause no major confusion. There are actually two implications $\longrightarrow$ and $\Longrightarrow$. The two mean the same thing, except that $\longrightarrow$ is HOL's "real" implication, whereas $\Longrightarrow$ comes from Isabelle's meta-logic and expresses inference rules. Thus $\Longrightarrow$ cannot appear inside a HOL formula. For the purpose of this paper the two may be identified. However, beware that $\longrightarrow$ binds more tightly than $\Longrightarrow$: in $\forall x.\ P \longrightarrow Q$ the $\forall x$ covers $P \longrightarrow Q$, whereas in $\forall x.\ P \Longrightarrow Q$ it covers only P.

Set comprehension is written $\{x.\ P\}$ rather than $\{x \mid P\}$ and is also available for tuples, e.g. $\{(x,\ y,\ z).\ P\}$.

2 Syntax and Operational Semantics

Everything is based on an unspecified type *state* of states. This could be a mapping from variables to values, but to keep things abstract we leave this open. The type *bexp* of boolean expressions is defined as an abbreviation:

types *bexp* = *state* $\Rightarrow$ *bool*

This model of boolean expressions requires a few words of explanation. Type *bool* is HOL's predefined type of propositions. Thus all the usual logical connectives like $\wedge$ and $\vee$ are available. Instead of modelling the syntax of boolean expressions, we model their semantics. For example, if states are mappings from variables to values, the programming language expression `x != y` becomes $\lambda s.\ s\ x \neq s\ y$.

The syntax of our programming language is defined by a recursive datatype (not shown). Statements in this language are called *commands*. Command *Do* f, where f is of type *state* $\Rightarrow$ *state set*, represents an atomic command that leads in one step from some state s to a new state $t \in f\ s$, or blocks if $f\ s$ is empty. Thus *Do* can represent many well-known constructs such as `skip` (*Do* $(\lambda s.\ \{s\})$), `abort` (*Do* $(\lambda s.\ \{\})$), and (random) assignment. This is the only source of nondeterminism, but other constructs, like a binary choice between commands, are easily simulated.

In addition we have sequential composition (*c1*; *c2*), conditional (*IF b THEN c1 ELSE c2*), iteration (*WHILE b DO c*) and the procedure call command *CALL*. There is only one parameterless procedure in the program. Hence *CALL* does not even need to mention the procedure name. There is no separate syntax for procedure declarations. Instead we introduce a new constant

consts *body* :: *com*

that represents the body of the one procedure in the program. Since *body* is unspecified, this is completely generic.

The semantics of commands is defined operationally, by the simplest possible scheme, a so-called *evaluation* or *big-step* semantics. Execution is defined via triples of the form $s\ -c\rightarrow\ t$ which should be read as "execution of c starting in state s may terminate in state t". This allows for different kinds of nondeterminism: there may be other terminating executions $s\ -c\rightarrow\ u$ with $t \neq u$, there may be nonterminating computations, and there may be blocking computations. Nontermination and blocking is only discussed in the context of total correctness. Execution of commands is defined inductively in the standard fashion and requires no comments. See §1.3 for the notation.

$t \in f\ s \Longrightarrow s\ -Do\ f\rightarrow\ t$
$[\![s0\ -c1\rightarrow\ s1;\ s1\ -c2\rightarrow\ s2]\!] \Longrightarrow s0\ -c1;\ c2\rightarrow\ s2$
$[\![b\ s;\ s\ -c1\rightarrow\ t]\!] \Longrightarrow s\ -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow\ t$
$[\![\neg\ b\ s;\ s\ -c2\rightarrow\ t]\!] \Longrightarrow s\ -IF\ b\ THEN\ c1\ ELSE\ c2\rightarrow\ t$
$\neg\ b\ s \Longrightarrow s\ -WHILE\ b\ DO\ c\rightarrow\ s$
$[\![b\ s;\ s\ -c\rightarrow\ t;\ t\ -WHILE\ b\ DO\ c\rightarrow\ u]\!] \Longrightarrow s\ -WHILE\ b\ DO\ c\rightarrow\ u$
$s\ -body\rightarrow\ t \Longrightarrow s\ -CALL\rightarrow\ t$

This semantics turns out not to be fine-grained enough. The soundness proof for the partial correctness Hoare logic below proceeds by induction on the call depth during execution. To make this work we define a second semantics $s\ -c-n\rightarrow\ t$ which expresses that the execution uses at most n nested procedure invocations, where n is a natural number. The rules are straightforward: n is just passed around, except for procedure calls, where it is decremented (*Suc* n is $n\ +\ 1$):

$t \in f\ s \Longrightarrow s\ -Do\ f-n\rightarrow\ t$
$[\![s0\ -c1-n\rightarrow\ s1;\ s1\ -c2-n\rightarrow\ s2]\!] \Longrightarrow s0\ -c1;\ c2-n\rightarrow\ s2$
$[\![b\ s;\ s\ -c1-n\rightarrow\ t]\!] \Longrightarrow s\ -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow\ t$
$[\![\neg\ b\ s;\ s\ -c2-n\rightarrow\ t]\!] \Longrightarrow s\ -IF\ b\ THEN\ c1\ ELSE\ c2-n\rightarrow\ t$
$\neg\ b\ s \Longrightarrow s\ -WHILE\ b\ DO\ c-n\rightarrow\ s$
$[\![b\ s;\ s\ -c-n\rightarrow\ t;\ t\ -WHILE\ b\ DO\ c-n\rightarrow\ u]\!] \Longrightarrow s\ -WHILE\ b\ DO\ c-n\rightarrow\ u$
$s\ -body-n\rightarrow\ t \Longrightarrow s\ -CALL-Suc\ n\rightarrow\ t$

By induction on $s\ -c-m\rightarrow\ t$ we show monotonicity w.r.t. the call depth:

lemma $s\ -c-m\rightarrow\ t \Longrightarrow \forall\, n.\ m \leq n \longrightarrow s\ -c-n\rightarrow\ t$

With the help of this lemma we prove the expected relationship between the two semantics:

lemma *exec-iff-execn*: $(s\ -c\rightarrow\ t) = (\exists\, n.\ s\ -c-n\rightarrow\ t)$

Both directions are proved separately by induction on the operational semantics.

3 Hoare Logic for Partial Correctness

As motivated in §1.1, auxiliary variables will be an integral part of our framework. This means that assertions must depend on them as well as on the state. Initially we do not fix the type of auxiliary variables but parameterize the type of assertions with a type variable $'a$:

types $'a\ assn = 'a \Rightarrow state \Rightarrow bool$

Reasoning about recursive procedures requires a context to store the induction hypothesis about recursive *CALL*s. This context is a set of Hoare triples:

types $'a\ cntxt = ('a\ assn \times com \times 'a\ assn)set$

In the presence of only a single procedure the context will always be empty or a singleton set. With multiple procedures, larger sets can arise. Contexts are denoted by C and D.

Validity (w.r.t. partial correctness) is defined as usual, except that we have to take auxiliary variables into account as well:

$$\models \{P\}c\{Q\} \ \equiv\ \forall s\ t.\ s\ {-c\rightarrow}\ t \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)$$

The state of the auxiliary variables (*auxiliary state* for short) is always denoted by z.

Validity of a context and a Hoare triple in a context are defined as follows:

$$\|\!\models C \ \equiv\ \forall (P,c,Q) \in C.\ \models \{P\}c\{Q\}$$
$$C \models \{P\}c\{Q\} \ \equiv\ \|\!\models C \longrightarrow \models \{P\}c\{Q\}$$

Note that $\{\} \models \{P\}\ c\ \{Q\}$ is equivalent to $\models \{P\}\ c\ \{Q\}$.

Unfortunately, this is not the end of it. As we have two semantics, $-c\rightarrow$ and $-c-n\rightarrow$, we also need a second notion of validity parameterized with the recursion depth n:

$$\models n\ \{P\}c\{Q\} \ \equiv\ \forall s\ t.\ s\ {-c-n\rightarrow}\ t \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)$$
$$\|\!\models n\ C \ \equiv\ \forall (P,c,Q) \in C.\ \models n\ \{P\}c\{Q\}$$
$$C \models n\ \{P\}c\{Q\} \ \equiv\ \|\!\models n\ C \longrightarrow \models n\ \{P\}c\{Q\}$$

Finally we come to the proof system for deriving triples in a context:

$C \vdash \{\lambda z\ s.\ \forall t \in f\ s.\ P\ z\ t\}\ Do\ f\ \{P\}$
$[\![C\vdash\{P\}c1\{Q\}; C\vdash\{Q\}c2\{R\}]\!] \Longrightarrow C \vdash \{P\}\ c1;\ c2\ \{R\}$
$[\![C \vdash \{\lambda z\ s.\ P\ z\ s \wedge b\ s\}\ c1\ \{Q\};\ C \vdash \{\lambda z\ s.\ P\ z\ s \wedge \neg\ b\ s\}\ c2\ \{Q\}]\!]$ $\Longrightarrow C \vdash \{P\}\ IF\ b\ THEN\ c1\ ELSE\ c2\ \{Q\}$
$C \vdash \{\lambda z\ s.\ P\ z\ s \wedge b\ s\}\ c\ \{P\} \Longrightarrow C \vdash \{P\}\ WHILE\ b\ DO\ c\ \{\lambda z\ s.P\ z\ s\wedge\neg b\ s\}$
Consequence: $[\![C \vdash \{P'\}\ c\ \{Q'\}; \forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)]\!]$ $\Longrightarrow C \vdash \{P\}\ c\ \{Q\}$
CALL: $\{(P,\ CALL,\ Q)\} \vdash \{P\}\ body\ \{Q\} \Longrightarrow \{\} \vdash \{P\}\ CALL\ \{Q\}$
Assumption: $\{(P,\ CALL,\ Q)\} \vdash \{P\}\ CALL\ \{Q\}$

Note that Hoare triples to the left of $\vdash$ are written as real triples, whereas to the right of $\vdash$ both the customary $\{P\}\ c\ \{Q\}$ syntax and ordinary triples are permitted.

The rule for *Do* is the generalization of the assignment axiom to an arbitrary nondeterministic state transformation. The next 3 rules are familiar, except for their adaptation to auxiliary variables. The *CALL* rule embodies induction and has already been motivated in §1.1. Note that it is only applicable if the context is empty. This shows that we never need nested induction. For the same reason the assumption rule is stated with just a singleton context.

The consequence rule is unusual but not completely new. Modulo notation it is identical to a slight reformulation by Olderog [21] of a rule by Cartwright and Oppen [5]. A different reformulation of the rule seems to have appeared for the first time in the work by Morris [14]. A more recent reinvention and reformulation is due to Hofmann [10]:

$$\forall s\ t\ z.\ P\ z\ s \longrightarrow Q\ z\ t \vee (\exists z'.\ P'\ z'\ s \wedge (Q'\ z'\ t \longrightarrow Q\ z\ t))$$

Although logically equivalent to our side condition, the symmetry of our version appeals not just for aesthetic reasons but because one can actually remember it!

Our system differs from earlier Hoare logics for partial correctness because we have followed Kleymann [24] who realized that a rule like the above consequence rule subsumes the normal consequence rule — thus the latter has become superfluous.

The proof of the soundness theorem

theorem $C \vdash \{P\}c\{Q\} \implies C \models \{P\}c\{Q\}$

requires a generalization: $\forall n.\ C \models n\ \{P\}\ c\ \{Q\}$ is proved instead, from which the actual theorem follows directly via lemma *exec-iff-execn*. The generalization is proved by induction on $C \vdash \{P\}\ c\ \{Q\}$.

The completeness proof follows the *most general triple* approach [6]:

$$MGT :: com \Rightarrow state\ assn \times com \times state\ assn$$
$$MGT\ c \equiv (\lambda z\ s.\ z = s,\ c,\ \lambda z\ t.\ z\ {-c\rightarrow}\ t)$$

There are a number of points worth noting. For a start, the most general triple equates the type of the auxiliary state z with type *state*. The precondition equates the auxiliary state with the initial state, so to speak making a copy of it. Therefore the postcondition can refer to this copy and thus the initial state. Finally, the postcondition is the strongest postcondition w.r.t. the given precondition and command.

It is easy to see that $\{\} \vdash MGT\ c$ implies completeness:

lemma *MGT-implies-complete*:
$\{\} \vdash MGT\ c \implies \{\} \models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q{::}state\ assn\}$

Simply apply the consequence rule to $\{\} \vdash MGT\ c$ to obtain $\{\} \vdash \{P\}\ c\ \{Q\}$ — the side condition is discharged with the help of $\{\} \models \{P\}\ c\ \{Q\}$ and a little predicate calculus reasoning. The type constraint $Q{::}state\ assn$ is required because pre and postconditions in $MGT\ c$ are of type *state assn*, not *'a assn*.

In order to discharge $\{\} \vdash MGT\ c$ one proves

lemma *MGT-lemma*: $C \vdash MGT\ CALL \implies C \vdash MGT\ c$

The proof is by induction on c. In the *WHILE*-case it is easy to show that $\lambda z\, t.\ (z, t) \in \{(s, t).\ b\ s \wedge s -c\rightarrow t\}^*$ is invariant. The precondition $\lambda z\, s.\ z{=}s$ establishes the invariant and a reflexive transitive closure induction shows that the invariant conjoined with $\neg\ b\ t$ implies the postcondition $\lambda z\, t.\ z -WHILE\ b\ DO\ c\rightarrow t$. The remaining cases are trivial.

We can now derive $\{\} \vdash MGT\ c$ as follows. By the assumption rule we have $\{MGT\ CALL\} \vdash MGT\ CALL$, which implies $\{MGT\ CALL\} \vdash MGT\ body$ by the *MGT-lemma*. From the *CALL* rule it follows that $\{\} \vdash MGT\ CALL$. Applying the *MGT-lemma* once more we obtain the desired $\{\} \vdash MGT\ c$ and hence completeness:

theorem $\{\} \models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q{::}state\ assn\}$

This is the first proof of completeness in the presence of (unbounded) nondeterminism. Earlier papers, if they considered completeness at all, restricted themselves to deterministic languages. However, our completeness proof follows the one by Apt [2] quite closely. This will no longer be the case for total correctness.

4 Hoare Logic for Total Correctness

4.1 Termination

To express total correctness, we need to talk about guaranteed termination of commands. Due to nondeterminism, the existence of a terminating computation in the big-step semantics does not guarantee that all computations from some state terminate. Hence we inductively define a new judgement $c \downarrow s$ that expresses guaranteed termination of c started in state s:

$f\ s \neq \{\} \implies Do\ f \downarrow s$
$[\![c1 \downarrow s0;\ \forall s1.\ s0 -c1\rightarrow s1 \longrightarrow c2 \downarrow s1]\!] \implies (c1;\ c2) \downarrow s0$
$[\![b\ s;\ c1 \downarrow s]\!] \implies IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$
$[\![\neg\ b\ s;\ c2 \downarrow s]\!] \implies IF\ b\ THEN\ c1\ ELSE\ c2 \downarrow s$
$\neg\ b\ s \implies WHILE\ b\ DO\ c \downarrow s$
$[\![b\ s;\ c \downarrow s;\ \forall t.\ s -c\rightarrow t \longrightarrow WHILE\ b\ DO\ c \downarrow t]\!] \implies WHILE\ b\ DO\ c \downarrow s$
$body \downarrow s \implies CALL \downarrow s$

The first rule expresses that if $Do\ f$ blocks, i.e. there is no next state in $f\ s$, we do not consider this a normal termination. Thus $\downarrow$ rules out both infinite and blocking computations. The remaining rules are self-explanatory.

By induction on $\downarrow$ it is easily shown that if *WHILE* terminates in the sense of $\downarrow$ then one must eventually reach a state where the loop test becomes false:

lemma $[\![\ (WHILE\ b\ DO\ c) \downarrow f\ k;\ \ \forall i.\ f\ i -c\rightarrow f(Suc\ i)\ ]\!] \implies \exists i.\ \neg b(f\ i)$

The inductive proof requires $f\ k$ rather than the more intuitive $f\ 0$.

It follows that the executions of the body of a terminating *WHILE*-loop form a well-founded relation (for *wf* see below):

lemma *wf-WHILE*: $wf\ \{(t,s).\ WHILE\ b\ DO\ c \downarrow s \wedge b\ s \wedge s\ -c\rightarrow\ t\}$

Now that we have termination, we can define total validity, $\models_t$, as partial validity and guaranteed termination:

$$\models_t \{P\}c\{Q\} \ \equiv\ \models \{P\}c\{Q\} \wedge (\forall z\ s.\ P\ z\ s \longrightarrow c{\downarrow}s)$$

For validity of a context and validity of a Hoare triple in a context we follow the corresponding definitions for partial correctness:

$$|\models_t C \ \equiv\ \forall (P,c,Q) \in C.\ \models_t \{P\}c\{Q\}$$
$$C \models_t \{P\}c\{Q\} \ \equiv\ |\models_t C \longrightarrow \models_t \{P\}c\{Q\}$$

4.2 Hoare Logic

To distinguish the proofs of partial and total correctness the latter use the symbol $\vdash_t$. The rules for $\vdash_t$ differ from the ones for $\vdash$ only in the two places where nontermination can arise (loops and recursion) and in the consequence rule:

$[\![wf\ r;\ \forall s'.\ C \vdash_t \{\lambda z\ s.\ P\ z\ s \wedge b\ s \wedge s' = s\}\ c\ \{\lambda z\ s.\ P\ z\ s \wedge (s,\ s') \in r\}]\!]$ $\Longrightarrow C \vdash_t \{P\}\ WHILE\ b\ DO\ c\ \{\lambda z\ s.\ P\ z\ s \wedge \neg\ b\ s\}$
$[\![wf\ r;\ \forall s'.\ \{(\lambda z\ s.\ P\ z\ s \wedge (s,\ s') \in r,\ CALL,\ Q)\} \vdash_t$ $\{\lambda z\ s.\ P\ z\ s \wedge s = s'\}\ body\ \{Q\}]\!]$ $\Longrightarrow \{\} \vdash_t \{P\}\ CALL\ \{Q\}$
$[\![C \vdash_t \{P'\}\ c\ \{Q'\};$ $(\forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)) \wedge$ $(\forall s.\ (\exists z.\ P\ z\ s) \longrightarrow (\exists z.\ P'\ z\ s))]\!]$ $\Longrightarrow C \vdash_t \{P\}\ c\ \{Q\}$

Before we discuss these rules in turn, a note on *wf*, which means well-founded: a relation r is well-founded iff there is no infinite descending chain

$$\ldots, (s_3, s_2), (s_2, s_1), (s_1, s_0) \in r.$$

The *WHILE*-rule is fairly standard: in addition to invariance one must also show that the state goes down w.r.t. some well-founded relation r. The only notable feature is the universal quantifier $(\forall s')$ that allows the postcondition to refer to the initial state. If you are used to more syntactic presentations of Hoare logic you may prefer a side condition that s' is a new variable. But since we embed Hoare logic in a language with quantifiers, why not use them to good effect?

The *CALL*-rule is like the one for partial correctness except that use of the induction hypothesis is restricted to those cases where the state has become smaller w.r.t. r. The $\forall s'$ fulfills a similar function as in the *WHILE*-rule. See §4.4 for an application of this rule which elucidates how $\forall s'$ is handled.

The consequence rule is like its cousin for partial correctness but with a version of precondition strengthening conjoined that takes care of the auxiliary state z: $\forall s.\ (\exists z.\ P\ z\ s) \longrightarrow (\exists z.\ P'\ z\ s)$.

Our rules for total correctness are similar to those by Kleymann [13]. The difference in the *WHILE*-rule is that he has a well-founded relation on some arbitrary type α together with a function from *state* to α, which we have collapsed to a well-founded relation on *state*. This is equivalent but avoids the additional type α. The same holds for the *CALL*-rule. As a consequence our *CALL*-rule is much simpler than the one by Kleymann (and ultimately Sokolowski [25]) because we avoid the additional existential quantifiers over values of type α. Finally, the side condition in our rule of consequence looks quite different from the one by Kleymann, although the two are in fact equivalent:

lemma $((\forall s\ t.\ (\forall z.\ P'\ z\ s \longrightarrow Q'\ z\ t) \longrightarrow (\forall z.\ P\ z\ s \longrightarrow Q\ z\ t)) \wedge$
$(\forall s.\ (\exists z.\ P\ z\ s) \longrightarrow (\exists z.\ P'\ z\ s)))$
$= (\forall z\ s.\ P\ z\ s \longrightarrow (\forall t. \exists z'.\ P'\ z'\ s \wedge (Q'\ z'\ t \longrightarrow Q\ z\ t)))$

Kleymann's version (the proposition to the right of the =) is easier to use because it is more compact, whereas our new version clearly shows that it is a conjunction of the side condition for partial correctness with precondition strengthening, which is not obvious in Kleymann's formulation. Further equivalent formulations are explored by Naumann [15].

As usual, soundness is proved by induction on $C \vdash_t \{P\}\ c\ \{Q\}$:

theorem $C \vdash_t \{P\}c\{Q\} \implies C \models_t \{P\}c\{Q\}$

The *WHILE* and *CALL*-cases require well-founded induction along the given well-founded relation.

The key difference to previous work in the literature (Kleymann, America and de Boer, Apt, etc) emerges in the completeness proof. For total correctness, the most general triple used to be turned around: $\lambda z\ t.\ z\ {-c\rightarrow}\ t$ becomes the weakest precondition of $\lambda z\ s.\ z = s$. However, this only works if the programming language is deterministic. Hence we leave the most general triple as it is and merely add the termination requirement to the precondition:

$MGT_t\ c\ \equiv\ (\lambda z\ s.\ z = s \wedge c{\downarrow}s,\ c,\ \lambda z\ t.\ z\ {-c\rightarrow}\ t)$

The first two lemmas on the way to the completeness proof are unchanged:

lemma $\{\} \vdash_t MGT_t\ c \implies \{\} \models_t \{P\}c\{Q\} \implies \{\} \vdash_t \{P\}c\{Q{::}state\ assn\}$
lemma $C \vdash_t MGT_t\ CALL \implies C \vdash_t MGT_t\ c$

However, if we now try to continue following the proof at the end of §3 to derive $\{\} \vdash_t MGT_t\ c$ we can no longer do so directly because the *CALL*-rule has changed. What we would need is the following lemma:

lemma *CALL-lemma*:
$\{(\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge (s,s') \in rcall,\ CALL,\ \lambda z\ s.\ z\ {-body\rightarrow}\ s)\} \vdash_t$
$\{\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge s = s'\}\ body\ \{\lambda z\ s.\ z\ {-body\rightarrow}\ s\}$

where *rcall* is some suitable well-founded relation. From that lemma the *CALL*-rule infers $\{\} \vdash_t \{\lambda z\ s.\ z = s \wedge CALL \downarrow s\}\ CALL\ \{\lambda z\ s.\ z\ {-CALL\rightarrow}\ s\}$ which is exactly $\{\} \vdash_t MGT_t\ CALL$. Completness follows trivially via the two lemmas

further up. However, before we can even start to prove the hypothetical *CALL-lemma*, we need to provide the well-founded relation *rcall*, which turns out to be the major complicating factor.

Given a terminating *WHILE*, the iterated executions of the body directly yield the well-founded relation that proves termination. In contrast, given a terminating *CALL*, the big-step semantics does not yield a well-founded relation on states that decreases between the beginning of the execution of the body and a recursive call. The reason is that the recursive call is embedded in the body and thus the big-step semantics is too coarse. Informally what we want is the relation $\{(s', s) \mid \text{starting the body in state } s \text{ leads to a recursive } CALL \text{ in state } s'\}$.

4.3 The Termination Ordering

In order to formalize the above informal description of the termination ordering we define a very fine-grained small-step semantics that one can view as an abstract machine operating on a command stack. Each step $(cs,\ s) \rightarrow (cs',\ s')$ (partially) executes the topmost element of the command stack *cs*, possibly replacing it with a list of new commands. Note that $x \# xs$ is the list with head x and tail xs.

$t \in f\ s \Longrightarrow (Do\ f \# cs,\ s) \rightarrow (cs,\ t)$
$((c1;\ c2) \# cs,\ s) \rightarrow (c1 \# c2 \# cs,\ s)$
$b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2) \# cs,\ s) \rightarrow (c1 \# cs,\ s)$
$\neg\ b\ s \Longrightarrow ((IF\ b\ THEN\ c1\ ELSE\ c2) \# cs,\ s) \rightarrow (c2 \# cs,\ s)$
$\neg\ b\ s \Longrightarrow ((WHILE\ b\ DO\ c) \# cs,\ s) \rightarrow (cs,\ s)$
$b\ s \Longrightarrow ((WHILE\ b\ DO\ c) \# cs,\ s) \rightarrow (c \# (WHILE\ b\ DO\ c) \# cs,\ s)$
$(CALL \# cs,\ s) \rightarrow (body \# cs,\ s)$

Note that a separate *SKIP* command would obviate the need for lists: simply replace [] by *SKIP* and # by ;.

The above semantics is intentionally different from the customary structural operational semantics. The latter features the following rule:

$$(c1,s) \rightarrow (c1',s') \Longrightarrow (c1;c2,s) \rightarrow (c1';c2,s')$$

In case *c1* is a nest of semicolons, it is not flattened as above, and hence one cannot easily see what the next atomic command is. Which we need to see, to define *rcall*, the well-founded ordering required for the application of the *CALL* rule in the completeness proof in §4.2 above:

$$rcall \equiv \{(t,s).\ body{\downarrow}s \wedge (\exists\, cs.\ ([body],\ s) \rightarrow^* (CALL \# cs,\ t))\}$$

theorem *wf rcall*

The amount of work to prove this theorem is significant and should not be underestimated, but for lack of space we cannot discuss the details. The complexity of the proof is due to the two notions of (non)termination, $\downarrow$ and infinite $\rightarrow$ reductions, which need to be related. However, abolishing $\downarrow$ would help very little:

the lengthy proofs are those about $\rightarrow$, and one would then need to replace a few slick proofs via $\downarrow$ by more involved ones via $\rightarrow$.

To finish the completeness proof in §4.2 it remains to prove *CALL-lemma*. It cannot be proved directly but needs to be generalized first:

lemma $\{(\lambda z\ s.\ (z{=}s \wedge body{\downarrow}s) \wedge (s,t) \in rcall,\ CALL,\ \lambda z\ s.\ z\ {-}body{\rightarrow}\ s)\} \vdash_t$
$\{\lambda z\ s.\ (z{=}s \wedge body{\downarrow}t) \wedge (\exists\, cs.\ ([body],t) \rightarrow^* (c\#cs,s))\}\ c\ \{\lambda z\ s.\ z\ {-}c{\rightarrow}\ s\}$

This lemma is proved by induction on c. The *WHILE*-case is a little involved and requires a local reflexive transitive closure induction.

The actual *CALL-lemma* follows easily, as does completeness:

theorem $\{\} \models_t \{P\}c\{Q\} \implies \{\} \vdash_t \{P\}c\{Q{::}state\ assn\}$

4.4 Example

To elucidate the use of our very semantic-looking proof rules we will now verify the example from §1.1, showing only the key steps and minimizing Isabelle-specific detail. We start by declaring a type *variables* and defining *state* to be *variables* $\Rightarrow$ *nat* — the variables in the example program range only over natural numbers. The program variable `i` is represented by a constant i of type *variables*. The body of the recursive procedure is defined by translating tests and assignments into functions on states. Updating a function s at point x with value e is a predefined operation written $s(x := e)$.

$body \equiv IF\ \lambda s.\ s\ i = 0\ THEN\ Do(\lambda s.\{s\})$
$ELSE\ (Do(\lambda s.\ \{s(i := s\ i - 1)\});\ CALL;\ Do(\lambda s.\ \{s(i := s\ i + 1)\}))$

We will now prove the desired correctness statement:

lemma $\{\} \vdash_t \{\lambda z\ s.\ s\ i = z\ N\}\ CALL\ \{\lambda z\ s.\ s\ i = z\ N\}$

As a first step we apply the *CALL*-rule where we instantiate r to $\{(t,\ s).\ t\ i < s\ i\}$ — well-foundedness of this relation is proved automatically. This leaves us with the following goal:

$1.\ \forall s'.\ \{(\lambda z\ s.\ s\ i = z\ N \wedge s\ i < s'\ i,\ CALL,\ \lambda z\ s.\ s\ i = z\ N)\} \vdash_t$
$\{\lambda z\ s.\ s\ i = z\ N \wedge s = s'\}\ body\ \{\lambda z\ s.\ s\ i = z\ N\}$

Isabelle always numbers goals. In this case there is only one. We get rid of the leading $\forall s'$ via HOL's $\forall$-introduction rule which turns it into $\bigwedge s'$, the universal quantifier of Isabelle's meta-logic. Roughly speaking this means that s' is now considered an arbitrary but fixed value.

After unfolding the *body* we apply the *IF*-rule and are left with two subgoals:

$1.\ \bigwedge s'.\ \{(\lambda z\ s.\ s\ i = z\ N \wedge s\ i < s'\ i,\ CALL,\ \lambda z\ s.\ s\ i = z\ N)\} \vdash_t$
$\{\lambda z\ s.\ (s\ i = z\ N \wedge s = s') \wedge s\ i = 0\}\ Do\ (\lambda s.\ \{s\})\ \{\lambda z\ s.\ s\ i = z\ N\}$
$2.\ \bigwedge s'.\ \{(\lambda z\ s.\ s\ i = z\ N \wedge s\ i < s'\ i,\ CALL,\ \lambda z\ s.\ s\ i = z\ N)\} \vdash_t$
$\{\lambda z\ s.\ (s\ i = z\ N \wedge s = s') \wedge s\ i \neq 0\}$
$Do\ (\lambda s.\ \{s(i := s\ i - 1)\});\ CALL;\ Do\ (\lambda s.\ \{s(i := s\ i + 1)\})$
$\{\lambda z\ s.\ s\ i = z\ N\}$

Both are easy to prove. During the proof of the second one we provide the intermediate assertions $\lambda z\ s.\ 0 < z\ N \wedge s\ i = z\ N - 1 \wedge s\ i < s'\ i$ and $\lambda z\ s.\ 0 < z\ N \wedge s\ i = z\ N - 1$. This leads to the following subgoal for the *CALL*:

$$
\begin{array}{l}
1.\ \bigwedge s'.\ \{(\lambda z\ s.\ s\ i = z\ N \wedge s\ i < s'\ i,\ CALL,\ \lambda z\ s.\ s\ i = z\ N)\} \vdash_t \\
\qquad \{\lambda z\ s.\ 0 < z\ N \wedge s\ i = z\ N - 1 \wedge s\ i < s'\ i\}\ CALL \\
\qquad \{\lambda z\ s.\ 0 < z\ N \wedge s\ i = z\ N - 1\}
\end{array}
$$

Applying consequence and assumption rules we are left with

$$
\begin{array}{l}
1.\ \bigwedge s'.\ (\forall s\ t.\ (\forall z.\ s\ i = z\ N \wedge s\ i < s'\ i \longrightarrow t\ i = z\ N) \longrightarrow \\
\qquad\qquad (\forall z.\ 0 < z\ N \wedge s\ i = z\ N - 1 \wedge s\ i < s'\ i \longrightarrow \\
\qquad\qquad\qquad 0 < z\ N \wedge t\ i = z\ N - 1)) \wedge \\
\qquad\qquad (\forall s.\ (\exists z.\ 0 < z\ N \wedge s\ i = z\ N - 1 \wedge s\ i < s'\ i) \longrightarrow \\
\qquad\qquad\quad (\exists z.\ s\ i = z\ N \wedge s\ i < s'\ i))
\end{array}
$$

which is proved automatically. This concludes the sketch of the proof.

5 More Procedures

We now generalize from a single procedure to a whole set of procedures following the ideas of von Oheimb [20]. The basic setup of §2 is modified only in a few places:

- We introduce a new basic type *pname* of procedure names.
- Constant *body* is now of type $pname \Rightarrow com$.
- The *CALL* command now has an argument of type *pname*, the name of the procedure that is to be called.
- The call rule of the operational semantics now says

$$s\ -body\ p\rightarrow\ t \Longrightarrow s\ -CALL\ p\rightarrow\ t$$

Note that this setup assumes that we have a procedure body for each procedure name. In particular, *pname* may be infinite.

5.1 Hoare Logic for Partial Correctness

Types *assn* and and *cntxt* are defined as in §3, as are $\models \{P\}\ c\ \{Q\}$, $|\!\models C$, $\models n\ \{P\}\ c\ \{Q\}$ and $|\!\models n\ C$. However, we now need an additional notion of validity $C\ |\!\models\ D$ where D is a set as well. The reason is that we can now have mutually recursive procedures whose correctness needs to be established by simultaneous induction. Instead of sets of Hoare triples we may think of conjunctions. We define both $C\ |\!\models\ D$ and its relativized version:

$$C\ |\!\models\ D \equiv |\!\models\ C \longrightarrow |\!\models\ D$$
$$C\ |\!\models n\ D \equiv |\!\models n\ C \longrightarrow |\!\models n\ D$$

Our Hoare logic defines judgements of the form $C \Vdash D$ where both C and D are (potentially infinite) sets of Hoare triples; $C \vdash \{P\}\ c\ \{Q\}$ is simply an abbreviation for $C \Vdash \{(P,c,Q)\}$. With this abbreviation the rules for ";", *IF*, *WHILE* and consequence are exactly the same as in §3. The remaining rules are

$\bigcup p.\ \{(P\ p,\ CALL\ p,\ Q\ p)\} \Vdash \bigcup p.\ \{(P\ p,\ body\ p,\ Q\ p)\} \Longrightarrow \{\} \Vdash \bigcup p.\ \{(P\ p,\ CALL\ p,\ Q\ p)\}$
$(P,\ CALL\ p,\ Q) \in C \Longrightarrow C \vdash \{P\}\ CALL\ p\ \{Q\}$
$\forall (P,\ c,\ Q) \in D.\ C \vdash \{P\}\ c\ \{Q\} \Longrightarrow C \Vdash D$
$[\![C \Vdash D;\ (P,\ c,\ Q) \in D]\!] \Longrightarrow C \vdash \{P\}\ c\ \{Q\}$

Note that $\bigcup p.$ is the indexed union $\bigcup_p$.

The *CALL* and the assumption rule are straightforward generalizations of their counterparts in §3. The fact that *CALL*-rule reasons about *all* procedures simultaneously merely simplifies notation: arbitrary subsets of procedures work just as well. The final two rules are structural rules and could be called conjunction introduction and elimination, because they put together and take apart sets of triples.

Soundness is proved as before, by induction on $C \Vdash D$:

theorem $C \Vdash D \implies C \mid\!\models D$

But first we generalize from $C \mid\!\models D$ to $\forall n.\ C \mid\!\models n\ D$. Now the *CALL*-case can be proved by induction on n.

The completeness proof also resembles the one in §3 closely: the most general triple *MGT* is defined exactly as before, and the lemmas leading up to completness are simple generalizations:

lemma $\{\} \vdash MGT\ c \Longrightarrow \models \{P\}c\{Q\} \Longrightarrow \{\} \vdash \{P\}c\{Q{::}state\ assn\}$
lemma $\forall p.\ C \vdash MGT(CALL\ p) \implies C \vdash MGT\ c$
lemma $\{\} \Vdash \bigcup p.\ \{MGT(CALL\ p)\}$
theorem $\models \{P\}c\{Q\} \implies \{\} \vdash \{P\}c\{Q{::}state\ assn\}$

5.2 Hoare Logic for Total Correctness

Hoare logic for total correctness of mutually recursive procedures has not received much attention in the literature. Sokolowski's system [25], the only one that comes with a completness proof, is seriously incomplete, as it lacks rules of adaption to deal with the problem described in §1.1.

Our basic setup of termination and validity is as in §4 but extended by one more notion of validity:

$C \mid\!\models_t D \equiv \mid\!\models_t C \longrightarrow \mid\!\models_t D$

The rules for *Do*, ";", *IF*, *WHILE* and consequence are exactly the same as in §4.2. In addition we have the two structural rules called conjunction introduction and elimination from §5.1 above (but with $\vdash_t$ instead of $\vdash$). Only the *CALL*-rule changes substantially and becomes

$[\![\ wf\ r;$
$\quad \forall q\ pre.\ (\bigcup p.\ \{(\lambda z\ s.\ P\ p\ z\ s \wedge ((p,s),(q,pre)) \in r, CALL\ p, Q\ p)\})$

$$\vdash_t \{\lambda z\ s.\ P\ q\ z\ s \wedge s = pre\}\ body\ q\ \{Q\ q\}\ \rrbracket$$
$$\Longrightarrow \{\} \Vdash_t \bigcup p.\ \{(P\ p,\ CALL\ p,\ Q\ p)\}$$

This rule appears to be genuinely novel. To understand it, imagine how you would simulate mutually recursive procedures by a single procedure: you combine all procedure bodies into one procedure and select the correct one dynamically with the help of a new program variable which holds the name of the currently called procedure. The well-founded relation in the above rule is of type $((pname \times state) \times (pname \times state))set$ thus simulating the additional program variable by making *pname* a component of the termination relation.

We consider an example from [12] which the authors claim is difficult to treat with previous approaches [25, 22].

```
proc pedal = if n=0 ∨ m=0 then skip
             else if n < m then (n:=n-1; m:=m-1; CALL coast)
             else (n:=n-1; CALL pedal)
proc coast = if n<m then (m:=m-1; CALL coast) else CALL pedal
```

One possible termination ordering (which is all we are interested in) is the reverse lexicographic product of the relation $\{(\texttt{pedal}, \texttt{coast})\}$ on *pname* with the lexicographic ordering on $(\texttt{n}, \texttt{m})$. If `coast` calls `pedal`, $(\texttt{n}, \texttt{m})$ is unchanged and the relation on *pname* decreases. In all other cases either `n` decreases or `n` is unchanged and `m` decreases.

Soundness and completeness are proved almost exactly as for a single procedure. We do not even need to show the theorems. Previous work on total correctness of mutually recursive procedures is either incomplete [25] or lacks completeness proofs [22, 12].

6 Expressiveness and Relative Completeness

In the literature, most completeness results for Hoare logics are qualified with the word *relative*, meaning relative to the completeness of the deductive system for the assertion language, which enters the picture in the consequence rule. This issue is absent in our formalization for the following reason: both $\models$ and $\vdash$ are specified in the same finite logical system, HOL. Thus they both inherit HOL's incompleteness. In particular, there must be valid Hoare triples whose validity is not provable in HOL. What the completeness theorem tells us is that both $\models$ and $\vdash$ are equally incomplete. This is important because it means we never need to resort to the operational semantics to prove some Hoare triple, we can always do it just as well in the Hoare logic.

The second important issues that we have ignored so far is expressiveness, i.e. the ability to express the intermediate predicates that may arise in a proof. In the following discussion we restrict attention to programs where the boolean expressions and the functions in the *Do*-commands are definable in the assertion language.

Clearly HOL is expressive as the completeness proofs can be formalized in it. We will narrow things down to weaker logical systems, although an analysis of the precise proof theoretic strength required is beyond the scope of this paper. For partial correctness the customary result that first-order arithmetic is expressive still holds, essentially because the most general triple can be expressed in it. The details are standard. For total correctness matters change. First-order arithmetic is still expressive for bounded nondeterminism (as shown by Apt [3] for Dijkstra's guarded commands) but fails to be so in the presence of unbounded nondeterminism [3, 4]. The reason is that we now have to formalize assertions about termination. Apt solves the problem by enriching the assertion language with a least fixedpoint operator, i.e. moving towards the μ-calculus. Essentially we have used the same trick: termination ($\downarrow$) is defined inductively, which can be expressed as a least fixedpoint (and this is in fact what Isabelle/HOL translates inductive definitions into internally). Therefore first-order arithmetic enriched with least fixedpoints is expressive in our setting, too.

However, there is one more complication: our proof rules for loops and procedure calls employ arbitrary well-founded orderings on the state space. Fortunately we can dispense with general well-founded orderings. Studying the completeness proof in §4, we find that two termination orderings suffice, namely the one in lemma *wf-WHILE* for loops (§4.1) and *rcall* for procedures (§4.3). Hence we could specialize the two rules with these most general termination orderings, thus removing the well-foundedness premise while retaining completeness. And if we examine the definition of these orderings, we find that they require the same ingredients as the most general triple, namely the transition semantics and the termination predicate ($\downarrow$). Thus the version of the μ-calculus used by Apt [3, 4], or any reasonable logic that can express most general triples, is expressive for procedures as well. In contrast, Apt and Plotkin require (recursive) ordinals on top of their μ-calculus. They are aware that the ordinals are strictly speaking not necessary (Hitchcock and Park [7] do without them) but leave it as an open question to find a *syntax directed* system without ordinals. Our proof system provides one answer.

Acknowledgments

I am indebted to Thomas Kleymann and David von Oheimb for providing the logical foundations, to Krzysztof Apt and Kamal Lodaya for very helpful comments, and to Markus Wenzel for the Isabelle document preparation system.

References

[1] Pierre America and Frank de Boer. Proving total correctness of recursive procedures. *Information and Computation*, 84:129–162, 1990. 103

[2] Krzysztof Apt. Ten Years of Hoare's Logic: A Survey — Part I. *ACM Trans. Programming Languages and Systems*, 3(4):431–483, 1981. 103, 109

[3] Krzysztof Apt. Ten Years of Hoare's Logic: A Survey — Part II: Nondeterminism. *Theoretical Computer Science*, 28:83–109, 1984. 103, 117

[4] Krzysztof Apt and Gordon Plotkin. Countable nondeterminism and random assignment. *Journal of the ACM*, 33:724–767, 1986. 103, 117

[5] Robert Cartwright and Derek Oppen. The logic of aliasing. *Acta Informatica*, 15:365–384, 1981. 108

[6] Gerald Arthur Gorelick. A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report 75, Dept. of Computer Science, Univ. of Toronto, 1975. 103, 108

[7] Peter Hitchcock and David Park. Induction rules and termination proofs. In M. Nivat, editor, *Automata, languages, and programming*, pages 225–251. North Holland, 1973. 117

[8] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:567–580,583, 1969. 103

[9] C. A. R. Hoare. Procedures and parameters: An axiomatic approach. In E. Engeler, editor, *Semantics of algorithmic languages*, volume 188 of *Lecture Notes in Mathematics*, pages 102–116. Springer-Verlag, 1971. 103

[10] Martin Hofmann. Semantik und Verifikation. Lecture notes, Universität Marburg. In German, 1997. 108

[11] Peter V. Homeier and David F. Martin. Mechanical verification of mutually recursive procedures. In M. A. McRobbie and J. K. Slaney, editors, *Automated Deduction — CADE-13*, volume 1104 of *Lect. Notes in Comp. Sci.*, pages 201–215. Springer-Verlag, 1996. 104

[12] Peter V. Homeier and David F. Martin. Mechanical verification of total correctness through diversion verification conditions. In J. Grundy and M. Newey, editors, *Theorem Proving in Higher Order Logics (TPHOLs'98)*, volume 1479 of *Lect. Notes in Comp. Sci.*, pages 189–206. Springer-Verlag, 1998. 116

[13] Thomas Kleymann. Hoare logic and auxiliary variables. *Formal Aspects of Computing*, 11:541–566, 1999. 111

[14] J. H. Morris. Comments on "procedures and parameters". Undated and unpublished. 103, 108

[15] David Naumann. Calculating sharp adaptation rules. *Information Processing Letters*, 77:201–208, 2000. 111

[16] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications*. Wiley, 1992. 104

[17] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. In V. Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *Lect. Notes in Comp. Sci.*, pages 180–192. Springer-Verlag, 1996. 103

[18] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. *Formal Aspects of Computing*, 10:171–186, 1998. 103

[19] Tobias Nipkow, Lawrence Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lect. Notes in Comp. Sci.* Springer-Verlag, 2002. 104

[20] David von Oheimb. Hoare logic for mutual recursion and local variables. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, volume 1738 of *Lect. Notes in Comp. Sci.*, pages 168–180. Springer-Verlag, 1999. 114

[21] Ernst-Rüdiger Olderog. On the notion of expressiveness and the rule of adaptation. *Theoretical Computer Science*, 24:337–347, 1983. 103, 108

[22] P. Pandya and M. Joseph. A structure-directed total correctness proof rule for recursive procedure calls. *The Computer Journal*, 29:531–537, 1986. 116

[23] Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1994. 103

[24] Thomas Schreiber. Auxiliary variables and recursive procedures. In *TAPSOFT'97: Theory and Practice of Software Development*, volume 1214 of *Lect. Notes in Comp. Sci.*, pages 697–711. Springer-Verlag, 1997. 103, 108

[25] Stefan Sokołowski. Total correctness for procedures. In *Mathematical Foundations of Computer Science (MFCS)*, volume 53 of *Lect. Notes in Comp. Sci.*, pages 475–483. Springer-Verlag, 1977. 103, 111, 115, 116

A Fixpoint Theory for Non-monotonic Parallelism

Yifeng Chen

Department of Mathematics and Computer Science, University of Leicester
University Road, Leicester LE1 7RH, UK

Abstract. This paper studies parallel recursions. The trace specification language used in this paper incorporates sequentiality, nondeterminism, reactiveness (including infinite traces), conjunctive parallelism and general recursion. The language is the minimum of its kind and thus provides a context in which we can study parallel recursions in general. In order to use Tarski's theorem to determine the fixpoints of recursions, we need to identify a well-founded partial order. A theorem of this paper shows that no appropriate order exists. Tarski's theorem alone is not enough to determine the fixpoints of parallel recursions. Instead of using Tarski's theorem directly, we reason about the fixpoints of terminating and nonterminating behaviours separately. Such reasoning is supported by the laws of a new composition called *partition*. We propose a fixpoint technique called the *partitioned fixpoint*, which is the least fixpoint of the nonterminating behaviours after the terminating behaviours reach their greatest fixpoint. The surprising result is that although a recursion may not be monotonic with regard to the lexical order, it must have the partitioned fixpoint, which equals the least lexical-order fixpoint. Since the partitioned fixpoint is well defined in any complete lattice, the results are applicable to various semantic models. Major existing fixpoint techniques simply become special cases of the partitioned fixpoint. For example, an Egli-Milner-monotonic recursion has its least Egli-Milner fixpoint, which can be shown to be the same as the partitioned fixpoint. The new technique is more general than the least Egli-Milner fixpoint in that the partitioned fixpoint can be determined even when a recursion is not Egli-Milner monotonic. Examples of non-monotonic recursions with fair-interleaving parallelism are studied. Their partitioned fixpoints are shown to be consistent with our intuitions.

1 Introduction

Recursions are notoriously tricky to model in denotational semantics. A general recursion is normally written as an equation: $X = f(X)$ in which X is called the *recursive argument*, and $f(X)$ called the *recursion*. For example $X = (x := x + 1 \mathbin{;} X)$ defines a recursion that increases variable x infinitely many times sequentially. If nondeterminism is allowed, the equation does not guarantee a unique fixpoint. Among all fixpoints, we must determine a fixpoint that is consistent with our understanding and at the same time convenient to

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 120–134, 2002.
© Springer-Verlag Berlin Heidelberg 2002

our semantic studies. The fixpoint of a recursion $f(X)$ is normally written $\phi X \cdot f(X)$ or ϕf for short. For example, a loop $\texttt{do}\ b \rightarrow P\ \texttt{od}$ is defined by $\phi X \cdot (\texttt{if}\ b\ \texttt{then}\ (P \fatsemi X)\ \texttt{else}\ \mathbb{I})$ where $\mathbb{I}$ (skip, no operation) is the unit of sequential composition. The simplest recursion $\phi X \cdot X$ corresponds to an empty loop $(\texttt{do}\ \texttt{true} \rightarrow \mathbb{I}\ \texttt{od})$ whose body is a skip.

Dijkstra's original Guarded-Command Language (GCL for short [9]) allows only finite nondeterminism. This restriction reflects computability, but it also limits the use of unboundedly nondeterministic specifications for *program refinement* [1]. Dijkstra dropped the restriction in his later work [11]. Recursions with unbounded nondeterminism are monotonic but may not be continuous. Tarski's fixpoint theorem [23] is a standard technique to determine the least fixpoint of a recursion that is monotonic with regard to a well-founded partial order (see Section 2). All recursions must be monotonic with regard to the order, and their least-fixpoint semantics must be consistent with our intuitions. Various partial orders with numerous variations have been proposed (e.g. [1, 3, 12, 14, 18, 22]). All of them work well in some circumstances but none of them is universally applicable.

Another restriction of GCL is that a recursion must be a guarded loop. Guardedness simplifies semantics by requiring the recursive argument to appear only in the second argument of any sequential composition. A guarded recursion is hence monotonic with regard to many partial orders. A semantic model without general recursions cannot incorporate procedure calls. Dijkstra studied general recursions in his later paper [10] based on the refinement order. Nelson [18], instead, used the Egli-Milner order with regard to which unguarded sequential recursions are also monotonic.

GCL uses a healthiness condition to exclude 'miracles'. A miracle is a non-executable specification that allows no behaviour from some initial states. Miracles are found useful for specification purposes including detection of pre-compilation errors and type conflicts [25]. Complete theories of program development have been developed based on semantics with miracles [1]. Our study on global synchrony [7] used miracles for the compositional reasoning of safety and liveness properties. The inclusion of miracles is also essential to the integrity of a semantic space. A semantic space containing miracles is normally a complete lattice (under the refinement order). Complete lattices are simple and rich, and enjoy better properties than domains in general [5, 15]. It is hence not surprising that most modern semantic models allow miracles (e.g. [12, 15, 18]). For example, Nelson dropped the restriction in his generalisation of Dijkstra's calculus [18].

GCL does not allow reactiveness. Reactive processes [2, 4, 7, 8, 14, 15, 19, 21, 22]. are very different from sequential programs. Park [19] observed that neither the least nor the greatest fixpoint alone is applicable to reactiveness. Dunne [12] used the Egli-Milner order [17, 18] and showed that the order is perhaps more appropriate for reactiveness than the refinement order.

A model allowing infinite reactive behaviours (e.g. [7, 8, 21, 22]) is much trickier than a model without them (e.g. [2, 14, 15, 22]). If we intend to rea-

son about safety and liveness properties, the modelling of such behaviours is inevitable.

Another challenge is involved with the loops whose bodies are skip (i.e. the unit of sequential composition) or any other command that does not generate intermediate states. If we intend to unify reactiveness and sequentiality, skip-like state transitions are inevitable. This problem is approached in different ways. For example, ACP [2] does not allow skip but uses a silent event to represent a similar but different concept. Timed CSP [8] allows zero-time transitions but does not guarantee a valid semantics for every recursion. We believe that it is essential to define valid semantics for any recursion, as was done in domain theory.

An infinite loop whose body takes some finite time greater than zero must take infinite time [8], but the infinite loop of a zero-time transition (e.g. skip) is less obvious: should it be zero, nondeterministically arbitrary or infinite? 'Zero time' (known as *zeno effect* or perhaps better termed *infinitesimal time*) is needed when the concrete amount of time taken by a program is irrelevant to us as far as it terminates. We argue that the sequential composition of finitely many zero-time transitions should take zero time, while the sequential composition of infinitely many such transitions takes infinite time.

The final challenge comes from parallelism. Even the simplest forms of parallelism complicate semantic studies tremendously. For example, conjunction as a parallel composition (e.g. in CSP [14] and Logs [7]) is not monotonic with regard to the Egli-Milner order. Thus the techniques developed in [12, 18] are not applicable in a language with conjunctive parallel composition.

In this paper we study a specification language consisting of five commands: specification of reactiveness, sequential composition, nondeterministic choice, parallel composition and recursion. The language allows unbounded nondeterminism, general recursions, reactiveness, infinite behaviours, empty loop body and parallelism. It is the *minimum* language of this kind. A similar language *Logic of Global Synchrony* (or Logs for short [7]) allowing multiple program variables has been successfully applied to the specifications of PRAM [13] and BSP [16]. Four contributions are made in this paper:

1. Tarski's fixpoint theorem is shown *not* to be directly applicable to our trace language due to the non-existence of an appropriate order;
2. the *partitioned fixpoint* is proposed to determine the fixpoints of the recursions in our language and shown to be the least lexical-order fixpoint, although the recursions may not be monotonic with regard to the order;
3. existing major fixpoint theories become special cases of the new technique, which is also applied to fair-interleaving parallelism.

Section 2 reviews relational semantics of sequential programming. A parallel specification language is introduced in Section 3. Section 4 introduces a technique called *partitioned fixpoint*. Section 5 first presents five partial orders and then shows that none of them makes Tarski's theorem applicable to the language. A theorem shows that no other partial order can be used. Section 6 introduces

a technique called *partitioned fixpoint*. In Section 7 the technique is used to determine the fixpoints of recursions in the language and shown to be more general than existing fixpoint techniques.

2 Sequential Semantics

A sequential specification allowing nondeterminism is a binary relation between the initial and final states (or equivalently, a function mapping each initial state to a set of final states). In this paper, we follow [15, 25] and always write relations as predicates on some dashed and undashed variables. For example, $x\,R\,x' \;\widehat{=}\; (x' = x+1)$ corresponds to a binary relation $\{(a,\, a{+}1) \mid a \in S\}$ that represents an assignment statement $x := x + 1$ where S is the state space. The undashed variable x denotes the observation on the initial state, while the dashed variable x' denotes the observation on the final state.

The sequential composition of two relations P and Q is simply their relational composition: $P \mathbin{;} Q \;\widehat{=}\; \exists x_0 \cdot (P[x_0/x'] \;\wedge\; Q[x_0/x])$. Nondeterministic choice becomes disjunction. For example, the relation $x \leqslant x'$ is a sequential specification with unbounded nondeterminism. A specification Q is considered 'better' or 'more refined' than another specification P, if Q is more deterministic. This is simply modelled by relational containment: $P \supseteq Q$.

However the above modelling is not concrete enough to distinguish nonterminating computations from terminating ones. In Z [25], nontermination is represented as a special state $\uparrow$. A sequential specification then becomes a relation in $\mathcal{P}(S_\uparrow \times S_\uparrow)$. For example, the assignment statement $x := x+1$ is represented as a relation $x\ R\ x' \;\widehat{=}\; (x = \uparrow \;\vee x' = x+1)$ under *total correctness* [20].

Hoare and He [15] later proposed a more elegant presentation using a pair of truth-valued special variables $\mathtt{ok}$ and $\mathtt{ok}'$, which denote the proper start and the successful termination respectively. For example, the assignment statement $x := x + 1$ is then represented as a predicate on four variables

$$(x, \mathtt{ok})\ \ R\ \ (x', \mathtt{ok}') \;\;\widehat{=}\;\; (\mathtt{ok} \;\Rightarrow\; x' = x + 1 \wedge \mathtt{ok}')\,.$$

It represents a computation that always terminates successfully (i.e. $\mathtt{ok}' = \mathtt{true}$) and increases the value of x if it has started properly (i.e. $ok = \mathtt{true}$). If it never starts properly, its behaviour becomes chaotic. Miracles are allowed as infeasible specifications, which cannot be implemented by executable programs.

Sequential specifications form a complete lattice [15] in which specifications are partially-ordered by the refinement order $\supseteq$, the bottom denoted by $\bot$ is $\mathtt{true}$, the top denoted by $\top$ is $\neg\mathtt{ok}$, the glb is set union $\cup$, and the lub is set intersection $\cap$. Thus a recursive function is a function that transforms each binary relation to another relation. Since all compositions are supposed to be monotonic, so are all recursions.

A monotonic function has a least fixpoint in a complete lattice [23]. All compositions are monotonic and thus any sequential recursion has its least $\supseteq$-fixpoint μf. A semantic space may form different complete lattices under different orders, each of which determines a unique least fixpoint. For example, if the refinement

order is reversed, we will obtain the least $\subseteq$-fixpoint, which equals the greatest $\supseteq$-fixpoint νf. We may further consider a fixpoint starting from an arbitrary element A. If $A \supseteq f(A)$, the monotonic function then has its least $\supseteq$-fixpoint $\mu_A f$ in the sub complete lattice whose top and bottom are $\top$ and A respectively. The case of $A \subseteq f(A)$ can be treated similarly.

The requirement of complete lattice can be relaxed to a *well-founded partial order* in which any non-empty subset has a glb. It can be shown that any monotonic function f has a least fixpoint in a well-founded order, provided that the function has *some* fixpoint [6].

3 A Simple Parallel Specification Language

In the previous section, we discussed two different ways of representing termination using $\uparrow$ and $\mathtt{ok}/\mathtt{ok}'$. In this section, we study a more expressive specification language that incorporates reactive behaviours. The same specification language [7] has been successfully applied to derive parallel algorithms of matrix multiplication, dynamic load balancing and dining-philosopher problem [7] for Parallel Random-Access Machine [13] and Bulk-Synchronous Parallelism [24].

A specification in this language denotes the observation on the initial state, final state and all intermediate states of a reactive process. In order to support reasoning about safety and liveness properties, we also allow infinite sequences. We will use two special trace variables tr and tr' to replace $\mathtt{ok}$ and $\mathtt{ok}'$. A computation starts properly if tr is a finite sequence. Similarly, a computation terminates successfully if tr' is a finite sequence.

Our language is thus a direct generalisation of Z-style sequential specification. In this paper we focus on specifications of traces, although the language can be generalised to any partially-ordered behaviour-based model such as real time, timed trace, branching time and their combinations, and the results obtained will still be valid.

Let S be the state space, and $S^{*\infty}$ the set of all sequences of states (including the infinite ones). For any two sequences $s, t \in S^{*\infty}$, $s \frown t$ denotes their concatenation. If s is infinite, then $s \frown t = s$. Two traces are ordered $s \leqslant t$ iff s is a prefix of t. The difference $t - s$ is a suffix of t such that $s \frown (t - s) = t$. $|s|$ denotes the length of the sequence. The length of the empty sequence $[\,]$ is 0. s_k denotes the k-th element of the sequence ($0 \leqslant k < |s|$).

Trace interleaving $s \mathbin{|||} t$ is the *set* of all fair interleavings of the two traces s and t. All elements of s and t must appear in every fair interleaving of them in alternating order. For example, the fair interleaving of $[0, 0, \cdots]$ and $[1, 1, \cdots]$ is the set of all traces containing ω-infinitely-many 0s and 1s.

A specification is a predicate on four variables x, x', tr and tr', which denote the initial state, the final state, the trace of intermediate states before the start and the trace of intermediate states before the end, respectively. We assume that a computation can only extend its trace history, i.e. $tr \leqslant tr'$. Anything sequentially following a nonterminating specification cannot be observed. Thus any specification's behaviour is arbitrary if tr is infinite. These restrictions first

imposed by Hoare and He are vital to the simplicity and validity of semantic models.

The most basic specification $\langle r \rangle$ of reactiveness is characterised by a relation $r = r(x, s, x')$ of the initial state x, the final state x' and a trace s of intermediate states. The trace s corresponds to the difference $tr' - tr$ of the traces tr and tr'. The specification $\langle r \rangle$ is defined by

$$\langle r \rangle \quad \widehat{=} \quad (|tr| = \infty \vee r[(tr' - tr)/s]) \wedge (tr \leqslant tr') .$$

This definition satisfies all the above restrictions. It represents a computation that only starts properly if the trace tr of intermediate states has not been infinite, and appends the trace s to the current trace tr and produces the trace tr' before the end. If tr is already infinite, then its tr' becomes an arbitrary extension of tr and the final state x' is chaotic.

Extreme specifications become special specifications of reactiveness: magic with no behaviours is $\top \widehat{=} \langle \texttt{false} \rangle$, chaos with all behaviours is $\bot \widehat{=} \langle \texttt{true} \rangle$, termination with all terminating behaviours is $\rhd \widehat{=} \langle |s| < \infty \rangle$ and nontermination with all nonterminating behaviours is $\lhd \widehat{=} \langle |s| = \infty \rangle$.

Sequential composition is still relational composition. Nondeterministic choice is disjunction. Recursion will be defined in section 7.

$\langle r \rangle$			specification of reactiveness
$P \fatsemi Q$	$\widehat{=}$	$\exists x_0 tr_0 \cdot P[x_0/x', tr_0/tr'] \wedge Q[x_0/x, tr_0/tr]$	sequential composition
$P \cup Q$	$\widehat{=}$	$P \vee Q$	nondeterministic choice
$P \cap Q$	$\widehat{=}$	$P \wedge Q$	parallel composition
ϕf			recursion

Conjunction is the simplest form of parallelism. It turns out to be very powerful for specification. In variable-sharing parallel programming, communication interference should be avoided. This becomes the same as avoiding miracles in refinement calculus. For example, to prove livelock freedom of an algorithm for the dining-philosopher problem, it is sufficient to conjoin the specifications of non-blocked philosophers and livelocked philosophers together and show that the algorithm renders the conjunction of livelocked philosophers to be a magic and leaves only the non-blocked philosophers [7]. In Section 7, we will also briefly discuss a more advanced form of *fair-interleaving parallelism*.

The specification of reactiveness is in fact the normal form of all specifications. The compositions $\langle p \rangle \cup \langle q \rangle$, $\langle p \rangle \cap \langle q \rangle$ and $\langle p \rangle \fatsemi \langle q \rangle$ can be reduced to $\langle p \vee q \rangle$, $\langle p \wedge q \rangle$ and

$$\langle \exists uyv \cdot p(x, u, y) \wedge ((s = u \wedge |u| = \infty) \vee (q(y, v, x') \wedge s = u \frown v \wedge |u| < \infty)) \rangle$$

respectively. The recursion defined in section 7 is also reducible to the normal form. All specifications of reactiveness form a complete lattice in which the order is $\supseteq$, glb is $\cup$, lub is $\cap$, top is $\top$, bottom is $\bot$. Any specification $\langle p \rangle$ has its

complement $\langle \neg p \rangle$. The complement of a specification P is denoted by $\sim P$. The difference $\langle p \rangle - \langle q \rangle$ between two relations is defined by $\langle p \wedge \neg q \rangle$.

Many useful specification commands can be derived from the basic ones.

$\texttt{idle}$	$\widehat{=}$	$\langle s=[x] \wedge x=x' \rangle$	one-step idle
$x :\in E$	$\widehat{=}$	$\langle x' \in E \wedge s=[\,] \rangle$	nondeterministic gassignment
$x := e$	$\widehat{=}$	$x :\in \{e\}$	deterministic assignment
$\mathbb{I}$	$\widehat{=}$	$x := x$	skip, no operation
$(b)_\top$	$\widehat{=}$	$x :\in \{x \mid b(x)\}$	if b then skip else magic
$\texttt{if}\ b\ \texttt{then}\ P\ \texttt{else}\ Q$	$\widehat{=}$	$((b)_\top \fatsemi P) \cup ((\neg b)_\top \fatsemi Q)$	if b then P else Q

4 Partitions

Our reasoning about recursions will rely on a derived composition called *partition*. A partition has a general form $(P\ {}_A|_B\ Q)$ where the two parameter specifications A and B called *partitioning elements* are complements of each other:

Definition 1 $P\ {}_A|_B\ Q \ \widehat{=}\ (P \wedge A) \vee (Q \wedge B)$ *where* $A \vee B = \bot$ *and* $A \wedge B = \top$.

In fact we need to write only one of the partitioning elements and use $P\ {}_A|\ Q$ to denote $P\ {}_A|_{\sim A}\ Q$, and $P\ |_A\ Q$ to denote $P\ {}_{\sim A}|_A\ Q$. In this paper, we use only the latter notation. If A and B are $\rhd$ and $\lhd$ respectively, we simply write a partition as $P|Q$, which combines the terminating behaviours of P and nonterminating behaviours of Q. For example, $P|\top$ and $\top|P$ extract the terminating and nonterminating behaviours from P respectively. Partitions satisfy some readily-proved laws whose flexible use can make reasoning simple and elegant. We list only those to be used in this paper.

Law 1

(1) $P\,|_A P \ = \ P$ (2) $P\,|_A Q \ = \ Q\,|_{\sim A} P$

(3) $P\,|_\bot Q \ = \ Q$ (4) $P\,|_\top Q \ = \ P$

(5) $(P\,|_A R)\,|_A Q \ = \ P\,|_A Q$ (6) $P\,|_A (R\,|_A Q) \ = \ P\,|_A Q$

(7) $P\,|_A \top \ = \ P \cap \sim A$ (8) $\top\,|_A P \ = \ P \cap A$

(9) $P\,|_A \bot \ = \ P \cup A$ (10) $\bot\,|_A P \ = \ P \cup \sim A$

(11) $P \fatsemi Q \ = \ (P|\top \fatsemi Q|\top) \mid ((\top|P \fatsemi Q) \cup (P \fatsemi \top|Q))$

With partitions, we can reason about terminating and nonterminating behaviours separately. For example, a sequential composition $P \fatsemi Q$ terminates if and only if both P and Q terminate. That means the terminating behaviours of the sequential composition are 'pure' in the sense that they are only related to the terminating behaviours of P and Q. On the other hand, $P \fatsemi Q$ never terminates if and only if either P or Q never terminates. The nonterminating behaviours of $P \fatsemi Q$ are 'mixed' with terminating and nonterminating behaviours of P (see Law 1(11)). The nonterminating behaviours will no longer be 'mixed', if the terminating part reaches a fixpoint and becomes 'constant'. This motivates us to first determine the greatest fixpoint of terminating part, and then determine the least fixpoint of nonterminating part (refer to Section 6).

5 Non-monotonicity of Parallelism

To determine fixpoints using Tarski's theorem, we need a well-founded partial order. Note that program refinement and fixpoint calculation are separate issues. They may use different partial orders.

The nonterminating behaviour of a sequential specification in Z is simple. It has only two possibilities: either 'empty' (i.e. termination $\emptyset$) or 'full' (i.e. nontermination $\{\uparrow\}$). Thus the nonterminating behaviours of any two sequential specifications are automatically ordered. This makes the required partial order easy to define. The nonterminating behaviour of a reactive process can be far more complicated. For example, the recursions $\phi X \cdot \langle s = [0] \rangle \mathbin{;} X$ and $\phi X \cdot \langle s = [1] \rangle \mathbin{;} X$ generate two different infinite traces. No meaningful partial order between them can be possibly defined.

We consider five partial orders for our specification language. The refinement order $\supseteq$ is relational containment, whose bottom is $\bot$. $\subseteq$ is the reverse refinement order, whose bottom is $\top$. Three other partial orders are defined:

Definition 2

$$\begin{array}{lcl} P \sqsubseteq_\pi Q & \widehat{=} & (P|\top \subseteq Q|\top) \wedge (\top|P \supseteq \top|Q \\ P \sqsubseteq_\varepsilon Q & \widehat{=} & P \sqsubseteq_\pi Q \wedge ((Q|\top - P|\top) \mathbin{;} \lhd \subseteq \top|P) \\ P \sqsubseteq_\lambda Q & \widehat{=} & P \sqsubseteq_\pi Q \vee (P|\top \subset \top|Q) \,. \end{array}$$

The order $\sqsubseteq_\varepsilon$ (with additional conjunct) is sharper than $\sqsubseteq_\pi$, which is sharper than $\sqsubseteq_\lambda$ (with additional disjunct). Nontermination $\lhd$ is the bottom shared by $\sqsubseteq_\pi$, $\sqsubseteq_\varepsilon$ and $\sqsubseteq_\lambda$. Termination $\rhd$ is the top of $\sqsubseteq_\pi$ and $\sqsubseteq_\lambda$.

The pairwise order $\sqsubseteq_\pi$ is a combination of the reverse refinement order for terminating behaviours and the refinement order for nonterminating behaviours. Unfortunately, the sequential composition is not monotonic with regard to the pairwise order. For example, the program

$$x := 0 \mathbin{;} \texttt{if}\ \ (x = 1)\ \ \texttt{then}\ \ \phi X \cdot (\texttt{idle} \mathbin{;} X)\ \ \texttt{else}\ \ \mathrm{II}\,. \tag{1}$$

always terminates and equals $x := 0$. However if we replace $x := 0$ with a $\sqsubseteq_\pi$-greater program $(x := 0 \cup x := 1)$, the new program may generate nonterminating behaviours and thus becomes incomparable with (1).

The Egli-Milner order $\sqsubseteq_\varepsilon$ [12, 17, 18] is a 'revised' pairwise order. Operator $-$ calculates the difference between two relations (like set minus, see Section 3). The additional conjunct of the Egli-Milner order is tricky. It is designed to force sequential composition to be monotonic. It requires a greater specification to contain limited additional terminating behaviours so that the potential nonterminating behaviours generated from these terminating behaviours are contained by any smaller specification. For example, *similar to the Egli-Milner order for sequential specifications* [17], two always-terminating specifications are $\sqsubseteq_\varepsilon$-comparable if and only if they are the same relation. This has solved the nonmonotonicity problem of sequential composition of (1) in that the programs $x := 0$ and $(x := 0 \cup x := 1)$ are simply not $\sqsubseteq_\varepsilon$-comparable. The definition (see [12]) adopted here allows reactiveness and is hence slightly more general than its original form for sequential semantics [20].

The lexical order $\sqsubseteq_\lambda$ is new. It is similar to but finer than the pairwise order in that the nonterminating part does not need to be ordered if the terminating part is strictly ordered. The lexical order is the main order that we will investigate in this paper.

Theorem 1 *The following table lists the monotonicity results of the compositions with regard to the five partial orders.* `yes` *and* `no` *indicate 'monotonic' and 'non-monotonic' respectively.*

Order	*Bottom*	$X \cup P$	$X \cap P$	$P \fatsemi X$	$X \fatsemi P$
$\supseteq$	$\bot$	yes	yes	yes	yes
$\subseteq$	$\top$	yes	yes	yes	yes
$\sqsubseteq_\pi$	$\lhd$	yes	yes	yes	no
$\sqsubseteq_\varepsilon$	$\lhd$	yes	no	yes	yes
$\sqsubseteq_\lambda$	$\lhd$	yes	no	no	no

The proof involves routine manipulation of definitions and is thus omitted.

To determine the fixpoints of recursions using Tarski's theorem, we need to choose an order that yields semantics consistent with our intuitions on the behaviours of recursions. Any calculation of Tarski's least fixpoint starts from the bottom of a well-founded partial order. Let $\sqsubseteq_*$ be the order that we are after. Note that $\sqsubseteq_*$ should be a *partial order*, if we want to *uniquely* pinpoint fixpoints using Tarski's theorem. Let $\bot_*$ denote the bottom of the order.

A semantics allowing unbounded nondeterminism and infinite behaviours must distinguish possible nontermination from necessary nontermination. In particular, the empty loop $\phi X \cdot X$ has two equivalent forms:

$$\phi X \cdot (\mathbb{I} \fatsemi X) \quad \text{or} \quad (\texttt{do}\ \texttt{true} \rightarrow \mathbb{I}\ \texttt{od}) \tag{2}$$

where $\mathbb{I}$ is the unit of sequential composition. The corresponding function $f(X) = X$ of the empty loop immediately reaches its least fixpoint $\bot_*$. The empty loop *never* terminates, and its semantics must not contain any terminating behaviour; otherwise, for example, if its semantics were chaos $\bot$, we would have an undesirable inequality:

$$(\phi X \cdot X) \fatsemi \langle\, s = [1] \,\rangle \quad \neq \quad (\phi X \cdot X)$$

in which tr and tr' can be equal on the right-hand side but cannot not be equal on the left-hand side. The inequality suggests that the behaviour of a nonterminating process could be altered if it is followed by another process that generates an intermediate state 1. Such counterintuitive interpretation is the result of the incorrect semantic assumption on the empty loop. Thus we conclude that $\bot_* \subseteq \lhd$.

On the other hand, the empty loop is an executable program that at least generates some outputs. Thus its semantics must not be miraculous, i.e. $\top \subset \bot_*$.

In summary, the required order $\sqsubseteq_*$ must satisfy:

(A) $\sqsubseteq_{*}$ is a well-founded partial order,
(B) $\top \subset \bot_{*} \subseteq \lhd$ where $\bot_{*}$ is the bottom of the order,
(C) all compositions of our language are $\sqsubseteq_{*}$-monotonic.

None of the five orders that we have considered satisfies all three criteria.

	$\supseteq$	$\subseteq$	$\sqsubseteq_{\varepsilon}$	$\sqsubseteq_{\pi}$	$\sqsubseteq_{\lambda}$
(A)	✓	✓	✓	✓	✓
(B)			✓	✓	✓
(C)	✓	✓			

A natural question is: *Does there exist any other order to make all compositions monotonic?* Unfortunately the answer is no, as reported next.

Indeed the following theorem has ruled out the existence of any such order.

Theorem 2 (Non-monotonicity of parallelism) *No order satisfying* (A), (B) *and* (C) *above exists.*

Proof. Suppose that $\sqsubseteq_{*}$ is an order satisfying (A), (B) and (C). Let P, Q and R be three specifications in our language. We construct two recursions:

$$\begin{array}{rcl} f(X) & \widehat{=} & P \mid ((X \cap Q) \cup ((X|\top \mathbin{;} \lhd) \cap R \cap \bot_{*})) \\ g(X) & \widehat{=} & P \mid ((X \cap R) \cup ((X|\top \mathbin{;} \lhd) \cap Q \cap \bot_{*})) \,. \end{array}$$

Both recursions must be $\sqsubseteq_{*}$-monotonic according to (C). Since $\bot_{*}$ is the bottom of the order $\sqsubseteq_{*}$, we thus have $\bot_{*} \sqsubseteq_{*} \rhd$. This leads to

$$\begin{array}{rclcrcl} P|(Q \cap \bot_{*}) & = & f(\bot_{*}) & \sqsubseteq_{*} & f(\rhd) & = & P|(R \cap \bot_{*}) \\ P|(R \cap \bot_{*}) & = & g(\bot_{*}) & \sqsubseteq_{*} & g(\rhd) & = & P|(Q \cap \bot_{*}) \,. \end{array}$$

The order $\sqsubseteq_{*}$ is a *partial* order according to (A). Thus $P|(Q \cap \bot_{*}) = P|(R \cap \bot_{*})$ must hold for arbitrary specifications P, Q and R. Let $P = Q = \top$ and $R = \bot_{*}$. We then have $\top = \bot_{*}$, which contradicts (B). Thus the order that satisfies all three criteria (A), (B) and (C) does not exist. □

Tarski's fixpoint theorem alone is not applicable to the language. This, however, does not exclude the existence of a least fixpoint with regard to some partial order. With additional information, we may still be able to determine such a fixpoint. We now take advantage of just that.

6 Fixpoints of Non-monotonic Functions

In this section we will introduce a more general fixpoint technique for parallel recursions. We shall use a parameterised lexical order $\sqsubseteq_{\lambda(A)}$ of which the original lexical order $\sqsubseteq_{\lambda}$ is a special case when $A = \lhd$.

Definition 3
$P \sqsubseteq_{\lambda(A)} Q \;\widehat{=}\; (P\,|_A \top \subseteq Q\,|_A \top) \vee (P\,|_A \top = Q\,|_A \top \wedge \top\,|_A P \supseteq \top\,|_A Q)$

The specifications of our language form a complete lattice under the refinement order $\supseteq$. Any $\supseteq$-monotonic function f has its least $\supseteq$-fixpoint μf and greatest $\supseteq$-fixpoint νf. The *partitioned fixpoint* $\psi_A f$ is the least $\supseteq$-fixpoint of the right-hand part after the left-hand part reaches its greatest $\supseteq$-fixpoint.

Definition 4 (Partitioned fixpoint) $\psi_A f \;\widehat{=}\; \mu X \cdot f(\nu f \mid_A X)$

Some previous fixpoints become special cases of the partitioned fixpoint. For example, the least $\supseteq$-fixpoint becomes a special case when $A = \bot$:

$$\psi_\bot f \;=\; \mu X \cdot f(\nu f \mid_\bot X) \;=\; \mu X \cdot f(X) \;=\; \mu f\,. \tag{3}$$

When $A = \top$:

$$\psi_\top f \;=\; \mu X \cdot f(\nu f \mid_\top X) \;=\; \mu X \cdot f(\nu f) \;=\; f(\nu f) \;=\; \nu f\,. \tag{4}$$

And when $A \supseteq f(A)$ we have the following readily-proved theorem:

Theorem 3 *If f is $\supseteq$-monotonic and $A \supseteq f(A)$, then $\psi_A f = \mu_A f$.*

However the partitioned fixpoint $\psi_A f$ is more general and is, surprisingly, well-defined in some cases that A and $f(A)$ are not $\supseteq$-comparable or f is not even $\sqsubseteq_{\lambda(A)}$-monotonic. The following theorem on the partitioned fixpoints will be the key to our modelling of nontermination.

Theorem 4 (Partitioned fixpoint) *If a $\supseteq$-monotonic function f satisfies $f(X \mid_A \top) = f(X \mid_A \top) \mid_A \top$ for any specification X, then $\psi_A f$ is the least $\sqsubseteq_{\lambda(A)}$-fixpoint of function f.*

Proof. We first notice that $\psi_A f \mid_A \top$ reaches its greatest $\supseteq$-fixpoint $\nu f \mid_A \top$ directly. According to the definition, $\psi_A f$ is the least $\supseteq$-fixpoint of the function $\lambda X \cdot f(\nu f \mid_A X)$. Thus we have:

$$\begin{array}{ll}
\psi_A f \mid_A \top & \\
= & \text{definition of } \psi_A f \\
f(\nu f \mid_A \psi_A f) \mid_A \top & \\
= & \text{distributivity of } (\cdot \mid_A \top) \\
f((\nu f \mid_A \psi_A f) \mid_A \top) \mid_A \top & \\
= & \text{mid-part elimination Law 1(5)} \\
f(\nu f \mid_A \top) \mid_A \top & \\
= & \text{distributivity of } (\cdot \mid_A \top) \\
f(\nu f) \mid_A \top & \\
= & \text{definition of fixpoint} \\
\nu f \mid_A \top\,. &
\end{array}$$

Thus $\psi_A f$ is a fixpoint of f:

$$
\begin{array}{ll}
\psi_A f & \\
= & \psi_A f \text{ is the least } \supseteq\text{-fixpoint of } \lambda X \cdot f(\nu f \mid_A X) \\
f(\nu f \mid_A \psi_A f) & \\
= & \text{inverse of mid-part elimination Law 1(5)} \\
f((\nu f \mid_A \top) \mid_A \psi_A f) & \\
= & \text{a proved fact} \\
f((\psi_A f \mid_A \top) \mid_A \psi_A f) & \\
= & \text{mid-part elimination Law 1(5)} \\
f(\psi_A f). &
\end{array}
$$

We now need to show that $\psi_A f$ is the least $\sqsubseteq_{\lambda(A)}$-fixpoint. Let L be a fixpoint of f i.e. $L = f(L)$. νf is the greatest $\supseteq$-fixpoint and hence $\nu f \subseteq L$. Thus $(\nu f \mid_A \top) \subseteq (L \mid_A \top)$ due to the monotonicity of $(\cdot \mid_A \top)$.

1. If $(\psi_A f \mid_A \top) \subset (L \mid_A \top)$, then we immediately have $\psi_A f \sqsubseteq_{\lambda(A)} L$.
2. If $\nu f \mid_A \top = \psi_A f \mid_A \top = L \mid_A \top$, then

$$
\begin{aligned}
L &= f(L) \\
&= f((L \mid_A \top) \mid_A L) \\
&= f((\nu f \mid_A \top) \mid_A L) \\
&= f(\nu f \mid_A L)
\end{aligned}
$$

or L is a fixpoint of $\lambda X \cdot f(\nu f \mid_A X)$ and hence $\psi_A f \supseteq L$ and $(\top \mid_A \psi_A f) \supseteq (\top \mid_A L)$. Thus $\psi_A f \sqsubseteq_{\lambda(A)} L$.

Thus $\psi_A f$ is indeed the least $\sqsubseteq_{\lambda(A)}$-fixpoint. □

7 Applications of Partitioned Fixpoint

The technique of partitioned fixpoint of complete lattices can be applied to our specification language. Let the fixpoint of any recursion $f(X)$ be the partitioned fixpoint: $\phi f \mathrel{\widehat{=}} \psi_{\lhd} f$.

For example, the recursion $f(X) \mathrel{\widehat{=}} ((X \fatsemi \rhd) \cap \rhd) \cup \mathbb{I}$ reaches its least Egli-Milner fixpoint $\rhd$ in three steps: $f(\lhd) = \mathbb{I}$, $f(\mathbb{I}) = \rhd$ and $f(\rhd) = \rhd$. However the fixpoint cannot be determined using Tarski's theorem based on the Egli-Milner order because $\mathbb{I} \not\sqsubseteq_{\varepsilon} \rhd$. Fortunately, the function $\lambda X \cdot (X|\top)$ distributes all recursions in the language. Theorem 4 is hence applicable and leads to the right fixpoint $\rhd$.

Proposition 5 *Let $(X, \sqsubseteq_1)$ and $(X, \sqsubseteq_2)$ be two partial orders, and L_1 and L_2 be the least fixpoints of a function (on X) with regard to $\sqsubseteq_1$ and $\sqsubseteq_2$ respectively. If $\sqsubseteq_1 \subseteq \sqsubseteq_2$ then $L_1 = L_2$.*

Let $f(X)$ be a recursion in the language. Thus f is $\supseteq$-monotonic (refer to Theorem 1) and satisfies $f(X|\top) = f(X|\top) \mid \top$. Previous fixpoints now become special cased of the partitioned fixpoint:

1. our calculation in (3) guarantees that the least $\supseteq$-fixpoint μf *equals* the partitioned fixpoint $\psi_\perp f$ with partitioning element $\perp$;
2. similarly, our calculation in (4) guarantees that the greatest $\supseteq$-fixpoint νf *equals* the partitioned fixpoint $\psi_\top f$ with partitioning element $\top$;
3. Theorem 4 states that the least $\sqsubseteq_\lambda$-fixpoint always exists and *equals* the partitioned fixpoint $\psi_\lhd f$ with partitioning element $\lhd$;
4. the pairwise order is sharper than the lexical order $\sqsubseteq_\pi \subseteq \sqsubseteq_\lambda$; thus according to Proposition 5, if there exists the least $\sqsubseteq_\pi$-fixpoint then it must *equal* the partitioned fixpoint $\psi_\lhd f$;
5. similarly, $\sqsubseteq_\varepsilon \subseteq \sqsubseteq_\lambda$; thus if there exists a least $\sqsubseteq_\varepsilon$-fixpoint then it must *equal* $\psi_\lhd f$.

We may also consider a more realistic form of parallelism that combines conjunctive parallelism for the initial and final states and fair-interleaving parallelism for intermediate states.

Definition 5 $\langle\, p(x,s,x') \,\rangle \;|||\; \langle\, q(x,s,x') \,\rangle$
$$\widehat{=}\quad \langle\, \exists uv \cdot p(x,u,x') \wedge q(x,v,x') \wedge s \in (u \mathbin{|||} v) \,\rangle$$

From a common initial state, two specifications in the above composition must agree on the same final state, but their intermediate states are fairly interleaved.

The composition is *commutative* and *associative* and *distributes nondeterministic choice*. It terminates if both specifications in the composition terminate and thus guarantees the distributivity condition of Theorem 4.

Law 2 $P \,|||\, Q \;=\; (P|\top \,|||\, Q|\top) \mid (\top|P \,|||\, Q \;\cup\; P \,|||\, \top|Q)$.

However $|||$ is not monotonic with regard to $\sqsubseteq_\pi$, $\sqsubseteq_\varepsilon$ or $\sqsubseteq_\lambda$. Let Z_1 be the nonterminating specification $\langle\, s = [0,1,2,\cdots] \,\rangle$, which has a tricky property: $(\lhd \;|||\; Z_1) \cap Z_1 = \top$. Let Z_0 be the terminating specification $\langle\, s = [\,] \,\rangle$ and $Z_n \;\widehat{=}\; Z_{n-1} \;|||\; Z_1$ for any $n \geqslant 0$. The recursion $f(X) \;\widehat{=}\; (X \;|||\; Z_1) \cup Z_0$ is a counterexample, which can be approximated as follows:

$$\begin{array}{rcl}
f^0(\lhd) & = & \lhd \\
f^1(\lhd) & = & \lhd \;|||\; Z_1 \cup Z_0 \\
f^2(\lhd) & = & \lhd \;|||\; Z_2 \cup Z_1 \cup Z_0 \\
\cdots & = & \cdots\cdots \\
f^n(\lhd) & = & \lhd \;|||\; Z_n \cup \bigcup_{k<n} Z_k \\
\cdots & = & \cdots\cdots
\end{array}$$

Nontermination $\lhd$ is the bottom of all three orders. However $f(\lhd) \not\sqsubseteq_\pi f^2(\lhd)$, $f(\lhd) \not\sqsubseteq_\varepsilon f^2(\lhd)$ and $f(\lhd) \not\sqsubseteq_\lambda f^2(\lhd)$ in that the nonterminating behaviours of $f^2(\lhd)$ (including Z_1) are not contained by $f^1(\lhd)$.

Fortunately the partitioned fixpoint of the example can be calculated:

$$\begin{array}{ll}
\phi f & \\
= & \qquad\qquad\qquad\qquad \text{definition} \\
\mu X \cdot f(\nu f \mid X) &
\end{array}$$

$$
\begin{array}{ll}
= & \nu f \mid \top = Z_0 \\
\mu X \cdot f(Z_0 \mid X) & \\
= & \text{definition of } f \\
\mu X \cdot ((Z_0 \mid X) \mathbin{|||} Z_1) \cup Z_0 & \\
= & \text{Law 2} \\
\mu X \cdot ((\top \mid X) \mathbin{|||} Z_1) \cup Z_1 \cup Z_0 & \\
= & f \text{ is continuous } \mu f = \bigcap_{n>1} f^n(\bot) \text{ and } \top \mid \bot = \lhd \\
\bigcap_{n>1} \left(\lhd \mathbin{|||} Z_n \cup \bigcup_{k \leqslant n} Z_k \right) &
\end{array}
$$

The final result is also the least lexical-order fixpoint according to Theorem 4.

8 Conclusions

In this paper we have studied the modelling of recursions in the style of relational semantics. Most results obtained are also applicable to other formalism such as predicate-transformer semantics and axiomatic semantics.

The partitioned fixpoint requires a precondition that terminating behaviours of any composition must not depend on the nonterminating behaviours of its arguments. This requirement is weak; all program constructions that we know (including negation and implication) satisfy it. The additional information provided by the distributivity precondition is vital for determining the least lexical-order fixpoint. Without that information, Tarski's theorem alone is not enough to tackle non-monotonic recursions. Our semantics can be made more concrete by adding a pair of 'fresh' variables to denote divergent points. Arbitrary non-termination containing all infinite behaviours can then be distinguished from any intermediate failure (e.g. the empty loop): the former never diverges, while the latter diverges after some point. The specification language provides a context in which we can study recursions in general. The obtained results are also applicable to other models of resource cumulation.

This paper partly arose from a DPhil thesis. The author is grateful to his supervisor J.W. Sanders for various discussions, comments, suggestions and review of the draft of this paper, and to Roland Backhouse and Steve Dunne for pointing out errors in early versions of this paper. The author also gratefully acknowledge the insightful discussions with Ian Hayes, Jifeng He and Gavin Lowe and wide-ranging comments of anonymous referees.

References

[1] R. J. R. Back and K. Sere. Stepwise refinement of action systems. In *Mathematics of Program Construction*, volume 375 of *LNCS*, pages 115–138. Springer-Verlag, 1989. 121

[2] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985. 121, 122

[3] M. M. Bonsangue and J. N. Kok. The weakest precondition calculus: Recursion and duality. *Formal Aspects of Computing*, 6(A):788–800, 1994. 121

[4] S. Brookes. Full abstraction for a shared-variable parallel language. *Information and Computation*, 127(2):145–163, 1996. 121

[5] Y. Chen. How to write a healthiness condition. In *2nd International Conference on Integrated Formal Methods*, volume 1945 of *LNCS*, pages 299–317. Springer-Verlag, 2000. 121

[6] Y. Chen. A fixpoint theorey for non-monotonic parallelism. Technical Report 38, Department of Maths & Computer Science, University of Leicester, 2001. 124

[7] Y. Chen and J. W. Sanders. Logic of global synchrony. In *12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 487–501. Springer-Verlag, 2001. 121, 122, 124, 125

[8] J. Davies and S. Schneider. A brief history of timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995. 121, 122

[9] E. W. Dijkstra. Guarded commands, nondeterminacy and the formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975. 121

[10] E. W. Dijkstra and C. S. Scholten. Semantics of recursive procedures. EWD 859, 1983. 121

[11] E. W. Dijkstra and A. J. M. von Gasteren. A simple fixed-point argument without the restriction to continuity. *Acta Informatica*, 23(1):1–7, 1986. 121

[12] S. E. Dunne. Recasting Hoare and He's relational theory of programs in the context of general correctness. Technical report, School of Computing and Mathematics University of Teesside, 2000. 121, 122, 127

[13] S. Fortune and J. Wyllie. Parallelism in random access machines. In *10th Annual ACM Symposium on Theory of Computing*, pages 114–118, 1978. 122, 124

[14] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. 121, 122

[15] C. A. R. Hoare and J. He. *Unifying Theories of Programming*. Prentice Hall, 1998. 121, 123

[16] W. F. McColl. Scalability, portability and predictability: The BSP approach to parallel programming. *Future Generation Computer Systems*, 12:265–272, 1996. 122

[17] C. C. Morgan and A. McIver. Unifying wp and wlp. *Information Processing Letters*, 59:159–163, 1996. 121, 127

[18] G. Nelson. A generalisation of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, 1989. 121, 122, 127

[19] D. M. R. Park. On the semantics of fair parallelism. In *Abstract Software Specification*, volume 86 of *LNCS*, pages 504–526. Springer-Verlag, 1980. 121

[20] G. D. Plotkin. Lecture notes on domain theory. The Pisa Notes, 1983. 123, 127

[21] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981. 121

[22] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998. 121

[23] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955. 121, 123

[24] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990. 124

[25] J. Woodcock and J. Davis. *Using Z : specification, refinement, and proof*. Prentice Hall, 1991. 121, 123

Greibach Normal Form in Algebraically Complete Semirings

Zoltán Ésik*[1] and Hans Leiß**[2]

[1] Dept. of Computer Science
University of Szeged, Szeged, Hungary
esik@inf.u-szeged.hu
[2] Centrum für Informations- und Sprachverarbeitung
University of Munich, Munich, Germany
leiss@cis.uni-muenchen.de

Abstract. We give inequational and equational axioms for semirings with a fixed-point operator and formally develop a fragment of the theory of context-free languages. In particular, we show that Greibach's normal form theorem depends only on a few equational properties of least pre-fixed-points in semirings, and elimination of chain- and deletion rules depend on their inequational properties (and the idempotency of addition). It follows that these normal form theorems also hold in non-continuous semirings having enough fixed-points.

Keywords: Greibach normal form, context-free languages, pre-fixed-point induction, equational theory, Conway algebra, Kleene algebra, algebraically complete semirings

1 Introduction

It is well-known that the equational theory of context-free languages, i.e. the equivalence problem for context-free grammars, is not recursively enumerable. This may have been the reason why little work has been done to develop a formal theory for the rudiments of the theory of context-free languages.

In contrast, the equational theory of regular languages is decidable, and several axiomatizations of it appeared, using regular expressions as a notation system. In the 1970s, axiomatizations by schemata of equations between regular expressions were conjectured by Conway[8]. Salomaa[23] gave a finite first-order axiomatization based on a version of the unique fixed-point rule. Redko[21] showed that the theory does not have a finite equational basis. Twenty years later, Pratt[20] showed that a finite equational axiomatization is possible if one extends the regular operations $+, \cdot$ and * by the left- and right *residuals* $/$ and $\backslash$ of $\cdot$. The

* This author was supported by BRICS (Aalborg) and the National Foundation of Hungary for Scientific Research, grant T35169.

** This author was supported by a travel grant from the Humboldt-Foundation.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 135–150, 2002.
© Springer-Verlag Berlin Heidelberg 2002

important new axiom was $(a/a)^* = (a/a)$, the axiom of 'pure induction'. (For a recent extension of Pratt's methods, see Santocanale[24].) Earlier, Krob[16] confirmed several conjectures of Conway including the completeness of Conway's group identities. He also gave several finite axiomatizations, including a system having, in addition to a finite number of equational axioms, a Horn formula expressing that a^*b is the *least solution of* $ax + b \leq x$. See also Boffa[7], Bloom and Ésik[6], Bernátsky e.a.[4]. Independently, Kozen[14] defined a *Kleene algebra* as an idempotent semiring equipped with a * operation subject to the above Horn-formula and its dual asserting that b^*a is the least solution of $xa + b \leq x$. He gave a direct proof of the completeness of the Kleene algebra axioms with respect to the equational theory of the regular sets.

With a least-fixed-point operator μ, these axioms of KA can be expressed as $a^*b = \mu x(ax + b)$ and $ba^* = \mu x(xa + b)$. Hence it is natural to extend the regular expressions by a construction $\mu x.r$, which gives a notation system for context-free languages. Extensions of KA by μ have been suggested in [17] to axiomatize fragments of the theory of context-free languages.

In this paper we look at axioms for semirings with a least-fixed-point operator that are sufficient to prove some of the normal form theorems for context-free grammars. In particular, we derive the Greibach[11] normal form theorem using only equational properties of least fixed-points. Our proof gives the efficient algorithm of Rosenkrantz[22], but avoids the analytic method of power series of his proof. Our axioms also imply that context-free grammars have normal forms without chain rules or deletion rules. An important aspect is that we do not use the idempotency of $+$, except for the elimination of deletion rules, and so the classical theorems are extended to a wide class of semirings.

Recently, Parikh's theorem, another classical result on context-free languages, has been treated in a similar spirit. Hopkins and Kozen[13] generalized this theorem to an equation schema valid in all commutative idempotent semirings with enough solutions for recursion equations, also replacing analytic methods by properties of least fixed-points. A purely equational proof is given in [1].

2 Park μ-Semirings and Conway μ-Semirings

We will consider *terms*, or *μ-terms* defined by the following syntax, where x ranges over a fixed countable set X of variables:

$$T ::= x \mid 0 \mid 1 \mid (T + T) \mid (T \cdot T) \mid \mu x\, T$$

For example $\mu x(x + 1)$ is a term. To improve readability, we write $\mu x.t$ instead of $\mu x\, t$ when the term t is 0, 1, a variable or not concretely given. The variable x is *bound* in $\mu x.t$. We identify any two terms that only differ in the names of the bound variables. The set *free*(t) of *free variables* of a term t is defined as usual. A term is *closed* if it has no free variables and *finite* if it has no subterm of the form $\mu x.t$. We will write $t(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_n)$, to indicate that the free variables of t belong to $\{x_1, \ldots, x_n\}$. *Simultaneous substitution* $t[\vec{t}/\vec{x}]$ of

$\vec{t} = (t_1, \ldots, t_n)$ for $\vec{x}$ is defined as usual. By $\mu x.t[s/y]$ we mean $\mu x(t[s/y])$, not $(\mu x.t)[s/y]$.
We are interested in interpretations where $\mu x.t$ provides a solution to $x = t$.

Definition 1. A *μ-semiring* is a semiring $(A, +, \cdot, 0, 1)$ with an interpretation $(\cdot)_A$ of the terms t as functions $t_A : A^X \to A$, such that

1. for each environment $\rho \in A^X$, all variables $x \in X$ and all terms t, t':
   (a) $0_A(\rho) = 0$, $1_A(\rho) = 1$, $x_A(\rho) = \rho(x)$, $(t + t')_A(\rho) = t_A(\rho) + t'_A(\rho)$, $(t \cdot t')_A(\rho) = t_A(\rho) \cdot t'_A(\rho)$,
   (b) the 'substitution lemma' holds, i.e. $(t[t'/x])_A(\rho) = t_A(\rho[x \mapsto t'_A(\rho)])$,
2. for all terms t, t' and $x \in X$, if $t_A = t'_A$, then $(\mu x.t)_A = (\mu x.t')_A$.

A *weak ordered μ-semiring* is a μ-semiring A equipped with a partial order $\leq$ such that all term functions t_A are monotone with respect to the pointwise order. An *ordered μ-semiring* is a weak ordered μ-semiring A such that for any terms t, t' and $x \in X$, if $t_A \leq t'_A$ in the pointwise order, then $(\mu x.t)_A \leq (\mu x.t')_A$.

In a μ-semiring A, the value $t_A(\rho)$ does not depend on $\rho(x)$ if x does not have a free occurrence in t. As usual, $\rho[x \mapsto a]$ is the same as ρ except that it maps x to a. A term equation $t = t'$ *holds* or *is satisfied* in a μ-semiring A, if $t_A = t'_A$. A term inequation $t \leq t'$ *holds* in a μ-semiring A equipped with a partial order $\leq$, if $t_A \leq t'_A$ in the pointwise order on A^X. An implication $t = t' \to s = s'$ holds in A, if for all $\rho \in A^X$, whenever $t_A(\rho) = t'_A(\rho)$, then also $s_A(\rho) = s'_A(\rho)$. Likewise for implications with inequations.

Definition 2. A *strong μ-semiring* is a μ-semiring where $\forall x(t = t') \to \mu x.t = \mu x.t'$ holds, for all terms t, t' and $x \in X$. A *strong ordered μ-semiring* is a weak ordered μ-semiring where $\forall x(t \leq t') \to \mu x.t \leq \mu x.t'$ holds, for all terms t, t' and variables $x \in X$.

The validity of $\forall x(t = t') \to \mu x.t = \mu x.t'$ implies condition 2. in Definition 1.

Definition 3. A *Park μ-semiring* is a weak ordered μ-semiring satisfying the *fixed-point inequation (1)* and the pre-fixed-point *induction axiom (2)*, also referred to as the *Park induction rule*, for all terms t and $x, y \in X$:

$$t[\mu x.t/x] \leq \mu x.t, \tag{1}$$

$$t[y/x] \leq y \to \mu x.t \leq y. \tag{2}$$

Proposition 1. *Any Park μ-semiring A is a strong ordered μ-semiring satisfying the* composition identity (3) *and the* diagonal identity (4)*, for all terms t, t' and all variables x, y:*

$$\mu x.t[t'/x] = t[\mu x.t'[t/x]/x] \tag{3}$$

$$\mu x.\mu y.t = \mu x.t[x/y]. \tag{4}$$

Note that taking t' to be x in (3) gives the *fixed point equation* for t,

$$\mu x.t = t[\mu x.t/x]. \tag{5}$$

Proof. To prove that A is a strong ordered μ-semiring, suppose for terms t, t' and $\rho \in A^X$ that $t_A(\rho[x \mapsto a]) \leq t'_A(\rho[x \mapsto a])$, for all $a \in A$. Since t_A is monotone, it follows that every pre-fixed-point of the map $a \mapsto t'_A(\rho[x \mapsto a])$ is a pre-fixed-point of the map $a \mapsto t_A(\rho[x \mapsto a])$. Hence, $(\mu x.t)_A(\rho) \leq (\mu x.t')_A(\rho)$. Equations (3) and (4) are established in Niwinski [19]. □

Definition 4. A *Conway μ-semiring* is a μ-semiring satisfying the *Conway identities* (3) and (4), for all terms t, t' and variables x, y.

3 Algebraically Complete Semirings

An *ordered semiring* is a semiring $(S, +, \cdot, 0, 1)$ equipped with a partial order $\leq$ such that the $+$ and $\cdot$ operations are monotone in both arguments. Note that if $+$ is idempotent and 0 is the least element, then $\leq$ is the *semilattice order*, i.e. $x \leq y$ iff $x + y = y$. Clearly, each weak ordered μ-semiring is an ordered semiring.

With $z \notin \mathit{free}(t)$, the *left iteration* t^ℓ and the *right iteration* t^r of a term t are

$$t^\ell := \mu z(zt + 1) \qquad \text{and} \qquad t^r := \mu z(tz + 1).$$

Definition 5. An *algebraically complete semiring* is an ordered semiring which is a Park μ-semiring and satisfies the inequations

$$x^r y \leq \mu z(xz + y) \tag{6}$$
$$yx^\ell \leq \mu z(zx + y). \tag{7}$$

By Proposition 1, every algebraically complete semiring satisfies the composition (3) and diagonal identities (4), hence also the fixed-point identity (5).

Proposition 2. *Any algebraically complete semiring S satisfies the (in)equations*

$$0 \leq x \tag{8}$$
$$x^r y = \mu z(xz + y) \tag{9}$$
$$yx^\ell = \mu z(zx + y) \tag{10}$$
$$x^r = x^\ell. \tag{11}$$

Proof. (8) follows from (6) and (2): we have $0 = 1^r \cdot 0 \leq \mu x.x \leq x$ in S. The parts of (9) and (10) beyond (6) and (7) follow from the fixed-point inequation, monotonicity and the induction rule. As for (11), we have

$$\begin{aligned}
(xy)^r &= \mu z((xy)z + 1) \\
&= x \cdot \mu z(y(xz + 1)) + 1 && \text{by (3) for } t := xz + 1 \text{ and } t' := yz, \\
&= x \cdot \mu z((yx)z + y) + 1 \\
&= x \cdot (yx)^r y + 1, && \text{by (9).}
\end{aligned}$$

With $x = 1$, we get $y^r = y^r y + 1$, which by the Park induction rule gives $y^\ell = \mu z(zy+1) \leq y^r$. Similarly, using (10) we get $y^r \leq y^\ell$, so that $y^\ell = y^r$. □

By (11), two possible definitions of iteration, left and right iteration, coincide in any algebraically complete semiring. With $z \notin \mathit{free}(t)$, we *define*

$$t^* := \mu z(tz + 1). \tag{12}$$

On algebraically complete semirings A, we obtain a *-operation with $a^* = a^r = a^\ell$ for all a.

We call these semirings *complete* since they contain the least-pre-fixed-point of each definable function, and *algebraic* since the context-free or 'algebraic' languages, which subsume the regular ones via (12), are the prime example.

Example 1. A *continuous semiring* is a semiring $S = (S, +, \cdot, 0, 1)$ with a complete partial order $\leq$ such that 0 is its least element and $+$ and $\cdot$ are continuous, i.e., they preserve in each argument the sup of any directed nonempty set. Any continuous semiring S gives rise to an algebraically complete semiring where $(\mu x.t)_S$ is the least solution of the fixed-point equation $x = t$ (see [6]).

Let $\mathbb{N}$ denote the set of nonnegative integers and let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. Equipped with the usual order and $+$ and $\cdot$ operations, $\mathbb{N}_\infty$ is a continuous semiring. Also, every finite ordered semiring having 0 as least element is continuous. Thus, $\mathbb{N}_\infty$ and the boolean semiring $\mathbb{B} = \{0, 1\}$ are algebraically complete semirings.

Other prime examples of continuous semirings are the semiring $\mathbf{L}_A$ of all languages in A^*, where A is a set, $+$ is set union, $\cdot$ is concatenation and $\leq$ is set inclusion, and the semiring $\mathbb{N}_\infty\langle\langle A^*\rangle\rangle$ of power series over A with coefficients in $\mathbb{N}_\infty$, equipped with the pointwise order.

The set $\mathbf{R}_M$ of all binary relations on the set M, where $+$ is union, $\cdot$ the relation product, 0 the empty relation, 1 the diagonal on M and $\leq$ is inclusion, is a continuous semiring. In this example, r^* is the reflexive transitive closure of r.

Example 2. The context-free languages in $\mathbf{L}_A$ form an algebraically complete semiring as do the algebraic power series in $\mathbb{N}_\infty\langle\langle A^*\rangle\rangle$. Unless A is empty, neither of these semirings is continuous. Given a set A of binary relations over the set M, let $\mathbf{R}_M(A)$ be the values in $\mathbf{R}_M$ of all μ-terms with parameters from A. Then $\mathbf{R}_M(A)$ is an algebraically complete semiring, which is generally not continuous. These semirings are non-continuous since the partial order is not complete.

Example 3. There exist algebraically complete idempotent semirings that cannot be embedded in a continuous (idempotent) semiring. We argue as follows. The first-order theory of (idempotent) algebraically complete semirings is recursively enumerable. The context-free languages over A are free for the class of semirings that can be embedded in continuous idempotent semirings (see [17]). Since their equational theory is not r.e. when $|A| \geq 2$, the equational theory of idempotent continuous semirings is not r.e. In fact, when $|A| \geq 2$, the free idempotent algebraically complete semiring on A does not embed in a continuous semiring.

Proposition 3. *In any algebraically complete semiring, for all elements a,*

$$(\sum_{i=0}^{n} a^i) \leq a^*. \tag{13}$$

For any integer $n \geq 0$, we will denote by n also the term which is the n-fold sum of 1 with itself. When n is 0, this is just the term 0.

Proposition 4. *In any algebraically complete semiring, for any element $a \geq 1$,*

$$a^* \;=\; a^* + 1 \;=\; a^* + a^* \;=\; a^* \cdot a^* \;=\; a^{**}. \tag{14}$$

Proof. (Sketch) The inequations $a^* \geq x$ follow from (9) using (1) and (2), the reverse ones from monotonicity and $a^* + a^* = 2a^* \leq a^*a^* \leq a^{**}$ using (13). □

Remark 1. An element x of an ordered semiring is *reflexive* if $1 \leq x$ and *transitive* if $xx \leq x$. In an ordered semiring which is a Park μ-semiring, we call $x^{\odot} := \mu z(1 + zz + x)$ the *reflexive transitive closure* of x. We remark that in an algebraically complete semiring, we have $x^* \leq x^{\odot}$ and

$$x^{\odot} \leq x^* \quad \Longleftrightarrow \quad x^* + x^* \leq x^*.$$

So when $+$ is idempotent as in $\mathbf{R}_M$ or $\mathbf{L}_A$, then iteration x^* coincides with reflexive transitive closure $x^{\odot}$, see also [18, 20, 4, 7]. In $\mathbb{N}_\infty$, $0^* \neq 0^{\odot} = 1^* = \infty$.

Proposition 5. *In any algebraically complete semiring, we have for $n \in \mathbb{N}$*

$$0^* \;=\; 1 \quad \textit{and} \quad (n+1)^* \;=\; 1^*. \tag{15}$$

Proof. By induction on n, using (13) and (14). □

A *morphism* between μ-semirings or Conway μ-semirings A, B is any function $h : A \to B$ that commutes with the term functions: if $h^X : A^X \to B^X$ is the pointwise extension of h, then $t_B \circ h^X = h \circ t_A$, for all terms t. A morphism of Park μ-semirings and algebraically complete semirings is a μ-semiring morphism which is a monotone function. A morphism of continuous semirings is a semiring morphism which is a continuous function.
It is not difficult to prove that for any set A, the power series semiring $\mathbb{N}_\infty\langle\langle A^*\rangle\rangle$, is the free continuous semiring generated by A: For any continuous semiring S and function $h : A \to S$, there is a unique morphism of continuous semirings $\overline{h} : \mathbb{N}_\infty\langle\langle A^*\rangle\rangle \to S$ extending h. In particular, $\mathbb{N}_\infty$ is the initial continuous semiring. It is also an algebraically complete semiring and a symmetric inductive *-semiring (cf. section 4). In [10], it has been shown that $\mathbb{N}_\infty$ is initial in the category of (symmetric) inductive *-semirings. The proof of the following similar result is too long to be included here:

Theorem 1. *If t is a closed term, then for some $c \in \mathbb{N}_\infty$, equation $t = c$ holds in all algebraically complete semirings.*

Corollary 1. *$\mathbb{N}_\infty$ is initial in the class of all algebraically complete semirings, and $\mathbb{B}$ is initial in the class of all idempotent algebraically complete semirings.*

4 Algebraic Conway Semirings

Next we turn to equational notions derived from algebraically complete semirings, and connect these with related notions in the literature.

Definition 6. An *algebraic Conway semiring* is a Conway μ-semiring that satisfies (9), (10) and (11).

Thus, any algebraically complete semiring is an algebraic Conway semiring. In [6], a *Conway semiring* is defined to be a semiring S with an operation $^* : S \to S$ subject to the equations

$$(x+y)^* = (x^*y)^*x^* \quad \text{and} \quad (xy)^* = 1 + x(yx)^*y.$$

Any algebraic Conway semiring is a Conway semiring (using $x^r = x^\ell$ for x^*):

Proposition 6. *For any terms t and s the following equations hold in any algebraic Conway semiring:*

$$(t+s)^* = (t^*s)^*t^* \tag{16}$$
$$(ts)^* = 1 + t(st)^*s. \tag{17}$$

Proof. For (16), note

$$\begin{aligned}
(x+y)^r &= \mu z((x+y)z+1) \\
&= \mu z(xz+yz+1) \\
&= \mu z \mu v(xv+yz+1) && \text{by (4)} \\
&= \mu z(x^r(yz+1)) && \text{by (9)} \\
&= \mu z((x^ry)z+x^r) \\
&= (x^ry)^rx^r.
\end{aligned}$$

As for equation (17), note that in the proof of Proposition 2, we have already derived $(xy)^r = x(yx)^ry+1$ from the composition identity and (9) only. □

Equation (9) gives $\mu z(zt+s) = st^\ell$ for $z \notin \mathit{free}(st)$. Its generalization (18) allows us to eliminate left-recursion, which is essential for the Greibach-Normal-Forms.

Proposition 7. *For any terms t and s which may have free occurrences of the variable z, the following equations hold in any algebraic Conway semiring:*

$$\mu z(zt+s) = \mu z(st^*). \tag{18}$$

Proof. By (4) and (10), we have $\mu z(zt+s) = \mu z(\mu x(xt+s)) = \mu z(st^\ell)$. □

Remark 2. Conway semirings are left- and right-linear versions of algebraic Conway semirings: they satisfy the composition identity $\mu z.t[s/z] = t[\mu z.s[t/z]/z]$ for the terms $t = xz+1$ and $s = yz$ (resp. $t = zx+1$ and $s = zy$) and the diagonal identity $\mu z.t[z/v] = \mu z.\mu v.t$ for the term $t = xv + yz + 1$ (resp. $t = vx + zy + 1$), which are right-(resp. left-)linear in the recursion variables.

The left- and right-linear versions of algebraically complete semirings are the *symmetric inductive* **-semirings* of [10]. An *inductive* **-semiring* is an ordered semiring with a *-operation, satisfying

$$xx^* + 1 \leq x^* \quad \text{and} \quad xz + y \leq z \to x^*y \leq z.$$

A *symmetric* inductive *-semiring also satisfies $zx + y \leq z \to yx^* \leq z$. It follows that an inductive *-semiring satisfies $x^*x + 1 = x^*$ and has a monotone *-operation. Propositions 3 – 5 hold in all inductive *-semirings.

Proposition 8. [10] *Every inductive* **-semiring is a Conway semiring.*

A *Kozen semiring*, called *Kleene algebra* in [14], is an idempotent symmetric inductive *-semiring in which the partial order is given by $x \leq y :\iff x+y = y$. Any idempotent algebraically complete semiring is a Kleene algebra.

5 Term Vectors and Term Matrices

We write $\vec{t}$ for the term vector $(t_1, \ldots, t_n)$, $n \geq 1$. When $\vec{x} = (x_1, \ldots, x_n)$ is a vector of *different* variables, we define the term vector $\mu\vec{x}.\vec{t}$ by induction on n:

- If $n = 1$, then $\mu\vec{x}.\vec{t} := (\mu x_1.t_1)$.
- If $n = m + 1 > 1$, $\vec{y} = (x_1, \ldots, x_m)$, $\vec{z} = (x_n)$, $\vec{r} = (t_1, \ldots, t_m)$, $\vec{s} = (t_n)$, put

$$\mu\vec{x}.\vec{t} := (\mu\vec{y}.\vec{r}[\mu\vec{z}.\vec{s}/\vec{z}],\ \mu\vec{z}.\vec{s}[\mu\vec{y}.\vec{r}/\vec{y}]).$$

This definition is motivated by the Bekić–De Bakker–Scott rule [3, 2]. For term verctors $\vec{t} = (t_1, \ldots, t_n)$ and $\vec{t'} = (t'_1, \ldots, t'_n)$ of dimension $n \geq 1$, we say that the equation $\vec{t} = \vec{t'}$ *holds* in a μ-semiring A if each equation $t_i = t'_i$ does.
The following facts are proven in [6] in the more general context of Conway theories (Conway algebras). See Chapter 6, Section 2.

Theorem 2. *Suppose that A is a Conway μ-semiring. Then for each term vector $\vec{t}$ and vector $\vec{x}$ of different variables as above, the equation*

$$\mu\vec{x}.\vec{t} = (\mu\vec{y}.\vec{r}[\mu\vec{z}.\vec{s}/\vec{z}],\ \mu\vec{z}.\vec{s}[\mu\vec{y}.\vec{r}/\vec{y}]) \tag{19}$$

holds in A for each *way of splitting $\vec{x}$ and $\vec{t}$ into two parts as $\vec{x} = (\vec{y}, \vec{z})$ and $\vec{t} = (\vec{r}, \vec{s})$ such that the dimension of $\vec{y}$ agrees with the dimension of $\vec{r}$.*

The vector versions of the composition and diagonal identities, and hence the vector version of the fixed-point equation, hold in any Conway μ-semiring:

Theorem 3. *For all term vectors $\vec{t}, \vec{s}$ and variable vectors $\vec{x}, \vec{y}$ of the same size, any Conway μ-semiring satisfies*

$$\mu\vec{y}.\vec{t}[\vec{s}/\vec{y}] = \vec{t}[\mu\vec{x}.\vec{s}[\vec{t}/\vec{x}]/\vec{x}], \tag{20}$$

$$\mu\vec{x}.\mu\vec{y}.\vec{t} = \mu\vec{x}.\vec{t}[\vec{x}/\vec{y}], \tag{21}$$

$$\mu\vec{x}.\vec{t} = \vec{t}[\mu\vec{x}.\vec{t}/\vec{x}]. \tag{22}$$

Moreover, writing $\mu\vec{x}.\vec{t} = (r_1, \ldots, r_n)$, the permutation identity

$$\mu(x_{1\pi}, \ldots, x_{n\pi}).(t_{1\pi}, \ldots, t_{n\pi}) = (r_{1\pi}, \ldots, r_{n\pi})$$

holds for all permutations $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$.

Proposition 9. *Suppose that $\vec{t}$ and $\vec{r}$ are term vectors of dimension m, n, respectively. Moreover, suppose that the components of the vectors of variables $\vec{x}$, $\vec{y}$ of dimension m and n, respectively, are pairwise distinct. Then*

$$\mu(\vec{x}, \vec{y}).(\vec{t}, \vec{r}) = \mu(\vec{x}, \vec{y}).(\vec{t}[\vec{r}/\vec{y}], \vec{r}) \tag{23}$$

holds in any Conway μ-semiring.

The fact that the fixed-point inequation and -induction rule extend to vector versions is essentially due to Bekić [3] and de Bakker and Scott [2]. See also [9].

Theorem 4. *In any Park μ-semiring,*

$$\vec{t}[\mu\vec{x}.\vec{t}/\vec{x}] \leq \mu\vec{x}.\vec{t} \quad \text{and} \quad \vec{t}[\vec{y}/\vec{x}] \leq \vec{y} \;\rightarrow\; \mu\vec{x}.\vec{t} \leq \vec{y}$$

hold for all term vectors $\vec{t}$ and vectors $\vec{x}, \vec{y}$ of variables of the same size.

By induction on the dimension, using the Bekić-Scott-equations, one obtains:

Lemma 1. *Let A be a μ-semiring. For all vectors $\vec{t}, \vec{t}'$ of terms and $\vec{x}$ of variables of the same dimension, if $\vec{t}_A = \vec{t}'_A$, then $(\mu\vec{x}.\vec{t})_A = (\mu\vec{x}.t')_A$.*

A *term matrix* $T = (t_{i,j})$ of size $n \times m$, where $n, m \geq 1$, consists of a term vector $\vec{t}$ of length nm, listing the entries of T by rows, and the dimension (n, m).
We denote by 1_n the $n \times n$ matrix whose diagonal entries are 1 and whose other entries are 0, and by $0_{n,m}$ the $n \times m$ matrix whose entries are all 0. When S and T are term matrices of appropriate size, we define $S + T$ and ST in the obvious way. Suppose that T is a term matrix and X is a variable matrix of the same size $n \times m$, with pairwise distinct variables, and let $\vec{t}$ and $\vec{x}$ be obtained by listing their entries by rows. Then $\mu X.T$ is the term matrix of size $n \times m$ consisting of the term vector $\mu\vec{x}.\vec{t}$ and the dimension (n, m).
For square matrices T, we can define the left- and right iterations T^ℓ and T^r, using μ. Independently of μ, we now define a matrix T^* by induction on the dimension of T and then relate T^* to T^ℓ and T^r.

Definition 7. For an $n \times n$ term matrix T, define a matrix T^* inductively:

1. If $n = 1$ and $T = (\, t \,)$ for some term t, then $T^* := (\, t^* \,)$.
2. If $n = k + l$, where $k \geq l = 1$, and

$$T = \begin{pmatrix} R & S \\ U & V \end{pmatrix} \quad \text{where } R \text{ is } k \times k \text{ and } V \text{ is } l \times l, \tag{24}$$

then

$$T^* := \begin{pmatrix} \overline{R} & \overline{S} \\ \overline{U} & \overline{V} \end{pmatrix} \quad \text{where} \quad \begin{array}{ll} \overline{R} = (R + SV^*U)^* & \overline{S} = \overline{R}SV^* \\ \overline{U} = \overline{V}UR^* & \overline{V} = (V + UR^*S)^*. \end{array} \tag{25}$$

When $T = (\, t_{ij} \,)$ and $S = (\, s_{ij} \,)$ are term matrices of the same size, we say that $T = S$ *holds* in a μ-semiring A if each equation $t_{ij} = s_{ij}$ holds in A.

Theorem 5. *([6], Ch.9, Theorem 2.1) Let A be an algebraic Conway semiring. Suppose that T is an $n \times n$ term matrix, S is an $n \times m$ (resp. $m \times n$) term matrix and let X be an $n \times m$ (resp. $m \times n$) matrix of new variables. Then the equations*

$$\mu X(TX + S) = T^*S \tag{26}$$

$$\mu X(XT + S) = ST^* \tag{27}$$

hold in A. Moreover, (25) holds, if T splits like (24) for any k, l and submatrices of appropriate dimensions.

In particular, the coincidence of left- and right iteration for matrices holds in A,

$$T^\ell := \mu X(XT + 1_n) = \ T^* \ = \ \mu X(TX + 1_n) =: T^r. \tag{28}$$

Lemma 2. *If A is a μ-semiring, so is $Mat_{n\times n}(A)$, for each $n \geq 1$.*

Proof. For each term t we define a term matrix t' of size $n \times n$ inductively,

$$\begin{array}{ll} x' := (x_{i,j}), & (t_1 + t_2)' := t_1' + t_2', \\ 0' := 0_{n,n}, & (t_1 \cdot t_2)' := t_1' \cdot t_2', \\ 1' := 1_n, & (\mu x.t)' := \mu x'.t'. \end{array}$$

using different new variables $x_{i,j}$ and $+$ and $\cdot$ for matrices on the right hand side. Let $M := Mat_{n\times n}(A)$. Each $\rho : X \to M$ is obtained from some $\hat{\rho} : X \to A$ such that $\rho(x) = (\hat{\rho}(x_{i,j}))$ when $x' = (x_{i,j})$. We define $t_M : M^X \to M$ by $t_M(\rho) := t'_A(\hat{\rho})$. Using Lemma 1, one can check that M is a μ-semiring. □

Theorem 6. *Let $n \geq 1$. If A is an algebraic Conway semiring, so is $Mat_{n\times n}(A)$. If A is an algebraically complete semiring, then so is $Mat_{n\times n}(A)$.*

Proof. By Lemma 2, $M := Mat_{n\times n}(A)$ is a μ-semiring. If A is algebraic Conway, M satisfies the Conway identities (3) and (4), by Theorem 3. If A is an algebraically complete μ-semiring, then M, ordered componentwise, is a Park μ-semiring by Theorem 4. By (26) – (28), M satisfies (9) and (10) and hence is algebraically complete. □

By Proposition 6 and Proposition 7, it follows immediately:

Corollary 2. *Let X be an $m \times n$ matrix of distinct variables, T an $n \times n$ and S an $m \times n$ term matrix whose terms may contain variables of X. Then*

$$TT^* + 1_n = T^*, \tag{29}$$

$$\mu X(XT + S) = \mu X(ST^*) \tag{30}$$

hold in any algebraic Conway semiring A.

6 Normal Forms

In this section we present a Greibach normal form theorem applicable to all algebraically complete semirings. The following normal form theorem is standard.

Theorem 7. *(See, e.g., [6], Chapter 9, Theorem 1.4, Remark 1.5) In algebraic Conway semirings, any μ-term is equivalent to the first component of a term vector of the form $\mu(x_1, \ldots, x_n).(p_1, \ldots, p_n)$, where each p_i is a finite term.*

Let K be the set of terms $\{0, 1, \ldots\} \cup \{1^*\}$, which, by Theorem 1, amount to *all* closed terms over algebraically complete semirings.

Proposition 10. *In algebraic Conway semirings, $kx = xk$ holds for all $k \in K$.*

A *monomial* is a term of the form ku, where $k \in K$ and u is a product of variables. When u is the empty product, the monomial ku is called *constant*. The *leading factor* of a monomial ku, where $u = x_1 \cdots x_n$ is a nonempty product of variables, is the variable x_1. A *polynomial* is any finite sum of monomials. In particular, 0 is a polynomial.

Definition 8. A term vector $\mu\vec{x}.\vec{t}$, where $\vec{t} = (t_1(\vec{x}, \vec{y}), \ldots, t_n(\vec{x}, \vec{y}))$, is a *context-free grammar* if each t_i is a polynomial. The context-free grammar $\mu\vec{x}.\vec{t}(\vec{x}, \vec{y})$ *has no chain rules*, if no t_i has a monomial of the form kx where $k \in K \setminus \{0\}$ and $x \in \vec{x}$; it *has no ϵ-rules* if no t_j has a monomial of the form k where $k \in K \setminus \{0\}$. A context-free grammar $\mu\vec{x}.\vec{t}$ is in *Greibach normal form* if each t_i is a polynomial which is a sum of non-constant monomials whose leading factors are among the *parameters* $y_1, \ldots, y_m$.

The next theorem is a first version of Greibach's normal form theorem. The algorithm in the proof is due to Rosenkrantz [22] (cf. [12], Algorithm 4.9.1). We use properties of least pre-fixed-points rather than power series to prove its correctness, and thus show that it holds in any algebraic Conway semiring.
If $\mu\vec{x}.\vec{t}$ has dimension n and $m \leq n$, we denote by $(\mu\vec{x}.\vec{t})_{[m]}$ the vector whose components are the first m components of $\mu\vec{x}.\vec{t}$. We write $(\mu\vec{x}.\vec{t})_1$ for $(\mu\vec{x}.\vec{t})_{[1]}$.

Theorem 8. *Let $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{z} = (z_1, \ldots, z_p)$ be distinct variables and $\mu\vec{x}.\vec{t}(\vec{x}, \vec{z})$ a context-free grammar that has no chain-rules and no ϵ-rules. Then there is a context-free grammar*

$$\mu(x_1, \ldots, x_n).(s_1, \ldots, s_n)(x_1, \ldots, x_n, z_1, \ldots, z_p)$$

in Greibach normal form, such that $m \leq n \leq m + m^2$ and the equation

$$\mu\vec{x}.\vec{t} = (\mu(x_1, \ldots, x_n).(s_1, \ldots, s_n))_{[m]}$$

holds in any algebraic Conway semiring.

Proof. By distributivity, we can write

$$t_j(\vec{x}, \vec{z}) = \sum_{k=1}^{m} (x_k \cdot t_{kj}(\vec{x}, \vec{z})) + r_j(\vec{x}, \vec{z}),$$

where r_j is 0 or a sum of non-constant monomials whose leading factors are parameters; constant monomials $\neq 0$ do not occur since $\mu\vec{x}.\vec{t}$ has no ϵ-rules. So we can write $\mu\vec{x}.\vec{t}$ as

$$\mu\vec{x}(\vec{x} \cdot T(\vec{x}, \vec{z}) + \vec{r}(\vec{x}, \vec{z})),$$

using the $m \times m$ matrix $T = (t_{ij})$ and $\vec{r} = (r_1, \ldots, r_m)$. With an $m \times m$ matrix $Y = (\, y_{ij} \,)$ of new variables, consider the term

$$\mu(\vec{x}, Y).(\vec{r}Y + \vec{r}, TY + T). \tag{31}$$

Then in all algebraic Conway semirings, we have:

$$\begin{aligned}
(\mu(\vec{x}, Y).(\vec{r}Y + \vec{r}, TY + T))_{[m]} &= \mu\vec{x}(\vec{r}T^*T + \vec{r}) && \text{by (19)} \\
&= \mu\vec{x}(\vec{r}(T^*T + 1_m)) && \text{by (26)} \\
&= \mu\vec{x}(\vec{r}T^*) && \text{by (29)} \\
&= \mu\vec{x}(\vec{x}T + \vec{r}) && \text{by (30)} \\
&= \mu\vec{x}.\vec{t}.
\end{aligned}$$

It remains to be shown that (31) contains no essential left recursion. First, each component of $\vec{r}Y + \vec{r}$ is of the form

$$(\vec{r}Y)_j + r_j = \sum_{k=1}^{m} (r_k \cdot y_{kj}) + r_j,$$

which is 0 or can be written as a sum of non-constant monomials whose leading factors are parameters. Second, each component of $TY + T$ is of the form

$$\sum_{k=1}^{m} t_{ik} \cdot y_{kj} + t_{ij}. \tag{32}$$

By Proposition 9 leading factors x_u in summands of t_{ik} and t_{ij} can be replaced by $(\vec{r}Y)_u + r_u$. Since $\mu\vec{x}.\vec{t}$ has no chain-rules, none of the t_{ik} or t_{ij} is a constant $k \in K \setminus \{0\}$, so y_{kj} is not a leading factor of $t_{ik} \cdot y_{kj}$ and no monomial in the new polynomials is a constant $\neq 0$. □

Example 4. Let G be the context-free grammar

$$A = BC + a, \quad B = Ab + CA, \quad C = AB + CC$$

over the alphabet $\{a, b\}$. In matrix notation, this is

$$(A, B, C) = (A, B, C) \cdot T + (a, 0, 0) \qquad \text{where} \quad T = \begin{pmatrix} 0 & b & B \\ C & 0 & 0 \\ 0 & A & C \end{pmatrix}. \quad (33)$$

By the proof, the least solution of (33) is the same as the least solution of the (essentially) right-recursive system

$$\begin{array}{ccc} (A, B, C) & = & (a, 0, 0) \cdot Y + (a, 0, 0) \\ Y & = & T \cdot Y + T \end{array} \qquad \text{where} \quad Y = \begin{pmatrix} Y_{1,1} & Y_{1,2} & Y_{1,3} \\ Y_{2,1} & Y_{2,2} & Y_{2,3} \\ Y_{3,1} & Y_{3,2} & Y_{3,3} \end{pmatrix}.$$

Multiplying out gives

$$\begin{array}{lll} A = aY_{1,1} + a & Y_{2,2} = CY_{1,2} & Y_{1,3} = bY_{2,3} + BY_{3,3} + B \\ B = aY_{1,2} & Y_{2,3} = CY_{1,3} & Y_{3,1} = AY_{2,1} + CY_{3,1} \\ C = aY_{1,3} & Y_{1,1} = bY_{2,1} + BY_{3,1} & Y_{3,2} = AY_{2,2} + CY_{3,2} + A \\ Y_{2,1} = CY_{1,1} + C & Y_{1,2} = bY_{2,2} + BY_{3,2} + b & Y_{3,3} = AY_{2,3} + CY_{3,3} + C \end{array}$$

Finally, plug in the right hand sides for A, B, C in the Y-equations.

For algebraically complete semirings, we can show a slightly more general version of the Greibach normal form theorem, based on the following

Lemma 3 (Elimination of chain rules). *For every context-free grammar $\mu\vec{x}.\vec{t}(\vec{x}, \vec{z})$ there is a context-free grammar $\mu\vec{x}.\vec{s}$ without chain rules, such that*

$$\mu\vec{x}.\vec{t} = \mu\vec{x}.\vec{s}$$

holds in all algebraically complete semirings. If $\mu\vec{x}.\vec{t}$ has no ϵ-rules, then $\mu\vec{x}.\vec{s}$ has no ϵ-rules.

The proof has to be omitted due to space limits. It follows that in Theorem 8, restricted to algebraically complete semirings, one can drop the assumption that $\mu\vec{x}.\vec{t}$ has no chain rules. It is substantially more difficult to get rid of ϵ-rules:

Lemma 4 (Elimination of ϵ-rules). *Let $\vec{t}(\vec{x}, \vec{z})$ be an m-tuple of polynomials in $\vec{x} = x_1, \ldots, x_m$ with parameters $\vec{z}$. There are constants $\vec{k} \in K^m$ and polynomials $\vec{s}(\vec{x}, \vec{z})$ without non-zero constant monomials such that*

$$\mu\vec{x}.\vec{t} = \vec{k} + \mu\vec{x}.\vec{s}$$

holds in all continuous semirings and in all idempotent algebraically complete semirings. (Idempotency is not used for $\mu\vec{x}.\vec{t} \leq \vec{k} + \mu\vec{x}.\vec{s}$.)

Proof. (Idea) We can here only sketch the construction of $\vec{k}$ and $\vec{s}$. Write $\vec{t}(\vec{x},\vec{z}) = \vec{q}(\vec{x},\vec{z}) + \vec{p}(\vec{x}) + \vec{c}$ where $\vec{q}(\vec{x},\vec{z})$ sums the monomials containing at least one of the parameters $\vec{z}$, $\vec{p}(\vec{x})$ the non-constant monomials not containing a parameter, and $\vec{c}$ the constant monomials. Put $\vec{k} := \mu\vec{x}(\vec{p}(\vec{x}) + \vec{c})$. By Theorem 1, $\vec{k} \in \mathbb{N}_\infty^m$. Then write $\vec{t}(\vec{x}+\vec{k},\vec{z})$ as a sum of non-constant and constant monomials, using the semiring equations, which gives $\vec{s}(\vec{x},\vec{z})$ via $\vec{t}(\vec{x}+\vec{k},\vec{z}) = \vec{s}(\vec{x},\vec{z}) + \vec{k}$.
For example, if $\vec{t}(\vec{x}) = (x+1, 1, xy)$, then $\vec{t}(\vec{x}) = \vec{q}(\vec{x}) + \vec{p}(\vec{x}) + \vec{c}$ with $\vec{q}(\vec{x}) = \vec{0}$, $\vec{p}(\vec{x}) = (x, 0, xy)$, and $\vec{c} = (1, 1, 0)$. So $\vec{k} = \mu\vec{x}(\vec{p}(\vec{x}) + \vec{c}) = \mu\vec{x}.\vec{t} = (1^*, 1, 1^*)$ and

$$\vec{t}(\vec{x}+\vec{k}) = (x + 1^* + 1, 1, xy + x1^* + 1^*y + 1^*1^*) = \vec{s}(\vec{x}) + \vec{k}$$

for $\vec{s}(\vec{x}) = (x, 0, xy + 1^*(x+y))$. Hence $\vec{k} + \mu\vec{x}.\vec{s} = \vec{k} + \vec{0} = \mu\vec{x}.\vec{t}$. □

We don't know if Lemma 4 holds for algebraically complete semirings in general, though we know that it does if $\mu\vec{x}.\vec{t}$ is of size 1. Hence, of the version of Greibach's normal form theorem involving elimination of ϵ-rules we only have:

Theorem 9. *For each context-free grammar $\mu\vec{x}.\vec{t}$ of length m there is $\vec{k} \in K^m$ and a context-free grammar $\mu\vec{x}.\vec{r}$ in Greibach normal form such that $\mu\vec{x}.\vec{t} = \vec{k} + (\mu\vec{x}.\vec{r})_{[m]}$ holds in all continuous semirings and in all idempotent algebraically complete semirings.*

Proof. By Lemma 4, there are $\vec{k} \in K^m$ and a context-free grammar $\mu\vec{x}.\vec{s}$ without ϵ-rules such that $\mu\vec{x}.\vec{t} = \vec{k} + \mu\vec{x}.\vec{s}$ holds in all continuous semirings and in all idempotent algebraically complete semirings. By Lemma 3, we may assume that $\mu\vec{x}.\vec{s}$ does not have chain-rules. Hence, by Theorem 8, there is a context-free grammar $\mu(\vec{x},\vec{y}).\vec{r}$ such that $\mu\vec{x}.\vec{s} = (\mu(\vec{x},\vec{y}).\vec{r})_{[m]}$ holds in all algebraic Conway semirings, hence in all algebraically complete semirings. □

Since the set of context-free languages over A form an idempotent algebraically complete semiring, Theorem 9 implies the classical Greibach normal form theorem (cf. [11, 12]). Together with Theorem 7, we obtain:

Corollary 3. *For each term t, either t is closed and for some $k \in K$, $t = k$ holds in all algebraically complete semirings, or t is not closed and for some $k \in K$ and some term $\mu\vec{x}.\vec{s}$ in Greibach normal form, $t = k + (\mu\vec{x}.\vec{s})_1$ holds in all continuous semirings and in all idempotent algebraically complete semirings.*

7 Open Problems

Problem 1. Find concrete representations of the free algebraically complete (idempotent) semirings.

We conjecture that the one-generated free algebraically complete (idempotent) semiring consists of the algebraic series in $\mathbb{N}_\infty\langle\langle a^*\rangle\rangle$ (regular = context-free languages in $\{a\}^*$, respectively), where a is a single letter. When $|A| \geq 2$, it is

not true that the free algebraically complete semiring on A is the semiring of algebraic series in $\mathbb{N}_\infty\langle\langle A^*\rangle\rangle$. Also, when $|A| \geq 2$, the free algebraically complete idempotent semiring on A is not the semiring of context-free languages in A^*.

Problem 2. Does ϵ-elimination hold in all *non-idempotent* algebraically complete semirings? Does it hold in all algebraic Conway semirings satisfying $1^* = 1^{**}$?

Problem 3. To what extent do the normal form theorems hold when, as in process algebra, we only have one-sided distributivity of multiplication over sum?

Problem 4. Is every Kleene algebra embeddable in an idempotent algebraically complete semiring? Is every symmetric inductive *-semiring embeddable in an algebraically complete semiring?

If so, then the Horn theory of Kleene algebras, which is undecidable ([15]), is the same as the rational Horn theory of idempotent algebraically closed semirings.

References

[1] L. Aceto, Z. Ésik and A. Ingólfsdóttir. A fully equational proof of Parikh's theorem. *BRICS Report Series*, RS-01-28, Aarhus, 2001. 136

[2] J. W. de Bakker and D. Scott. A theory of programs. *IBM Seminar*, August, 1969. 142, 143

[3] H. Bekić. Definable operations in general algebra, and the theory of automata and flowcharts. *Technical Report*, IBM Laboratory, Vienna, 1969. 142, 143

[4] L. Bernátsky, S. L. Bloom, Z. Ésik, and Gh. Stefanescu. Equational theories of relations and regular sets, extended abstract. In *Proceedings of the Conference on Words, Combinatorics and Semigroups,Kyoto, 1992*, pages 40–48. World Scientific Publishing Co. Pte. Ltd., 1994. 136, 140

[5] S. L. Bloom and Z. Ésik. Iteration algebras, *Int. J. Foundations of Computer Science*, 3(1991), 245–302.

[6] S. L. Bloom and Z. Ésik. *Iteration Theories*, Springer, 1993. 136, 139, 141, 142, 144, 145

[7] M. Boffa. Une condition impliquant toutes les identités rationnelles. *RAIRO Inform. Théor. Appl.*, 29 (1995), 515–518. 136, 140

[8] J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, London, 1971. 135

[9] Z. Ésik. Completeness of Park induction, *Theoretical Computer Science*, 177 (1997), 217–283. 143

[10] Z. Ésik and W. Kuich. Inductive $*$-semirings. To appear in *Theoretical Computer Science.* 140, 142

[11] S. A. Greibach. A new normal-form theorem for context-free, phrase-structure grammars. *J. of the Association for Computing Machinery*, 12 (1965), 42–52. 136, 148

[12] M. Harrison. *Introduction to Formal Languages.* Addison Wesley, Reading, 1978. 145, 148

[13] M. W. Hopkins and D. Kozen. Parikh's theorem in commutative Kleene algebra. In *Proc. Symp. Logic in Computer Science (LICS'99)*, IEEE Press, 1999, 394–401. 136

[14] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. In *6th Ann. Symp. on Logic in Computer Science, LICS'91.* Computer Society Press, 1991, 214–225. 136, 142

[15] D. Kozen. On the complexity of reasoning in Kleene algebra. In *Proc. 12th Symp. Logic in Computer Science*, IEEE Press, 1997, 195–202. 149

[16] D. Krob. Complete systems of B-rational identities. *Theoret. Comput. Sci.*, 89 (1991), 207–343. 136

[17] H. Leiß. Towards Kleene Algebra with Recursion. In *Proc. 5th Workshop on Computer Science Logic, CSL '91.* Springer LNCS 626, 242–256, 1991. 136, 139

[18] K. C. Ng and A. Tarski. Relation algebras with transitive closure. *Notices of the American Math. Society*, 24:A29–A30, 1977. 140

[19] D. Niwinski. Equational μ-calculus. In *Computation Theory (Zaborow, 1984)*, pages 169–176, Springer LNCS 208, 1984. 138

[20] V. R. Pratt. Action Logic and Pure Induction. In *Logics in AI: European Workshop JELIA '90.* Springer LNCS 478, 97–120, 1990. 135, 140

[21] V. N. Redko. On the determining totality of relations for the algebra of regular events. (Russian) *Ukrain. Mat. Ž.*, 16 (1964), 120–126. 135

[22] D. J. Rosenkrantz. Matrix equations and normal forms for context-free grammars. *Journal of the Association for Computing Machinery*, 14 (1967), 501–507. 136, 145

[23] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the Association for Computing Machinery*, 13 (1966), 158–169. 135

[24] L. Santocanale. On the equational definition of the least prefixed point. In *MFCS 2001*, pages 645-656, Springer LNCS 2136, 2001. 136

Proofnets and Context Semantics for the Additives

Harry G. Mairson[1] and Xavier Rival[2]

[1] Computer Science Department, Brandeis University
Waltham, Massachusetts 02454
mairson@cs.brandeis.edu
[2] École Normale Superieure
45 rue d'Ulm, 75005 Paris
rival@di.ens.fr

Abstract. We provide a *context semantics* for Multiplicative-Additive Linear Logic (MALL), together with *proofnets* whose reduction preserves semantics, where proofnet reduction is equated with cut-elimination on MALL sequents. The results extend the program of Gonthier, Abadi, and Lévy, who provided a "geometry of optimal λ-reduction" (context semantics) for λ-calculus and Multiplicative-Exponential Linear Logic (MELL). We integrate three features: a semantics that uses *buses* to implement slicing; a proofnet technology that allows *multidimensional boxes* and *generalized garbage*, preserving the linearity of additive reduction; and finally, a *read-back procedure* that computes a cut-free proof from the semantics, a constructive companion to full abstraction theorems.

Linear Logic [4, 7] models computation and reasoning that is sensitive to the notion of *consumable resources*. Its *multiplicative* fragment ($\otimes$, $\parr$) allows linear products (pairing and unpairing), implementing functions: a *context* pairs a continuation and an argument, a *function* unpairs and connects the two. Its *additive* fragment ($\oplus$, &) allows linear sums (injection and case dispatch), implementing features of *processes* in the style of CSP or CCS [3, 17, 12]. The *exponential* fragment implements sharing of resources: arguments, control contexts. We can then implement, for example, graph reduction technology for λ-calculus with control operators (`call/cc`, `abort`, jumps), and related mechanical proof systems for classical logic—taking care of the *sharing* and *copying* implicit in these calculi [16, 18, 11].

This logic was subsequently augmented with *proofnets* [8, 13] a proof notation which eliminates the irrelevant sequentialization that complicates cut elimination. Further, Geometry of Interaction (GoI) developed the idea that proof reduction can be seen as a local interaction process [5, 6]. GoI was simplified in the "geometry of optimal λ-reduction" by Gonthier, Abadi and Lévy [9, 10] in the context of the MELL fragment. By introducing simple data-structures, known as *context semantics*, they reduced Hilbert spaces to Dilbert spaces, and developed a proofnet technology which implemented the context semantics locally. Reduction on proofnets preserves the semantics, and Lamping's algorithm

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 151–166, 2002.
© Springer-Verlag Berlin Heidelberg 2002

for optimal reduction of λ-terms [14] is a method of graph reduction. They further indicated how to *read back* any part of the Böhm tree (normal form) of a λ-term from its context semantics. Can this program be carried out for full Linear Logic? We extend their results to the MALL fragment (multiplicatives and additives): this may be a step towards a satisfactory proofnet syntax for full Linear Logic with a good characterization of proofs.

The MALL fragment is problematic since it does not, like MLL, have a nice cut-elimination procedure. Additive cut-elimination is not really linear, since proof structure is *discarded*—how do we do this *locally*? How do we reduce cuts involving auxiliary cut formulas? The latter also involves (additive) *copying.* Part of our work will be to understand and improve these reduction procedures for MALL, incorporating both better proofnets and better MALL syntax.

The main contributions of this paper are to provide an integrated development of (1) a context semantics for the MALL fragment; (2) a proofnet technology allowing normalization of MALL proofs, using the ideas of *multidimensional boxes* and *generalized garbage*; and (3) a read-back procedure that inputs a valid context semantics and outputs a normalized proofnet.

Section 1 defines context semantics; Section 2 presents a proofnet syntax that implements this semantics locally. Then, we show in Section 3 correctness of proofnets normalization and in Section 4 the existence and the correctness of the read-back algorithm.

1 Context Semantics for MALL Prooftrees

A brief MALL tutorial is found in Appendix A. Our semantics is described by *contexts* comprising *eigenweights* and *command strings*. Contexts relate the structure of formulas and proofs.

Definition 1 (Eigenweight, eigenvalue). *An* eigenweight *is a variable ranging over the booleans* $\mathcal{B} = \{0, 1\}$*. If* W *is a set of eigenweights, an* eigenvalue with base W *is a function* $\omega : W \to \mathcal{B}$*. Each eigenweight corresponds to a* &*-link (rule) in the proof. The value* 0 *(resp.* 1*) characterizes the left (resp. right) part of the subproof above the link.*

Definition 2 (Context). $\mathcal{T} = \{l, r, g, d\}$ *comprise the* tokens*;* l *and* r *(*left *and* right*) are the* multiplicative tokens*,* g *and* d *(*gauche *and* droite*) are the* additive tokens*. The* command strings $\mathcal{S}$ *are defined by* $s \longrightarrow \diamond \mid t.s$ *where* $t \in \mathcal{T}$*. Given a set* W *of eigenweights, the* contexts with base W is the set $\mathcal{C}_W$ *of pairs* (s, ω) *where* $s \in \mathcal{S}$ *and* $\omega : W \to \mathcal{B}$*.*

Given W, the set C_F of *valid command strings* for a formula F is: $C_{A\otimes B} = C_{A\parr B} = l.C_A \cup r.C_B$ for the multiplicatives, $C_{A\&B} = C_{A\oplus B} = g.C_A \cup d.C_B$ for the additives, and $C_V = C_{V^\perp} = \mathcal{S}$ for variables. Thus a command string describes a possible path in a formula, and eigenweights define slices in proofs. A context defines a position in the additive structure of a proof: a slice, and a path in formulas.

Definition 3 (Semantics of a prooftree). *Let π be a prooftree of $\vdash \Gamma$ and W be a set of eigenweights, one for each &-rule in π. Let $\mathcal{F}_\pi$ be the set of occurrences of formulas in π. The* ports *of π are the formulas in Γ. For each eigenvalue $\omega : W \to \mathcal{B}$, we define a binary relation $\to_\omega$ on $\mathcal{F} \times \mathcal{S}$. This relation is the union of the contributions of all the links in the proof, each contribution defined in Figure 1. Let $\leftrightarrow^\star_\omega$ be the reflexive transitive closure of $\to_\omega$.*

The context semantics *of π is the* partial *function $[\![\pi]\!] : \Gamma \times \mathcal{S} \times (W \to \mathcal{B}) \to \Gamma \times \mathcal{S}$ such that $[\![\pi]\!](F, s, \omega) = (F', s')$ if and only if $(F, s) \leftrightarrow^\star_\omega (F', s')$.*

$$\frac{}{\vdash A, A^\perp}\,\mathrm{Ax} \qquad (A, s) \to_\omega (A^\perp, s) \qquad (A^\perp, s) \to_\omega (A, s)$$

$$\frac{\vdash \Gamma^1, A \qquad \vdash \Delta^1, A^\perp}{\vdash \Gamma^0, \Delta^0}\,\mathrm{Cut} \qquad \begin{array}{c}(F^0 \in \Gamma^0, s) \to_\omega (F^1 \in \Gamma^1, s)\\ (F^0 \in \Delta^0, s) \to_\omega (F^1 \in \Delta^1, s)\\ (A, s) \to_\omega (A^\perp, s)\\ (A^\perp, s) \to_\omega (A, s)\end{array}$$

$$\frac{\vdash \Gamma^1, A \qquad \vdash \Delta^1, B}{\vdash \Gamma^0, \Delta^0, A \otimes B}\,\otimes \qquad \begin{array}{c}(F^0 \in \Gamma^0, s) \to_\omega (F^1 \in \Gamma^1, s)\\ (F^0 \in \Delta^0, s) \to_\omega (F^1 \in \Delta^1, s)\\ (A \otimes B, l.s) \to_\omega (A, s)\\ (A \otimes B, r.s) \to_\omega (B, s)\end{array}$$

$$\frac{\vdash \Gamma^1, A, B}{\vdash \Gamma^0, A \parr B}\,\parr \qquad \begin{array}{c}(F^0 \in \Gamma^0, s) \to_\omega (F^1 \in \Gamma^1, s)\\ (A \parr B, l.s) \to_\omega (A, s)\\ (A \parr B, r.s) \to_\omega (B, s)\end{array}$$

$$\frac{\vdash \Gamma^1, A \qquad \vdash \Gamma^2, B}{\vdash \Gamma^0, A \,\&\, B}\,\&(w) \qquad \begin{array}{c}\text{if} \quad \omega(w_i) = 0, \quad \text{then:}\\ (F^0 \in \Gamma^0, s) \to_\omega (F^1 \in \Gamma^1, s)\\ (A \& B, g.s) \to_\omega (A, s)\\ \text{if} \quad \omega(w_i) = 1, \quad \text{then:}\\ (F^0 \in \Gamma^0, s) \to_\omega (F^1 \in \Gamma^2, s)\\ (A \& B, d.s) \to_\omega (B, s)\end{array}$$

$$\frac{\vdash \Gamma^1, A}{\vdash \Gamma^0, A \oplus B}\,\oplus_0 \qquad \begin{array}{c}(F^0 \in \Gamma^0, s) \to_\omega\\ (F^1 \in \Gamma^1, s)\\ (A \oplus B, g.s) \to_\omega\\ (A, s)\end{array}$$

Fig. 1. Context semantics of prooftrees

Intuitively, an eigenvalue defines a *slice* in the proof—where *left* or *right* is chosen for each &-rule in the proof. Given ω, the relation $\to_\omega$ defines the paths going up in the proof that is included in this slice. The transitive closure $\to^\star_\omega$ of $\to_\omega$ defines upwards paths in the proof that are contained in the slice defined by ω; a maximal upwards path starts either at an hypothesis or at a cut formula and ends at an axiom formula. The reflexive, symmetric transitive closure $\leftrightarrow^\star_\omega$ of $\to_\omega$ defines all the valid paths in the slice defined by ω. This compositional semantics is easily adapted to the proofnets defined in Section A.

Example 1. Consider the proof π (with the convention that distinct occurrences of a same formula get distinct marks):

$$\frac{\dfrac{\overline{(A^\perp)^2, A^1} \qquad \overline{(A^\perp)^3, A^2}}{(A^\perp)^1, A^0}\,Cut \qquad \dfrac{\overline{(A^\perp)^5, A^3}}{(A^\perp)^4, A \oplus B}\,\oplus_l}{(A^\perp)^0, A \,\&\, (A \oplus B)}\,\&(w)$$

Then, if $\omega_i(w) = i$ (for $i = 0, 1$), we have: $((A^\perp)^0, \diamond) \to_{\omega_0} ((A^\perp)^1, \diamond) \to_{\omega_0} ((A^\perp)^2, \diamond)$, $(A^1, \diamond) \to_{\omega_0} ((A^\perp)^2, \diamond)$, $((A^\perp)^3, \diamond) \to_{\omega_0} (A^2, \diamond)$, and $(A\&(A\oplus B), g.\diamond) \to_{\omega_0} (A^0, \diamond) \to_{\omega_0} (A^2, \diamond)$. The formulas A^1 and $(A^\perp)^3$ are cut together, thus $[\![\pi]\!]((A^\perp)^0, \diamond, \omega_0) = (A\&(A\oplus B), g.\diamond)$. Similarly, we have $[\![\pi]\!](A\&(A \oplus B), d.g.\diamond, \omega_1) = ((A^\perp)^0, \diamond)$.

2 Proofnets

Given a semantics for MALL proofs, we provide a proofnet syntax based on *bus notation* [9], which supports a simpler definition of both semantics and local reduction.

Syntax and semantics: We replace the single wires of the proofnets from Appendix A by *buses* of wires. In a proof with n &-links in bijection with n eigenweight variables $w_0, \ldots, w_{n-1}$, edges of the encoded proofnet are composed of (1) n *weight wires*, one for each variable w_i; and (2) one *command wire* (for the command string). We draw weight wires on the left, and the command wire on the right.

We use three types of ternary nodes: the *multiplicative* nodes, the *additive* nodes and the *weight* nodes. A node has two *auxiliary ports* (above the triangle) and one *principal port* (below the triangle). Multiplicative and additive nodes act on the command wire, and a weight node acts on a specific weight wire (see Figure 2). Each proofnet edge is labeled by a formula. A wire ends at a port of another node, a proofnet port, or a plug.

Definition 4 (Proofnets with bus-notation). *The recursive encoding of a prooftree π into a proofnet $\langle\pi\rangle$ is done according to the rules in Figure 3.*

We henceforth discuss proofnets using this bus notation. Note the *lamination* of $\langle\pi_i\rangle$ with the eigenweights of π_{1-i} in rules Cut, $\otimes$, and $\parr$—weight wires from $\langle\pi_{1-i}\rangle$ are added to $\langle\pi_i\rangle$, but do nothing.

Context semantics for proofnets is similar to that for prooftrees. Given an eigenvalue ω, we replace the relation $\to_\omega$ by the relation $\hookrightarrow_\omega$ on $\mathcal{E}\times\mathcal{S}$, where a formula occurrence $\phi \in \mathcal{F}$ in the prooftree is represented by a proofnet edge $e_\phi \in \mathcal{E}$ (we write $\textsc{p}$, $\textsc{l}$, $\textsc{r}$ for the principal, and left and right auxiliary ports of a node):

Additive node: $(\textsc{p}, g.s) \hookrightarrow_\omega (\textsc{l}, s)$ and $(\textsc{p}, d.s) \hookrightarrow_\omega (\textsc{r}, s)$.
Multiplicative node: $(\textsc{p}, l.s) \hookrightarrow_\omega (\textsc{l}, s)$ and $(\textsc{p}, r.s) \hookrightarrow_\omega (\textsc{r}, s)$.
Weight node: $\omega(w) = 0 \Rightarrow (\textsc{p}, s) \hookrightarrow_\omega (\textsc{l}, s)$ and $\omega(w) = 1 \Rightarrow (\textsc{p}, s) \hookrightarrow_\omega (\textsc{r}, s)$.

Nodes in proofnets act on contexts as *routers*: in this sense we can say that proofnets are a low-level encoding of context semantics.

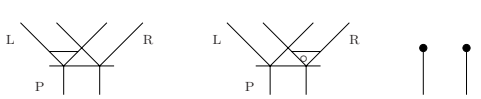


Fig. 2. Weight node, command node (○), plug

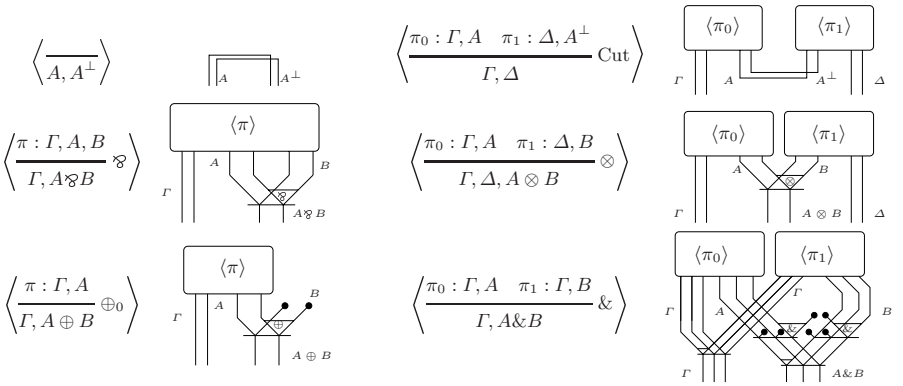
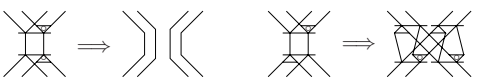
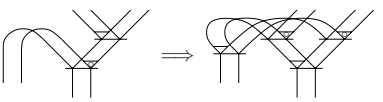


Fig. 3. Proofnets with bus notation

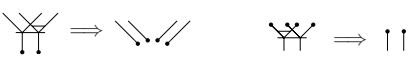
Local reduction rules: We use the *local, semantics preserving* rules of [9]–an advantage of our proofnet syntax—plus some extra ones. Assume only one eigenweight variable in the figures—more variables (and the full suite of rules) can be deduced from those that are presented. *Cut-rules:* These make two nodes interact on their principal ports. If they are on the same wire and of the same type they disappear (this will be the case, for instance, when an immediate cut will be reduced), else they duplicate each other. This rule will be used primarily for box copying.



η-rule: This corresponds to the duplication of a node by a weight node (it plays a part in duplication of boxes and in η-expansion of proofs).



Plugs: Their occurrence means nodes might be useless. Thus we have *garbage collection* rules for deleting such nodes and propagate plugs.



3 Cut Elimination

Cut elimination needs to preserve semantics. Consider first the case of *multiplicative cuts*, the easiest:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A} \quad \dfrac{\pi_1}{\Delta, B}}{\Gamma, \Delta, A \otimes B}\otimes \quad \dfrac{\dfrac{\pi_2}{\Pi, A^\perp, B^\perp}}{\Pi, A^\perp \parr B^\perp}\parr}{\Gamma, \Delta, \Pi}\,Cut \quad \Longrightarrow \quad \dfrac{\dfrac{\pi_0}{\Gamma, A} \quad \dfrac{\dfrac{\pi_1}{\Delta, B} \quad \dfrac{\pi_2}{\Pi, A^\perp, B^\perp}}{\Delta, \Pi, A^\perp}\,Cut}{\Gamma, \Delta, \Pi}\,Cut$$

In the encoded proofnet, we have two multiplicative nodes facing each other on their principal ports, so the cut rule can be applied and make these two nodes disappear. The resulting proofnet is the encoding of the prooftree obtained by eliminating the cut in the above prooftree (i.e., with two cuts on A and B).

The case of *additive cuts* is a little more complicated:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A} \quad \dfrac{\pi_1}{\Gamma, B}}{\Gamma, A \,\&\, B}\,\&(w) \quad \dfrac{\dfrac{\pi_2}{\Delta, A^\perp}}{\Delta, A^\perp \oplus B^\perp}\,\oplus_0}{\Gamma, \Delta}\,Cut \quad \Longrightarrow \quad \dfrac{\dfrac{\pi_0}{\Gamma, A} \quad \dfrac{\pi_2}{\Delta, A^\perp}}{\Gamma, \Delta}\,Cut$$

This step can be handled on the encoded proofnet—but in several stages: duplication of the additive node corresponding to the $\oplus$-link and of the proof $\langle \pi_2 \rangle$ by a w-weight node, annihilation of the two resulting $\oplus$-nodes by the &-nodes (Cut rule), garbage collection of the proof $\langle \pi_1 \rangle$ and of the right duplicate of $\langle \pi_2 \rangle$ and then of the w-weight nodes.

This is not completely satisfactory because garbage collection modifies the semantics (by selecting its "meaningful" part). In the following, we will clearly distinguish the cut-elimination step and the garbage collection step: the first one preserves the semantics while the second selects the good part. In order to do so, we will have to introduce *garbage* explicitly in the proofs (i.e., parts that would disappear in the usual MALL prooftrees) and to remove it after the normalization. The partial additive cut-elimination step is described in Figure 4. This approach is related to the $\flat$-rules of [6, 15], which we discuss below.

Finally, we consider the difficult case of *additive cuts on auxiliary formulas.* Given cut formulas $F, F^\perp$, if F is auxiliary and $F^\perp$ is principal , then the proof of $F^\perp$ is just copied (in the same slice!). But if *both* are auxiliary, we have a proof

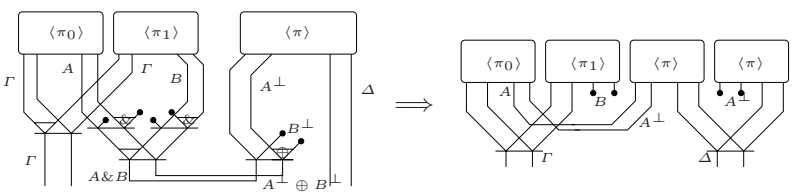


Fig. 4. Additive cut-elimination, generation of garbage

like:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A, F} \quad \dfrac{\pi_1}{\Gamma, B, F}}{\Gamma, A \,\&B, F}\,\&(w_0) \qquad \dfrac{\dfrac{\pi_2}{\Delta, C, F^\perp} \quad \dfrac{\pi_3}{\Delta, D, F^\perp}}{\Delta, C\&D, F^\perp}\,\&(w_1)}{\Gamma, \Delta, A \,\&B, C\&D}\,Cut$$

This cut can be reduced in different ways, copying *either* side of the proof. The same happens with the encoded proofnet. Proofnets here fail to avoid useless sequentializations: the two rewritings are symmetric, and there is no reason to choose one instead of the other.

Our solution is to merge the *boxes* corresponding to each &-link into a *two dimensional box*. Add a rule in the prooftree syntax corresponding to n &-links in parallel; proofnet syntax then has boxes with multiple &-nodes. An n-dimensional &-link (or "higher order box") is encoded like a "normal" &-link: at each port of the subnet encoding the box, we have a tree of weight nodes with 2^n (instead of 2) leaves. The order of the weight nodes is arbitrary, since the η-rule can permute them. A cut involving auxiliary ports of two boxes is shown in Figure 5. Dually, we introduce *multidimensionality* in MALL *proofs:*

Definition 5 (Generalized &-rule). *The* $\&(n, k, g_0)$-rule *(where* $n, k \in \mathbb{N}$*, and* $g_0 \in \mathcal{B}^k$*) has a conclusion:* $\Gamma, A_0^1 \& A_1^1, \ldots, A_0^n \& A_1^n$*, and* 2^{n+k} *hypotheses* $\pi(b)$ *where* $b \in \mathcal{B}^{n+k}$*, including (1)* 2^n *proofs* $\pi(b, g_0) : \Gamma, A_{b_1}^1, \ldots, A_{b_n}^n$ *where* $b \in \mathcal{B}^n$*, and (2)* $2^n(2^k - 1)$ *garbage proofs* $\pi(b, g) : \Gamma, A_{b_1}^1, \ldots, A_{b_n}^n, \bullet$ *where* $b \in \mathcal{B}^n$ *and* $g \in \mathcal{B}^k \setminus \{g_0\}$*.*

The $\bullet$ symbol marks garbage. These *plugs* guarantee that the garbage is disconnected; later we will see that detecting this disconnectedness in the semantics is decidable, which facilitates read-back. Moreover, some rules will be useful to introduce and handle garbage:

$$\dfrac{\Gamma, X \quad \Delta, Y}{\Gamma, \Delta, \bullet}\,\mathbf{G} \qquad \dfrac{\Gamma, \bullet, \bullet}{\Gamma, \bullet}\,\bullet$$

The "usual" &-rule now corresponds to the $\&(1, 0)$-rule (one principal conclusion $A\&B$, no garbage, and two hypotheses). The problem of adding garbage to

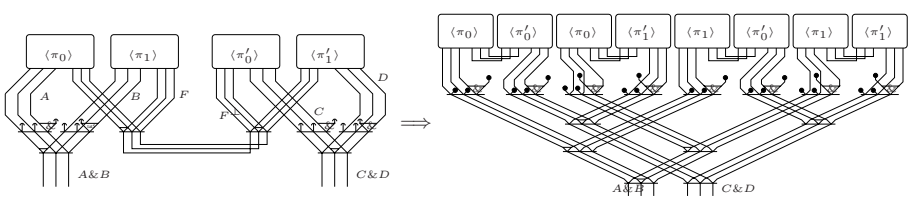


Fig. 5. Generation of a higher dimensional box (of dimension 2)

prooftrees occurs during elimination of an immediate additive cut, which we instead describe as:

$$\dfrac{\dfrac{\dfrac{\pi_0}{\Gamma, A_0} \quad \dfrac{\pi_1}{\Gamma, A_1}}{\Gamma, A_0 \& A_1}\,\&(1,0)(w) \quad \dfrac{\dfrac{\pi_2}{\Delta, A_0^\perp}}{\Delta, A_0^\perp \oplus A_1^\perp}\,\oplus_0}{\Gamma, \Delta}\,Cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\pi_0}{\Gamma, A_0} \quad \dfrac{\pi_2}{\Delta, A_0^\perp}}{\Gamma, \Delta}\,Cut \quad \dfrac{\dfrac{\pi_1}{\Gamma, A_1} \quad \dfrac{\pi_2}{\Delta, A_0^\perp}}{\Gamma, \Delta, \bullet}\,\mathbf{G}$$

$$\dfrac{\Gamma, \Delta \qquad \Gamma, \Delta, \bullet}{\Gamma, \Delta}\,\&(0,1,(w \mapsto 0))$$

More generally, a $\&(n,0)$-rule corresponds to the parallelization of n &-links. The cut-elimination on extended prooftrees is quite similar to the cut-elimination on usual prooftrees. The generalized &-link behaves as follows: (1) a $\&(n_0, k_0, g_0)$-link cut against a $\&(n_1, k_1, g_1)$-link on auxiliary ports results in a $\&(n_0+n_1, k_0+k_1, (g_0, g_1))$-link— for instance, in Fig. 5, two $\&(1,0)$-links are cut against each other and we get a $\&(2,0)$-link; (2) a $\&(n,k,g)$-link cut on one of its principal ports F against a $\oplus_i$-link results in a $\&(n-1, k+1, g')$-link, where g' is obtained from g by assigning i to the eigenweight corresponding to the principal port F.

An extended MALL proof can be translated back to a MALL proof (erasure of the garbage in the proof), so extended MALL is as expressive as MALL.

Theorem 1 (Cut-elimination on proofnets). *Let π_0 be an extended MALL prooftree. If π_0 can be reduced to the prooftree π_1, then $\langle \pi_0 \rangle \longrightarrow^\star_{Cut+\eta} \langle \pi_1 \rangle$. If π_1 is normal, $\langle \pi_1 \rangle$ is also normal (there is no local cut-redex in $\langle \pi_1 \rangle$).*

Theorem 2 (Correctness). *Using the notation of Theorem 1, $[\![\pi_0]\!] = [\![\pi_1]\!]$. Or equivalently, $[\![\langle \pi_0 \rangle]\!] = [\![\langle \pi_1 \rangle]\!]$.*

Garbage collection does the same work on the prooftree and on the proofnet. In both cases it does modify the semantics. These theorems can be summarized by the diagram:

$$\begin{tikzcd}
\pi_0 \arrow[r, "Cut"] \arrow[d] & \pi_1 \arrow[r, "GC"] \arrow[d] & \pi_2 \arrow[d] \\
\langle \pi_0 \rangle \arrow[r, "Cut+\eta"] \arrow[d] & \langle \pi_1 \rangle \arrow[r, "GC"] \arrow[d] & \langle \pi_2 \rangle \\
[\![\pi_0]\!] \arrow[r, equal] & [\![\pi_1]\!] &
\end{tikzcd}$$

4 Read-Back and Completeness

Read-back consists of building a cut-free proof from a valid semantics, deriving normalized proofs without normalizing.

Theorem 3 (Definition of read-back). *There exists an algorithm $\mathcal{R}$ that inputs the context semantics S of a proof π_0 and outputs a normal proofnet π_1 such that $[\![\pi_1]\!] = S$.*

We output a proofnet instead of a prooftree only to elide unnecessary sequentializations. The proof is constructive: a top-down algorithm that determines what the external structure (i.e., close to the ports) of the proofnet is, and then recursively reapplies itself on subcomponents of the initial semantics. Briefly, readback is decidable because the semantics is finitely representable.

The three main steps of the recursive deconstruction are: (1) determination of the meaningful slice of the outermost boxes (and at the same time, removing garbage of outermost boxes); (2) determination of the paths to the principal ports of the outermost boxes and of the structure of the normal proofnet outside the outermost boxes; and (3) re-application of the algorithm to each slice of the outermost boxes, after having eliminated the totally useless eigenweights for each component.

Outermost boxes and garbage:

Definition 6 (ω-slice).

*Let π a be proof and ω an eigenvalue. We write $p \in \mathcal{P}$ for the ports of π. For each command string s, we write $|s| = s \setminus \{g, d\}$ for the command string containing only multiplicative information. The ω-*slice *$[\![\pi]\!]_\omega$ of π is defined by $[\![\pi]\!](p, s) = (p', s') \iff [\![\pi]\!]_\omega(p, |s|) = (p', |s'|)$.*

Proposition 1. *Given the semantics of a proofnet coding a MALL proof in normal form, each ω-slice gives the context semantics of a normalized MLL proofnet.*

Proposition 2. *Two different boxes that branch on the same eigenweight cannot be in the same slice—an important case being at the top level. As a consequence, if a box B is* top-level *(i.e., not contained in any other box), there is only one copy of B in the proofnet.*

Box absorption causes copying of &-boxes into different slices, so we specify *occurrences* of boxes by choosing values for some subset of eigenweights, as well as a path of $\otimes, \parr, \&$ and $\oplus_i$ nodes leading to the box.

If $[\![\pi]\!]$ is the context semantics of a top-level $\&(n, k)$-box occurrence, some eigenweights $w_1, \ldots, w_n$ are "good" eigenweights and give useful slices of the box; the other eigenweights $w_{n+1}, \ldots, w_{n+k}$ are "bad" eigenweights and indicate garbage. How can we tell the good from the bad?

Lemma 1. *Let $[\![\pi]\!]$ be the context semantics of a* top-level *$\&(n, k)$-box occurrence with eigenweights W. Then p is a bad eigenweight if there exists a (bad) setting of p to 0 or 1 such that, for any setting of eigenweights $W - p$, the defined ω-slice does not give a MLL proofnet.*

That the box be top-level is important in the above argument: two occurrences of the same box (in different slices, by Proposition 2), may be cut against $\oplus_0$ in one case and $\oplus_1$ in the other, so there is no unique bad setting to the eigenweight.

Example 2. As an example of this situation, consider

$$\dfrac{\dfrac{\dfrac{C, \Gamma, A^\perp}{C, \Gamma, A^\perp \oplus B^\perp}\oplus_0 \quad \dfrac{D, \Gamma, B^\perp}{D, \Gamma, A^\perp \oplus B^\perp}\oplus_1}{C \&_p D, \Gamma, A^\perp \oplus B^\perp}\&(p) \quad \dfrac{\Delta, A \quad \Delta, B}{\Delta, A \&_q B}\&(q)}{C \&_p D, \Gamma, \Delta}\text{Cut}$$

where we annotate the implicit boxes with eigenvariables: in the normal form, p may be at top level, but q is not. While q is a "bad" eigenweight in the semantics, we cannot tell whether $q = 0$ or $q = 1$ results in garbage: in slice $p = 0$, $q = 1$ gives garbage, and in slice $p = 1$, $q = 0$ gives garbage. This is, essentially, why all garbage cannot be determined at the top level of the read-back algorithm.

By iterating the use of Lemma 1, we can detect each of the k bad eigenweights and its bad setting, and then project out the good part of the semantics that does not involve garbage.

Definition 7 (projection). *Let $[\![\pi]\!]$ be the context semantics of a top-level $\&(n, k, g_0)$-box with bad eigenweights $w_1, \ldots, w_n$; the only assignment to $w_1, \ldots, w_n$ that does not lead to garbage is $g_0(w_1), \ldots g_0(w_n)$. Then the* projection *of $[\![\pi]\!]$ is obtained by restricting it to eigenvalues g such that $\forall i, g(w_i) = g_0(w_i)$.*

The first step of the algorithm consists in determining the garbage-eigenweights of the outermost box together with their good setting (i.e., eliminate garbage) as described above. Observe that Proposition 2 assures us that top-level boxes have not been copied: when this condition fails, we may not be able to recover garbage immediately.

External structure of the proofnet: We now determine the structure of the proofnet that is external to the outermost box. This is done in two steps: (1) localize the &-links that correspond to the principal ports of the outermost boxes, and (2) recover the proofnet structure external to these boxes via a kind of projection. This is done by considering paths in the main formulas that end at a &-node.

Identifying paths to principal ports of outermost boxes: Let F be a formula at a port and $\underline{\&}$ an occurrence of a &-connective in F. We call $\underline{\&}$ *primary* in F if either $F = A \ \underline{\&} B$, or $F = A \ \circ B$ ($\circ \in \{\otimes, \parr, \oplus\}$) and $\underline{\&}$ is primary in A or B. How do we determine if a primary occurrence is a port of an outermost box?

Definition 8 (&-path). *A command string s codes a* &-path *to a primary &-connective of a formula F if one of the following is satisfied: (1) $F = A \ \& B$ and $s = \diamond$; (2) $F = A \ \oplus B$ and $s = g.s'$ and s' codes a &-path of A; (3) $F = A \ \oplus B$ and $s = d.s'$ and s' codes a &-path of B; (4) $F = A \ \otimes B$ or $F = A \parr B$ and $s = l.s'$ and s' codes a &-path of A; (5) $F = A \ \otimes B$ or $F = A \parr B$ and $s = r.s'$ and s' codes a &-path of B.*

Proposition 3. *A box with command string s contains different slices determined by eigenweight w if there exists a port p and command string s coding a &-path to a primary &-link, such that for any command strings s_0, s_1 and eigenvalue $\omega : W \mapsto \mathcal{B}$, we have $[\![\pi]\!](p, s.g.s_0, \omega) = (p', c') \Longrightarrow \omega(w) = 0$ and $[\![\pi]\!](p, s.d.s_1, \omega) = (p', c') \Longrightarrow \omega(w) = 1$.*

Informally, Proposition 3 just says that w is a good eigenweight if whenever we take a path from a proofnet port to a primary &-node, w is always 0 when we go left and 1 when we go right. (Recall that the branching at a &-node is done both by the associated eigenweights *and* by the command string.)

If a top-level box has a &-connective at its port, the connective is primary, though every primary &-connective is not necessarily at the port of a top-level box.

Recovering proofnet structure external to outermost boxes: The identification of primary &-connectives and the &-paths to them uncovers a forest of trees, where each tree is located at a different external port of the proofnet, and constructed from the &-paths. By simultaneously examining the logical formula at a port, we can also recover the logical connectives along the path. Binary links in the trees only occur at $\otimes$ and $\parr$ nodes.

Definition 9 ($\circ$-removal). *Let p be a port with formula $A \circ B$ ($\circ \in \{\otimes, \parr\}$). A* $\circ$-removal *is a modification of the semantics that splits p into ports p' with formula A and p'' with formula B. (An $\oplus_i$-removal is defined similarly but p is then just replaced by another port instead of two.)*

Definition 10 (partition). *A* partition *is a minimal sequence of removals of nodes $n_1, \ldots, n_k$ where n_i ($i < k$) are $\parr$ or $\oplus_i$ nodes and n_k is a $\otimes$ node, where after the successive removals of $n_1, \ldots, n_{k-1}$, the removal of n_k divides the semantics into two disjoint sets S and S' of paths, such that the set of ports referenced in S are disjoint from those referenced in S'.*

Lemma 2. *If the semantics has a void partition, there is at most one top-level box, and every primary &-node is a port.*

Our read-back procedure iterates the search for partitions. It keeps track of the nodes removed from each partition which makes the reconstruction of the proofnet possible at the end. Each *component* of the partition is guaranteed to have at most one top-level box.

Completely useless eigenweights: The last step divided the proofnet (the semantics) into several components. Each component corresponds to a top-level box, or is empty and then will not be considered any more. Before reapplying the algorithm on a slice of one component, it is useful to get rid of useless eigenweights corresponding to boxes of the other components.

An eigenweight w is *totally useless* if and only if the paths in the proofnet do not depend on it, which can be decided looking at the semantics $[\![\pi]\!]$ of a component, as it is equivalent to: $\forall F_0, F_1, s_0, s_1, \forall \omega. [\![\pi]\!](F_0, s_0, \omega \mid_{w=0}) =$

$[\![\pi]\!](F_1, s_1) \iff [\![\pi]\!](F_0, s_0, \omega\mid_{w=1}) = [\![\pi]\!](F_1, s_1)$. This deconstruction step corresponds in the reconstruction stage to the *re-lamination* of the components of the proofnets once they have been re-computed from their semantics.

Correctness of read-back and consequences:

Theorem 4 (Correctness of read-back). *If π_0 is a MALL proofnet, and read-back applied to $[\![\pi_0]\!]$ outputs a proofnet π_2, then $\pi_0 \rightarrow_{Cut+\eta} \pi_1 \rightarrow_{GC+Cut+\eta} \pi_2$, where the first arrow corresponds to normalization and the second to full η-expansion (i.e., nodes connected to auxiliary ports of boxes are absorbed) and to garbage collection.*

The *η-expansion* mentioned above comes from the fact that the context semantics cannot distinguish proofs that are equivalent modulo the absorption and the duplication of a link by a box. This property of the semantics is absolutely essential to ensure correctness of the semantics with respect to the reduction, since reduction of non immediate cuts involves absorptions and duplications.

Example 3 (η-equivalence of prooftrees). Here are two different proofs with the same context semantics:

$$\dfrac{\dfrac{\dfrac{}{F, F^\perp} \quad \dfrac{}{F, F^\perp}}{F, F^\perp \& F^\perp}\,\&(w)}{F \oplus G, F^\perp \& F^\perp}\,\oplus_0 \qquad \dfrac{\dfrac{\dfrac{}{F, F^\perp}}{F \oplus G, F^\perp}\,\oplus_0 \quad \dfrac{\dfrac{}{F, F^\perp}}{F \oplus G, F^\perp}\,\oplus_0}{F \oplus G, F^\perp \& F^\perp}\,\&(w)$$

We note $\omega_i(w) = i$. The semantics S of the two proofs above is defined by: $S(F \oplus G, g.s, \omega_0) = (F\&G, g.s)$, $S(F\&G, g.s, \omega_0) = (F \oplus G, g.s)$, $S(F \oplus G, g.s, \omega_1) = (F\&G, d.s)$, and $S(F\&G, d.s, \omega_1) = (F \oplus G, g.s)$.

Theorem 3 also means that context semantics characterizes what a MALL proof is. Indeed, if Γ are formulas, W a set of eigenweights and $S \in \Gamma \times \mathcal{S} \times (W \rightarrow \mathcal{B}) \rightarrow \Gamma \times \mathcal{S}$ a finitely representable function, then either $\mathcal{R}(S)$ is a normalized proofnet π such that $[\![\pi]\!] = S$, or $\mathcal{R}(S)$ is undefined. In the second case, S is not the semantics of any proofnet: on the contrary there would exist a proofnet π' such that $[\![\pi']\!] = S$ and by Theorem 4, $\pi' \rightarrow_{Cut+\eta+GC} \pi$. Therefore, Theorems 3 and 4 relate a form of *full completeness* of the context semantics; they can be summarized by the diagram:

$$\begin{tikzcd}
\pi_0 \arrow[r, "Cut"] \arrow[d] & \pi_1 \arrow[r] \arrow[d] & \pi_2 \arrow[d] \\
\langle \pi_0 \rangle \arrow[r, "Cut+\eta"] \arrow[dr] & \langle \pi_1 \rangle \arrow[r, "Cut+GC+\eta"] \arrow[d] & \langle \pi_2 \rangle \\
 & [\![\pi_0]\!] \arrow[ur, bend right, "\mathcal{R}"'] &
\end{tikzcd}$$

5 Conclusions, Related and Future Work

Girard [8] provides a syntax with weights, a sequentialization procedure and a cut-elimination procedure restricted to so called *ready cuts*, i.e., cuts that are

not in boxes. Tortora de Falco provided a more complete study of the reduction of the proofnets in [20]. His syntax involves boxes, and the problems he encounters are related to ours. He proves a restricted confluence property that should be extendable to our settings without any problem. Following our work, we discovered that the notion of multiboxes appears in a manuscript of Tortora De Falco [19]. Our notion of generalized boxes is in the same spirit as his multiboxes; however, we have integrated a reduction-preserving semantics akin to Laurent [15], together with a more generalized notion of garbage that preserves the linearity of additive cut-elimination.

We designed proofnets with bus notation to encode the context semantics precisely and locally, allowing a fine study of the reduction process. Then we could see exactly which step endangers the preservation of the semantics under reduction, and postpone it. We can then discuss *optimal reduction* of additive proofnets.

On the semantics point of view, our work is related to the Geometry of Interaction and to all its simplified versions [6, 10, 15]. The semantics exposed in this paper is quite close to the token machine of [15]. In this setting, Laurent essentially proves correctness of the semantics with respect to prooftree normalization. Our proofnet syntax gives a sort of low-level implementation of this semantics. We extend the introduction of garbage (corresponding to $\flat$-rules in [15] and [6]) to the generalized &-connector, and hence to multidimensional garbage.

In proving a correspondence between normalization of sequents and proofnets, we modified the rules for MALL, introducing both *garbage* and *parallelization*. Read-back involved the detection of garbage and parallelization. The existence of read-back corresponds to a form of completeness of the semantics. The η-equivalence and the choice of a η-expanded form are the price for this nice property.

Among the continuations of this work, the first one is its extension to full Linear Logic. The case of the units should not be too hard. The extension to the exponentials is probably more challenging since the !-rule acts on a large bunch of formulas (by checking that all these formulas are of the form $?F$): This might be problematic, especially to get a local (optimal) reduction.

Last, the correctness of read back expresses that the context semantics enjoys some completeness property. The read-back algorithm could probably be reformulated in the game semantics framework, which might be the starting point to some comparisons between the concrete insight given by the context semantics and the concurrent games constructions of Abramsky and Melliès [2, 1].

Acknowledgments. We wish to thank J. Feret, A. Miné and to an anonymous referee for their helpful comments on a preliminary version of this paper.

References

[1] S. Abramsky and P.-A. Melliès. Concurrent games and full completeness. In *LICS'99*, pages 431–442. IEEE, July 1999. 163

[2] Samson Abramsky and Guy McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (extended abstract). In *Proceedings of 1996 Workshop on Linear Logic*, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996. 163

[3] G. Bellin and P. J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, December 1994. 151

[4] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987. 151, 165

[5] Jean-Yves Girard. Geometry of interaction I: Interpretation of system F. In *Logic Colloquium '88*, pages 221–260. North-Holland, 1989. 151, 166

[6] Jean-Yves Girard. Geometry of interaction III: The general case. In *Advances in Linear Logic*, pages 329–389. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca. 151, 156, 163, 166

[7] Jean-Yves Girard. Linear logic: its syntax and semantics. In *Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca. 151, 165

[8] Jean-Yves Girard. Proof-nets: The parallel syntax for proof-theory. In *Logic and Algebra*. Marcel Dekker, 1996. 151, 162, 166

[9] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optmnal lambda reduction. In *POPL'92*, pages 15–26, Albuquerque, January 1992. ACM Press. 151, 154, 155, 166

[10] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. Linear logic without boxes. In *LICS'92*, pages 223–34. IEEE, Los Alamitos, 1992. 151, 163, 166

[11] Timothy G. Griffin. The formulae-as-types notion of control. In *POPL'90*, pages 47–57. ACM Press, New York, 1990. 151

[12] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1985. & 0-13-153289-8. 151

[13] Y. Lafont. From proof nets to interaction nets. In *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995. Proceedings of the 1993 Workshop on Linear Logic, Cornell Univesity, Ithaca. 151, 166

[14] John Lamping. An algorithm for optimal lambda-calculus reductions. In *POPL'90*, pages 16–30. ACM Press, January 1990. 152

[15] Olivier Laurent. A token machine for full geometry of interaction (extended abstract). In *TLCA'01*, volume 2044, pages 283–297. LNCS, Springer-Verlag, May 2001. 156, 163, 166

[16] Julia L. Lawall and Harry G. Mairson. Sharing continuations: proofnets for languages with explicit control. In *ESOP'2000*, volume 1782. LNCS, Springer-Verlag, 2000. 151

[17] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999. 151

[18] Chetan R. Murthy. Extracting constructive content from classical proofs. Technical Report TR90-1151, Cornell University, Computer Science Department, August 1990. 151

[19] Lorenzo Tortora de Falco. The additive multiboxes. *Annals of Pure and Applied Logic*. To appear. 163

[20] Lorenzo Tortora de Falco. Additives of linear logic and normalization- part 1: a (restricted) church-rosser property. *Theoretical Computer Science*. 163

$$\frac{}{\vdash A, A^\perp}\,\mathrm{Ax} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}\,\mathrm{Cut} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\,\otimes$$

$$\frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\,\parr \qquad \frac{\vdash \Gamma, A \quad \vdash \Gamma, B}{\vdash \Gamma, A \,\&\, B}\,\& \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, A \oplus B}\,\oplus_0 \qquad \frac{\vdash \Gamma, B}{\vdash \Gamma, A \oplus B}\,\oplus_1$$

Fig. 6. The rules of the Multiplicative and additive fragment

A MALL: Proofs, Nets, Reduction

Definition 11 (Formula). MALL formulas *are generated from the grammar* $F \longrightarrow V \mid V^\perp \mid F \otimes F \mid F \parr F \mid F \& F \mid F \oplus F$ *where* V *ranges over variables,* $\otimes$ *and* $\parr$ *(resp.,* $\&$ *and* $\oplus$*) are the conjunction and disjunction of the multiplicative (additive) component, and* $(-)^\perp$ *is the involutive negation on literals.*

Atomic negation is extended to a defined involutive connector using the De Morgan identities $(A \otimes B)^\perp = A^\perp \parr B^\perp$ and $(A \& B)^\perp = A^\perp \oplus B^\perp$. We use right-handed *sequents*—multisets of formulas $\vdash F_0, \ldots, F_{n-1}$—where all sequent formulas play the same role. A well-known interpretation of the connectives is *economic* [7]. Negation represents need, and involution ($A^{\perp\perp} = A$) means if you need to need, you have; $\vdash \Gamma, A^\perp$ means you need (a proof of) A to produce (a proof of) Γ. Figure 6 gives the MALL rules: Ax and Cut are the identity rules. The rules $\otimes$ and $\parr$ (resp. $\&$, $\oplus_0$ and $\oplus_1$) form the multiplicative (resp. additive) fragment. Note that in these rules, we need both $\Gamma^\perp$ and $\Delta^\perp$ to produce $A \otimes B$, but only $\Gamma^\perp$ to produce $A \& B$.

Definition 12 (Prooftree). *A* prooftree *(or* MALL-prooftree*) is a tree whose leaves are sequents, linked by the rules showed in Figure 6.*

There is exactly one *introduction* rule for each additive or multiplicative connector (except $\oplus$). The *principal formula* of a link is the new formula introduced, and the other formulas are *auxiliary*. The *cut formulas* of a cut-link are the two hypotheses that are *eliminated* in the conclusion (A and $A^\perp$ in Figure 6). An *immediate cut* is a cut link whose cut formulas are the principal formulas of the two links above the cut. The *ports* of a prooftree are the formulas in the final proof link. Since full linear logic has a cut-elimination procedure, so does MALL (see [4]):

Theorem 5 (Cut-elimination). *There exists an algorithm which inputs a prooftree* π *of MALL sequent* S*, and outputs a prooftree* π' *of the same sequent* S *without any occurrence of the* Cut*-rule.*

Cut-elimination for MALL is described by a collection of local rewriting rules that push Cut-links upwards and make them disappear; these rules appear in the proof of Theorem 5. The rules are not confluent, partly because the prooftree syntax introduces unnecessary sequentializations. For instance, if we start with the proof:

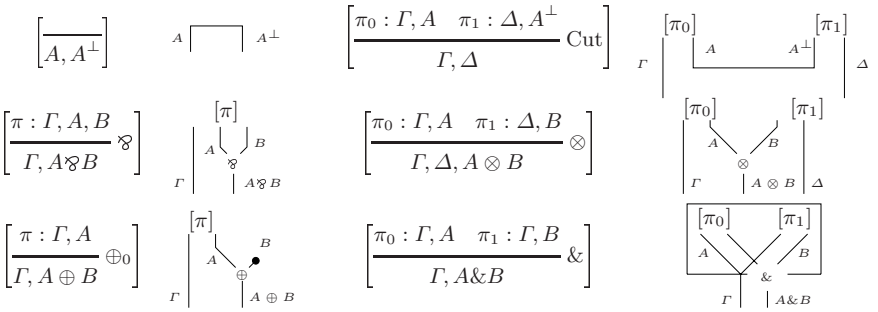


Fig. 7. Proofnets with boxes

$$\dfrac{\dfrac{\dfrac{\pi_0}{\vdash \Gamma, A, B, F}}{\vdash \Gamma, A \parr B, F}\ \parr \qquad \dfrac{\dfrac{\pi_1}{\vdash \Delta, C, F^\perp}}{\vdash \Delta, C \oplus D, F^\perp}\ \oplus_0}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\ Cut$$

then we can rewrite it to either of the following:

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash \Gamma, \Delta, A, B, C}}{\vdash \Gamma, \Delta, A \parr B, C}\ \parr}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\ \oplus_0 \qquad \dfrac{\dfrac{\dfrac{\vdots}{\vdash \Gamma, \Delta, A, B, C}}{\vdash \Gamma, \Delta, A, B, C \oplus D}\ \oplus_0}{\vdash \Gamma, \Delta, A \parr B, C \oplus D}\ \parr$$

Proofnets [8, 13] eliminate such useless sequentializations—the two proofs above have the same meaning, and the semantics should not distinguish them.

The &-connector is unique in its problematic *additive sharing* of Γ in the &-rule (see Figure 6). This non-linear phenomenon is represented in proofnet syntax either by drawing a *box* around the two subproofs (becoming the left and the right side of the box) above a &-link, or by adding a Boolean *eigenweight* to all the formulas in the proof. In the latter, the formulas on the left and right sides of the &-link get opposite boolean values, making their distinction possible. Each side is called a *slice*. These two approaches (boxes and weights) are equivalent.

Definition 13 (Proofnets with boxes). *The inductive encoding* $[.]$ *of prooftrees into* proofnets *is shown in Figure 7. A* port *of the proofnet* $[\pi]$ *is a proofnet* wire *corresponding to a port of* π*.*

Cut elimination annihilates reciprocal links (& and $\oplus$, or $\otimes$ and $\parr$). *Geometry of Interaction* [5, 6] provides a mathematical framework for this phenomenon, where a semantics (see [9, 10, 15]) is defined that is preserved by reduction.

A Tag-Frame System of Resource Management for Proof Search in Linear-Logic Programming

Joshua S. Hodas[1], Pablo López[2⋆], Jeffrey Polakow[1], Lubomira Stoilova[1⋆⋆], and Ernesto Pimentel[2]

[1] Department of Computer Science, Harvey Mudd College
Claremont, CA 91711, USA
{hodas,jpolakow,lstoilova}@cs.hmc.edu
http://www.cs.hmc.edu/~hodas

[2] Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos. 29071 Málaga. España
{lopez,ernesto}@lcc.uma.es
http://www.lcc.uma.es/~[lopez,ernesto]

Abstract. In programming languages based on linear logic, the program can grow and shrink in a nearly arbitrary manner over the course of execution. Since the introduction of the I/O model of proof search [11, 12], a number of refinements have been proposed with the intention of reducing its degree of non-determinism [3, 4, 12, 13, 14]. Unfortunately each of these systems has had some limitations. In particular, while the *resource management* systems of Cervesato et al. [3, 4] and the *frame* system of López and Pimentel [14] obtained the greatest degree of determinism, they required global operations on the set of clauses which were suitable only for interpreter-based implementations. In contrast the *level-tags* system of Hodas, et al. relied only on relabeling tags attached to individual formulas, and was hence appropriate as the specification of an abstract machine. However it retained more non-determinism than the resource management systems. This led to a divergence in the operational semantics of the interpreted and compiled versions of the language Lolli. In this paper we propose a *tag-frame* system which recaptures the behavior of the resource management systems, while being appropriate as a foundation of a compiled implementation.

1 Introduction

In ordinary, pure logic programs, the program is flat, with all clauses available at all times. In languages with intuitionistic implications in goals, such as the

⋆ López and Pimentel were supported in part by the project TIC2001-2705-C03-02 funded by the Spanish Ministry of Science and Technology. During the summer of 2001, López was also supported in part by a travel grant from Harvey Mudd College.

⋆⋆ Stoilova was supported in part by a grant from the Harvey Mudd College Computer Science Clinic Program.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 167–182, 2002.
© Springer-Verlag Berlin Heidelberg 2002

hereditary Harrop formulas of λ-Prolog, some management of clauses is necessary as program execution moves through such a goal. This is because an implicational goal causes a formula to be added temporarily to the program, in a manner similar to a constrained use of `assert`/`retract`. Since the program grows and shrinks as a stack, however, the bookkeeping is straightforward. In contrast, in languages based on linear logic, such as Lolli [10, 11, 12] and Lygon [5, 6, 8, 9], because there are restrictions on the number of times an assumption can be used in a proof, the context can grow and shrink in a nearly arbitrary manner over the course of execution.

Hodas and Miller introduced the I/O model of proof search in order to deal with the most serious source of non-determinism in the management of program clauses during search [11, 12]. Since that time a number of refinements of that system have been proposed with the intention of reducing or eliminating other sources of non-determinism [3, 4, 9, 12, 13, 14]. Unfortunately, each of these systems has had some limitations. For example, the *resource management* systems of Cervesato, et al. [3, 4], and the *frame* system of López and Pimentel [14] obtain the greatest degree of determinism. However, they require global operations on the active set of clauses that are both time consuming and unrealistic as a part of the behavior of an abstract-machine compilation target for the language. In contrast the *level-tags* system of Hodas, et al. relies only on manipulation and examination of tags attached to individual formulas. As formulas are used their tag values are changed, as opposed to the formula actually being removed from the program context. This system is therefore more appropriate as the specification of an abstract machine. Unfortunately, it poorly handles a variety of language features in comparison to the resource management systems. In addition, it has been formulated only for a smaller fragment of the logic. This has led to a divergence in the operational semantics of the interpreted and compiled versions of the language Lolli.

In this paper we propose a *tag-frame* system which derives from, and captures the positive aspects of, all the above systems. In particular, it recaptures all the determinism of the resource management systems, but in a manner that requires only the manipulation and examination of tags labelling formulas: formulas are simply marked (rather than removed) when they are used, and there are no global manipulations of the set of formulas. Further, it is possible to determine whether a formula is actually available for backchaining simply by examining its tags. In fact, whereas the level-tags system contained certain operations that required examining or modifying tags on all the formulas in a context, this system manipulates tags only on individual formulas.

This paper is arranged as follows. In Section 2 we review the principal sources of controllable non-determinism in linear logic proof search and describe the various previous systems mentioned above. In Section 3 we introduce the Tag Frame proof system, and describe its key qualitative properties. In Section 4 we present various formal properties of the system, leading up to a statement of soundness and completeness. Proofs of these properties are not contained in this paper, but will be available in a technical report. Finally, Sections 5 and 6 describe relevant related work and some future areas of focus, respectively.

$$\frac{}{\Delta^I \backslash \Delta^I \Longrightarrow A \gg A}\ \textbf{atomic} \gg_{\textbf{io}} \qquad \frac{}{\Delta\Delta^O \backslash \Delta^O \Longrightarrow \top}\ \top_{io}$$

$$\frac{\Delta^I \backslash \Delta^M \Longrightarrow D \gg A \quad \Delta^M \backslash \Delta^O \Longrightarrow G}{\Delta^I \backslash \Delta^O \Longrightarrow G \multimap D \gg A}\ \multimap \gg_{io} \qquad \frac{\Delta^I \backslash \Delta^O \Longrightarrow D \gg A}{\Delta^I D \backslash \Delta^O \Longrightarrow A}\ \textbf{pick}_{io}$$

$$\frac{\Delta^I \backslash \Delta^O \Longrightarrow G_1 \quad \Delta^I \backslash \Delta^O \Longrightarrow G_2}{\Delta^I \backslash \Delta^O \Longrightarrow G_1 \,\&\, G_2}\ \&_{io} \qquad \frac{\Delta\Delta^O D \backslash \Delta^O \Longrightarrow G}{\Delta\Delta^O \backslash \Delta^O \Longrightarrow D \multimap G}\ \multimap_{io}$$

$$\frac{\emptyset \backslash \_ \Longrightarrow G}{\Delta^I \backslash \Delta^I \Longrightarrow !G}\ !_{io}$$

Fig. 1. The I/O proof system of Hodas and Miller

2 Background

The need to consider novel strategies to control unnecessary non-determinism in proof search was obvious from the time researchers first considered the development of logic programming languages based on intuitionistic linear logic. While Hodas and Miller showed that a fragment of linear logic corresponding to linear hereditary Harrop formulas admitted goal-directed search, and hence could be considered as an *abstract logic programming language*, a naive implementation of the system, derived directly from the proof rules of linear logic, would be unusable due to the degree of non-determinism in the management of formulas in the program [11, 12].

In this section we present a historic overview of the refinements of proof search to date. It is necessary to present all these systems, because our proposal builds on and derives features from all of them. Each system is presented briefly, noting only the key ideas. The reader is referred to the individual papers for more exposition.

2.1 The I/O Model

The first and most serious source of non-determinism is apparent in the left-hand rule for linear implication:

$$\frac{\Delta_1 \longrightarrow A \quad \Delta_2, B \longrightarrow C}{\Delta_1, \Delta_2, A \multimap B \longrightarrow C}\ \multimap_L$$

In order to apply this rule during bottom-up search for a proof, it is necessary to determine an effective splitting of the assumptions into the two sub-contexts Δ_1 and Δ_2. However the number of such splittings to consider is exponential in the number of formulas in the context, and most of these splittings will generally not work. Hodas and Miller proposed a lazy system, which they called the *I/O model*, in which the left premiss is proved in the context of all available formulas, and those assumptions it does not use are then made available to, and must be used in, the proof of the right premiss. They showed that the resulting system

$$\frac{}{\Delta^I \backslash \Delta^I \Longrightarrow_0 A \gg A}\ \textbf{atomic} \gg_{io_\top} \qquad \frac{}{\Delta^I \backslash \Delta^I \Longrightarrow_1 \top}\ \top_{io_\top}$$

$$\frac{\Delta^I \backslash \Delta^M \Longrightarrow_{v_1} D \gg A \quad \Delta^M \backslash \Delta^O \Longrightarrow_{v_2} G}{\Delta^I \backslash \Delta^O \Longrightarrow_{v_1 \vee v_2} G \multimap D \gg A}\ \multimap \gg_{io_\top} \qquad \frac{\Delta^I \backslash \Delta^O \Longrightarrow_v D \gg A}{\Delta^I D \backslash \Delta^O \Longrightarrow_v A}\ \textbf{pick}_{io_\top}$$

$$\frac{\Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \quad \Delta^I \backslash \Delta^O \Longrightarrow_0 G_2}{\Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \&00_{io_\top} \qquad \frac{\Delta\Delta^O D \backslash \Delta^O \Longrightarrow_0 G}{\Delta\Delta^O \backslash \Delta^O \Longrightarrow_0 D \multimap G}\ \multimap 0_{io_\top}$$

$$\frac{\Delta^I \backslash \Delta_1^O \Longrightarrow_1 G_1 \quad \Delta^I \backslash \Delta_2^O \Longrightarrow_1 G_2}{\Delta^I \backslash \Delta_1^O \cap \Delta_2^O \Longrightarrow_1 G_1 \,\&\, G_2}\ \&11_{io_\top} \qquad \frac{\Delta\Delta^O D \backslash \Delta^O \_ \Longrightarrow_1 G}{\Delta\Delta^O \backslash \Delta^O \Longrightarrow_1 D \multimap G}\ \multimap 1_{io_\top}$$

$$\frac{\Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \quad \Delta^I \backslash \Delta_2 \Delta^O \Longrightarrow_1 G_2}{\Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \&01_{io_\top} \qquad \frac{\emptyset \backslash \emptyset \Longrightarrow_v G}{\Delta^I \backslash \Delta^I \Longrightarrow_0 !G}\ !_{io_\top}$$

$$\frac{\Delta^I \backslash \Delta_1 \Delta^O \Longrightarrow_1 G_1 \quad \Delta^I \backslash \Delta^O \Longrightarrow_0 G_2}{\Delta^I \backslash \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \&10_{io_\top}$$

Fig. 2. The I/O$_\top$ proof system of Hodas

was sound and complete relative to the traditional formulation of the rules for the fragment they were interested in.

The intent of the I/O model is captured in the proof system presented in Figure 1. This is essentially the I/O system of Hodas and Miller, recast in a somewhat different style similar to one used by Cervesato, et al. [3, 4], which we will adopt for the remainder of this paper. In this system, the left rules have been replaced by judgments for using a selected clause to prove an atomic goal formula. Proofs in this system are therefore necessarily *uniform*, in the sense of Miller, et al. [15], and *focused*, in the sense of Andreoli [1]. The left-hand side of the sequent now features two sets of assumptions, separated by a slash. The first is the set of assumptions passed into the proof as input, the second is the set of assumptions left over at the end of the proof, and passed back out as output.

Note that, throughout the paper, we present the proof system for only a fragment of the logic sufficient to elucidate the problems with which we are concerned. In particular, we assume all formulas in the context are linear; that is, there are no negative uses of !. We further omit the multiplicative conjunction, $\otimes$, as its behavior in goals is mimicked by negative uses of the linear implication, $\multimap$. Finally, we also eliminate the additive disjunction, $\oplus$, as well as negative uses of the additive conjunction, &.

2.2 The I/O$_\top$ Model

While the I/O model deals with the most glaring source of needless nondeterminism, it introduces a new source in the $\top_R$ rule. In the standard system, the axiom rule for $\top$:

$$\frac{}{\Delta \longrightarrow \top}\ \top_R$$

$$\frac{}{\emptyset; \Delta^I \setminus \Delta^I \Longrightarrow_0 A \gg A}\ \textbf{atomic} \gg_{rm3} \qquad \frac{}{\Xi; \Delta^I \setminus \Delta^I \Longrightarrow_1 \top}\ \top_{rm3}$$

$$\frac{\emptyset; \Xi\ \Delta^I \setminus \Delta^M \Longrightarrow_0 D \gg A \quad (\Xi \cap \Delta^M); (\Delta^I \cap \Delta^M) \setminus \Delta^O \Longrightarrow_v G}{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v G \multimap D \gg A}\ \multimap \gg 0_{rm3}$$

$$\frac{\emptyset; \Xi\ \Delta^I \setminus \Delta^M \Longrightarrow_1 D \gg A \quad \emptyset; \Delta^M \setminus \Delta^O \Longrightarrow_v G}{\Xi; \Delta^I \setminus (\Delta^I \cap \Delta^O) \Longrightarrow_1 G \multimap D \gg A}\ \multimap \gg 1_{rm3}$$

$$\frac{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v D \gg A}{\Xi\ D; \Delta^I \setminus \Delta^O \Longrightarrow_v A}\ \textbf{pick\_strict}_{rm3} \qquad \frac{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v D \gg A}{\Xi; \Delta^I D \setminus \Delta^O \Longrightarrow_v A}\ \textbf{pick\_lax}_{rm3}$$

$$\frac{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \quad \Xi\ (\Delta^I - \Delta^O); \emptyset \setminus \emptyset \Longrightarrow_v G_2}{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_0 G_1 \,\&\, G_2}\ \&0_{rm3}$$

$$\frac{\Xi; \Delta^I \setminus \Delta^M \Longrightarrow_1 G_1 \quad \Xi\ (\Delta^I - \Delta^M); \Delta^M \setminus \Delta^O \Longrightarrow_v G_2}{\Delta^I \setminus (\Delta_1^O \cap \Delta_2^O) \Longrightarrow_v G_1 \,\&\, G_2}\ \&1_{rm3}$$

$$\frac{\Xi\ D; \Delta^I \setminus \Delta^O \Longrightarrow_v G}{\Xi; \Delta^I \setminus \Delta^O \Longrightarrow_v D \multimap G}\ \multimap 0_{rm3} \qquad \frac{\emptyset \setminus \emptyset \Longrightarrow_v G}{\emptyset; \Delta^I \setminus \Delta^I \Longrightarrow_0 !G}\ !_{rm3}$$

Fig. 3. The $\mathcal{RM}_3$ proof system of Cervesato, et al.

means that the goal succeeds in any context, effectively consuming any formulas that have been passed to this branch of the proof. In the I/O system, however, the rule must select some subset, Δ, of the currently available formulas to consume, passing along the rest, Δ^O, as available to the proofs of subsequent goals. This selection is of course exponential, as was the original splitting discussed above.

The solution is to replace this explicit consumption with a flag which indicates whether a goal of $\top$ has been seen at the top-level in a given sub-proof. If so, any assumptions left over at the end of proof construction can be considered to have been consumed by that goal. This is captured in the system in Figure 2, which is a variant of the system first formulated by Hodas [10]. The flag, which represents implicit consumption or weakening of unused assumptions, appears as a subscript of the sequent arrow.

2.3 Resource Management Systems

The *resource management* model of Cervesato, Hodas, and Pfenning [3, 4], presented in Figure 3, attempts to deal principally with the inefficient treatment of the additive conjunction, &, in goals. The key advance is in noticing that it does not make sense to provide the attempt to prove the right premiss with the entire input context to work with, since it is barred, in the end, from making use of any formulas that were not used in the proof of the left premiss. Thus, the input to the proof of the right premiss should be exactly those assumptions that were used in the proof of the left premiss. Similarly, all of those assumptions *must* be used, none can be left as output. To accomplish this, the system divides the input context into two parts. The first is a *strict* context of formulas which must be used in this sub-proof. The second is a *lax* context of formulas treated in the

$$\frac{}{\emptyset; \Pi/\Pi \Longrightarrow_0 A \gg A}\ \textbf{atomic}\gg\mathcal{F}' \qquad \frac{}{\Delta; \Pi/\Pi \Longrightarrow_1 \top}\ \top\,\mathcal{F}'$$

$$\frac{\emptyset; \Delta :: \Pi/\Delta' :: \Pi' \Longrightarrow_0 D \gg A \quad \Delta'; \Pi'/\Pi'' \Longrightarrow_v G}{\Delta; \Pi/\Pi'' \Longrightarrow_v G \multimap D \gg A}\ \multimap_0\gg\mathcal{F}'$$

$$\frac{\emptyset; \Delta :: \Pi/\Delta' :: \Pi' \Longrightarrow_1 D \gg A \quad \emptyset; \Delta' :: \Pi'/\Delta'' :: \Pi'' \Longrightarrow_v G}{\Delta; \Pi/\Pi'' \Longrightarrow_1 G \multimap D \gg A}\ \multimap_1\gg\mathcal{F}'$$

$$\frac{\Delta; \Pi/\Pi' \Longrightarrow_v D \gg A}{D\ \Delta; \Pi/\Pi' \Longrightarrow_v A}\ \textbf{pick}\,\Delta\,\mathcal{F}' \qquad \frac{\Delta; \Pi/\Pi' \Longrightarrow_v D \gg A}{\Delta; \Pi \triangleleft D/\Pi' \Longrightarrow_v A}\ \textbf{pick}\,\Pi\,\mathcal{F}'$$

$$\frac{\Delta; \Pi/\Pi' \Longrightarrow_0 G_1 \quad \Delta\ (\Pi - \Pi'); \mathtt{nil}/\mathtt{nil} \Longrightarrow_v G_2}{\Delta; \Pi/\Pi' \Longrightarrow_0 G_1 \,\&\, G_2}\ \&_0\,\mathcal{F}'$$

$$\frac{\Delta; \Pi/\Pi' \Longrightarrow_1 G_1 \quad \Delta\ (\Pi - \Pi'); \Pi'/\Pi'' \Longrightarrow_v G_2}{\Delta; \Pi/\Pi'' \Longrightarrow_v G_1 \,\&\, G_2}\ \&_1\,\mathcal{F}'$$

$$\frac{D\ \Delta; \Pi/\Pi' \Longrightarrow_v G}{\Delta; \Pi/\Pi'' \Longrightarrow_v D \multimap G}\ \multimap\mathcal{F}' \qquad \frac{\emptyset; \mathtt{nil}/\mathtt{nil} \Longrightarrow_v G}{\emptyset; \Pi/\Pi \Longrightarrow_0 !\,G}\ !\,\mathcal{F}'$$

Fig. 4. The $\mathcal{F}'$ frames proof system of López and Pimentel

normal manner. Complexity arises when one or the other conjuncts includes $\top$ as a subgoal.

2.4 The Frame System

While the $\mathcal{RM}_3$ proof system of Cervesato, et al. achieves reduced non-determinism and early failure in many uses of the additive conjunction, &, it unfortunately imposes an extra burden on the treatment of the linear implication in clauses and, similarly, of the multiplicative conjunction, $\otimes$, which it mimics. In particular, since the proofs of the two premises of the rule freely share the pool of input formulas and may divide them up arbitrarily, the proof of the first premiss has no strictness constraints. This is accomplished by adding the contents of the current strict context into the lax context since they are *allowed* to be used, they just are not *required* to be used. However, before the second premiss can be proved, what is left from the original two contexts must be disentangled, necessitating the context intersections to be computed.

The *frame* system, $\mathcal{F}$, of López and Pimentel [14], a variant of which is shown in Figure 4, aims to eliminate this extra cost by replacing the lax context with a stack of contexts, referred to as frames. The nesting of applications of the left rule for linear implication is reflected in the stack of frames. When a strict context is added to the lax, it is pushed on the front, rather than being intermingled. The ensuing disentanglement can therefore be accomplished in constant time. Note that in the rule **pick** $\Pi\,\mathcal{F}'$, the expression $\Pi \triangleleft D$ refers to the stack Π with the formula D inserted into one of its frames. Thus this rule corresponds to selecting the formula D from some arbitrary frame in the stack.

In the case of logic programming, as opposed to theorem proving, it is not clear that the frames formulation can be implemented as given. This is because in a working language implementation the order of clauses is significant, due to the top-down search for clauses in Prolog-like languages. Since the formulas that are strict at a particular moment (due to having been used in the first branch of the proof of an additive conjunction, for example) may occur anywhere in the program, they cannot be isolated from the rest of the clauses in a separate frame. Thus the strict and lax contexts will need to be represented by markers on the individual formulas, and disentangling the two contexts requires traversing the entire context to locate formulas tagged as belonging to the topmost frame. The system, nevertheless, provides crucial inspiration for our solution to these problems. (Watkins, in an unpublished note, took inspiration from the level-tags model to develop a system that is essentially isomorphic to the frame system [17]. That work also provided helpful inspiration for our tag-frame system.)

2.5 The $\mathcal{LRM}$ Level-Tag Model

The systems $\mathcal{RM}_3$ and $\mathcal{F}$ minimize needless backtracking during search. However, because the need to move formulas around between strict and lax contexts and to perform operations such as intersection on contexts requires manipulating large dynamic structures they are best suited to interpreter-based implementations. Hodas, Watkins, Tamura, and Kang [13], building on work of Tamura and Kaneda [16] proposed the *level-tags* proof system, $\mathcal{LRM}$, for a fragment of linear hereditary Harrop formulas which was better suited to implementation as an abstract machine.

In this system, each sequent is adorned with two *level indices*, L and U. Their values, which rise and fall during proof search, determine the availability and strictness of formulas in the context. Each formula in the program is similarly adorned with two indices. The first is the *consumption level*: a formula may only be used if its value matches the current value of L. The second tag is used initially to indicate the smallest value of L at which the formula is allowed to exist without having been consumed. Thus it controls strictness. Once a formula has been used, its consumption level is set to 0, so that it is unavailable, and the other tag is set to the current value of U, indicating at what point it was used.

As proof search passes through the various operators, the structure of the contexts remains stable. Only the tags on the formulas are manipulated. Thus while there are still a number of rules which require examining or manipulating the tags on all of the formulas in the context, the data structures being manipulated are simple. Because our use of tags shares only the basic inspiration of this system, and the details differ significantly, we omit the actual $\mathcal{LRM}$ proof system.

3 The Tag-Frame System

The *tag-frame* system, $\mathcal{TF}$, presented in Figure 5, was motivated by the desire to reduce or remove the overhead of the global context operations such as context

$$\frac{\forall D^t \in \Delta_I \quad t \notin \delta}{\Delta_I \backslash \Delta_I \overset{\delta::\pi}{\underset{\sigma}{\longrightarrow}}{}^{\sigma}_{0} A \gg A}\ \textbf{atomic} \gg \mathcal{TF} \qquad \frac{}{\Delta_I \backslash \Delta_I \overset{\delta::\pi}{\underset{\delta\cup\sigma}{\longrightarrow}}{}^{\sigma}_{1} \top}\ \top\mathcal{TF}$$

$$\frac{\Delta_I \backslash \Delta_M \overset{\{t\}::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{0} D \gg A \quad \Delta_M \backslash \Delta_O \overset{\pi}{\underset{\sigma''}{\overset{\sigma'-\{t\}}{\longrightarrow}}}{}^{G}_{v}}{\Delta_I \backslash \Delta_O \overset{\pi}{\underset{\sigma''}{\longrightarrow}}{}^{\sigma}_{v} G \multimap D \gg A}\ \multimap_0 \gg \mathcal{TF} \quad (t\ new)$$

$$\frac{\Delta_I \backslash \Delta_M \overset{\{t\}::\delta::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{1} D \gg A \quad \Delta_M \backslash \Delta_O \overset{\{t\}::\delta::\pi}{\underset{\sigma''}{\longrightarrow}}{}^{\sigma'-\{t\}}_{v} G}{\Delta_I \backslash \Delta_O \overset{\delta::\pi}{\underset{(\sigma''-\{t\})\cup\delta}{\longrightarrow}}{}^{\sigma}_{1} G \multimap D \gg A}\ \multimap_0 \gg \mathcal{TF} \quad (t\ new)$$

$$\frac{\Delta_L D^d \Delta_R \backslash \Delta_O \overset{\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{v} D \gg A}{\Delta_L D^t \Delta_R \backslash \Delta_O \overset{\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{v} A}\ \textbf{pick}\ \mathcal{TF} \quad t \in \pi, d \in \sigma \qquad \frac{D^t \Delta_I \backslash D^{\text{-}} \Delta_O \overset{\delta::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{v} G}{\Delta_I \backslash \Delta_O \overset{\delta::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{v} D \multimap G}\ \multimap \mathcal{TF} \quad (t \in \delta)$$

$$\frac{\Delta_I \backslash \Delta_M \overset{\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\{d\}}_{0} G_1 \quad \Delta_M \backslash \Delta_O \overset{\sigma'}{\underset{\sigma''}{\overset{::\mathtt{nil}}{\longrightarrow}}}{}^{\sigma}_{v} G_2}{\Delta_I \backslash \Delta_O \overset{\pi}{\underset{\sigma''}{\longrightarrow}}{}^{\sigma}_{0} G_1 \,\&\, G_2}\ \&_0\,\mathcal{TF} \quad (d\ new)$$

$$\frac{\Delta_I \backslash \Delta_M \overset{\delta::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\{d\}}_{1} G_1 \quad \Delta_M \backslash \Delta_O \overset{\sigma'}{\underset{\sigma''}{\overset{::\pi}{\longrightarrow}}}{}^{\sigma}_{v} G_2}{\Delta_I \backslash \Delta_O \overset{\delta::\pi}{\underset{\sigma''}{\longrightarrow}}{}^{\sigma}_{v} G_1 \,\&\, G_2}\ \&_1\,\mathcal{TF} \quad (d\ new)$$

$$\frac{\forall D^s \in \Delta_I \quad s \notin \delta \quad \Delta_I \backslash \_ \overset{\{t\}::\mathtt{nil}}{\underset{\_}{\longrightarrow}}{}^{\sigma}_{\_} G}{\Delta_I \backslash \Delta_I \overset{\delta::\pi}{\underset{\sigma}{\longrightarrow}}{}^{\sigma}_{0} !\,G}\ !\,\mathcal{TF} \quad (t\ new)$$

Fig. 5. The tag-frame proof system, $\mathcal{TF}$

intersections present in $\mathcal{RM}_3$ while retaining that system's ability to prune failed searches early. For a number of years the authors were convinced that it should be possible to design a system in which there was minimal structural manipulation of contexts, and the availability of a formula at a given point in a proof tree could be determined by examining just some annotations on the formula, which are adjusted as proof search progressed.

The tag-frame system demonstrates those attributes. The key idea is that formulas are each marked with a single tag value. Depending on its value, and the values adorning the sequent, that tag can indicate whether a formula is available for use, whether it must be used in the current sub-proof, and whether it has been used, and if so where. In contrast to the level-tags system, rules in $\mathcal{TF}$ require changing the tags only on individual formulas. While two rules still require scanning tags throughout the context, we believe, as discussed in Section 6, that that work can be amortized to reduce the effort considerably.

A $\mathcal{TF}$ sequent is of the general form:

$$\Delta_I \backslash \Delta_O \overset{\delta::\pi}{\underset{\sigma'}{\longrightarrow}}{}^{\sigma}_{v}\, G$$

The contexts Δ_I, and Δ_O are, as usual, the input and output contexts. However, in this system they contain exactly the same formulas. They differ only in how the formulas are marked. Not all formulas in Δ_I may actually be available for use in the current sub-proof. They may already have been used (and marked as such) at a previous point.

The stack adorning the upper-left of the sequent arrow is a stack of frames of tag values. The management of this stack corresponds fairly directly to the management of context frames in the system $\mathcal{F}'$. The topmost frame, δ, consists of tag values that denote strict formulas which must be used in the current sub-proof, while the tags in all the frames below are used to mark formulas that are lax. That is, formulas marked with those tags may be used, but need not be.

The set σ adorning the upper-right of the sequent contains tags that may be used to mark formulas as they are used. Any tag in this set may be used. The set σ' on the lower-left of the sequent arrow contains the tags that adorn formulas, in either of the contexts, which have been consumed. This includes both formulas that are explicitly consumed in the **pick** $\mathcal{TF}$ rule, as well as formulas from the strict context that are implicitly consumed by an instance of $\top$. Finally, the variable v on the lower-right of the sequent is the traditional $\top$-flag.

Section 4 states several key properties of the system, but the following properties of the annotation frames and stacks are worth keeping in mind while examining the rules:

- The sets δ, π (or, rather, the union of the sets comprising π), and σ are pairwise disjoint.
- The sets δ and σ are never empty. The root sequent of a tag-frame proof generally has a singleton σ. It also has a singleton δ and an empty π. All formulas in the initial Δ_I are marked with the one tag in δ, indicating that all initial assumptions are strict.
- The only formulas in Δ_I available for use in the current sub-proof are those whose tags are in δ and π
- The set σ' always contains σ. If the goal G contains an instance of $\top$ at an appropriate position, then σ' also contains δ. The set σ' may also contain additional tags that are new, local tags created above this point in the tree which bleed out of scope due to the nature of the rules for additive conjunction.

  To understand this, consider the $\&\,0$ rule. The left premiss introduces a new tag d to mark its consumption so that the right premiss has a trace to follow. This new tag d will occur in that sequent's σ' which then plays the role of δ for the right premiss, so that that premiss will use exactly the same assumptions as the left. If the goal of the right premiss includes an appropriately placed instance of $\top$, then σ' will be contained in σ''; and, hence, d will be transferred to the conclusion of the rule.

The role of the the various tag frames is seen in how they are used and changed in the rules. For example, the use of δ to impose strictness is seen in **atomic**$\gg\mathcal{TF}$ in which this identity axiom can be applied only if there are no strict formulas (ones labeled with tags in δ) in the input context. The $!\mathcal{TF}$ rule has a similar constraint, since the sub-proof of G may not consume any extent formulas. In a related fashion, the $\top\mathcal{TF}$ rule copies all of δ to σ' in order to indicate that all the strict formulas have been consumed. When a formula is added to the assumptions in an application of the $\multimap\mathcal{TF}$ rule, it is marked with a tag chosen from δ, indicating that the formula must be used in this sub-proof. Because the strictness constraint on that formula will be enforced at the leaves of the proof, it is not necessary to check the formula's tag on exit from the rule, as indicated by the wildcard tag on the formula in the output context.

In order to relax any current strictness constraints in the first premiss of the $\multimap\gg\mathcal{TF}$ rules, a new frame is pushed onto the stack on top of δ. This frame consists of a single new tag. Since that tag is new, there are no formulas in Δ_I marked with it, and the proof of the left premiss thus begins with no formulas labeled as strict. If that premiss encounters a $\top$ goal, then the right premiss is similarly proved with no strictness constraints. Because the only formulas that may become labeled with the new tag are those added to the context by instances of $\multimap$ within D, and since such formulas are removed from the context before exiting the sub-proof that added them, we can see that no formulas in Δ_I or Δ_M will be labeled with the new tag, even though that tag may occur in σ' on exit from the left premiss (if G contains an instance of $\top$). Therefore, in order to slow the growth of the frames, the new tag is removed from σ' if present, before that frame is used in the right premiss. It is similarly removed from σ'' on exit from the rule if necessary.

In the rules for $\&$, the proof of the left premiss is started with a fresh tag as the only element of σ. Thus, it will be easy to identify the formulas consumed in that sub-proof, as they will all be marked with tags from σ', and that set will not overlap with the current δ, π, or σ. If the proof of the left premiss does not encounter a $\top$, then the proof of the right premiss must use exactly those formulas marked with tags from σ'. This is accomplished by using that set in place of δ (making those formulas strict), and setting an empty stack beneath (meaning that no other formulas may be used). If $\top$ is encountered in the proof of the left premiss, then the proof of the right premiss is allowed access to the formulas marked with tags in π, since they were implicitly consumed in the left branch, and so may be used in the right if desired.

4 Soundness and Completeness

In this section we discuss the soundness and completeness of $\mathcal{TF}$ with respect to the frame system variant $\mathcal{F}'$. Due to space limitations full proofs are not included in the paper; however, formal results are explained to gain insight and provide a detailed account of the inner workings of $\mathcal{TF}$.

Since the $\mathcal{F}'$ proof system (Figure 4) is a direct reformulation of a part of the original $\mathcal{F}$ [14] for the fragment of linear hereditary Harrop formulas we are dealing with in this paper, the soundness and completeness of $\mathcal{F}'$ w.r.t. to $\mathcal{F}$ are trivially proved. In particular, note that the right and pick rules in Figure 4 are a simplification of those of the original $\mathcal{F}$ system. On the other hand, $\mathcal{F}$ left rules are replaced by rewrite judgements in $\mathcal{F}'$. The reader is referred to the original paper [14] for further details on the frame systems.

It is worth noting that in the $\mathcal{F}'$ system the stack structure is imposed on the contexts (i.e. on the logic program and the residue) so that clauses are ordered according to their level of strictness. In contrast, in the $\mathcal{TF}$ system this structure is imposed on the tags, and therefore formulas with any given tag are scattered through the context.

It should be clear that for the $\mathcal{TF}$ system to work properly, it is essential to avoid tag clashes. As described in Section 3, a few $\mathcal{TF}$ rules have the proviso "*new t*". The intended meaning is that the tag t is new in the sense that it has never been used in the $\mathcal{TF}$-proof so far. While this loose notion is sufficient for an informal description of $\mathcal{TF}$, a more formal way to express uniqueness of tags is needed to prove the results presented in this section. To that end we assume the existence of a countably infinite set of tags $\mathcal{T}$, and extend $\mathcal{TF}$-sequents by adding a signature (also countable) of unused, available tags $\Sigma \subseteq \mathcal{T}$

$$\Sigma : \Delta_I \backslash \Delta_O \; {}^{\delta::\pi}_{\sigma'} \!\!\longrightarrow\!\!{}^{\sigma}_{v} \; G$$

where Σ contains tags that are neither in $\delta \cup \sigma$ nor in π, and no formula in Δ_I is tagged with a tag of Σ. Thus, each time we have a proviso "*new t*", a tag t is taken (and removed) from Σ. In addition, when rules with two premises are applied bottom-up (namely, $\&\,\mathcal{TF}$ and $\multimap\mathrm{L}\,\mathcal{TF}$), the signature Σ is split into two countably infinite disjoint signatures as shown below

$$\frac{\Sigma_1 : \Delta_I \backslash \Delta_M \; {}^{\pi}_{\sigma'} \!\!\longrightarrow\!\!{}^{\{d\}}_{0} \; G_1 \quad \Sigma_2 : \Delta_M \backslash \Delta_O \; {}^{\sigma'::\mathtt{nil}}_{\sigma''} \!\!\longrightarrow\!\!{}^{\sigma}_{v} \; G_2}{\Sigma : \Delta_I \backslash \Delta_O \; {}^{\pi}_{\sigma''} \!\!\longrightarrow\!\!{}^{\sigma}_{0} \; G_1 \,\&\, G_2} \; \&_0\,\mathcal{TF}$$

with the proviso $\Sigma = \{d\} \dot\cup \Sigma_1 \dot\cup \Sigma_2$, $\Sigma_i (i = 1, 2)$ being infinite.

Next, to complete the formalization of uniqueness of tags and ensure that strictness tags and consumption markers are appropriately used in a $\mathcal{TF}$-proof, the notion of *tag-consistency* is introduced.

Definition 1 (Tag-consistency). *Let $\hat{\pi}$ denote the union of the multisets comprising the stack π, and $[\Delta]_\xi$ denote the multiset of formulas D such that $D^t \in \Delta$ when $t \in \xi$. Then, a $\mathcal{TF}$-sequent*

$$\Sigma : \Delta_I \backslash \Delta_O \; {}^{\delta::\pi}_{\sigma'} \!\!\longrightarrow\!\!{}^{\sigma}_{v} \; G$$

is tag-consistent *if and only if:*

1. *Σ, δ, $\hat{\pi}$, and σ are pairwise disjoint*

2. *δ and σ are non-empty*
3. $[\Delta_I]_\Sigma = \emptyset$

Since the $\mathcal{TF}$ rules preserve tag-consistency, it is easily proved that given a provable, tag-consistent $\mathcal{TF}$-sequent, every $\mathcal{TF}$-sequent involved in its $\mathcal{TF}$-proof is tag-consistent, thus there are no tag clashes.

In addition to uniqueness of tags, contexts and tags occurring in a $\mathcal{TF}$-proof satisfy certain non-trivial properties that are essential to prove soundness and completeness. These properties are gathered in the following:

Theorem 1 (Consumption Invariants). *For all Δ_I, Δ_O, Σ, δ, π, σ, σ', v, and G such that*

$$\Sigma : \Delta_I \backslash \Delta_O \xrightarrow[\sigma']{\delta::\pi}{}^{\sigma}_{v} G$$

is provable and tag-consistent, it holds that:

1. $\sigma' = \sigma \cup \rho$ *or* $\sigma' = \sigma \cup \delta \cup \rho$ *where* $\rho \subseteq \Sigma$
2. $\forall t \notin \sigma \cup \Sigma :\ [\Delta_O]_{\{t\}} \subseteq [\Delta_I]_{\{t\}}$
3. $\forall t \in \sigma :\ [\Delta_O]_{\{t\}} \supseteq [\Delta_I]_{\{t\}}$
4. $[\Delta_O]_{\sigma'} = [\Delta_I]_\delta \uplus [\Delta_I]_\sigma \uplus ([\Delta_I]_{\hat{\pi}} - [\Delta_O]_{\hat{\pi}})$
5. $\forall \eta . \eta \cap (\delta \cup \hat{\pi} \cup \sigma' \cup \Sigma) = \emptyset :\ [\Delta_O]_\eta = [\Delta_I]_\eta$
6. $[\Delta_O]_\mathcal{T} = [\Delta_I]_\mathcal{T}$

Proof. By induction on the structure of the $\mathcal{TF}$ proofs.

The first property describes the composition of σ', i.e. the set of tags establishing the overall consumption. In particular, note that σ is always included in σ'. In addition, σ' may also include δ depending on whether or not a $\top$ occurred in the proof of G. Finally, σ may include some additional tags globally referred to as ρ. This set of tags, ρ, accounts for the local tags exported by the $\&\mathcal{TF}$ rules. Note that these rules are the only ones that export local tags, since the $\multimap\!\gg\mathcal{TF}$ rules remove their new tags from the output.

The next two properties establish essential consumption invariants relating the input and the output of the $\mathcal{TF}$-proofs. In particular, the second property means that if a formula D^t occurs in the output Δ_O, and the tag t is neither a consumption marker nor new, then D^t was in the input Δ_I. In addition, the third property says that every formula marked as consumed in the input is also marked with the same consumption marker in the output.

The fourth property states that the consumed formulas in the output, $[\Delta_0]_{\sigma'}$, are those that were strict in the input, $[\Delta_I]_\delta$, plus those already consumed, $[\Delta_I]_\sigma$, plus the portion of lax resources that have been consumed in the proof of G, $[\Delta_I]_{\hat{\pi}} - [\Delta_O]_{\hat{\pi}}$. This property is also referred to as the *local consumption property.*

Finally, the fifth and sixth properties say that formulas not involved in the $\mathcal{TF}$-proofs are silently returned and that the input and output contexts have the same cardinality, respectively.

The invariants relating input and output in $\mathcal{TF}$-proofs are stronger than those in other resource management systems [3, 4, 14]. Weaker versions of these invariants relating the strict and lax portions of the input and the output are stated in the following:

Corollary 1. *For all Δ_I, Δ_O, Σ, δ, π, σ, σ', v, G such that*

$$\Sigma : \Delta_I \backslash \Delta_O \xrightarrow[\sigma']{\delta::\pi}{}^{\sigma}_{v} G$$

is provable and tag-consistent, it holds that:

1. $[\Delta_O]_\delta \subseteq [\Delta_I]_\delta$
2. $[\Delta_O]_\pi \subseteq [\Delta_I]_\pi$
3. $\pi \cap \sigma' = \emptyset$
4. $\sigma \subseteq \sigma'$

Proof. Immediate from the Consumption Invariants theorem.

Note that the third property ensures that returned lax resources are still available to be consumed elsewhere.

We are now in a position to state the logical relationship between $\mathcal{TF}$ and $\mathcal{F}'$. Note that the contexts of an $\mathcal{F}'$-proof comprise just a portion of those of the corresponding $\mathcal{TF}$-proof. In particular, consumed resources are not kept in the input contexts of a $\mathcal{F}'$-proof whereas they are kept in a $\mathcal{TF}$-proof. On the other hand, the output contexts of a $\mathcal{F}'$-proof can only contain lax resources, while the output contexts of a $\mathcal{TF}$-proof may contain strict, lax, and consumed resources.

Theorem 2 (Soundness). *The $\mathcal{TF}$ proof system is sound with respect to the $\mathcal{F}'$ proof system; that is, for all Δ_I, Δ_O, Σ, δ, π, σ, σ', v, G such that*

$$\Sigma : \Delta_I \backslash \Delta_O \xrightarrow[\sigma']{\delta::\pi}{}^{\sigma}_{v} G$$

is provable and tag-consistent, it holds that:

$$[\Delta_I]_\delta \,;\, [\Delta_I]_\pi \,/\, [\Delta_O]_\pi \Longrightarrow_v G$$

Proof. By induction on the structure of the $\mathcal{TF}$ proofs.

Theorem 3 (Completeness). *The $\mathcal{TF}$ proof system is complete with respect to the $\mathcal{F}'$ proof system. That is, for all Δ, Π, G, Π', and v, if*

$$\Delta; \Pi / \Pi' \Longrightarrow_v G$$

then for all Δ_I, Σ, δ, π, σ such that:

1. *δ, $\hat{\pi}$, and σ are pairwise disjoint*
2. *δ and σ are non-empty*
3. $[\Delta_I]_\delta = \Delta$
4. $[\Delta_I]_\pi = \Pi$
5. $[\Delta_I]_\Sigma = \emptyset$

there are Δ_O and σ' satisfying:

1. $\Delta_I \backslash \Delta_O \xrightarrow[\sigma']{\delta::\pi}{}^{\sigma}_{v} G$
2. $[\Delta_O]_\pi = \Pi'$
3. $[\Delta_O]_{\sigma'} - [\Delta_I]_\sigma = \Delta \uplus (\Pi - \Pi')$

Proof. The two first consequences are proved by induction on the structure of the $\mathcal{TF}$ proofs, and the third one is direct consequence of the local consumption property and the correspondence among contexts of both sequents.

5 Related Work

The tag-frame system presented in this paper is tailored to a particular strategy for solving the linearity constraints which arise during bottom-up proof construction. This particular strategy is motivated by a logic programming interpretation of proof search. However, other strategies are certainly possible.

Harland and Pym presented a proof-system for linear logic which is independent of any strategy for satisfying the linearity constraints [7]. This is accomplished by making the linearity constraints explicit in the proof system as boolean expressions attached to each formula. The intuition is that formulas whose associated expressions evaluate to false are not actually in the sequent. This mechanism relieves the need for splitting up the resources between premises in the multiplicative rules; instead each premiss receives its own copy of the context, where the boolean expressions are adjusted to insure that one copy of each formula is annotated with an expression that will evaluate to false, so that, under the preceding intuition, only one premiss "actually" contains each formula. The axiom rules of the system place conditions upon the boolean expressions which ensure the linearity constraints are properly maintained.

Varying when and how the constraints on the boolean expressions are solved produces different strategies for linear proof search. The lazy resource distribution of the I/O model [11] corresponds to solving the constraints of each multiplicative branch upon reaching the end of that branch, before moving to the next branch. Subsequent work improving the efficiency of Lolli proof search, [3, 4, 14] and the work presented in this paper, corresponds to solving constraints generated in a multiplicative branch even before the end of the branch is reached, when possible. In particular, the constraints are checked at each leaf. In some sense tag-frames can be seen as encoding the low-level implementation details of a particular constraint checking strategy.

Making linear constraints explicit in a proof system may be understood as a way of dealing with partial information during proof construction: the distribution of linear resources throughout a proof will, in general, not be completely known until the proof is completely constructed. Andreoli has given a general reformulation of focusing proofs [1] which is not specific to linear logic [2]. This presentation also relies upon constraints to represent partial information, about unification as well as linearity, during proof construction, and presents a general constraint-solving method.

6 Conclusions and Future Work

We believe that $\mathcal{TF}$ is nearly optimal in its behavior, given the trade-off between eliminating non-determinism and using low-level as opposed to high-level data structures. It is easy to compare the system rule-by-rule with the others to see that it does at worst the same work for each rule, and often less. We further believe the remaining linear-time step, scanning at the leaves for formulas that should have been used, can be made linear in the cardinality of δ by maintaining counts of tag usage. We plan to implement and test this in the immediate future.

References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 1992. 170, 180

[2] Jean-Marc Andreoli. Focusing and proof construction. *Annals of Pure and Applied Logic*, 107(1–3):131–163, 2001. 180

[3] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. In Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister, editors, *Proceedings of the Fifth International Workshop on Extensions of Logic Programming*, volume 1050 of *Lecture Notes in Artificial Intelligence*, pages 67–81. Springer-Verlag, March 1996. 167, 168, 170, 171, 178, 180

[4] Iliano Cervesato, Joshua S. Hodas, and Frank Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232(1-2), February 2000. 167, 168, 170, 171, 178, 180

[5] James Harland and David Pym. The uniform proof-theoretic foundation of linear logic programming. In V. Saraswat and K. Ueda, editors, *Proceedings of the 1991 International Logic Programming Symposium*, pages 304–318. M. I. T. Press, 1991. 168

[6] James Harland and David Pym. A uniform proof-theoretic investigation of linear logic programming. *Journal of Logic and Computation*, 4(2):175–207, April 1994. 168

[7] James Harland and David Pym. Resource distribution via boolean constraints. In W. McCune, editor, *Proceedings of the Fourteenth International Conference on Automated Deduction — CADE-14*, Townsville, Australia, 1997. 180

[8] James Harland, David Pym, and Michael Winikoff. Programming in Lygon: An overview. In M. Wirsing and M. Nivat, editors, *Algebraic Methodology and Software Technology*, pages 391–405, Munich, Germany, 1996. Springer-Verlag LNCS 1101. 168

[9] James Harland and Michael Winikoff. Implementing the linear logic programming language Lygon. In John Lloyd, editor, *Proceedings of the 1995 International Logic Programming Symposium*, pages 66–80, 1995. 168

[10] J. S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design and Implementation*. PhD thesis, University of Pennsylvania, Department of Computer and Information Science, 1994. 168, 171

[11] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science*, July 15–18 1991. 167, 168, 169, 180

[12] J. S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1994. Extended abstract in the Proceedings of the Sixth Annual Symposium on Logic in Computer Science, Amsterdam, July 15–18, 1991. 167, 168, 169

[13] J. S. Hodas, K. Watkins, N. Tamura, and K.-S. Kang. Efficient implementation of a linear logic programming language. In *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming*, pages 145–159, June 1998. 167, 168, 173

[14] Pablo López and Ernesto Pimentel. Resource management in linear logic proof search revisited. In *Logic for Programming and Automated Reasoning*, volume 1705, pages 304–319. Springer Verlag, 1999. 167, 168, 172, 177, 178, 180

[15] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991. 170

[16] N. Tamura and Y. Kaneda. Extension of WAM for a linear logic programming language. In T. Ida, A. Ohori, and M. Takeichi, editors, *Second Fuji International Workshop on Functional and Logic Programming*, pages 33–50. World Scientific, Nov. 1996. 173

[17] Kevin Watkins. Unpublished Note, 1999. 173

Resource Tableaux (Extended Abstract)

Didier Galmiche[1], Daniel Méry[1], and David Pym[2]

[1] LORIA, Nancy, France
{galmiche,dmery}@loria.fr
[2] University of Bath, England
d.j.pym@bath.ac.uk

Abstract. The logic of bunched implications, **BI**, provides a logical analysis of a basic notion of resource rich enough to provide a "pointer logic" semantics for programs which manipulate mutable data structures. We develop a theory of semantic tableaux for **BI**, so providing an elegant basis for efficient theorem proving tools for **BI**. It is based on the use of an algebra of labels for **BI**'s tableaux to solve the resource-distribution problem, the labels being the elements of resource models. For **BI** with inconsistency, $\perp$, the challenge consists in dealing with **BI**'s Grothendieck topological models within such a proof-search method, based on labels. We prove soundness and completeness theorems for a resource tableaux method **TBI** with respect to this semantics and provide a way to build countermodels from so-called dependency graphs. As consequences, we have two strong new results for **BI**: the decidability of propositional **BI** and the finite model property with respect to Grothendieck topological semantics. In addition, we propose, by considering partially defined monoids, a new semantics which generalizes the semantics of **BI**'s pointer logic and for which **BI** is complete

Keywords: **BI**; resources; semantics; tableaux; decidability; finite model property.

1 Introduction

The notion of *resource* is a basic one in many fields, including economics, engineering and psychology, but it is perhaps most clearly illuminated in computer science. The location, ownership, access to and, indeed, consumption of, resources are central concerns in the design of systems (such as networks, within which processors must access devices such as file servers, disks and printers) and in the design programs, which access memory and manipulate data structures (such as pointers).

The development of a mathematical theory of resource is one of the objectives of the programme of study of **BI**, the logic of bunch implications, introduced by O'Hearn and Pym [10, 12, 13]. The basic idea is to model directly the observed properties of resources and then to give a logical axiomatization. Initially, we

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 183–199, 2002.
© Springer-Verlag Berlin Heidelberg 2002

require the following properties of resource, beginning with the simple assumption of a set R of elements of a resource: a *combination*, $\circ$, of resources, together with a *zero* resource, e; a *comparison*, $\sqsubseteq$, of resources. Mathematically, we model this set-up with a (for now, commutative) preordered monoid, $\mathcal{R} = (R, \circ, e, \sqsubseteq)$, in which $\circ$, with unit e, is functorial with respect to $\sqsubseteq$. Taking such a structure as an algebra of *worlds*, we obtain a forcing semantics for (propositional) **BI** which freely combines multiplicative (intuitionistic linear $\otimes$ and $\multimap$) and additive (intuitionistic $\wedge$, $\rightarrow$ and $\vee$) structure. A significant variation takes classical additives instead. **BI** is described in necessary detail in § 2. For now, the key property of the semantics is the *sharing interpretation* [10].

The (elementary) semantics of the multiplicative conjunction, $m \models \phi_1 * \phi_2$ iff there are n_1 and n_2 such that $m \sqsubseteq n_1 \circ n_2$, $n_1 \models \phi_1$ and $n_2 \models \phi_2$, is interpreted as follows: the resource m is sufficient to support $\phi_1 * \phi_2$ just in case it can be divided into resources n_1 and n_2 such that n_1 is sufficient to support ϕ_1 and n_2 is sufficient to support ϕ_2. The assertions ϕ_1 and ϕ_2 – think of them as expressing properties of programs — *do not share* resources. In contrast, in the semantics of the additive conjunction, $m \models \phi_1 \wedge \phi_2$ iff $m \models \phi_1$ and $m \models \phi_2$, the assertions ϕ_1 and ϕ_2 *share* the resource m. Similarly, the semantics of the multiplicative implication, $m \models \phi \mathrel{-\!\!*} \psi$ iff for all n such that $n \models \phi$, $m \circ n \models \psi$, is interpreted as follows: the resource m is sufficient to support $\phi \mathrel{-\!\!*} \psi$ – think of the proposition as (the type of) a function – just in case for any resource n which is sufficient to support ϕ – think of it as the argument to the function – the combination $m \circ n$ is sufficient to support ψ. The function and its argument *do not share* resources. In contrast, in the semantics of additive implication, $m \models \phi \rightarrow \psi$ iff for all $n \sqsubseteq m$, if $n \models \phi$, then $n \models \psi$, the function and its argument *share* the resource n. For a simple example of resource as *cost*, let the monoid be given by the natural numbers with addition and unit zero, ordered by less than or equals. A more substantial example, "pointer logic", **PL**, and its spatial semantics, has been provided by Ishtiaq and O'Hearn [8]. In fact, the semantics of pointer logic is based on *partial monoids*, in which the operation $\circ$ is partially defined.

An elementary Kripke resource semantics, formulated in categories of *presheaves* on preordered monoids, has been defined for **BI** [10, 12, 13] but it is sound and complete only for **BI** without inconsistency, $\perp$, the unit of the additive disjunction. This elementary forcing semantics handles inconsistency only by denying the existence of a world at which $\perp$ is forced. The completeness of **BI** *with* $\perp$ for a monoid-based forcing semantics is achieved, firstly, in categories of *sheaves* on *open topological monoids* [10, 13, 14] and, secondly, in the more abstract topological setting of *Grothendieck sheaves* on preordered monoids [13, 14]. This latter more general semantics is sketched in § 2. In each of these cases, inconsistency is internalized in the semantics. The semantics of pointer logic can be incorporated into the Kripke semantics based on Grothendieck sheaves [13, 14]. But it suggests partial monoids as a basis for a "Kripke resource semantics".

BI provides a logical analysis of a basic notion of resource [13], quite different from linear logic's "number-of-uses" reading, which has proved rich enough to

provide both intuitionistic and classical (*i.e.*, additives) "pointer logic" semantics for programs which manipulate mutable data structures [8, 9, 14]. In this context, efficient and useful proof-search methods are necessary. For many logics, semantic tableaux have provided elegant and efficient bases for tools based on both proof-search and countermodel generation [2]. We should like to have bases for such tools for **BI** and **PL**. The main difficulty to be overcome in giving such a system for **BI** is the presence of multiplicatives. We need a mechanism for calculating the distribution of "resources" with multiplicative rules which, in **BI**'s sequent calculus, given in § 2, is handled via side-formulæ. A solution is a specific use of *labels* that allow the capture of the semantical relationships between connectives during proof-search or proof-analysis [1, 3, 5].

Recent work has proposed a tableaux calculus, with labels, for $\mathbf{BI}^{\perp}$, *i.e.*, **BI** without $\perp$, which captures the elementary Kripke resource semantics [4] but an open question until now has been whether a similar approach or calculus can be extended to full **BI**, including $\perp$, and thus provide a decision procedure for **BI** (decidability of **BI** has been conjectured, via a different method, in [13] but not explicitly proved). A real difficulty lies in the treatment of a monoid-based forcing semantics, like Grothendieck topological semantics [13], with such a labelled calculus. In § 3, we define a system of labelled semantic tableaux, **TBI**, in which the labels are drawn from **BI**'s algebra of worlds and which use **BI**'s forcing semantics, based on Grothendieck sheaves. The rules are similar to the ones of [4] but the specific way to deal with $\perp$ topologically involves delicate new closure and provability conditions. We obtain, in § 4, soundness and completeness theorems for **TBI** with respect to the Grothendieck topological semantics given in § 2. Moreover, we use our completeness proof to show that in the case of a *failed* tableau, *i.e.*, non-provability, we can *construct a countermodel* from a particular structure, called a *dependency graph*. Consequently, we obtain proofs of two *new* results for **BI**, namely, the *finite model property* with respect to Grothendieck topological semantics and the *decidability* for propositional **BI**, conjectured but not proved in [13]. Moreover, observing that a dependency graph only deals with the relevant resources needed to decide provability, we propose, in § 5, a new resource semantics for **BI** that corresponds to an alternative way of dealing with $\perp$ by considering partially defined monoids. This way was mentioned but not developed in [13, 14] and thus this new resource semantics, which generalizes the semantics of pointer logic [8], is complete and naturally derived from our study of resource tableaux. The identified relationships between resources, labels, dependency graphs, proof-search and resource semantics are also essential. For instance, dependency graphs are directly countermodels in this new semantics.

2 The Semantics and Proof Theory of BI

We review briefly the semantics and proof theory of **BI** (with $\perp$). The details are in [13, 14]. There is an elementary Kripke resource semantics which, because of the interaction between $-\!\!*$ and $\perp$ [13, 14], is complete only for $\mathbf{BI}^{\perp}$. In

order to have completeness with $\bot$, it is necessary to use the topological setting introduced in [13, 14, 10] and described below, which is a significant step over the elementary case.

Definition 1 (GTM). *A* Grothendieck topological monoid *(GTM) is given by a quintuple* $\mathcal{M} = \langle M, \circ, e, \sqsubseteq, J \rangle$*, where* $\langle M, \circ, e, \sqsubseteq \rangle$ *is a preordered commutative monoid, in which* $\circ$ *is functorial w.r.t.* $\sqsubseteq$*, and* J *is a map* $J : M \to \wp(\wp(M))$ *satisfying the following:*

1. Sieve: *for any* $m \in M$*,* $S \in J(m)$ *and* $m' \in S$*,* $m \sqsubseteq m'$*;*
2. Maximality: *for any* n' *such that* $n' = n$*,* $\{n'\}$ *is in* $J(n)$*;*
3. Stability: *for any* $m, n \in M$ *and* $S \in J(m)$ *such that* $m \sqsubseteq n$*, there exists* $S' \in J(n)$ *such that for any* $n' \in S'$*, there exists* $m' \in S$ *such that* $m' \sqsubseteq n'$*;*
4. Transitivity: *for any* $m \in M$*,* $S \in J(m)$ *and* $\{S_{m'} \in J(m')\}_{m' \in S}$*,* $\bigcup_{m' \in S} S_{m'} \in J(m)$*;*
5. Continuity: *for any* $m, n \in M$ *and* $S \in J(m)$*,* $\{m' \circ n \mid m' \in S\} \in J(m \circ n)$*.*

Such a J *is usually called a* Grothendieck topology*.*

Definition 2 (GTI). *Let* $\mathcal{M}$ *be a GTM and* $\mathcal{P}(L)$ *be the collection of* **BI** *propositions over a language* L *of propositional letters, a* Grothendieck Topological Interpretation *is a function* $[\![-]\!] : L \to \wp(M)$ *satisfying:*

6. *(K): for any* $m, n \in M$ *such that* $n \sqsubseteq m$*,* $n \in [\![\mathrm{p}]\!]$ *implies* $m \in [\![\mathrm{p}]\!]$*;*
7. *(Sh): for any* $m \in M$ *and* $S \in J(m)$*, if, for all* $m' \in S$*,* $m' \in [\![\mathrm{p}]\!]$*, then* $m \in [\![\mathrm{p}]\!]$*.*

It is shown in [13, 14] that given an interpretation which makes (K) and (Sh) hold for atomic propositions, (K) and (Sh) also hold for any proposition of **BI** in that interpretation.

Definition 3 (GRM). *A* Grothendieck resource model *(GRM) is a triple* $\mathcal{G} = \langle \mathcal{M}, \models, [\![-]\!] \rangle$ *in which* $\mathcal{M} = \langle M, \circ, e, \sqsubseteq, J \rangle$ *is a GTM,* $[\![-]\!]$ *is a GTI and* $\models$ *is a forcing relation on* $M \times \mathcal{P}(L)$ *satisfying the following conditions:*

- $m \models \mathrm{p}$ *iff* $m \in [\![\mathrm{p}]\!]$
- $m \models \top$ *iff always*
- $m \models \bot$ *iff* $\emptyset \in J(m)$
- $m \models \phi \wedge \psi$ *iff* $m \models \phi$ *and* $m \models \psi$
- $m \models \phi \vee \psi$ *iff there exists* $S \in J(m)$ *such that for any* $m' \in S$*,* $m' \models \phi$ *or* $m' \models \psi$
- $m \models \phi \to \psi$ *iff for any* $n \in M$ *such that* $m \sqsubseteq n$*, if* $n \models \phi$*, then* $n \models \psi$
- $m \models \top$ *iff there exists* $S \in J(m)$ *such that for any* $m' \in S$*,* $e \sqsubseteq m'$
- $m \models \phi * \psi$ *iff there exists* $S \in J(m)$ *such that for any* $m' \in S$*, there exist* $n_\phi, n_\psi \in M$ *such that* $n_\phi \circ n_\psi \sqsubseteq m'$*,* $n_\phi \models \phi$ *and* $n_\psi \models \psi$
- $m \models \phi \mathrel{-\!\!*} \psi$ *iff for any* $n \in M$ *such that* $n \models \phi$*,* $m \circ n \models \psi$*.*

We make the following important remark which will prove useful later: if a world m is *inconsistent*, *i.e.*, is such that $m \models \bot$, then, by the continuity axiom of J, for any world n, $m \circ n$ is also inconsistent.

Definition 4. *Bunches are given by the grammar:* $\Gamma ::= \phi \mid \emptyset_a \mid \Gamma\,;\Gamma \mid \emptyset_m \mid \Gamma\,,\Gamma$. *Equivalence,* $\equiv$, *is given by commutative monoid equations for "," and ";", whose units are* $\emptyset_m$ *and* $\emptyset_a$ *respectively, together with the evident substitution congruence for sub-bunches – we write* $\Gamma(\Delta)$ *to denote a sub-bunch* Δ *of* Γ *– determined by the grammar.*

Let $\mathcal{G}$ be a *GRM* and ϕ_Γ be the formula obtained from a bunch Γ by replacing each ";" by $\wedge$ and each "," by $*$ with association respecting the tree structure of Γ. A sequent $\Gamma \vdash \phi$ is said to be *valid in* $\mathcal{G}$, written $\Gamma \models_{\mathcal{G}} \phi$, if and only if, for any world $m \in M$, $m \models \phi_\Gamma$ implies $m \models \phi$. A sequent $\Gamma \vdash \phi$ is *valid*, written $\Gamma \models \phi$, iff, for any *GRM* $\mathcal{G}$, it is valid in $\mathcal{G}$.

Definition 5 (LBI). **BI***'s sequent calculus,* **LBI***, is defined as follows:*

$$\frac{}{\phi \vdash \phi}\ Axiom \qquad \frac{\Gamma \vdash \phi \quad \Delta(\phi) \vdash \psi}{\Delta(\Gamma) \vdash \psi}\ Cut \qquad \frac{\Gamma(\Delta) \vdash \phi}{\Gamma(\Delta;\Delta') \vdash \phi}\ W \qquad \frac{\Gamma(\Delta;\Delta) \vdash \phi}{\Gamma(\Delta) \vdash \phi}\ C$$

$$\frac{}{\Gamma(\bot) \vdash \phi}\ \bot L \quad \frac{\Gamma \vdash \phi}{\Delta \vdash \phi}\ (\Gamma \equiv \Delta)\ E \quad \frac{\Gamma(\emptyset_m) \vdash \phi}{\Gamma(I) \vdash \phi}\ I\,L \qquad \frac{}{\emptyset_m \vdash I}\ I\,R$$

$$\frac{\Gamma(\emptyset_a) \vdash \phi}{\Gamma(\top) \vdash \phi}\ \top L \qquad \frac{}{\emptyset_a \vdash \top}\ \top R \quad \frac{\Gamma \vdash \phi \quad \Delta(\Delta',\psi) \vdash \chi}{\Delta(\Delta',\Gamma,\phi \mathrel{-\!\!*} \psi) \vdash \chi}\ \mathrel{-\!\!*} L \qquad \frac{\Gamma,\phi \vdash \psi}{\Gamma \vdash \phi \mathrel{-\!\!*} \psi}\ \mathrel{-\!\!*} R$$

$$\frac{\Gamma(\phi,\psi) \vdash \chi}{\Gamma(\phi * \psi) \vdash \chi}\ * L \qquad \frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma,\Delta \vdash \phi * \psi}\ * R \qquad \frac{\Gamma \vdash \phi \quad \Delta(\Delta';\psi) \vdash \chi}{\Delta(\Delta';\Gamma;\phi \to \psi) \vdash \chi}\ \to L \qquad \frac{\Gamma;\phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ \to R$$

$$\frac{\Gamma(\phi_1;\phi_2) \vdash \psi}{\Gamma(\phi_1 \wedge \phi_2) \vdash \psi}\ \wedge L \qquad \frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma;\Delta \vdash \phi \wedge \psi}\ \wedge R \qquad \frac{\Gamma(\phi) \vdash \chi \quad \Delta(\psi) \vdash \chi}{\Gamma(\phi \vee \psi);\Delta(\phi \vee \psi) \vdash \chi}\ \vee L \qquad \frac{\Gamma \vdash \phi_i\ (i=1,2)}{\Gamma \vdash \phi_1 \vee \phi_2}\ \vee R.$$

A proposition ϕ is a *theorem* of **LBI** iff $I \vdash \phi$. The Cut-elimination theorem holds for **LBI** [13]. Moreover, *soundness* and *completeness*, via a term model construction and a Hilbert-type system for **BI**, with respect to GRMs are proved in [13, 14]. As a corollary, we obtain *validity*, *i.e.*, a proposition ϕ is valid iff for any *GRM* $\mathcal{G}$, $e \models_{\mathcal{G}} \phi$.

3 Resource Tableaux for BI

We set up the theory of labelled semantic tableaux for **BI**. We assume a basic knowledge of tableaux systems [2]. We begin with algebras of labels, which provide the connection between the underlying syntactic tableaux and the semantics of the connectives used to regulate the multiplicative structure. In the case of $\mathbf{BI}^{\bot}$, we can provide an algebra which syntactically reflects the elementary semantics [4]. For **BI** and its Grothendieck topological semantics, the analysis is more delicate. A key step in this semantic analysis is the use of *dependency graphs*, explained in § 3.3.

3.1 A Labelling Algebra

We define a set of labels and constraints and a corresponding labelling algebra, *i.e.*, a preordered monoid whose elements are denoted by labels.

Definition 6. *A* labelling language *consists of the following symbols: a unit symbol* 1*, a binary function symbol* $\circ$*, a binary relation symbol* $\leq$*, a countable set of constants* $c_1, c_2, \ldots$ *.* Labels *are inductively defined from the unit* 1 *and the constants as expressions of the form* $x \circ y$ *in which* x *and* y *are labels.* Atomic labels *are labels which do not contain any* $\circ$*, while* compound labels *contain at least one* $\circ$*.* Label constraints *are expressions of the form* $x \leq y$*, where* x *and* y *are labels.*

Definition 7. *Labels and constraints are interpreted in an order-preserving preordered commutative monoid of labels, or* labelling algebra $\mathcal{L} = \langle L, \circ, 1, \leq \rangle$*, more precisely: 1.* L *is a set of labels; 2.* $\leq$ *is a preorder; 3. Equality on labels is defined by :* $x = y$ *iff* $x \leq y$ *and* $y \leq x$*; 4.* $\circ$ *is a binary operation on* L *such that: associativity:* $(x \circ y) \circ z = x \circ (y \circ z)$*, commutativity:* $x \circ y = y \circ x$*, identity:* $x \circ 1 = 1 \circ x = x$*, compatibility:* $x \circ z \leq y \circ z$ *if* $x \leq y$*. We say that* x *is a* sublabel *of* y *(notation:* $x \preceq y$*), if there exists a label* z *such that* $y = x \circ z$*. We say* $x \prec y$ *if* $x \preceq y$ *and* $x \neq y$*.* $\wp(x)$ *denotes the set of the sublabels of* x*.*

For notational simplicity, we can omit the binary symbol $\circ$ when writing labels. We deal with *partially defined* labelling algebras, obtained from sets of constraints by means of a *closure operator*.

Definition 8. *The* domain *of a set* K *of label constraints is the set of all sublabels occurring in some constraints of* K*,* i.e.*,* $\mathcal{D}(K) = \bigcup_{x \leq y \in K} (\wp(x) \cup \wp(y))$*. The* closure $\overline{K}$ *of* K *is defined as follows:*

1. $K \subseteq \overline{K}$*;*
2. *Reflexivity: if* $x \in \mathcal{D}(\overline{K})$*, then* $x \leq x \in \overline{K}$*;*
3. *Transitivity: if* $x \leq y \in \overline{K}$ *and* $y \leq z \in \overline{K}$*, then* $x \leq z \in \overline{K}$*;*
4. *Compatibility: if* $x \circ z$ *or* $y \circ z \in \mathcal{D}(\overline{K})$*, then* $x \leq y \in \overline{K}$ *implies* $x \circ z \leq y \circ z \in \overline{K}$*.*

We do not distinguish between the closure of a set of label constraints and the (partially defined) labelling algebra it generates.

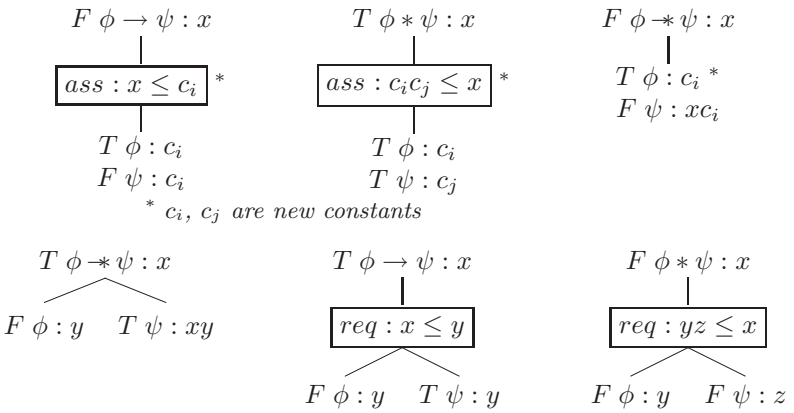
3.2 Expansion Rules

We can now define the expansion rules of **TBI**.

Definition 9. *A* signed formula *is a triple* $\langle Sg, \phi, l \rangle$*, denoted* $Sg\ \phi : l$*,* Sg *(*$\in \{F, T\}$*) being the sign of the formula* ϕ *(*$\in \mathcal{P}(L)$*) and* l *(*$\in \mathcal{L}$*) its label.*

Definition 10 (TBI). *A* **TBI** tableau t *is a rooted tree whose nodes are labelled with a signed formula and built according to the following* expansion rules*:*

$$\begin{array}{c} F\ \phi \vee \psi : x \\ | \\ F\ \phi : x \\ F\ \psi : x \end{array} \qquad \begin{array}{c} T\ \phi \vee \psi : x \\ \diagup \quad \diagdown \\ T\ \phi : x \quad T\ \psi : x \end{array} \qquad \begin{array}{c} T\ \phi \wedge \psi : x \\ | \\ T\ \phi : x \\ T\ \psi : x \end{array} \qquad \begin{array}{c} F\ \phi \wedge \psi : x \\ \diagup \quad \diagdown \\ F\ \phi : x \quad F\ \psi : x \end{array}$$



Given a tableau branch $\mathcal{B}$, $\mathcal{F}(\mathcal{B})$ denotes the set of all its signed formulæ. Moreover, $\mathcal{B}$ is associated two particular sets of label constraints, $Ass(\mathcal{B})$, with elements "ass", and $Req(\mathcal{B})$, with elements "req", that are, respectively, the set of its *assertions* and of its *requirements*, or *obligations*. The *domain* $\mathcal{D}(\mathcal{B})$ of a branch $\mathcal{B}$ is the set of all sublabels occuring in its assertions, *i.e.*, $\mathcal{D}(\mathcal{B}) = \mathcal{D}(Ass(\mathcal{B}))$. $C(\mathcal{B})$ is the subset of all constants of $\mathcal{D}(\mathcal{B})$. The notation for branches extends to tableaux as follows: $f(t) = \bigcup_{\mathcal{B}\in t} f(\mathcal{B})$, where f is one of $\mathcal{F}$, Ass, Req, $\mathcal{D}$ or C.
The rules for $\wedge$ and $\vee$ are the usual α,β ones. Those introducing *assertions* including $F{\twoheadrightarrow}$ for which the assertion $c_i \leq c_i$ is implicitly assumed are called $\pi\alpha$ and those introducing *requirements* are called $\pi\beta$. Notice also that $\pi\alpha$ rules create new (atomic) labels while $\pi\beta$ reuse existing ones.

Definition 11. *Let ϕ be a* **BI** *proposition. A* tableau sequence for ϕ *is a sequence of tableaux $t_1, t_2, \ldots$ for which t_1 is the one-node tree defined by $\mathcal{F}(t_1) = \{F\ \phi : 1\}$, $Ass(t_1) = \{1 \leq 1\}$, $Req(t_1) = \emptyset$ and t_{i+1} is obtained from t_i by applying, on a branch of t_i, an expansion rule of Definition 10.*

Definition 12. *Two signed formulæ $T\ \phi : x$, $F\ \phi : y$ are* complementary *in a branch $\mathcal{B}$ if and only if $x \leq y \in \overline{Ass}(\mathcal{B})$, i.e., iff the constraint $x \leq y$ belongs to the reflexive, transitive and compatible closure of the assertions of $\mathcal{B}$.*

So far, the definitions, as well as the expansion rules, were exactly the same as the one presented in [4] for **BI** without $\perp$ and its elementary semantics. As we mentioned in the introduction, we aim to address the problem of inconsistency while keeping as much of the initial tableau system as possible. Therefore, we will not derive new expansion rules for $\perp$, rather, we will extend the definition of a closed tableau with an additional condition which takes the specificity of $\perp$ into account. This new condition introduces the notion of an *inconsistent* label which syntactically reflects the fact that Grothendieck models may have several worlds at which $\perp$ is forced and, as noticed in the remark following Definition 3, that compositions with such worlds are themselves inconsistent.

The crucial point here is that, since the case of $\bot$ is handled via the closure rule *solely by considerations on the labels*, proving a formula of full propositional **BI**, compared to **BI** without $\bot$, is only a matter of deciding when a branch containing $\bot$ should be considered closed or not, but the procedures which actually build the tableaux, dependency graphs, as well as the related properties (termination, finiteness) remain unchanged.

3.3 Resource Tableaux with $\bot$ and Dependency Graphs

Definition 13. *Let $\mathcal{B}$ be a branch, a label x is* inconsistent *in $\mathcal{B}$ if there exists a label y such that $y \leq x \in \overline{Ass}(\mathcal{B})$ and a label z in $\wp(y)$ (set of sub-labels of y) such that $T \bot : z$ occurs in $\mathcal{B}$. A label x is* consistent *in $\mathcal{B}$ if it is not inconsistent.*

Definition 14. *A tableau t is* closed *if, for all its branches $\mathcal{B}$, the following conditions are satisfied :*
(i) 1. there are two formulæ $T\ \phi : x$ and $F\ \phi : y$ that are complementary in $\mathcal{B}$, or 2. there is $F \top : x$ in $\mathcal{B}$, or 3. there is $F\ I : x$ in $\mathcal{B}$ with $1 \leq x \in \overline{Ass}(\mathcal{B})$, or 4. there is $T\ I : x$ in $\mathcal{B}$ with $1 \leq x \notin \overline{Ass}(\mathcal{B})$, or 5. there is $F\ \phi : x$ in $\mathcal{B}$ with x inconsistent in $\mathcal{B}$;
(ii) $\forall x \leq y \in Req(\mathcal{B})$, $x \leq y \in \overline{Ass}(\mathcal{B})$.
A tableau sequence $t_1, t_2, \ldots$ is closed *if it contains a closed tableau.*

A specific graph, called the dependency graph or Kripke resource graph, is built in parallel with the tableau expansion. It reflects the information that can be derived from a given set of assertions.

Definition 15. *Given a tableau branch $\mathcal{B}$, the associated dependency graph $DG(\mathcal{B}) = [N(\mathcal{B}), A(\mathcal{B})]$ is defined as the following directed graph: the set of nodes $N(\mathcal{B})$ is the set of labels $\mathcal{D}(\mathcal{B})$ and the set of arrows $A(\mathcal{B})$ is built from the set of assertions $\overline{Ass}(\mathcal{B})$ as follows: there is an arrow $x \rightarrow y$ in $A(\mathcal{B})$ iff there is an assertion $x \leq y$ in $\overline{Ass}(\mathcal{B})$.*

We can formally define a procedure that builds, in parallel with tableau expansions, the dependency graph $DG(\mathcal{B})$ of a branch $\mathcal{B}$ and so, the closure $\overline{Ass}(\mathcal{B})$. The expansion rules of a dependency graph are such that the given graph is only expanded by the $\pi\alpha$ rules, all the other rules, introducing neither new constants, nor new assertions, simply leave it unchanged. On a dependency graph $DG(\mathcal{B})$, the fact that a requirement $x \leq y$ holds with respect to $\overline{Ass}(\mathcal{B})$ corresponds to the existence of a path from the node x to the node y. We illustrate this point with two examples.

Figure 1 shows a closed tableau for the formula $((\mathrm{p} \mathrel{-\!\!*} \bot) * \mathrm{p}) \rightarrow \mathrm{q}$, which is therefore provable in **BI**. We remark that we reach, after step 3, a tableau with two branches. The first branch is closed since it contains complementary formulæ, namely, $\langle T\ \mathrm{p} : c_3, F\ \mathrm{p} : c_3 \rangle$. The second, however, contains no complementary formulæ. It is the point were the closure condition plays its role. We notice that the branch contains the formula $T \bot : c_2c_3$. Thus, c_2c_3 is what we have called an inconsistent label and, by assertion $ass_2 : c_2c_3 \leq c_1$, c_1 is also inconsistent.

Therefore, the branch is closed because it contains the formula $F\ \mathrm{q} : c_1$ with label c_1 being inconsistent.

The second example, *q.v.* Figure 2, leads to an unclosed tableau for the formula $((\mathrm{p} \mathrel{-\!\!*} \bot) \to \bot) \mathrel{-\!\!*} (((\mathrm{p} * \mathrm{p}) \mathrel{-\!\!*} \bot) \to \bot)$ which is therefore unprovable. After step 6, the tableau is completed and we are left with four branches to close. The second one is closed with $\langle T\ \mathrm{p} : c_3, F\ \mathrm{p} : c_3 \rangle$, the third is closed with $\langle T \bot : c_2c_3, F \bot : c_2c_3 \rangle$ and the fourth is closed with $\langle T \bot : c_2, F \bot : c_2 \rangle$. The first branch, on the contrary, remains open since the only way to close it would be to have $\langle T\ \mathrm{p} : c_3, F\ \mathrm{p} : 1 \rangle$, but $c_3 \leq 1$ cannot be deduced from the assertions of the branch. We will see in a next section how to build a countermodel from such an open branch.

We now show that this labelled calculus, whose restriction to $\mathbf{BI}^{\bot}$ is complete for the elementary semantics, is complete for **BI** with respect to the Grothendieck topological semantics.

4 Completeness of the TBI Calculus

We show the soundness and completeness of **TBI** with respect to GRMs. This deductive framework allows not only a proof procedure but also, in the case of non-provability, the systematic generation of countermodels.

4.1 Soundness

Soundness is proved in a classical way, subject to the usual adaptations to **BI** [13, 14], from a notion of *realizability* that is preserved by the expansion rules [4].

Definition 16. *Let* $\mathcal{G} = \langle \mathcal{M}, \models, [\![-]\!] \rangle$ *be a GRM and* $\mathcal{B}$ *be a tableau branch, a* realization of $\mathcal{B}$ in $\mathcal{G}$ *is a mapping* $\|-\| : \mathcal{D}(\mathcal{B}) \to M$*, from the domain of* $\mathcal{B}$

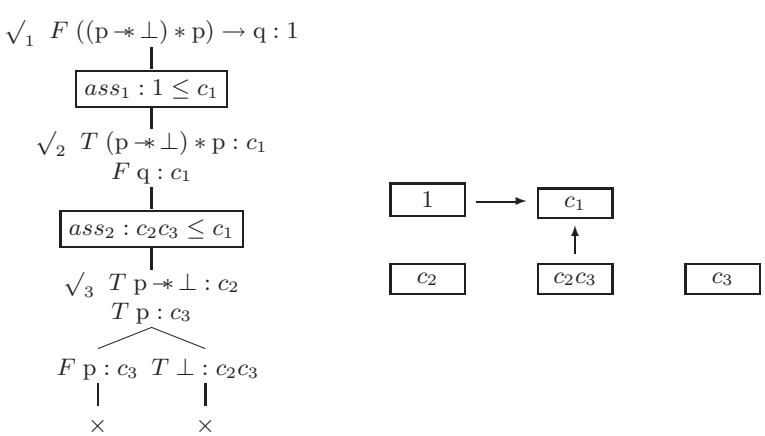


Fig. 1. Tableau and dependency graph for $((\mathrm{p} \mathrel{-\!\!*} \bot) * \mathrm{p}) \to \mathrm{q}$

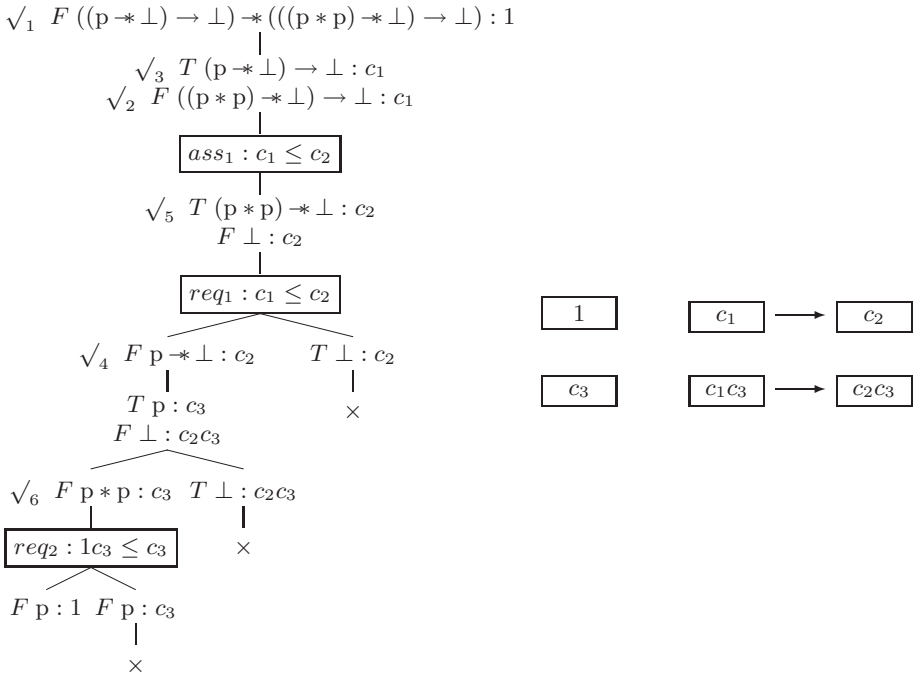


Fig. 2. Tableau and dependency graph for $((\mathrm{p} \mathrel{-\!\!*} \bot) \to \bot) \mathrel{-\!\!*} (((\mathrm{p} * \mathrm{p}) \mathrel{-\!\!*} \bot) \to \bot)$

to the worlds of M, that satisfies 1. $\|1\| = e$, 2. $\|x \circ y\| = \|x\| \circ \|y\|$, 3. for any $T\ \phi : x$ in $\mathcal{B}$, $\|x\| \models \phi$, 4. for any $F\ \phi : x$ in $\mathcal{B}$, $\|x\| \not\models \phi$, 5. for any $x \leq y$ in $Ass(\mathcal{B})$, $\|x\| \sqsubseteq \|y\|$.

Lemma 1. *Let t be a tableau, $\mathcal{B}$ a branch of t and $\|-\|$ a realization of $\mathcal{B}$ in a GRM $\mathcal{G}$. Then, for any $x \leq y \in \overline{Ass}(\mathcal{B})$, $\|x\| \sqsubseteq \|y\|$ holds in $\mathcal{G}$.*

Definition 17. *A tableau branch $\mathcal{B}$ is* realizable *if there exists a realization of $\mathcal{B}$ in some GRM $\mathcal{G}$. A tableau t is* realizable *if it contains a realizable branch.*

Lemma 2. *A closed tableau is not realizable.*

Proof Let t be a closed tableau that is also realizable. Then, t contains a branch $\mathcal{B}$ which is realizable in some *GRM* $\mathcal{G} = \langle \mathcal{M}, \models, [\![-]\!] \rangle$. If the branch is closed because of complementary formulæ $\langle T\ \phi : x, F\ \phi : y \rangle$ then, by definition, we have $x \leq y \in \overline{Ass}(\mathcal{B})$ which, by Lemma 1, implies $\|x\| \sqsubseteq \|y\|$. But, since $\|-\|$ realizes $\mathcal{B}$, we also have $\|x\| \models \phi$ and $\|y\| \not\models \phi$. Therefore, we reach a contradiction because, by property (K), we should have $\|y\| \models \phi$. If the branch is closed because of a formula $F\ \phi : x$, whose label x is inconsistent in $\mathcal{B}$, then, by definition, there exists a label y such that $y \leq x \in \overline{Ass}(\mathcal{B})$ and a label z in $\wp(y)$ such that $T\ \bot : z \in \mathcal{B}$. Since $\|-\|$ realizes $\mathcal{B}$ we have $x \not\models \phi$ and $z \models \bot$. Since

z is a sublabel of y, the continuity axiom of J implies that $y \models \bot$. Therefore, as Lemma 1 implies $\|y\| \sqsubseteq \|x\|$, (K) yields $x \models \bot$ and, once again, we reach a contradiction because, if $x \models \bot$ then, for any ϕ, we should have $x \models \phi$. Other cases are similar. □

Theorem 1 (soundness). *Let ϕ be a proposition of* **BI**. *If there exists a closed tableau sequence $\mathcal{T}$ for ϕ, then ϕ is valid in Grothendieck topological semantics.*

4.2 Countermodel Construction

We describe how to construct a countermodel of ϕ from an open branch in a tableau for ϕ. We obtain the finite model property and decidability for **BI**. The proof of the finite model property relies critically on the introduction of a special element, here called π, used to collect the inessential (and possibly infinite) parts of the model.

Definition 18. *Let $\mathcal{B}$ be a tableau branch. A signed formula $Sg\ X : x$ is* fulfilled, *or* completely analysed, *in $\mathcal{B}$, denoted $\mathcal{B} \Vdash Sg\ X : x$, if it satisfies one of the following conditions:*

1. $\mathcal{B} \Vdash F\ \bot : x$;
2. $\mathcal{B} \Vdash F\ I : x$ *iff* $1 \leq x \notin \overline{Ass}(\mathcal{B})$;
3. $\mathcal{B} \Vdash F\ \mathrm{p} : x$ *iff there is* $F\ \mathrm{p} : y \in \mathcal{B}$ *s.t.* $x \leq y \in \overline{Ass}(\mathcal{B})$;
4. $\mathcal{B} \Vdash F\ \phi \wedge \psi : x$ *iff* $\mathcal{B} \Vdash F\ \phi : x$ *or* $\mathcal{B} \Vdash F\ \psi : x$;
5. $\mathcal{B} \Vdash F\ \phi \vee \psi : x$ *iff* $\mathcal{B} \Vdash F\ \phi : x$ *and* $\mathcal{B} \Vdash F\ \psi : x$;
6. $\mathcal{B} \Vdash F\ \phi \rightarrow \psi : x$ *iff there is* $y \in \mathcal{D}(\mathcal{B})$ *s.t.* $x \leq y \in \overline{Ass}(\mathcal{B})$ *and both* $\mathcal{B} \Vdash T\ \phi : y$ *and* $\mathcal{B} \Vdash F\ \psi : y$;
7. $\mathcal{B} \Vdash F\ \phi * \psi : x$ *iff, for any* $y, z \in \mathcal{D}(\mathcal{B})$ *s.t.* $yz \leq x \in \overline{Ass}(\mathcal{B})$, $\mathcal{B} \Vdash F\ \phi : y$ *or* $\mathcal{B} \Vdash F\ \psi : z$;
8. $\mathcal{B} \Vdash F\ \phi \mathrel{-\!\!*} \psi : x$ *iff there exist* $y, xy \in \mathcal{D}(\mathcal{B})$ *s.t. both* $\mathcal{B} \Vdash T\ \phi : y$ *and* $\mathcal{B} \Vdash F\ \psi : xy$;
9. $\mathcal{B} \Vdash T\ \top : x$;
10. $\mathcal{B} \Vdash T\ I : x$ *iff* $1 \leq x \in \overline{Ass}(\mathcal{B})$;
11. $\mathcal{B} \Vdash T\ \mathrm{p} : x$ *iff there is* $T\ \mathrm{p} : y \in \mathcal{B}$ *s.t.* $y \leq x \in \overline{Ass}(\mathcal{B})$;
12. $\mathcal{B} \Vdash T\ \phi \wedge \psi : x$ *iff both* $\mathcal{B} \Vdash T\ \phi : x$ *and* $\mathcal{B} \Vdash T\ \psi : x$;
13. $\mathcal{B} \Vdash T\ \phi \vee \psi : x$ *iff* $\mathcal{B} \Vdash T\ \phi : x$ *or* $\mathcal{B} \Vdash T\ \psi : x$;
14. $\mathcal{B} \Vdash T\ \phi \rightarrow \psi : x$ *iff, for any* $y \in \mathcal{D}(\mathcal{B})$ *s.t.* $x \leq y \in \overline{Ass}(\mathcal{B})$, $\mathcal{B} \Vdash F\ \phi : y$ *or* $\mathcal{B} \Vdash T\ \psi : y$;
15. $\mathcal{B} \Vdash T\ \phi * \psi : x$ *iff there are* $y, z \in \mathcal{D}(\mathcal{B})$ *s.t.* $yz \leq x \in \overline{Ass}(\mathcal{B})$ *and both* $\mathcal{B} \Vdash T\ \phi : y$ *and* $\mathcal{B} \Vdash T\ \psi : z$;
16. $\mathcal{B} \Vdash T\ \phi \mathrel{-\!\!*} \psi : x$ *iff, for any* $y, xy \in \mathcal{D}(\mathcal{B})$, $\mathcal{B} \Vdash F\ \phi : y$ *or* $\mathcal{B} \Vdash T\ \psi : xy$.

Lemma 3. *Let $\mathcal{B}$ be a tableau branch. The property of being fulfilled, given in Definition 18, satisfies Kripke monotonicity, i.e., (i) $\mathcal{B} \Vdash F\ \phi : x$ and $y \leq x \in \overline{Ass}(\mathcal{B})$ imply $\mathcal{B} \Vdash F\ \phi : y$, and (ii) $\mathcal{B} \Vdash T\ \phi : x$ and $x \leq y \in \overline{Ass}(\mathcal{B})$ imply $\mathcal{B} \Vdash T\ \phi : y$.*

Definition 19. *A tableau branch $\mathcal{B}$ is* completed *if any signed formula $Sg\ \phi : x$ in $\mathcal{B}$ is fulfilled. A tableau is* completed *if it has a branch that is completed. A tableau branch $\mathcal{B}$ is an* H-branch *if it is open and completed.*

Lemma 4. *If $\mathcal{B}$ is an H-branch then, for any proposition ϕ, not both $\mathcal{B} \Vdash T\ \phi : x$ and $\mathcal{B} \Vdash F\ \phi : x$.*

The dependency graph related to a formula ϕ during the resource tableau construction represents the closure of the assertions in the sense of Definition 8 and so captures the computational content of ϕ. Therefore, if a formula ϕ happens to be unprovable, we should have enough information in its dependency graph to extract a countermodel for ϕ. For that, we must provide a preordered commutative monoid together with a Grothendieck topology and a forcing relation which falsifies ϕ in some world. The idea behind the countermodel construction is to regard the dependency graph itself as the desired countermodel, thereby considering it as a central semantic structure. For that, we take the nodes (labels) of the graph as the elements of a monoid whose composition law is given by the composition of the labels. The preordering relation is then given by the arrows and the forcing relation simply reflects the property of being fulfilled.

The key problem is that, since the closure operator induces a *partially defined* labelling algebra, the dependency graph only deals with those pieces of information (resources) that are relevant for deciding provability. Therefore, the monoidal law should be completed with suitable values for those compositions which are undefined. The problem of undefinedness is solved in Definition 20 by the introduction of a particular element, denoted π, to which all undefined compositions are mapped and for which the equation $(\forall x)(x \circ \pi = \pi \circ x = \pi)$, meaning that any composition with something undefined is itself undefined, is assumed.

However, we must be careful because introducing a new element may affect the property of a formula $\phi \multimap\!\!* \psi$ of being realized in a world x although the signed formula $T\ \phi \multimap\!\!* \psi : x$ was fulfilled in the dependency graph. Indeed, if π forces ϕ then, since $x \circ \pi = \pi$, we also need π to force ψ. But, if π forces any formula ψ, then everything works as it should. On the other hand, we know that an inconsistent world necessarily forces any formula ψ because $\bot \vdash \psi$ is an axiom. Therefore, making π an inconsistent world by setting $\emptyset \in J(\pi)$ just solves the problem.

Definition 20 (M-structure, π). *Let $\mathcal{B}$ be an H-branch. The M-structure $\mathcal{M}(\mathcal{B}) = \langle M, \circ, 1, \sqsubseteq, J \rangle$ is defined as follows: (i) M is the subset of labels of $\mathcal{D}(\mathcal{B})$ consistent in $\mathcal{B}$, extended with a particular element π; (ii) $\circ$ is a composition law defined by*

$$x \circ 1 = 1 \circ x = x \qquad x \circ y = y \circ x = \begin{cases} xy & \text{if } xy \in M \\ \pi & \text{otherwise;} \end{cases}$$

(iii) the relation $\sqsubseteq$ between elements of M is defined by

$$x \sqsubseteq y \quad \text{iff} \quad y = x = \pi \ \text{ or } \ x \leq y \in \overline{Ass}(\mathcal{B});$$

and (iv) the map $J : M \to \wp(\wp(M))$*, called the J-map of* $\mathcal{B}$*, is defined by*

$$J(\pi) = \{\{\pi\}, \emptyset\} \quad J(x) = \{\{y\} \mid y = x\}.$$

Lemma 5. *Let* $\mathcal{B}$ *be an H-branch, the M-structure* $\mathcal{M}(\mathcal{B}) = \langle M, \circ, 1, \sqsubseteq, J \rangle$ *is a GTM,* i.e.*, (i)* $\langle M, \circ, 1, \sqsubseteq \rangle$ *is a preordered commutative monoid, and (ii)* J *is a Grothendieck topology.*

Definition 21. *Let* $\mathcal{M}(\mathcal{B}) = \langle M, \circ, 1, \sqsubseteq, J \rangle$ *be the M-structure of an H-branch* $\mathcal{B}$ *and* $\mathcal{P}(L)$ *denote the collection of* **BI** *propositions over a language* L *of propositional letters. The interpretation* $[\![-]\!]_{\mathcal{B}} : L \to \wp(M)$ *is, for any atomic proposition* p*,* $[\![\mathrm{p}]\!]_{\mathcal{B}} = \{\pi\} \cup \{x \mid \mathcal{B} \Vdash T\ \mathrm{p} : x\}$.

Lemma 6. $[\![-]\!]_{\mathcal{B}}$ *is a GTI, i.e., it satisfies properties (K) and (Sh) of Definition 2.*

Theorem 2. *Let* $\mathcal{B}$ *be an H-branch. Then* $\langle \mathcal{M}(\mathcal{B}), \models, [\![-]\!]_{\mathcal{B}} \rangle$ *is a Grothendieck resource model of* $\mathcal{B}$*, i.e., for any proposition* ϕ*, we have: (i)* $\pi \models \phi$*; (ii)* $\mathcal{B} \Vdash T\ \phi : x$ *implies* $x \models \phi$*; (iii)* $\mathcal{B} \Vdash F\ \phi : x$ *implies* $x \not\models \phi$*.*

Returning to the example of Figure 2, we show how to build a countermodel from the open branch. As the reader might check, all formulæ in the open branch are fulfilled and $\mathcal{B}$ is therefore what we have called an H-branch. Firstly, following the steps of Definition 20, we build from $\mathcal{B}$ a GTM $\mathcal{M}(\mathcal{B}) = \langle M, \circ, 1, \sqsubseteq, J \rangle$.

(i) M is the subset of labels of $\mathcal{D}(\mathcal{B})$ that are consistent, to which we add the element π, *i.e.*, $M = \{1, c_1, c_2, c_3, c_1c_3, c_2c_3, \pi\}$. Notice that, because of the presence in $\mathcal{B}$ of both the assertion $ass_1 : c_1 \leq c_2$ and of the label c_2c_3, the label c_1c_3, although not initially present in $\mathcal{B}$, is added by the closure operation in order to respect the compatibility requirement.

(ii) The multiplication $\circ$ is

$\circ$	1	c_1	c_2	c_3	c_1c_3	c_2c_3	π
1	1	c_1	c_2	c_3	c_1c_3	c_2c_3	π
c_1	c_1	π	π	c_1c_3	π	π	π
c_2	c_2	π	π	c_2c_3	π	π	π
c_3	c_3	c_1c_3	c_2c_3	π	π	π	π
c_1c_3	c_1c_3	π	π	π	π	π	π
c_2c_3	c_2c_3	π	π	π	π	π	π
π	π	π	π	π	π	π	π

(iii) The preordering relation $\sqsubseteq$ reflects the structure of the assertions $\overline{Ass}(\mathcal{B})$. If we omit implicit reflexive relations, we have two non-trivial relations, namely, $c_1 \sqsubseteq c_2$ and $c_1c_3 \sqsubseteq c_2c_3$.

(iv) The Grothendieck topology J is given by the following table:

x	1	c_1	c_2	c_3	c_1c_3	c_2c_3	π
$J(x)$	$\{\{1\}\}$	$\{\{c_1\}\}$	$\{\{c_2\}\}$	$\{\{c_3\}\}$	$\{\{c_1c_3\}\}$	$\{\{c_2c_3\}\}$	$\{\{\pi\}, \emptyset\}$

Secondly, we apply Definition 21 to the only atomic proposition p occuring in the branch $\mathcal{B}$, which leads to the GTI $[\![\mathrm{p}]\!]_{\mathcal{B}} = \{\pi, c_3\}$. This, in turn, finally gives rise to the GRM $\mathcal{G} = \langle\, \mathcal{M}(\mathcal{B}), \models, [\![-]\!]_{\mathcal{B}} \,\rangle$, the desired countermodel. Now we check that (i) $c_1 \models (\mathrm{p} \mathrel{-\!\!*} \bot) \to \bot$ and (ii) $c_1 \not\models ((\mathrm{p} * \mathrm{p}) \mathrel{-\!\!*} \bot) \to \bot$. For (i), we have $c_3 \models \mathrm{p}$ because $c_3 \in [\![\mathrm{p}]\!]_{\mathcal{B}}$ and $c_2 c_3 \not\models \bot$ because $\emptyset \notin J(c_2 c_3)$. Thus, $c_2 \not\models \mathrm{p} \mathrel{-\!\!*} \bot$ and, since $c_1 \sqsubseteq c_2$ we obtain, by (K), $c_1 \not\models \mathrm{p} \mathrel{-\!\!*} \bot$. Therefore, we have $c_1 \models (\mathrm{p} \mathrel{-\!\!*} \bot) \to \bot$. For (ii), we notice that π is the only world that forces $\mathrm{p}*\mathrm{p}$. Thus, we have $c_2 \models (\mathrm{p} * \mathrm{p}) \mathrel{-\!\!*} \bot$ only if $c_2 \circ \pi \models \bot$, which is the case because $c_2 \circ \pi = \pi$ and $\pi \models \bot$. Note that it would not be the case in the elementary semantics for which no world can force $\bot$. On the other hand, $c_2 \not\models \bot$ because $\emptyset \notin J(c_2)$. Therefore, $c_1 \not\models ((\mathrm{p} * \mathrm{p}) \mathrel{-\!\!*} \bot) \to \bot$. Then the initial formula, although valid in the elementary semantics, is not provable in **BI** .

4.3 Completeness and Finite Model Property

A tableau construction procedure is an algorithm which, given a formula ϕ, builds a tableau sequence $t_1, t_2, \ldots, t_n$ until there exists a tableau t_i which is either closed or has an H-branch: Otherwise it does not terminate. **BI** has such a procedure, with $F\ \phi : 1$ as initial formula. Until $\mathcal{T}$ is closed or completed, choose an open branch $\mathcal{B}$; if there is an unfulfilled α or $\pi\alpha$ formula $(Sg\ \phi : x)$ in $\mathcal{B}$, then apply the related expansion rule; else if there is an unfulfilled β or $\pi\beta$ formula $(Sg\ \phi : x)$ in $\mathcal{B}$, then apply the corresponding expansion rule, with all labels for which the formula is not fulfilled.

When $\pi\alpha$ formulæ are in the scope of $\pi\beta$ formulæ, the fulfillment of $\pi\alpha$ formulæ requires the introduction of new constants which may destroy the fulfillment of $\pi\beta$ formulæ. In order to ensure termination of an H-branch construction, we need to control this introduction of constants and also to detect expansion sequences that are redundant. Concerning the first point, $F\ \phi \to \psi : x$ may simply be expanded in $F\ \psi : x$ when the branch $\mathcal{B}$ already contains $T\ \phi : y$ such that $y \leq x \in \overline{Ass}(\mathcal{B})$. Similar considerations apply to $T\ \phi * \psi : x$. Concerning the second point, we have to deal with expansions of the form $F\ \phi \mathrel{-\!\!*} \psi : x$ when x already contains a constant deriving from a previous occurrence of the same signed formula. With such expansions, we can have sequences such as $F\ \phi \mathrel{-\!\!*} \psi : x$, $F\ \phi \mathrel{-\!\!*} \psi : xc$ (c being introduced from the first expansion), $F\ \phi \mathrel{-\!\!*} \psi : xcc, \ldots$ in an H-branch. Then, we have a repetition of the same branch pattern (modulo additional c) but without more computational content allowing the possibility of closing the branch. This problem is solved with a specific notion of *expansion redundancy*, already introduced in [4] for the case of $\mathbf{BI}^{\perp}$, which ensures that a so-called non-redundant tableau is obtained. With these improvements, we can transform the semi-decision procedure into a decision procedure that terminates either with a closed tableau or with a finite H-branch. From such a branch, we can build a countermodel following Definition 20, and thus prove completeness, following an approach based on proof-search [11]. Moreover, as π captures the inessential parts of the model, the construction explained in Definition 20 always results in a finite countermodel when the corresponding H-branch is finite, so yielding the finite model property.

Theorem 3 (completeness). *If $I \models \phi$, then there is a closed tableau sequence for ϕ.*

Theorem 4 (finite model property). *If $I \not\vdash \phi$, then, there is a finite Grothendieck resource model such that $I \not\models \phi$.*

Corollary 1 (decidability). *Propositional* **BI** *is decidable.*

Note that full propositional linear logic, with exponentials, is undecidable even when restricted to the intuitionistic fragment, that the status of **MELL** is unknown, and that neither has the finite model property [6, 7].
From the capture of the semantics by labels, we provide a decision procedure for **BI** which builds countermodels in Grothendieck topological semantics. Their study gives us a better understanding of the semantic information necessary to analyse provability and of the relationships between the elementary and topological settings. As a consequence, we present, in the next section, a new, powerful result about **BI**'s semantics which generalizes previous work on pointer logic.

5 A New (Complete) Resource Semantics

In § 4, we have analysed how countermodels could be built from dependency graphs. We now observe that those models are very closely related to the ones recently proposed in the semantics of "pointer logic" [8, 13]. Indeed, the Grothendieck topology described in [13] exactly corresponds to our definition of the J-map. Moreover, in our models, a special element called π is used to capture undefinedness as the image of all undefined compositions and is the only one to force $\bot$ (because $\emptyset$ only belongs to $J(\pi)$).

A consequence of the completeness result for **TBI** (see Theorem 3) is that we can always restrict to such simple Grothendieck models and so obtain the completeness of **BI** with respect to a new Kripke resource semantics that is intermediate between the elementary and Grothendieck semantics. We sketch this new semantics.

Definition 22. *A* Kripke resource monoid *(KRM) is a preordered commutative monoid $\mathcal{M} = \langle M, \circ, e, \sqsubseteq \rangle$ in which M contains an element, denoted π, such that for any $m \in M$, $\pi \circ m = \pi$ and in which $\circ$ is functorial with respect to $\sqsubseteq$.*

Definition 23. *Let $\mathcal{M}$ be a KRM and $\mathcal{P}(L)$ be a language of* **BI** *propositions over a language L of propositional letters. Then, a* Kripke resource interpretation*, or KRI, is a function $[\![-]\!] : L \to \wp(M)$ satisfying Kripke monotonicity and such that for any $\mathrm{p} \in L$, $\pi \in [\![\mathrm{p}]\!]$.*

Definition 24. *A* Kripke resource model *is a triple $\mathcal{K} = \langle \mathcal{M}, \models, [\![-]\!] \rangle$ in which $\mathcal{M}$ is a KRM, $[\![-]\!]$ is a KRI and $\models$ is a forcing relation on $M \times \mathcal{P}(L)$ satisfying the following conditions:*

- $m \models \mathrm{p}$ *iff* $m \in [\![\mathrm{p}]\!]$
- $m \models \top$ *iff always*

- $m \models \bot$ *iff* $m = \pi$
- $m \models \phi \wedge \psi$ *iff* $m \models \phi$ *and* $m \models \psi$
- $m \models \phi \vee \psi$ *iff* $m \models \phi$ *or* $m \models \psi$
- $m \models \phi \rightarrow \psi$ *iff, for all* $n \in M$ *such that* $m \sqsubseteq n$, *if* $n \models \phi$, *then* $n \models \psi$
- $m \models I$ *iff* $e \sqsubseteq m$ *or* $m = \pi$
- $m \models \phi * \psi$ *iff there exist* $n_\phi, n_\psi \in M$ *such that* $n_\phi \circ n_\psi \sqsubseteq m$, $n_\phi \models \phi$ *and* $n_\psi \models \psi$
- $m \models \phi \mathrel{-\!\!*} \psi$ *iff, for all* $n \in M$ *such that* $n \models \phi$, $m \circ n \models \psi$.

Definition 25 (basic GRM). *A GRM* $\langle (M, \circ, e, \sqsubseteq, J), \models_{\mathcal{G}}, [\![-]\!]_{\mathcal{G}} \rangle$ *is* basic *iff* M *contains an element* π *such that for any* $m \in M$, $\pi \circ m = \pi$ *and* J *is basic,* i.e., *is given by* $J(m) = \{\{m\}\}$ *if* $m \neq \pi$ *and* $J(\pi) = \{\{\pi\}, \emptyset\}$.

Lemma 7. *The class of Kripke resource models coincides with the class of basic Grothendieck resource models.*

Proof Let $\mathcal{G}{=}\langle (M, \circ, e, \sqsubseteq, J), \models_{\mathcal{G}}, [\![-]\!] \rangle$ be a basic GRM. We must establish that $\langle (M, \circ, e, \sqsubseteq), \models_{\mathcal{G}}, [\![-]\!] \rangle$ is a Kripke model. Since $\mathcal{G}$ is basic, we simply show that $\models_{\mathcal{G}}$ satisfies the conditions of Definition 24. In the case of $\bot$, since $\emptyset$ only belongs to $J(\pi)$, the condition $\emptyset \in J(m)$ is equivalent to $m = \pi$. Now, for any world $m \neq \pi$, we have $J(m) = \{\{m\}\}$. Thus, in the case of I, the condition $(\exists S \in J(m))\ (\forall m' \in S)\ (e \sqsubseteq m')$ simplifies to $(\forall m' \in \{m\})\ (e \sqsubseteq m')$, which is equivalent to $e \sqsubseteq m$. The cases of $\vee$ and $*$ are similar. Conversely, endowing a Kripke model $\langle (M, \circ, e, \sqsubseteq), \models_{\mathcal{K}}, [\![-]\!] \rangle$ with the basic topology turns it into a basic Grothendieck model (a short calculation shows, for such a J, that Kripke monotonicity for $[\![-]\!]$ implies (Sh)). □

We have seen, in the semantics presented above, that π internalizes undefinedness and so corresponds to an alternative way of dealing with $\bot$ by considering a partially defined monoid, in which $\circ$ is a partial operation. Hence, we obtain a semantics which directly generalizes that taken in the analysis of pointer logic, in which the resource is computer memory, thereby emphasizing its utility in our analysis of resource:

- $m \models \bot$ iff never
- $m \models I$ iff $e \sqsubseteq m$
- $m \models \phi * \psi$ iff there exist $n, n' \in M$ such that $n \circ n' \downarrow$, $n \circ n' \sqsubseteq m$ $n \models \phi$ and $n' \models \psi$
- $m \models \phi \mathrel{-\!\!*} \psi$ iff for all $n \in M$ such that $n \models \phi$, $m \circ n \downarrow$ implies $m \circ n \models \psi$

where $\downarrow$ denotes definedness.

Theorem 5. BI *is sound and complete w.r.t. this "partial monoid" resource semantics.*

Proof The soundness is obvious since Grothendieck models include Kripke models. Turning to completeness, suppose that $I \not\models \phi$ then, by Theorem 3, there exists a tableau containing a H-branch from which one can construct a basic *GRM* which is a countermodel of ϕ following Definition 20. Lemma 7 then yields the corresponding Kripke countermodel for ϕ. Thus we observe that dependency graphs can be seen directly as countermodels in this new semantics. □

References

[1] V. Balat and D. Galmiche. *Labelled Deduction*, in Volume 17 of *Applied Logic Series*, Labelled Proof Systems for Intuitionistic Provability. Kluwer Academic Publishers, 2000. 185

[2] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Texts and Monographs in Computer Science. Springer Verlag, 1990. 185, 187

[3] D. M. Gabbay. *Labelled Deductive Systems*. OUP, 1996. 185

[4] D. Galmiche and D. Méry. Proof-search and countermodel generation in propositional BI logic - extended abstract -. In *4th Int. Symposium on Theoretical Aspects of Computer Software, TACS 2001, LNCS 2215*, 263–282, Sendai, Japan, 2001. Full version submitted. 185, 187, 189, 191, 196

[5] J. Harland and D. Pym. Resource-distribution via Boolean Constraints (Extended Abstract). In *14th Int. Conference on Automated Deduction, CADE-12, LNAI 814*, 222–236, Townsville, Queensland, Australia, July 1997. Full version to appear in ACM ToCL, 2003. 185

[6] Y. Lafont. The finite model property for various fragments of linear logic. *J. Symb. Logic* 62(4):1202–1208, 1997. 197

[7] P. Lincoln. Deciding provability of linear logic formulas. In *Advances in Linear Logic*, J.-Y.Girard, Y. Lafont and L. Regnier (editors), Cambridge Univ. Press, 1995, 109–122. 197

[8] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. 28th ACM Symp. on Principles of Prog. Langs., POPL 2001*, 14–26, London, UK, 2001. 184, 185, 197

[9] P. O'Hearn and J. Reynolds and H. Yang. Local Reasoning about Programs that Alter Data Structures. In *Proc. 15th Int. Workshop on Computer Science Logic, CSL'01, LNCS 2142*, 1–19, Paris, 2001. 185

[10] P. W. O'Hearn and D. Pym. The Logic of Bunched Implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999. 183, 184, 186

[11] M. Okada and K. Terui. Completeness proofs for linear logic based on proof search method (preliminary report). In *Type theory and its applications to computer systems*, 57–75, RIMS, Kyoto University, 1998. 196

[12] D. Pym. On bunched predicate logic. In *Proc. 14th Symposium on Logic in Computer Science*, 183–192, Trento, Italy, July 1999. IEEE Computer Society Press. 183, 184

[13] D. J. Pym. The Semantics and Proof Theory of the Logic of Bunched Implications. *Applied Logic Series*. Kluwer Academic Publishers, 2002. To appear; preprint available at http://www.cs.bath.ac.uk/∼pym/recent.html. 183, 184, 185, 186, 187, 191, 197

[14] D. J. Pym, P. W. O'Hearn and H. Yang. Possible Worlds and Resources: The Semantics of **BI**. Manuscript, http://www.cs.bath.ac.uk/∼pym/recent.html. 184, 185, 186, 187, 191

Configuration Theories

Pietro Cenciarelli

University of Rome, "La Sapienza"
Department of Computer Science - Via Salaria 113, 00198 Roma
cenciarelli@dsi.uniroma1.it

Abstract. A new framework for describing concurrent systems is presented. Rules for composing configurations of concurrent programs are represented by sequents $\Gamma \vdash_\rho \Delta$, where Γ and Δ are sequences of partially ordered sets (of events) and ρ is a matrix of monotone maps from the components of Γ to the components of Δ. Such a sequent expresses that whenever a configuration has certain specified subposets of events (Γ), then it extends to a configuration containing one of several specified subposets (Δ). The structural rules of Gentzen's sequent calculus are decorated by suitable operations on matrices, where *cut* corresponds to product. The calculus thus obtained is shown to be sound with respect to interpretation in *configuration structures* [GG90]. Completeness is proven for a restriction of the calculus to finite sequents. As a case study we axiomatise the *Java* memory model, and formally derive a non-trivial property of thread-memory interaction.

Keywords: semantics, concurrency, configuration structures, sequent calculus, Java.

1 Introduction

The Java language specification [GJS96] is very precise in describing how the events of a Java computation may depend on each other. For instance, it is required that, whenever a thread θ (a lightweight process) modifies the content of its *working memory* by assigning a value to an instance variable while holding a lock on some object, that value must be copied to the *main memory* before θ is allowed to release the lock [ibid. §17.6]. While it is relatively easy to write a denotational model of Java (say, as a Petri net or as an event structure [Cen00]), it is unclear whether such a model, a description of some large and complicated graph, would serve its purpose, e.g. to provide a usable mathematical framework for validating program logics or for proving, for example, that a process respects the above protocol on locks.

While writing an operational semantics of Java [CKRW98], the author realised that the rules of interaction that processes must obey could be conveniently formalised by using the same stuff of which models are made: posets of events. What a rule gives is a recipe for arranging events into legal configurations. From the work on Java the general idea originated of a context calculus

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 200–215, 2002.
© Springer-Verlag Berlin Heidelberg 2002

where posets of events representing fragments of concurrent computation combine into larger fragments according to language-dependent rules given in the form of axioms.

In the present paper we propose an axiomatic framework for describing concurrent systems. It is a sequent calculus, where sequents are made of posets and monotone injections specifying how the posets are allowed or required to match.

Sections 2 and 3 describe syntax and semantics of sequents. In Section 4 we give the structural rules of the calculus and prove soundness with respect to interpretation in *configuration structures* [GG90]. Completeness is proven in Section 5 for a restriction of the calculus to *finite* posets. The Java memory model is axiomatised by means of finite posets in Section 6 where a non-trivial property of thread interaction is proven formally. Note that the Java memory model to which we refer [GJS96, §17] is now under revision by the Java Community Process in order to make it support common optimising techniques that are currently disallowed. But of course the point of Section 6 is not to study specific features of the Java language, but to see how configuration theories fare in real life.

Notation. Here are some adopted notational conventions. An $m \times n$ *matrix* ρ in a *set* S is a doubly indexed family of elements ρ_{ij} of S ($i = 1 \ldots m$ and $j : 1 \ldots n$). When either m or n are 0, an $n \times m$ matrix is the empty family. The Greek letters ρ, σ, τ are used as metavariables for two-dimensional matrices. We write ρ_i instead of ρ_{i1} when ρ has size $m \times 1$. Similarly when ρ has size $1 \times n$.

If ρ and σ are matrices of size $m \times n$ and $r \times n$ respectively, we write $\rho\,;\sigma$ for the $(m + r) \times n$ matrix obtained by "placing ρ above σ": the ij-component of $\rho\,;\sigma$ is ρ_{ij} for $i \leq r$, while it is $\sigma_{(i-r)j}$ when $i > r$. Similarly, if ρ and σ are of size $m \times n$ and $m \times r$, we write $\rho\,,\sigma$ for the $m \times (n + r)$ matrix obtained by "placing ρ before σ": the ij-component of ρ, σ is ρ_{ij} for $j \leq n$, while it is $\sigma_{i(j-n)}$ when $j > n$.

We write function composition in diagrammatical order.

2 Poset Sequents

If A and B are partially ordered sets (posets), we write $p : A \rightarrowtail B$ for a monomorphism in the category of posets, that is, an injective function preserving the order of elements. In general, a momomorphism $p : A \rightarrowtail B$ in a category $\mathcal{C}$ is called *strong* when, for any commuting square $e\,v = u\,p$ in $\mathcal{C}$, where $e : C \to D$ is an epimorphism, there exists a unique diagonal $d : D \to A$ such that $v = d\,p$. Then, any strong mono which is epimorphic is an isomorphism. The strong monos p in the category of posets are exactly those which *reflect* the order, that is: $p(a) \leq p(b)$ implies $a \leq b$. By a simple argument, if A and B are *finite*, if p is as above and there exists a mono $B \rightarrowtail A$, then p must reflect the order. Moreover, p must be surjective (an epimorphism), and therefore it must be an isomorphism. This result is used in the proof of Proposition 11.

In general, we use Γ, $\Delta \ldots$ as metavariables for sequences of posets, A, $B \ldots$ for individual posets, and a, $b \ldots$ for elements of posets. However, when

the components of a sequence Γ are not introduced explicitly by an equation $\Gamma = A_1, \dots A_m$, we write Γ_i for the i-th element of Γ. Concatenation of sequences Γ and Δ is written Γ, Δ. If $\Gamma = A_1, \dots A_m$ and $\Delta = B_1 \dots B_n$ are finite sequences of posets, we write $\rho : \Gamma \to \Delta$ to mean that ρ is an $m \times n$ matrix of monos $\rho_{ij} : A_i \rightarrowtail B_j$. An $m \times 1$ matrix $\Gamma \to D$ is called an *interpretation* of Γ in D.

Definition 1 *A* poset sequent $\Gamma \vdash_\rho \Delta$ *(just* sequent *for short) consists of two finite sequences Γ and Δ of posets and an $m \times n$ matrix $\rho : \Gamma \to \Delta$ of monos.*

The posets in a sequent are meant to represent fragments of a configuration of events. The intuitive meaning of a sequent $\Gamma \vdash_\rho \Delta$ is that whenever a single configuration interprets *all* components of Γ, the interpretation extends along ρ to *at least one* component of Δ. Of course the Δ_i may include more events than are mentioned in Γ, thus specifying what is required to happen after (or must have happened before) a certain combination (Γ) of events.

Let $\Gamma \vdash_\rho \Delta$ and $\Pi \vdash_\sigma \Delta$ be sequents. It is easy to check that $\Gamma, \Pi \vdash_{\rho;\sigma} \Delta$ is a "well formed" sequent, that is: $(\rho; \sigma) : \Gamma, \Pi \to \Delta$. Similarly, if $\Gamma \vdash_\rho \Delta$ and $\Gamma \vdash_\sigma \Pi$ are sequents, so is $\Gamma \vdash_{\rho,\sigma} \Delta, \Pi$. Finally, if $\rho : \Gamma \to A$ and $\sigma : A \to \Delta$ are matrices of size $m \times 1$ and $1 \times n$ respectively, we can form their *product* $\rho\sigma : \Gamma \to \Delta$, of size $m \times n$, by using function composition to multiply components. Hence, if $\Gamma \vdash_\rho A$ and $A \vdash_\sigma \Delta$ are sequents, then so is $\Gamma \vdash_{\rho\sigma} \Delta$.

Example. Let $B, C \vdash_{f;g} A$ and $A \vdash_{u,v} D, E$ be sequents, where $f : B \rightarrowtail A$, $g : C \rightarrowtail A$, $u : A \rightarrowtail D$ and $v : A \rightarrowtail E$. Then, $B, C \vdash_\rho D, E$ is a sequent, where

$$\rho = \begin{bmatrix} f \\ g \end{bmatrix} [u\ v] = \begin{bmatrix} fu\ fv \\ gu\ gv \end{bmatrix},$$

with $fu : B \rightarrowtail D \dots$ and $gv : C \rightarrowtail E$ as required.

□

There is no general construction for multiplying two matrices $\Gamma \to \Delta \to \Pi$. Some notion of summation on morphisms of the form $\Gamma_i \rightarrowtail \Pi_j$ would be needed for that. However, in Section 5 we use the following construction for multiplying a matrix by a vector of row vectors. Let $\rho : \Gamma \to B_1, \dots B_n$ be an $m \times n$ matrix of monos, and let $\sigma : [\sigma^{(1)}, \dots \sigma^{(n)}]$ be a vector where each $\sigma^{(i)}$ is a $1 \times k_i$ matrix $B_i \to \Pi^{(i)}$. The product $\rho_{(\_)i}\sigma^{(i)} : \Gamma \to \Pi^{(i)}$ has size $m \times k_i$. Pasting all such matrices together horizontally we obtain a matrix $\rho \diamond \sigma = (\rho_{(\_)1}\sigma^{(1)}, \dots \rho_{(\_)n}\sigma^{(n)})$ of size $m \times (k_1 + \dots + k_n)$. The $\diamond$ construction is thus described by the following formation rule:

$$[\diamond] \quad \frac{\Gamma \vdash_\rho B_1, \dots B_n \quad B_1 \vdash_{\sigma^{(1)}} \Pi^{(1)} \ \dots \ B_n \vdash_{\sigma^{(n)}} \Pi^{(n)}}{\Gamma \vdash_{\rho \diamond \sigma} \Pi^{(1)}, \dots \Pi^{(n)}} \quad (\sigma = [\sigma^{(1)}, \dots \sigma^{(n)}])$$

Let L be a set of *labels* to be thought of as action names. An L-labelled sequent is a sequent $\Gamma \vdash_\rho \Delta$ where all components X of Γ and Δ are labelled by a function $X \to L$ and all components of ρ respect the labelling.

3 Configuration Structures

Definition 2 [GP95] *A* configuration structure *is a pair* $(E, \mathcal{C})$ *where* E *is a set, whose elements are called* events*, and* $\mathcal{C}$ *is a collection of subsets of* E*, called* configurations*.*

The events of a configuration structure (or just *structure* for short) can be viewed as occurrences of the actions a concurrent system may perform, while a configuration models a consistent state of the system, represented as the set of events occurred during computation up to that point. We write just $\mathcal{C}$ for $(E, \mathcal{C})$ when no confusion arises.

Configuration structures originate from [Win82], where they were introduced as an alternative way to address *event structures* [NPW81] (in the form known later as *prime event structures with binary conflict*). In [Win87] several closure conditions on the set of configurations of a structure $\mathcal{C}$ were given in order to get a precise match with *general event structures* (generalising those of [NPW81]). The requirements were: *finiteness* (if an event belongs to a configuration C, then it also belongs to a finite subconfiguration of C), *coincidence-freeness* (if two distinct events belong to a configuration C, then there exists a subconfiguration of C containing exactly one of them), closure under *bounded unions* and *non-emptyness* of $\mathcal{C}$.

In the framework of (general) event structures, configurations (as well as the order on events) are defined in terms of other mathematical structure. In the present paper, and following [GG90], we find it convenient to take the notion of configuration as primitive and that of order as derived. To this effect we adopt here (and do so implicitly) all of the above requirements *except* for closure under bounded unions, which is not needed for the treatment.

Let C be a configuration of a structure $\mathcal{C}$. We write $Sub(C)$ for the set $\{D \in \mathcal{C} \mid D \subseteq C\}$ of subconfigurations of C. Then, we let $\leq_C$ denote the binary relation on C such that $b \leq_C a$ if and only if, for all $D \in Sub(C)$, $a \in D$ implies $b \in D$. The set $\{b \in C \mid b \leq_C a\}$ is denoted by $C \downarrow a$, and similarly for $C \uparrow a$.

Proposition 3 *The relation* $\leq_C$ *is a* partial order*. Moreover, for all* $a \in C$*, the set* $C \downarrow a$ *is finite.*

The antisymmetry of $\leq$ (we omit indices when no confusion arises) is an immediate consequence of coincidence-freeness while finiteness of $C \downarrow a$ follows from the finiteness property on configurations (the converse does not hold). We use this property (only) in Section 6, where a formal rule expressing the groundedness of configurations is introduced to prove a property of Java. By the proposition above, we treat configurations as posets. If A is a poset, we write $(E_A, \mathcal{C}_A)$ for the structure whose events are the elements of A and whose configurations are the downwards closed subsets of A. When a poset A is treated as a configuration structure, it is meant $(E_A, \mathcal{C}_A)$.

In general, the collection of partial orders $\leq_C$, $C \in \mathcal{C}$, defined as above does not represent the causality relation on $\mathcal{C}$ faithfully. In particular, it does not hold that $a \leq_D b$ implies $a \leq_C b$ for $D \in Sub(C)$ (while the converse holds by definition). Calling *conservative* a structure where the above implication does hold, it is easy to check the following:

Proposition 4 *A configuration structure $\mathcal{C}$ is conservative if and only if it is has* downwards-closed bounded intersections*: for all $C \in \mathcal{C}$, for all $D, F \in Sub(C)$, and for all $a \in D \cap F$, if $b \leq_D a$ then $b \in F$.*

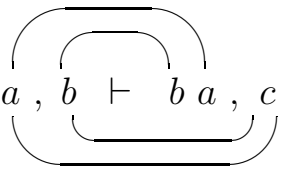
If C and D are configurations of a conservative structure, with $D \in Sub(C)$, the inclusion $D \subseteq C$ can be viewed as a morphism in the category of posets, which we write $(D, \leq_D) \hookrightarrow (C, \leq_C)$. We rely on conservativity in Definition 5, which is the heart of the present paper. Henceforth the configuration structures of discourse will implicitly be assumed conservative. Note that so are the *stable* structures of [GG01], which require closure under bounded intersections. These are precisely the structures where the order on a configuration determines its subconfigurations. Indeed all results in the present paper, which rely on weaker assumptions, specialise to stable structures.

Definition 5 *A structure $\mathcal{C}$ is said to* satisfy *a sequent $\Gamma \vdash_\rho \Delta$ when, for any configuration $C \in \mathcal{C}$ and interpretation $\pi : \Gamma \to C$, there exist a configuration $D \in \mathcal{C}$, a component $\Delta_k \in \Delta$ and a mono $q : \Delta_k \rightarrowtail D$ such that $C \in Sub(D)$ and, for all i, the following diagram commutes.*

$$\begin{tikzcd} \Gamma_i \arrow[r, "\rho_{ik}"] \arrow[d, "\pi_i"'] & \Delta_k \arrow[d, "q"] \\ C \arrow[r, hook] & D \end{tikzcd} \tag{1}$$

The notion of interpretation given above extends to labelled sequents and *labelled configuration structures* [GP95] in an obvious way. The reader should check that the above definition agrees with the intuitive meaning of sequents proposed in the previous section. Note that in the present setting we decided to attach no special *computational* meaning to the inclusions as $C \hookrightarrow D$ above. However, the notion of $\hookrightarrow$ must be strenthened in order to prove that satisfaction is preserved by *history preserving bisimulation* [GG01].

A sequent is called *valid* if it is satisfied by all structures. An example of valid sequent is $A \vdash_{id} A$. A slightly more complicated example is given in Diagram 2, which states that if a poset has an element a and it has an element b, then either a and b are the same element or they are distinct. The adopted graphical representation of sequents is to be read as follows: posets are separated by commas and no braces are used to hold elements of a set together. Vertical lines represent the order within each poset (where a below b means $a < b$), while their absence means no order. Links spanning across the turnstile represent the matrix of monos, where $a \frown b$ means $a \mapsto b$ by the corresponding matrix component.



(2)

By the above conventions, (2) stands for a sequent $A, B \vdash_\rho C, D$ where $A = \{a\}$, $B = \{b\}$, $C = \{a, b\}$ (with a and b unordered) and $D = \{c\}$. Moreover, $\rho_{11}(a) = a$, $\rho_{12}(a) = c$ and so on.

Example 6 $A, A \vdash_{id;id} A$ is *not* satisfied by the structure $A + A$, where $+$ denotes disjoint union. In fact, the two copies of A on the left side of the sequent are disjoint in the interpretation $A \xrightarrow{inl} A + A \xleftarrow{inr} A$, while the components of $(id; id)$ do overlap.

□

Here are other examples. The sequent $\vdash$ denotes *absurdity*. Note that this sequent features empty sequences as antecedent and succedent and it is meant as decorated by the empty matrix. A structure satisfying $\vdash A$ models a process where all runs are bound to produce a combination of events matching A. Let a and b be labelled by l_1 and l_2 respectively. Sequent (3.i) below is to be read: any l_1 action must be followed by an l_2 action, while sequent (3.ii) forbids l_2 actions to be preceded by (to depend causally on) l_1 actions.

(i) (ii)

(3)

By similar statements it is possible to describe the behaviour of concurrent programs axiomatically. This is shown in Section 6 where we shall develop further intuition on the meaning of sequents.

4 Configuration Theories

Definition 7 *A* configuration theory *is a set of sequents which is closed under the rule schemes of Table 1.*

The rule [l-weak] only allows the premises of a sequent to be weakened by the empty poset $\emptyset$. Left weakening by an arbitrary poset (as in [r-weak]) would be unsound, as in fact it would allow the inference of $A, A \vdash_{id;id} A$ from $A \vdash_{id} A$ (see Example 6). Rule [r-cut] is a special case of [$\diamond$], which was introduced as a formation rule in Section 2. Indeed [$\diamond$] can be derived from [r-cut]. Note that [$\diamond$] has an obvious dual, which is a general form (and is derivable from) [l-cut].

Table 1. Structural rules

[true]	$\dfrac{}{\vdash \emptyset}$	[iso]	$\dfrac{}{A \vdash_{\phi} B}$ (ϕ is iso)
[l-weak]	$\dfrac{\Gamma \vdash_{\rho} \Delta}{\Gamma, \emptyset \vdash_{\rho;\emptyset} \Delta}$	[r-weak]	$\dfrac{\Gamma \vdash_{\rho} \Delta}{\Gamma \vdash_{\rho,\sigma} \Delta, A}$ $(*)$
[l-contr]	$\dfrac{\Gamma, A, A \vdash_{\rho;\sigma;\sigma} \Delta}{\Gamma, A \vdash_{\rho;\sigma} \Delta}$	[r-contr]	$\dfrac{\Gamma \vdash_{\rho,\sigma,\sigma} \Delta, A, A}{\Gamma \vdash_{\rho,\sigma} \Delta, A}$
[l-exc]	$\dfrac{\Gamma, A, B, \Pi \vdash_{\rho;\sigma;\tau;\theta} \Delta}{\Gamma, B, A, \Pi \vdash_{\rho;\tau;\sigma;\theta} \Delta}$	[r-exc]	$\dfrac{\Gamma \vdash_{\rho,\sigma,\tau,\theta} \Delta, A, B, \Pi}{\Gamma \vdash_{\rho,\tau,\sigma,\theta} \Delta, B, A, \Pi}$
[l-cut]	$\dfrac{\Pi \vdash_{\tau} A \qquad \Gamma, A \vdash_{\rho;\sigma} \Delta}{\Gamma, \Pi \vdash_{\rho;\tau\sigma} \Delta}$	[r-cut]	$\dfrac{A \vdash_{\tau} \Pi \qquad \Gamma \vdash_{\rho,\sigma} \Delta, A}{\Gamma \vdash_{\rho,\sigma\tau} \Delta, \Pi}$

$(*)$ where σ is a column vector of monos $\sigma_i : \Gamma_i \rightarrowtail A$.

A *model* of a configuration theory is a structure which satisfies all sequents of the theory.

Theorem 8 *The rules of Table 1 are sound.*

Proof. It is required to prove that, if a configuration structure satisfies the premises of a rule, then it also satisfies the conclusion. We just prove the statement for the left and right cut rules. The argument is similar for the others.
[l-cut]. Let $\mathcal{C}$ satisfy the sequents $\Pi \vdash_{\tau} A$ and $\Gamma, A \vdash_{\rho;\sigma} \Delta$, let $C \in \mathcal{C}$ be a configuration, and let $\upsilon : \Gamma \to C$ and $\pi : \Pi \to C$ be matrices of monos. Satisfaction of τ yields an inclusion $C \hookrightarrow D$ and a map $q : A \rightarrowtail D$ making the $(*)$ square of diagram (4) commute for all i. Then, considering q in conjunction with the maps $\Gamma_j \xrightarrow{\upsilon_j} C \hookrightarrow D$, satisfaction of $\rho;\sigma$ yields an inclusion $D \hookrightarrow D'$, a component Δ_k and a map $q' : \Delta_k \rightarrowtail D'$ making all the rest of diagram (4) commute for all i and j. Since $\tau_i \sigma_k = (\tau\sigma)_{ik}$, we conclude that $\mathcal{C}$ satisfies $\Gamma, \Pi \vdash_{\rho;\tau\sigma} \Delta$ as required.

[r-cut]. Let $\mathcal{C}$ satisfy the sequents $A \vdash_{\tau} \Pi$ and $\Gamma \vdash_{\rho,\sigma} \Delta, A$, let $C \in \mathcal{C}$ be a configuration, and let $\upsilon : \Gamma \to C$ be a matrix of monos. Satisfaction of ρ, σ yields an injection $C \hookrightarrow D$ and moreover, for all $\Gamma_i \in \Gamma$, a commuting square as $(\star)$ below:

$$\begin{tikzcd} \Gamma_i \arrow[r, "\xi_i"] \arrow[d, "\upsilon_i"'] \arrow[dr, phantom, "(\star)"] & X \arrow[d, "q"] \\ C \arrow[r, hook] & D \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "\tau_k"] \arrow[d, "q"'] \arrow[dr, phantom, "(\star\star)"] & \Pi_k \arrow[d, "q'"] \\ D \arrow[r, hook] & D' \end{tikzcd}$$

where either $X = A$ and $\xi_i = \sigma_i$, or $X = \Delta_j$ for a component $\Delta_j \in \Delta$, and $\xi_i = \rho_{ij}$. In the last case the result follows immediately. Otherwise $X = A$ and, since τ is satisfied, there exists a component $\Pi_k \in \Pi$, an inclusion $D \hookrightarrow D'$ and a map $q' : \Pi_k \rightarrowtail D'$ such that the diagram $(\star\star)$ above commutes. Pasting $(\star)$ and $(\star\star)$ we get the required instance of diagram (1), where $\sigma_i \tau_\kappa = (\sigma\,\tau)_{ik}$.

□

$$\begin{tikzcd} \Gamma_j \arrow[rr, "\rho_{jk}"] \arrow[dd, "\upsilon_j"'] & & \\ & \Pi_i \arrow[r, "\tau_i"] \arrow[d, "\pi_i"] \arrow[dr, phantom, "(*)"] & A \arrow[r, "\sigma_k"] \arrow[d, "q"] & \Delta_k \arrow[d, "q'"] \\ & C \arrow[r, hook] & D \arrow[r, hook] & D' \end{tikzcd} \tag{4}$$

5 Completeness

There are valid sequents which cannot be derived from the inference rules of Table 1. One is Diagram (2) of Section 3. In this section we obtain a complete calculus at the cost of constraining sequents (but not the models) to be *finite*, that is, made of finite posets. Indeed, since Diagram (2) is finite, new rules are needed to achieve completeness. Note that the Java axioms of Section 6 are finite.

First we introduce a notion of order (in fact a preorder) on matrices of poset maps which is somewhat analogous to the notion of *rank* in linear algebra. Let $\rho : \Gamma \to A_1, \ldots A_m$ and $\sigma : \Gamma \to B_1, \ldots B_n$ be matrices of posets. We write $\rho \leq_\Gamma \sigma$ (omitting Γ when understood) if there exist a function on indices $f : \{1, \ldots n\} \to \{1, \ldots m\}$ and a family $\{\phi_j\}$ of monos $\phi_j : A_{f(j)} \rightarrowtail B_j$, $j = 1 \ldots n$, such that $\sigma_{ij} = \rho_{if(j)}\phi_j$, for all i. In this case we say that f and $\{\phi_j\}$ *witness* $\rho \leq \sigma$. The relation $\leq$ is reflexive and transitive. We call *equivalent* two matrices ρ and σ such that $\rho \leq \sigma$ and $\sigma \leq \rho$. The equivalence class of ρ is written $[\rho]$.

Proposition 9 *Let $\Gamma \vdash_\rho \Delta$ and $\Gamma \vdash_\sigma \Pi$ be sequents: $\rho \leq \sigma$ holds if and only if, whenever a structure satisfies σ, it also satisfies ρ.*

Proof. *If*: Each $\Pi_i \in \Pi$ (viewed as a configuration structure) satisfies σ, and so it must satisfy ρ. Hence, considering the interpretation $\sigma_{(\_)i} : \Gamma \to \Pi_i$, there exist a $\Delta_{f(i)} \in \Delta$ and a mono $\phi_i : \Delta_{f(i)} \rightarrowtail \Pi_i$ such that $\sigma_i = \rho_{f(i)}\phi_i$ as required.

Only if: Let $\rho \leq \sigma$, let $\mathcal{C}$) satisfy σ and let $\pi : \Gamma \to C \in \mathcal{C}$. There must exist a $D \in \mathcal{C}$, an inclusion $u : C \hookrightarrow D$, a $\Pi_k \in \Pi$ and a mono $q : \Pi_k \rightarrowtail D$ such that $\pi_i u = \sigma_{ik} q$, for all i. Since $\rho \leq \sigma$, there must exist $\Delta_{f(k)} \in \Delta$ and a mono $\phi_k : \Delta_{f(k)} \rightarrowtail \Pi_k$ such that $\rho_{if(k)}\phi_k = \sigma_{ik}$ for all i; hence $\pi_i u = \sigma_{ik} q = \rho_{if(k)}\phi_k q$ as required. □

By the above result the inference rule [sub] below is sound. This rule is used in Section 6. Moreover, since any matrix $\sigma : \Gamma \to \Pi$ is such that $\sigma \leq \epsilon$, where $\epsilon : \Gamma \to \varepsilon$ is the empty matrix and ε is the empty sequence, the *falsum* rule below is a special case of [sub]. Similarly, [sub] subsumes [iso], [r-weak], [r-contr] and [r-exc].

$$[\text{sub}] \qquad \frac{\Gamma \vdash_\rho \Delta}{\Gamma \vdash_\sigma \Pi} \quad (\sigma \leq \rho) \qquad\qquad [\text{falsum}] \qquad \frac{\Gamma \vdash}{\Gamma \vdash_\sigma \Pi}$$

Definition 10 *A matrix μ of size $m \times n$ is called* minimal *in its equivalence class when, for all $\rho \in [\mu]$ of size $m \times n'$, $n \leq n'$.*

We show that, when considering matrices $\Gamma \to \Delta$ where all components of Γ and Δ are *finite*, all minimal matrices in an equivalence class are *isomorphic*. In the rest of this section we assume, unless otherwise stated, that matrices are made of maps between finite posets. This assumption should also reassure the concerned reader that the side condition of [sub] can be checked effectively.

Proposition 11 *Let $\rho : \Gamma \to \Delta$ and $\mu : \Gamma \to \Pi$ be equivalent matrices, with μ of size $m \times n$ minimal in $[\rho]$. Then $\rho \leq \mu$ is witnessed by a family of* isomorphisms *$\phi_j : \Delta_{f(j)} \rightarrowtail \Pi_j$.*

Proof. Let ρ have size $m \times n'$ and let $\rho \leq \mu$ by $f : \{1, \ldots n\} \to \{1, \ldots n'\}$ and by a family of monos $\phi_j : \Delta_{f(j)} \rightarrowtail \Pi_j$. Let ρ' be the matrix obtained from ρ by deleting all the columns $\rho_{(\_)k} : \Gamma \to \Delta_k$ such that $k \neq f(j)$ for all j. Clearly $\rho \leq \rho'$, and moreover $\rho' \leq \mu$ by the same f and ϕ_j. Hence $\rho' \in [\mu]$. It follows that f must be injective, otherwise its image would be smaller than n, thus contraddicting the minimality of μ. So f is a bijection. Let $\mu \leq \rho'$ by a function g and a family $\psi_i : \Pi_{g(i)} \rightarrowtail \Delta_i$. By the same argument as above g must be a bijection, and hence all nodes in the bipartite directed graph whose edges are the ϕ_j and the ψ_i belong to (exactly) one cycle, which implies, from the remark at the the beginning of Section 2, that all ϕ_j are isos. □

A consequence of this result is that if two $m \times n$ matrices $\Gamma \to \Delta$ and $\Gamma \to \Pi$ are equivalent and minimal, there exists a family of n isomorphisms $\Delta_i \xrightarrow{\cong} \Pi_i$ through which all their components factorise. Hence, by a slight mathematical abuse, we say *the* minimal matrix of an equivalence class.

We can now define an operation that yields all possible mergings of two finite posets B and C which are consistent on some common intersection A.

Lemma 12 *Let A, B, C be finite posets and let $p : A \rightarrowtail B$ and $q : A \rightarrowtail C$ be monos. There exists a (possibly empty) matrix $(\pi;\tau) : B, C \to \Pi$ such that $p\,\pi_i = q\,\tau_i$ for all i, and moreover $(\pi;\tau) \leq (r;s)$ for all $r : B \rightarrowtail D$ and $s : C \rightarrowtail D$ such that $p\,r = q\,s$.*

Sketch of proof. Let K be any set of cardinality $k = |B|+|C|$ and let $K_1, \ldots K_n$ be the set of all posets whose underlying set is K. Their number (n) is $(k-1)3k/2$. For $j = 1 \ldots n$, consider the (finite) set of diagrams $B \rightarrowtail K_j \leftarrowtail C$ which commute with $(p;q)$. The components of Π are the images of all such diagrams, while π and τ are made of the injections. □

Of course, all matrices with the above property are equivalent. We let $\mu(p,q)$ be the minimal one of the equivalence class. By using this construction we can now introduce a new rule of inference which yields the *extension* of a sequent $\Gamma, A \vdash \Delta$ along a mono $A \rightarrowtail C$:

$$[\text{extend}] \quad \frac{\Gamma, A \vdash_{\rho;\sigma} B_1, \ldots B_n}{\Gamma, C \vdash_{(\rho \diamond \pi);\tau} \Pi^{(1)}, \ldots \Pi^{(n)}} \quad (*)$$

$(*)$ where $\pi = [\pi^{(1)}, \ldots \pi^{(n)}]$, $\tau = [\tau^{(1)}, \ldots \tau^{(n)}]$, $q : A \rightarrowtail C$ and, for all i, $(\pi^{(i)}, \tau^{(i)}) = \mu(\sigma_i, q) : (B_i, C) \to \Pi^{(i)}$.

Note that [extend] only makes sense in a calculus of finite posets, where $\mu(\_,\_)$ is defined.

Proposition 13 [extend] *is sound.*

Proof. Let $\Gamma, A \vdash_{\rho;\sigma} B_1, \ldots B_n$ be satisfied by $\mathcal{C}$), let $q : A \rightarrowtail C$ be a mono and let $(\zeta;p) : (\Gamma, C) \to D$ be an interpretation of (Γ, C) in a configuration $D \in \mathcal{C}$. Since $(\zeta; q\,p) : (\Gamma, A) \to D$, there exist $D' \in \mathcal{C}$, an inclusion $u : D \hookrightarrow D'$, a B_k and a mono $r : B_k \rightarrowtail D'$ such that $\sigma_k r = q\,p\,u$ and, for all i, $\rho_{ik} r = \zeta_i u$. Let $\mu(\sigma_k, q) = (\pi^{(k)}; \tau^{(k)}) : (B_k, C) \to \Pi^{(k)}$. Since $(\pi^{(k)}; \tau^{(k)}) \leq (r; p\,u)$, there exists $\Pi_h^{(k)} \in \Pi^{(k)}$ and a mono $\phi : \Pi_h^{(k)} \rightarrowtail D'$ such that $\pi_h^{(k)}\phi = r$ and $\tau_h^{(k)}\phi = p\,u$. Moreover, let $\rho \diamond \pi = (\rho_{(\_)1}\pi^{(1)}, \ldots \rho_{(\_)n}\pi^{(n)})$. The h-component of $(\rho \diamond \pi)^{(k)}$ is $\rho_{(\_)k}\,\pi_h^{(k)} : \Gamma \to \Pi_h^{(k)}$, and hence $\rho_{ik}\,\pi_h^{(k)}\phi = \rho_{ik}\,r = \zeta_i\,u$ as required.

□

Lemma 14 [extend] *preserves minimality.*

Proof. With no loss of generality we develop the argument for $A, B \vdash_\rho C, D$. Let $A, F \vdash_\sigma \Pi$ be the extension of ρ along a mono $q : B \rightarrowtail F$, and suppose that σ is not minimal. Let $\nu \in [\sigma]$ be minimal and let f be the funcion on indices witnessing $\sigma \leq \nu$. There must be a $\Pi_k \in \Pi^{(i)}$ such that $k \neq f(j)$ for all j. The matrix σ' obtained from σ by deleting $\sigma_{(\_)k}$ must still be in $[\sigma]$. Hence there must exist $\Pi_h \in \Pi^{(l)}$ and $p : \Pi_h \rightarrowtail \Pi_k$ through which the interpretation $\sigma_{(\_)k}$ of (A, F) factorises. However, it must be $l \neq i$, because otherwise $\mu(\rho_{2i}, q)$ would not be minimal. But then $\rho_{(\_)i} \simeq \rho_{(\_)l}$ contraddicting the minimality of ρ.

□

Theorem 15 (completeness) *The system of finite sequents which includes the axioms of Table 1 and* [extend] *is complete.*

Proof. Let $A_1, \ldots A_m \vdash_\rho \Delta$ be a valid sequent. It is required to prove that ρ is derivable. Consider the derivation:

$$
\begin{array}{rl}
[\text{l-weak}] & \dfrac{A_1 \vdash_{id} A_1}{} \\
[\text{extend}] & \dfrac{A_1, \emptyset \vdash_{id;\emptyset} A_1}{} \quad (\emptyset \rightarrowtail A_2) \\
[\text{l-weak}] & \dfrac{A_1, A_2 \vdash_\pi \Pi}{A_1, A_2, \emptyset \vdash_{\pi;\emptyset} \Pi} \\
 & \vdots \\
[\text{extend}] & \dfrac{\qquad\qquad}{A_1, \ldots A_m \vdash_\sigma B_1, \ldots B_n} \quad (\emptyset \rightarrowtail A_m)
\end{array}
$$

where the A_i are introduced one-by-one by m applications of [l-weak] and [extend]. Since these rules are sound and $A_1 \vdash_{id} A_1$ is valid, then so is also σ. Hence, by Proposition 9, $\sigma \in [\rho]$. Moreover [l-weak] preserves minimality trivially, while [extend] does so by Proposition 14. Since $A_1 \vdash_{id} A_1$ is minimal, then so is also σ. Therefore, from Proposition 11, $\rho \leq \sigma$ is witnessed by a family of isomorphisms $\phi_j : \Delta_{f(j)} \xrightarrow{\simeq} B_j$ such that $\rho_{if(j)}\, \phi_j = \sigma_{ij}$. Then, by n applications of [iso] and [r-cut] we derive:

$$
[\text{r-cut}]^n \quad \dfrac{A_1, \ldots A_m \vdash_\sigma B_1, \ldots B_n \qquad B_i \vdash_{\phi_i^{-1}} \Delta_{f(i)}}{A_1, \ldots A_m \vdash_{\sigma_{(\_)f(1)}, \ldots \sigma_{(\_)f(n)}} \Delta_{f(1)}, \ldots \Delta_{f(n)}}
$$

The rest of σ can then be adjoined by [r-weak]. □

6 The Theory of Java

The interaction of threads (lightweight processes) and memory in Java is described in the language specification [GJS96, Ch. 17] by means of eight kinds of actions. Besides *Lock* and *Unlock*, which we do not consider here, they are: *Use*, *Assign*, *Load*, *Store*, *Read* and *Write*. These are abstractions over corresponding Java bytecode instructions. We let u, a, l, s, r and w stand for events labelled respectively by these types of actions. Each thread has a *working memory* where private copies of shared variables are cached. Threads operate on their own working memory by *Use* and *Assign* actions. For example, a thread θ performing an assignment $x = x + 1$, first *uses* the content of its working copy of x to compute $x+1$, and then *assigns* the computed value v to x. However, v is not available to other threads unless θ decides (nondeterministically) to *store* the current value of its copy x to the main memory, where the *master copies* of all variables reside. The *Store* action is just a message sent *asynchronously* by θ to the main memory: the actual writing of v in the master copy of x is performed

by the main memory (possibly at a later time) with a *Write* action. Similarly *Read* and *Load* are used for a loosely coupled copying of data from the main memory to a thread's working memory.

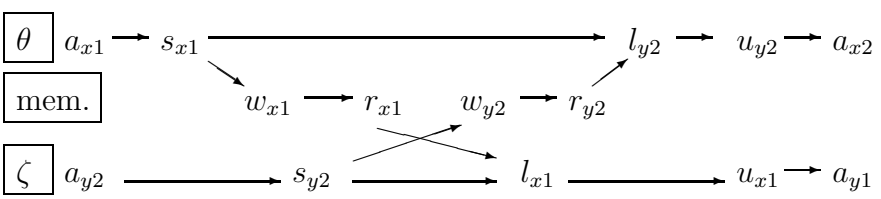
Following [CKRW98] we label events by 4-tuples of the form (α, θ, x, v), where $\alpha \in \{Use, Assign, Load, Store, Read, Write\}$, θ is a thread identifier, x is a variable and v is a value. We write $e : l$ to mean that event e has label l. Label components are omitted when undestood or irrelevant. Hence, if $u_{x1} : (Use, \zeta, x, 1)$, then u_{x1} represents the use of variable x, whose current value is 1, by a thread ζ, as in the example below for evaluating the right hand side of the assignment $y = x$, while $a : (Assign, y)$ stands for an assignment of an unspecified value to y by an unspecified thread.

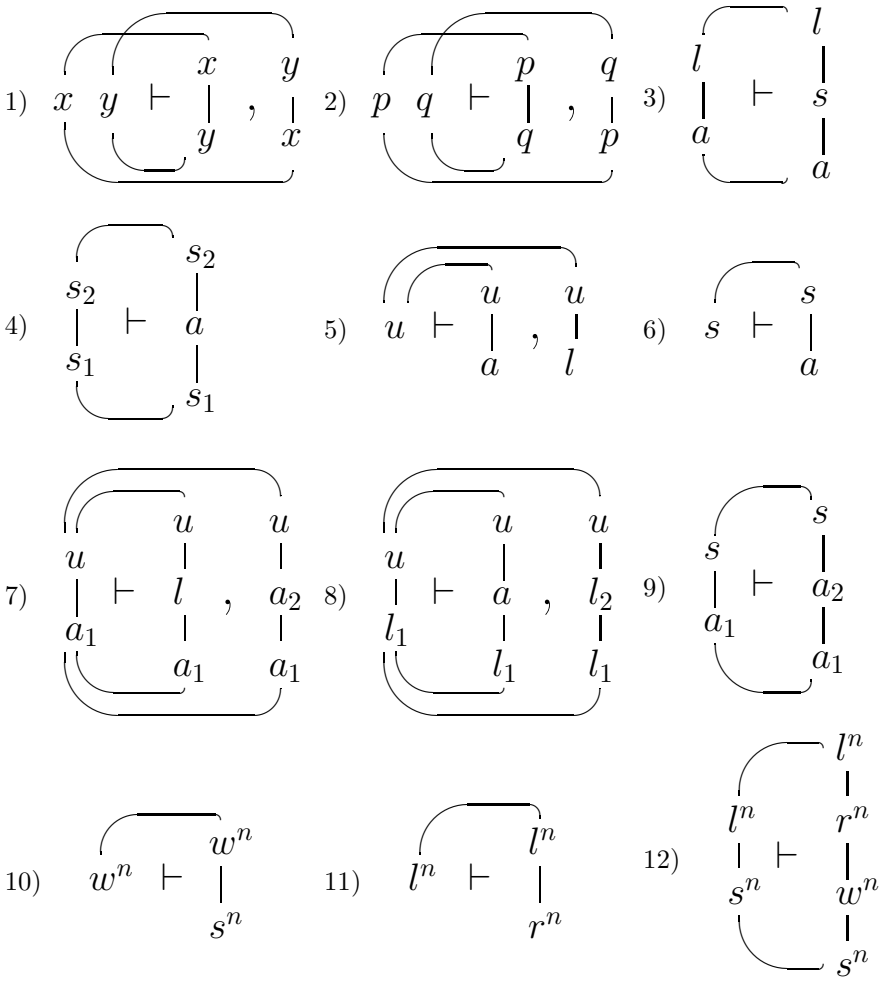
Table 2 shows a possible order of events which may occur when two threads θ and ζ, running in parallel left to right, execute respectively $(x = 1; x = y;)$ and $(y = 2; y = x;)$. The events are labeled as follows: $a_{x1} : (Assign, \theta, x, 1)$, $s_{x1} : (Store, \theta, x, 1)$, $l_{y2} : (Load, \theta, y, 2)$, $u_{y2} : (Use, \theta, y, 2)$, $a_{x2} : (Assign, \theta, x, 2)$, $w_{x1} : (Write, \theta, x, 1)$, $r_{x1} : (Read, \zeta, x, 1)$, $w_{y2} : (Write, \zeta, y, 2)$, $r_{y2} : (Read, \theta, y, 2)$, $a_{y2} : (Assign, \zeta, y, 2)$, $s_{y2} : (Store, \zeta, y, 2)$, $l_{x1} : (Load, \zeta, x, 1)$, $u_{x1} : (Use, \zeta, x, 1)$, $a_{y1} : (Assign, \zeta, y, 1)$. The execution ends with $x = y = 2$ in the working memory of θ and $x = y = 1$ in the working memory of ζ. Note that there is no causal dependency between actions performed by the memory on different variables, as between w_{x1} and w_{y2}. The ordering is legal according to the informal specification given in [GJS96].

Below we list 12 formal axiom schemes describing the protocol of memory and thread interaction in Java. They are subject to the side conditions given below, specifying what labels are to be attached to each event. By w^n we mean a *totally* ordered set $\{w_1 \leq w_2 \leq \ldots w_n\}$ of n events of type *Write*. Similarly for s^n, l^n and r^n. We do not consider synchronization by *lock* and *unlock*: the full theory, including synchronization by *lock* and *unlock* (6 additional axioms), is available at `http://cenciarelli.dsi.uniroma1.it/~cencia`.

As an example we explain axiom scheme (3). It represents all sequents of that form where $a : (Assign, \theta, x, v)$, $s : (Store, \theta, x, v)$ and $l : (Load, \theta, x)$ (the value being loaded in x is irrelevant). Hence: a *Store* action by θ on a variable x must intervene between an *Assign* by θ of x and a subsequent *Load* by θ of x. This is because a "thread is not permitted to lose its most recent assign" [GJS96, § 17.3].

Table 2. $(x = 1; x = y;) \mid\mid (y = 2; y = x;)$





The above sequents express the following requirements:

(1) $x : (\alpha, \theta)$ and $y : (\alpha', \theta)$, $\alpha, \alpha' \in \{Use, Assign, Load, Store\}$ (these are called *thread actions*). Intuitively, this axiom means that *the actions performed by any one thread are totally ordered* [GJS96, §17.2].

(2) $p : (\beta, \theta, x)$ and $q : (\beta', \theta, x)$, where $\beta, \beta' \in \{Read, Write\}$ (*memory actions*). *The actions performed by the main memory for any one variable are totally ordered* [ibid. §17.5].

(4) $s_1 : (Store, \theta, x, v_1)$, $s_2 : (Store, \theta, x, v_2)$ and $a : (Assign, \theta, x, v_2)$. *A thread is not permitted to write data from its working memory back to main memory for no reason* [ibid. §17.3].

(5) and (6) $u : (Use, \theta, x, v)$, $a : (Assign, \theta, x, v)$, $l : (Load, \theta, x, v)$ and $s : (Store, \theta, x, v)$. *Threads start with an empty working memory and new variables*

are created only in main memory and are not initially in any thread's working memory [ibid. §17.3].

(7) and (8) $a_1 : (Assign, \theta, x, v_1)$, $u : (Use, \theta, x, v_2)$, with $v_2 \neq v_1$, l and $l_2 : (Load, \theta, x, v_2)$, a_2 and $a : (Assign, \theta, x, v_2)$, $l_1 : (Load, \theta, x, v_1)$. *A Use action transfers the contents of the thread's working copy of a variable to the thread's execution engine* [ibid. §17.1].

(9) $a_1 : (Assign, \theta, x, v_1)$, $s : (Store, \theta, x, v_2)$, $a_2 : (Assign, \theta, x, v_2)$ and $v_2 \neq v_1$. *A Store action transmits the contents of the thread's working copy of a variable to main memory* [ibid. §17.1].

(10) and (11) $w_i : (Write, \theta, x, v_i)$, $s_i : (Store, \theta, x, v_i)$, $l_i : (Load, \theta, x, v_i)$ and $r_i : (Read, \theta, x, v_i)$, for $i = 1 \ldots n$. *Each Load or Write action is uniquely paired with a preceding Read or Store action respectively. Matching actions bear identical values* [ibid. §17.2,§17.3].

*(***12)** Labels as above. *The actions on the master copy of any given variable on behalf of a thread are performed by the main memory in exactly the order that the thread requested* [ibid. §17.3].

The Java language specification states that *a thread is not permitted to write data from its working memory back to main memory for no reason.* Axiom (4) alone does not seem to guarantee this property, and in fact [GJS96, §17.3] introduces explicitly a similar clause requiring that an assignment exists in between a *load* and a subsequent *store*. This is expressed formally by a version of axiom (4), call it (4-bis), where s_1 is replaced by $l : (Load, \theta, x, v_1)$. In [CKRW98] we proved that (4-bis) follows from the other axioms (in a non-obvious way). Here we are able to derive this sequent formally in the theory of Java. However, to do so we need a new rule, [grd], stating that configurations are *grounded*, that is: there are no infinite descending chains of events (see Proposition 3).

Let $s : A \rightarrowtail B$ and let $t : A \rightarrowtail D$. We write $t \ll s$ if there exists $r : A \rightarrowtail B$, $r \neq s$, such that for all $a \in A$:

- if $r(a) < s(a)$ then $D \uparrow t(a) \subseteq t(A)$;
- if $r(a) > s(a)$ then $D \downarrow t(a) \subseteq t(A)$ and there exists $a' \in A$ such that $r(a) < r(a') \leq s(a')$.

When A, B and D are *chains*, that is totally ordered sets, the condition expressed by $\ll$ allows the following inference:

$$[\text{grd}] \quad \frac{A \vdash_{\rho,\sigma} D, B}{A \vdash_{\rho} D} \quad (\text{if } A,\ B,\ D \text{ are chains and } \rho \ll \sigma)$$

By suitably generalising the relation $\ll$, this rule can be extended to arbitrary posets. The proof of soundness for [grd] is rather lengthy. Here we just give the intuition with an example: Let $A = \{a\}$, $B = \{a_1 < a_2\}$, $D = \{b < a_3\}$, let a and

the a_i be labelled by l and b by $l' \neq l$. Moreover, let $\sigma(a) = a_2$ and $\rho(a) = a_3$. It is easy to check that $\rho \ll \sigma$, where the required $r : A \rightarrowtail B$ is $r(a) = a_1$. To wit, σ can be viewed as iteratively "generating" events (namely a_2) below a. But iteration cannot go on indefinitely: the reader can verify that any configuration C of a structure $\mathcal{C}$ satisfying (ρ, σ) must feature a chain of l-actions preceeded by an l'-action (postulated by ρ; this justifies the notation $\rho \ll \sigma$), or otherwise no l-actions at all. Then, any interpretation $A \to C$ factorising through σ will also factorise through ρ, which means that $\mathcal{C}$ satisfies ρ.

Derivation of (4-bis). Events are meant as labelled according to the convention introduced above. Let $A = \{a < l < s\}$, $B = \{a_1 < s_1 < a_2 < l_1 < s_2\}$ and $D = \{a_3 < s_3 < l_2 < a_4 < s_4\}$. Let $\sigma : A \rightarrowtail B$ be the map $\sigma(a) = a_1$; the rest of σ is forced by the labels, and so is $\rho : A \rightarrowtail D$. The sequent $A \vdash_{\rho,\sigma} D, B$ can be derived from axioms (1), (3) and (4) using [extend] and [r-cut]. Since $\rho \ll \sigma$, [grd] yields $A \vdash_\rho D$. Moreover, let $E = \{l_3 < s_5\}$ and $F = \{l_4 < a_5 < s_6\}$, and let $\tau : E \rightarrowtail A$ and $\pi : E \rightarrowtail F$ be the obvious maps. From (1) and (6) we derive $E \vdash_{\tau,\pi} A, F$ and hence, by [r-cut], $E \vdash_{\tau\rho,\pi} D, F$. Since $\pi \leq (\tau\rho, \pi)$, [sub] yields $E \vdash_\pi F$ as required. □

Acknowledgements

Thanks to Alexander Knapp and Anna Labella for the many useful discussions.

References

[Cen00] P. Cenciarelli. Event Structures for Java. In S. Drossopoulou, S. Eisenbach, B. Jacobs, G. Leavens, P. Mueller, and A. Poetzsch-Heffter, editors, *Proceedings of the ECOOP 2000 Workshop on Formal Techniques for Java Programs, Cannes, France*, June 2000. 200

[CKRW98] P. Cenciarelli, A. Knapp, B. Reus, and M. Wirsing. An Event-Based Structural Operational Semantics of Multi-Threaded Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, 1523 LNCS. Springer, 1998. 200, 211, 213

[GG90] R. J. van Glabbeek and U. Goltz. Refinement of Actions in Causality Based Models. In W. P. de Roever J. W. de Bakker and G. Rozenberg, editors, *LNCS 430*, pages 267–300. Springer-Verlag, 1990. 200, 201, 203

[GG01] R. J. van Glabbeek and U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37:229–327, 2001. 204

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996. 200, 201, 210, 211, 212, 213

[GP95] R. J. van Glabbeek and G. D. Plotkin. Configuration structures (extended abstract). In D. Kozen, editor, *Proceedings of LICS'95*, pages 199–209. IEEE Computer Society Press, June 1995. 203, 204

[NPW81] M. Nielsen, G. D. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains: Part I. *Theoretical Computer Science*, 13(1):85–108, 1981. 203

[Win82] G. Winskel. Event Structure Semantics of CCS and Related Languages. *Springer LNCS*, 140, 1982. Proceedings ICALP'82. 203

[Win87] Glynn Winskel. Event Structures. In G. Rozemberg W. Brauer, W. Reisig, editor, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in LNCS. Springer-Verlag, 1987. 203

A Logic for Probabilities in Semantics

M. Andrew Moshier[1] and Achim Jung[2]

[1] Department of Mathematics, Computer Science and Physics, Chapman University
Orange, CA 92867, USA
moshier@chapman.edu

[2] School of Computer Science, The University of Birmingham
Edgbaston, Birmingham, B15 2TT, England
a.jung@cs.bham.ac.uk

Abstract. Probabilistic computation has proven to be a challenging and interesting area of research, both from the theoretical perspective of denotational semantics and the practical perspective of reasoning about probabilistic algorithms. On the theoretical side, the probabilistic powerdomain of Jones and Plotkin represents a significant advance. Further work, especially by Alvarez-Manilla, has greatly improved our understanding of the probabilistic powerdomain, and has helped clarify its relation to classical measure and integration theory. On the practical side, such researchers as Kozen, Segala, Desharnais, and Kwiatkowska, among others, study problems of verification for probabilistic computation by defining various suitable logics for the classes of processes under study. The work reported here begins to bridge the gap between the domain theoretic and verification (model checking) perspectives on probabilistic computation by exhibiting sound and complete logics for probabilistic powerdomains that arise directly from given logics for the underlying domains.

1 Introduction

The probabilistic powerdomain construction of Jones and Plotkin [17, 16] has proved to have applications beyond its origins as a tool for modelling probabilistic algorithms within domain theory. Edalat [11] employs the probabilistic powerdomain construction toward the study of fractals within a domain theoretic framework. Desharnais, *et al.* [9, 8, 7] study problems of verification for labelled Markov processes. And closer to the construction's origins, Mislove [29] and Tix [36] investigate how to integrate non-deterministic choice and probabilistic algorithms smoothly. McIver [28] looks at a similar problem from a more applied perspective.

The work of Desharnais, *et al.*, McIver, as well as Morgan, *et al.*, [30] are of particular interest to us because they involve the development of logics for reasoning about various probabilistic phenomena (such as labelled Markov processes). They suggest that a uniform treatment of how such logics may arise will prove to be useful. In this work, we provide such a treatment, showing how

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 216–232, 2002.
© Springer-Verlag Berlin Heidelberg 2002

to construct a logical description of the probabilistic powerspace for any stably compact topological space.

At the heart of our approach is an equivalence between (logical) *theories* and (denotational) *models*. On the logical side this means that we work with sets of axioms about concrete propositions and universally valid inference rules. On the semantic side we exhibit the structures which can be characterised by a logical theory. The classical example of such a correspondence is the *Stone Representation Theorem*: Every classical propositional theory corresponds uniquely to a totally disconnected compact Hausdorff space. The insight that Stone duality can be used to link denotational semantics and program logics is due to Smyth. It forms the basis of Abramsky's *Domain Theory in Logical Form* and was put to work in two substantial case studies, [2, 1]. Abramsky does not work with full propositional logic and Stone spaces but, rather, drops negation and implication, and employs the equivalence between theories of the remaining positive propositional logic and spectral spaces (which encompass all classical algebraic semantic domains, such as Scott-domains or bifinite domains). The class of spectral spaces, however, does not contain *continuous* spaces, such as the unit interval, and it is therefore not surprising that the setting needs to be further expanded in order to accommodate probabilities. Indeed, our work [22] is based on a further weakening of the logic by dropping the reflexivity axiom ($\phi \Vdash \phi$) from which the zero dimensionality of the corresponding spectral spaces arises.

This paper stresses the logical side of this correspondence, so it is not necessary to be an expert in the topological properties of stably compact spaces in order to appreciate the results reported below. We summarize the key topological properties in Section 2. The reader interested in a fuller story should consult [20, 24] or the forthcoming [13]. For our present purposes it is sufficient to recall a crucial result in the thesis of Alvarez-Manilla [4], where stably compact spaces are shown to be closed under the probabilistic powerspace construction. The only other known closure results for this construction concern dcpo's (trivially), continuous domains [17, 16], and Lawson-compact continuous domains [21], but unlike SCS none of these categories has a good logical description (via Stone duality, as explained above) nor many other closure properties as one needs for building a denotational semantics.

The logic, as we have said before, is propositional logic restricted to finite conjunction and disjunction (including true and false), and reflexivity is not assumed. Gentzen-style sequents

$$\phi_1, \ldots, \phi_n \Vdash \psi_1, \ldots, \psi_m$$

are the basic syntactic units (specifically, $\Vdash$ is part of the object syntax, as in Gentzen's sequent calculus, and not a meta-symbol denoting provability). The logic was first presented in [18] and [22], but it builds on the earlier [34, 20] and in essence is an elaboration of Abramsky's Domain Theory in Logical Form for continuous spaces. It is shown in [22] that despite non-reflexivity many important proof-theoretic concepts, such as cut elimination, still apply.

Under Stone duality, a proposition ϕ corresponds to an open set $o[\![\phi]\!]$; it was argued by Smyth [32, 38, 35] that this is in order: open sets correspond

to *semi-decidable* properties and these are precisely the ones which ought to be of relevance in program logics. In our setting, furthermore, a sequent $\Gamma \Vdash \Delta$ translates to a "strong containment" $o[\![\Gamma]\!] \Subset o[\![\Delta]\!]$ of open sets which is itself "observable" or "semi-decidable". However, we hasten to add that in the presence of non-determinism or probabilistic choice, the label "observable" has to be taken with a grain of salt.

From a motivational point of view, the language of "observable properties" is, nevertheless, useful for choosing the right primitives for a probabilistic logic. On the spatial side the probabilistic powerspace is gives a topology for the set of all *(normal) probability valuations on* X, i.e., those maps v which assign a probability (value in $[0,1]$) to all open subsets, and which have the following properties:

[Continuity] For directed sets $\{U_i\}_i$ of opens, $v(\bigcup_i U_i) = \sup_i\{v(U_i)\}$.
[Strictness] $v(\emptyset) = 0$.
[Modularity] For all opens U and V, $v(U) + v(V) = v(U \cap V) + v(U \cup V)$.
[Normalcy] $v(X) = 1$.

We call such functions *sub-probability valuations* if (4) is replaced by $v(X) \leq 1$). Probability valuations were first introduced into denotational semantics by the seminal work of Jones and Plotkin [17, 16], whereas earlier work, e.g. by Kozen [25], employed measures. The exact connection between valuations and measures has always been of interest in Mathematics. We only mention [31, 26, 5] and refer to [4] for a comprehensive treatment. On stably compact spaces, the connection is straight-forward: probability valuations extend uniquely to Radon measures and every Radon measure arises in this way. More importantly for us, Alvarez-Manilla shows that the set of (normal) valuations over a stably compact space can be given a stably compact topology that lies between the Scott topology and the topology of weak convergence. This opens the prospect that this probabilistic powerspace can be described logically. Even better, we now know [6] that the topology is actually equal to the weak topology (and is generally finer than the Scott topology). This is of relevance because it shows that Alvarez-Manilla's topology is precisely the weakest topology to make the integral $v \mapsto \int f dv$ a Scott continuous operation for every semi-continuous real-valued f. The canonical subbasic opens for the weak topology are the sets $O_q := \{v \in \mathsf{V}(X) \mid v(O) > q\}$ for O open in X, q a rational number between 0 and 1. In our probabilistic logic we should therefore be able to work with basic propositions ϕ_q, interpreted as "proposition ϕ holds with probability greater than q". This is indeed the approach that we shall take.

To give a *logic* for probability valuations, one needs to find the proof rules for entailments between propositions of this shape and show soundness and completeness with respect to the intended space of all probability valuations. The mathematical context for this becomes clearer by using a modicum of categorical terminology. The stably compact spaces introduced above form a subcategory SCS of the category Top of topological spaces and continuous functions. Also of interest is the category SCS^* where the objects are the same but morphisms are *saturated relations* (see Section 2 below for details). SCS can be identified with

a subcategory of SCS^*. On the logical side, non-reflexive propositional logic is an object in the category MLS, where morphisms between logics are entailment relations $\vdash$ very similar to the internal reasoning in a logic. The key result of [22] is that SCS^* and MLS are equivalent. This equivalence cuts down to one between SCS and $\mathsf{MLS}^{\mathsf{f}}$, where the entailment relations satisfy an additional property capturing functional behavior.

In order more fully to exploit this equivalence between semantics and logic, one then strives to lift it to *constructions*, that is, given a construction C (possibly in several variables) on SCS^*, one seeks a "logical" construction $\overline{\mathsf{C}}$ which respects the equivalence:

$$\begin{tikzcd}
\mathsf{SCS}^* \arrow[r, shift left, "\mathsf{lang}"] & \mathsf{MLS} \arrow[l, shift left, "\mathsf{spec}"] \\
\mathsf{SCS}^* \arrow[u, Rightarrow, "\mathsf{C}"] \arrow[r, shift left, "\mathsf{lang}"] & \mathsf{MLS} \arrow[u, Rightarrow, "\overline{\mathsf{C}}"'] \arrow[l, shift left, "\mathsf{spec}"]
\end{tikzcd}$$

Generally, $\overline{\mathsf{C}}$ is defined via proof rules, and the commutativity of the above diagram is shown by establishing $\mathsf{lang} \circ \mathsf{C} \approx \overline{\mathsf{C}} \circ \mathsf{lang}$.

For the probabilistic powerspace, we borrow an idea of Reinhold Heckmann's [15] and carry out the construction in four stages. This produces logical descriptions for all of the following spatial constructions, useful in their own rights:

- $\mathsf{C}\Omega(X)$, the space of Scott continuous functions from $\Omega(X)$ to $[0,1]$ with the compact-open topology (which coincides with both the weak and the Scott topology);
- $\mathsf{C}_s\Omega(X)$, the subspace of $\mathsf{C}\Omega(X)$ consisting of strict continuous functions;
- $\mathsf{V}(X)$, the subspace of $\mathsf{C}_s\Omega(X)$ consisting of modular strict continuous functions, i.e., sub-probability valuations;
- $\mathsf{V}^1(X)$, the subspace of $\mathsf{V}(X)$ consisting of normal probability valuations.

2 Stably Compact Spaces

A subset of a topological space X is *saturated* if and only if it is an intersection of opens. In particular, every open is saturated and the *saturation* of a subset is the intersection of its neighborhood filter. A subset is compact if and only if its saturation is compact. Compact saturated sets play a key role in our setting. A topological space is called *stably compact* if it is sober, locally compact and stable (i.e., finite sets of compact saturated subsets have compact intersection).

The reader may refer to [33, 35] for arguments in favor of regarding stably compact spaces as a suitable ambient category for carrying out domain theory. We note here that sobriety is needed to exploit Stone duality for representing

spaces by (sublattices of) their frame of opens (which we interpret as extensions of logical propositions). In contrast to Geometric Logic, we axiomatize the way-below relation between open sets, rather than inclusion, and local compactness is precisely the condition which guarantees that the former is rich enough to reconstruct the latter. Stability, finally, is natural because it allows us to deal with opens and compacts in the same logical framework, and is key to allowing us to use finitary logics.

Examples of stably compact spaces include various classes of domains in their Scott-topologies, such as continuous lattices, Scott-domains, bifinite domains, and FS-domains. Also included are all compact Hausdorff spaces.

We denote with $\Omega(X)$ the frame of open sets (ordered by inclusion) and with $\mathcal{K}(X)$ the lattice of compact saturated sets ordered by reversed inclusion. For a stably compact space both are continuous distributive lattices, in particular, $\mathcal{K}(X)$ is the set of closed sets for a topology on X, called the *co-compact topology*. We denote the resulting space by X_κ. From what we have said before it follows that both $\Omega(X)$ and $\mathcal{K}(X)$ are again stably compact when equipped with their Scott-topologies.

For morphisms, there is some choice. The first to come to mind are, of course, the topologically continuous functions, which give rise to the category SCS. However, we prefer to work in SCS^* where the morphisms from X to Y are the compact saturated subsets of $X_\kappa \times Y$ or equivalently, closed subsets of $X \times Y_\kappa$. Hence we refer to such relations simply as *closed relations*. Composition is the usual relational product. Although any relation $R\colon X \mathrel{-\!\!\!\!\!+\!\!\!\!\!\rightarrow} Y$ can be closed in $X \times Y_\kappa$ to yield a morphism in SCS^*, this process is not, in general, functorial.

For a continuous function $f\colon X \to Y$, the *hypergraph* $R_f := \{(x, y) \in X_\kappa \times Y \mid f(x) \leq_Y y\}$, where $\leq_Y$ is specialization in Y, is a closed relation and the assignment $f \mapsto R_f$ is a faithful functor $\mathsf{SCS} \Rightarrow \mathsf{SCS}^*$. Hence SCS can be identified with a subcategory of SCS^*, which turns out to have a co-reflection $\mathcal{K}$ which maps X to $\mathcal{K}(X)$ and $R\colon X \mathrel{-\!\!\!\!\!+\!\!\!\!\!\rightarrow} Y$ to the function $K \mapsto [K]R$.

We will also consider SCS^p where morphisms are (hypergraphs of) functions which are continuous with respect to both the original and the co-compact topology. These are known as *perfect maps*. SCS^* is order enriched by reversed inclusion between the graphs of saturated relations. It then turns out that a closed relation is (the hypergraph of) a perfect function if and only if it is an upper adjoint.

In previous work [22, 24, 19] we have shown that SCS^* enjoys a number of closure properties, to wit, disjoint union (product and coproduct in SCS^*), cartesian product (product in SCS), relation space (Kleisli exponential in SCS^*), lifting, and bilimits. In addition, [6] have shown that for a stably compact space X, the set $V^1(X)$ of probability valuations equipped with the weak topology, is stably compact. The weak topology is generated by sets of the form $O_p := \{v \in V^1(X) \mid v(O) > p\}$, where $O \in \Omega(X)$ and $0 < p < 1$. For a closed relation $R\colon X \mathrel{-\!\!\!\!\!+\!\!\!\!\!\rightarrow} Y$

it is natural to set $v\ V^1(R)\ v' \quad :\Longleftrightarrow \quad \forall U \in \Omega(Y).\ v(R^{-1}[U]) \leq v'(U)$[1]. One observes:

Proposition 1. *In general, $V^1(-)$ does not preserve composition in SCS^*. It is, however, a functor from SCS to SCS, which furthermore restricts and corestricts to SCS^p.*

3 The Multilingual Sequent Calculus

In this section we review the basic ideas of [22], where the category of multilingual sequent calculi (MLS) was first introduced. An algebra for two binary operations and two constants is called a *token algebra*. For example, any lattice $(L; \wedge, \top, \vee, \bot)$ is a token algebra, as is the appropriate term algebra $T(G)$ generated from a set G. For two token algebras L and M, a *consequence relation from L to M* is a relation $\vdash\ \subseteq \mathfrak{P}_{\mathsf{fin}}(L) \times \mathfrak{P}_{\mathsf{fin}}(M)$ obeying Gentzen's rules of positive sequent calculus:

$$\frac{}{\bot \vdash}\,(L\bot) \quad \frac{\phi, \Gamma \vdash \Delta \quad \psi, \Gamma \vdash \Delta}{\phi \vee \psi, \Gamma \vdash \Delta}\,(L\vee) \quad \frac{\phi, \psi, \Gamma \vdash \Delta}{\phi \wedge \psi, \Gamma \vdash \Delta}\,(L\wedge) \quad \frac{\Gamma \vdash \Delta}{\top, \Gamma \vdash \Delta}\,(L\top)$$

$$\frac{}{\vdash \top}\,(R\top) \quad \frac{\Gamma \vdash \Delta, \phi \quad \Gamma \vdash \Delta, \psi}{\Gamma \vdash \Delta, \phi \wedge \psi}\,(R\wedge) \quad \frac{\Gamma \vdash \Delta, \phi, \psi}{\Gamma \vdash \Delta, \phi \vee \psi}\,(R\vee) \quad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, \bot}\,(R\bot)$$

$$\frac{\Gamma \vdash \Delta}{\Gamma', \Gamma \vdash \Delta, \Delta'}\,(\mathsf{W})$$

The double lines in the above figures indicate that the rule applies in both directions. This differs from the usual presentation of a sequent calculus in two important ways. First, the tokens (formulas) on either side of a sequent are drawn from different sets. This immediately precludes closing under (Cut), and from including the usual identity axioms: $\phi \vdash \phi$. Second, in proof theory one typically only requires closure under forward application of the rules. However, in the presence of identity axioms and the (Cut) rule, such a relation is in fact also closed under backward application. Because we do not assume either identity axioms or closure under (Cut), we make the closure under backward application explicit. A third, less important difference is that we allow token algebras to be non-free. This, however, is just a convenience as the category MLS is equivalent to its full subcategory consisting of objects defined on free token algebras (which we examine in the next section). Consequence relations are the morphisms of the category MLS. Composition is defined by the following impoverished version of Gentzen's Cut rule. Given two consequence relations $\vdash : L \to M$ and $\vdash' : M \to N$, define $\vdash ; \vdash'$ by the rule:

$$\frac{\Gamma \vdash \phi \qquad \phi \vdash' \Lambda}{\Gamma \ \vdash ; \vdash' \ \Lambda}\,(\mathsf{S\text{-}Cut})$$

[1] For a closed relation $R\colon X \longrightarrow\!\!\!\!\!+\ Y$ we set $R^{-1}(U) := \{x \in X \mid \forall y \in Y.\ xRy \implies y \in U\}$.

This composition is associative, and consequence relations are closed under it. In case domain and target algebra are the same, one can consider Gentzen's original rule:

$$\frac{\Gamma \Vdash \Delta, \phi \qquad \phi, \Theta \Vdash \Lambda}{\Gamma, \Theta \ \Vdash \circ \Vdash \ \Delta, \Lambda}\,(\mathsf{Cut})$$

We employ (Cut) to define the objects (or, rather, identities) of our category. A *continuous sequent calculus on* L is a consequence relation $\Vdash_L$ from L to L satisfying $\Vdash_L \circ \Vdash_L \subseteq \Vdash_L$, and such that if $\Gamma, \Theta \Vdash_L \Delta, \Lambda$ holds where either Θ or Δ is empty, then there exists ϕ so that $\Gamma \Vdash_L \Delta, \phi$ and $\phi, \Theta \Vdash_L \Lambda$. That is, in a continuous sequent calculus (Cut) also applies in a limited backward form which is sufficient for the other inclusion $\Vdash_L \circ \Vdash_L \supseteq \Vdash_L$ to hold. (Note that we distinguish notationally between composition by (S-Cut) and (Cut), and between general and endo-relations.) We are now ready to define the category MLS: An object of the category MLS is a token algebra equipped with a continuous sequent calculus $L = (L, \Vdash_L)$. A morphism from L to M is a consequence relation $\vdash\colon L \to M$ that is *compatible* with L and M:

$$\Vdash_L ; \vdash \ = \ \vdash \ = \ \vdash ; \Vdash_M$$

This leads to the major result of [22]:

Theorem 1. *The categories* MLS *and* SCS^* *are equivalent.*

In one direction, the isomorphism is given by $\mathsf{spec}\colon \mathsf{MLS} \Rightarrow \mathsf{SCS}^*$, which assigns to a continuous sequent calculus the set of prime round filters, topologized in the usual way. We describe the inverse at the beginning of Section 5.

Like SCS^*, MLS is order-enriched (by inclusion between consequence relations). The equivalence preserves this enrichment and hence it restricts and corestricts to $\mathsf{SCS}^{\mathsf{p}}$ and $\mathsf{MLS}^{\mathsf{u}}$, the category of upper adjoint consequence relations. We will exhibit a general method for defining adjoints in MLS below.

4 Free Token Algebras

In Logic, formulas are normally built up freely from a set of atomic propositions. The analogous situation for MLS is given by a free term algebra $T(G)$ over a set of generators G. In this section, we explore how far the concepts of the multilingual sequent calculus can be expressed solely in terms of generators. This will provide us with the basic toolkit for doing domain constructions in a purely proof-theoretic fashion. First we note that consequence relations are completely determined by their behavior on generators.

Lemma 1 ([24]). *Let $L = T(G)$ and $M = T(H)$ be free token algebras and $R \subseteq \mathfrak{P}_{\mathsf{fin}}(G) \times \mathfrak{P}_{\mathsf{fin}}(H)$ be a relation. Denote with R^w the closure of R under weakening with generators and R^+ the further closure under the forward logical rules.*

1. *R^+ is a consequence relation.*
2. *R^+, when restricted to generators, equals R^w.*
3. *For an arbitrary consequence relation $\vdash$ from L to M, $\vdash = R^+$ where R is the restriction of $\vdash$ to generators.*

In general, a cut formula can not be restricted to generators but with a slight generalization we do succeed. For a set G, define a *diagonal pair on G* to be a pair $\langle \{C_i\}_i, \{D_j\}_j \rangle$, both sets of subsets of G, provided that for each choice function $f \in \prod_i C_i$ and choice function $g \in \prod_j D_j$, there exists i and j so that $f(i) = g(j)$. Given two consequence relations $\vdash\colon L \to T(G)$ and $\vdash'\colon T(G) \to N$, define $\vdash;\vdash'$ by the rule:

$$\frac{\begin{array}{cc} \Gamma \vdash \Delta_1 & \Theta_1 \vdash' \Lambda \\ \vdots & \vdots \\ \Gamma \vdash \Delta_m & \Theta_n \vdash' \Lambda \end{array}}{\Gamma\ (\vdash;\vdash')\ \Lambda}\ (\mathsf{Cut}^*)$$

subject to the condition that $\langle \{\Delta_i\}_{i=1}^m, \{\Theta_j\}_{j=1}^n \rangle$ is a diagonal pair on G.

The following justifies re-using ";" for composition:

Lemma 2 ([24]). *In the presence of the logical rules, (*S-Cut*) and (*Cut**) are interdefinable.*

For the identities we need to simulate the stronger requirement of idempotence with respect to (Cut).

Lemma 3. *For a consequence relation $\Vdash$ on a free token algebra $L = T(G)$, $\Vdash$ is a continuous sequent calculus if and only if $\Vdash;\Vdash\ \subseteq\ \Vdash$, and* [L-Int] *and* [R-Int] *where*

[L-Int] *If $\phi, \Gamma \Vdash \Delta$, then there exists a diagonal pair $\langle \{\Lambda_i\}_i, \{\Theta_j\}_j \rangle$ in G so that $\phi \Vdash \Lambda_i$ holds for each i, and $\Theta_j, \Gamma \Vdash \Delta$ holds for each j.*

[R-Int] *If $\Gamma \Vdash \Delta, \psi$, then there exists a diagonal pair $\langle \{\Lambda_i\}_i, \{\Theta_j\}_j \rangle$ in G so that $\Gamma \Vdash \Delta, \Lambda_i$ holds for each i, and $\Theta_j \Vdash \psi$ holds for each j.*

Our general strategy for defining functors $F\colon \mathcal{A} \Rightarrow \mathsf{MLS}$ will be the following:

1. [Basic tokens] For object A, define a set $G_F(A)$ and let the token algebra $F(A)$ be the term algebra $T(G_F(A))$ over $G_F(A)$.
2. [Proof rules] For a morphism $f\colon A \to B$, define $F^0(f)$ to be a relation from finite subsets of $G_F(A)$ to finite subsets of $G_F(B)$, and let $F(f)$ be $(F^0(f))^+$.
3. [Composition] Show that $F(g \circ f) = F(f); F(g)$. Because $F(-)$ is determined by its restriction to generators, this reduces to
   (a) [(Cut*) elimination] $F^0(f); F^0(g) \subseteq [F^0(g \circ f)]^w$; and
   (b) [(Cut*) introduction] $F^0(g \circ f) \subseteq [F^0(f); F^0(g)]^w$.
4. [Identities] Show that F preserves identities. In light of [(Cut*) elimination] above, this reduces to showing that $F^0(\Vdash)$ satisfies [L-Int] and [R-Int].

We label step (2) [Proof rules] because $F^0(f)$ can typically be presented in the form:

$$\frac{P(f\colon A \to B, \Gamma, \Delta)}{\Gamma F(f) \Delta}\,(F)$$

where P is some predicate on morphisms of $\mathcal{A}$ and finite sets of generators. The first two steps of the method are purely formal. The third and fourth steps constitute the verification that we have defined a functor. Also note that the conditions [(Cut^*) introduction], [L-Int] and [R-Int] are quite natural in traditional proof theory. They amount to the requirement that derivable sequents can always arise as the result of (Cut^*) of a specific form. This sort of meta-theorem is used, for example, to derive the Craig Interpolation Theorem: If $\Gamma \Rightarrow \Delta$, then there is a formula ϕ involving only non-logical symbols occurring in both Γ and Δ so that $\Gamma \Rightarrow \phi$ and $\phi \Rightarrow \Delta$. Thus the conditions on functors amount to a formalization of "good" behavior for constructions in the logic MLS.

Our principle tool for showing that two objects of MLS are isomorphic is the following.

Lemma 4. *Suppose L and M are continuous sequent calculi and $h\colon M \to L$ is a map between the underlying token algebras. Consider the following properties*

- [hom] *h is a homomorphism.*
- [smooth] *Whenever $\Gamma \Vdash_L h(\phi)$ then there exists $\phi' \in M$ such that $\phi' \Vdash_M \phi$ and $\Gamma \Vdash_L h(\phi')$. Likewise, with $h(\phi) \Vdash_L \Gamma$ we have $\phi \Vdash_M \phi'$ such that $h(\phi') \Vdash_L \Gamma$.*
- [$\Vdash$-preserving] *$\Delta \Vdash_M \Delta'$ implies $h(\Delta) \Vdash_L h(\Delta')$ (where $h(\Delta)$ is short for $h(\psi_1), \ldots, h(\psi_n)$ whenever $\Delta = \psi_1, \ldots, \psi_n$).*
- [$\Vdash$-reflecting] *$h(\Delta) \Vdash_L h(\Delta')$ implies $\Delta \Vdash_M \Delta'$.*
- [dense] *$\Gamma \Vdash_L \Gamma'$ implies that there exists $\phi \in M$ with $\Gamma \Vdash_L h(\phi) \Vdash_L \Gamma'$.*

Define relations $\vdash^h \subseteq \mathfrak{P}_{\mathsf{fin}}(L) \times \mathfrak{P}_{\mathsf{fin}}(M)$ and $\vdash_h \subseteq \mathfrak{P}_{\mathsf{fin}}(M) \times \mathfrak{P}_{\mathsf{fin}}(L)$ by setting

$$\Gamma \vdash^h \Delta \text{ if } \Gamma \Vdash_L h(\Delta), \text{ and } \Delta \vdash_h \Gamma \text{ if } h(\Delta) \Vdash_L \Gamma.$$

1. *If h is a smooth homomorphism then $\vdash^h$ and $\vdash_h$ are compatible consequence relations.*
2. *If h is a smooth homomorphism which is also $\Vdash$-preserving then $\vdash^h$ is the upper adjoint to $\vdash_h$. That is, $(\vdash^h; \vdash_h) \subseteq \Vdash_L$ and $\Vdash_M \subseteq (\vdash_h; \vdash^h)$.*
3. *If h is a smooth homomorphism which is also $\Vdash$-reflecting then $(\vdash_h; \vdash^h) \subseteq \Vdash_M$.*
4. *If h is a smooth homomorphism which is also dense then $\Vdash_L \subseteq (\vdash^h; \vdash_h)$.*

We observe that in the presence of $\Vdash$-preservation, the homomorphism condition is not needed. In practice, however, M is often a free token algebra $T(G)$ and h is defined as the homomorphic extension of a map from G to L. In this situation it is sufficient to check smoothness, $\Vdash$-preservation and reflection for lists Δ of generators only.

Also note that in the presence of $\Vdash$-reflection, smoothness is subsumed by density. With these two observations, the following extension from objects to functors becomes a straightforward corollary.

Lemma 5. *Suppose* $F\colon \mathcal{A} \Rightarrow \mathsf{MLS}$ *and* $G\colon \mathcal{A} \Rightarrow \mathsf{MLS}$ *are functors, and for each object* $A \in \mathcal{A}$, $h_A\colon G(A) \to F(A)$ *is a dense map between token algebras. If for each* $f\colon A \to B$ *in* $\mathcal{A}$,

$$\Gamma\ G(f)\ \Delta \text{ if and only if } h_A(\Gamma)\ F(f)\ h_B(\Delta)$$

then $\vdash^{h_A}$ *is a natural isomorphism from* F *to* G *with inverse* $\vdash_{h_A}$.

5 Domain Constructions in Logical Form

We will now illustrate how the general techniques of the previous section can be used for proving that an endofunctor functor $\overline{\mathsf{C}}$ in MLS is a logical description of an endofunctor C in SCS^* following the ideas outlined in Section 1. We start by defining a functor lang from SCS^* to MLS (which is in fact one half of the equivalence stated in Theorem 1). We set

$$G_{\mathsf{lang}}(X) := \{(O, K) \in \Omega(X) \times \mathcal{K}(X) \mid O \subseteq K\}$$

and let $\mathsf{lang}(X)$ be the free term algebra over these generators. For each closed relation $R\colon X \mathrel{-\!\!\!\vdash\!\!\!\rightarrow} Y$, define $\vdash_R = \mathsf{lang}(R)$ by the rule:

$$\frac{[\bigcap_{i=1}^{m} K_i]R \subseteq \bigcup_{j=1}^{n} O'_j}{(O_1, K_1), \ldots, (O_m, K_m) \vdash_R (O'_1, K'_1), \ldots, (O'_n, K'_n)}\ (\mathsf{lang})$$

We refer the reader to [22] for the proof that spec and lang determine an equivalence.

By a *construction* over spaces we mean a functor $\mathsf{C}\colon \mathsf{SCS}^* \Rightarrow \mathsf{SCS}^*$. We seek to find an analogue $\overline{\mathsf{C}}$ on the side of MLS, that is, we wish to show that the two functors $\mathsf{lang} \circ \mathsf{C}$ and $\overline{\mathsf{C}} \circ \mathsf{lang}$ are naturally isomorphic. For this we will employ the general technique described in the previous section, adapted to this special situation. Consider the objects first: Because SCS^* and MLS are isomorphic categories, we can replace $\mathsf{lang}(X)$ by an isomorphic "concrete" sequent calculus L, where the isomorphism is witnessed in the style of Lemma 4 by a dense, $\Vdash$-preserving and reflecting map sending tokens $\phi \in L$ to generators $\langle o_L[\![\phi]\!], k_L[\![\phi]\!]\rangle$ of $\mathsf{lang}(X)$. The task, then, is to define a sequent calculus $\overline{\mathsf{C}}(L)$ isomorphic to $\mathsf{lang} \circ \mathsf{C}(X)$. We do this by exhibiting a set of generators G_L for $\overline{\mathsf{C}}(L)$ together with interpretations $o_{\overline{\mathsf{C}}(L)}[\![-]\!]\colon G_L \to \Omega(\mathsf{C}(X))$ and $\kappa_{\overline{\mathsf{C}}(L)}[\![-]\!]\colon G_L \to \mathcal{K}(\mathsf{C}(X))$ such that the unique homomorphic extension of the map $g \mapsto \langle o_{\overline{\mathsf{C}}(L)}[\![g]\!], k_{\overline{\mathsf{C}}(L)}[\![g]\!]\rangle$ satisfies the conditions of Lemma 4.

For morphisms, the task is almost the same. We assume maps $o_L[\![-]\!]$, $\kappa_L[\![-]\!]$ and $o_K[\![-]\!]$, $\kappa_K[\![-]\!]$ which witness $L \cong \mathsf{lang}(X)$, and $M \cong \mathsf{lang}(Y)$, respectively. We also assume that the compatible consequence relation $\vdash\colon L \to M$ represents the SCS^* relation $R\colon X \mathrel{-\!\!\!\vdash\!\!\!\rightarrow} Y$ in the sense that

- $\forall \phi \in L, \psi \in M.\ \phi \vdash \psi$ if and only if $[\kappa_L[\![\phi]\!]]R \subseteq o_M[\![\psi]\!]$.

This property must be preserved by the spatial and the logical construction:

– $\forall \Gamma \subseteq G_L, \Delta \subseteq G_M.\ \Gamma\ \mathsf{C}(\vdash)\ \Delta$ if and only if $[\![\bigcap_{\phi\in\Gamma} \kappa_{\overline{\mathsf{C}}(L)}[\![\phi]\!]]\!]\mathsf{C}(R) \subseteq \bigcup_{\psi\in\Delta} o_{\overline{\mathsf{C}}(M)}[\![\psi]\!]$.

6 The Probabilistic Powerspace Construction

We are now ready to embark on our logical characterisation of the probabilistic powerspace of a stably compact space. Since a direct proof, despite the tools above, is still too complicated, we perform the construction in four stages, starting with the function space $\mathsf{C}\Omega(X) = [\Omega(X) \to [0,1]]$. This follows the strategy in [15].

We first observe that because both $\Omega(X)$ and $[0,1]$ are continuous lattices, $\mathsf{C}\Omega(X)$ is also a continuous lattice and therefore stably compact in its Scott-topology. The latter coincides with the weak topology generated by sets of the form

$$O_p := \{v \in \mathsf{C}\Omega(X) \mid v(O) > p\}$$

We therefore choose as generators for $\mathsf{C}\Omega(L)$ tokens ϕ_p where $\phi \in L$ and $0 < p < 1$ with the following interpretation function for open sets:

$$o_{\mathsf{C}\Omega}[\![\phi_p]\!] := \{v \in \mathsf{C}\Omega(X) \mid v(o_L[\![\phi]\!]) > p\}$$

For the compact interpretation we define $\overline{v}\colon \mathcal{K}(X) \to [0,1]$ by

$$\overline{v}(K) := \inf\{v(U) \mid U \supseteq K\}$$

and set

$$\kappa_{\mathsf{C}\Omega}[\![\phi_p]\!] := \{v \in \mathsf{C}\Omega(X) \mid \overline{v}(\kappa_L[\![\phi]\!]) \geq p\}$$

The consequence relation on $\mathsf{C}\Omega(L)$ is generated by the single proof rule

$$\frac{\phi \Vdash_L \psi \qquad p > q}{\phi_p \Vdash_{\mathsf{C}\Omega} \psi_q}\ (\mathsf{C}\Omega)$$

Using the general technique outlined in the previous section, it is now not too hard to show that this indeed is a logical description of $\mathsf{C}\Omega(X)$:

Proposition 2. *$\overline{\mathsf{C}\Omega}(L)$ and $\mathsf{lang}(\mathsf{C}\Omega(X))$ are isomorphic.*

The extension to morphisms is straightforward:

$$\frac{\phi \vdash \psi \qquad p > q}{\phi_p\ \mathsf{C}\Omega(\vdash)\ \psi_q}\ (\mathsf{C}\Omega)$$

and together with the previous proposition this yields:

Theorem 2. *The functor $\overline{\mathsf{C}\Omega}\circ\mathsf{lang}$ is naturally isomorphic to $\mathsf{lang}\circ\mathsf{C}\Omega$, in other words, $\overline{\mathsf{C}\Omega}\colon \mathsf{MLS} \Rightarrow \mathsf{MLS}$ is a logical description of the construction $\mathsf{C}\Omega\colon \mathsf{SCS}^* \Rightarrow \mathsf{SCS}^*$.*

We refine the isomorphism established in the preceding Theorem by restricting the construction to more specialized function spaces. Let us first consider the general situation. Suppose already have a logical description L of a space X and seek a logical description for a subspace $Y \subseteq X$. The idea is to keep the token algebra L but to strengthen the internal reasoning $\Vdash$ with additional proof rules, resulting in a consequence relation $\Vdash'$. This is in analogy to locale theory where a sublocale is defined as a congruence on the frame. In our setting, we intend to use Lemma 4 with h being the *identity* on L. It is then immediate that (hom) and ($\Vdash$-preservation) are satisfied, and that ($\Vdash$-reflection) cannot hold unless $Y = X$. What needs to be shown is smoothness and density, which can be expressed as $\Vdash ; \Vdash' = \Vdash' = \Vdash' ; \Vdash$. Since $\Vdash'$ is given by an additional proof rule, the inclusions $\Vdash ; \Vdash' \subseteq \Vdash'$ and $\Vdash' ; \Vdash \subseteq \Vdash'$ hold by convention, and it all boils down to showing the other directions. In the situation at hand, this will not be difficult.

Once this work is done, we conclude from Lemma 4 that $\mathsf{spec}(L, \Vdash')$ is a perfect subspace of $\mathsf{spec}(L, \Vdash) \cong X$, and it remains to show that this subspace is indeed the desired Y. To this end, one shows that for $x \in X$, the neighborhood filter is closed under the new proof rule if and only if $x \in Y$. This will complete the argument.

To restrict to those functions in $\mathsf{C}\Omega(X)$ which assign 0 to the empty set, we add the rule

$$\frac{}{\bot_p \Vdash} \ (\mathsf{Str})$$

The resulting construction is still functorial on all of SCS^* and MLS, respectively.

For modularity, note that our tokens stipulate lower bounds only. So we must break modularity into its constituent inequalities. Say that $v \colon \Omega(X) \to [0,1]$ is *sub-modular* if

$$v(U) + v(V) \leq v(U \cup V) + v(U \cap V)$$

and that v is *super-modular* if

$$v(U) + v(V) \geq v(U \cup V) + v(U \cap V)$$

These two properties are characterised by the following proof rules. For sub-modularity add:

$$\frac{\phi \Vdash_L \rho \quad \psi \Vdash_L \rho \quad \phi, \psi \Vdash_L \sigma \quad p + q > r + s}{\phi_p, \psi_q \Vdash_{\mathsf{V}(L)} \rho_r, \sigma_s} \ (\mathsf{Sub\text{-}mod})$$

and for super-modularity add:

$$\frac{\phi \Vdash_L \rho \quad \phi \Vdash_L \sigma \quad \psi \Vdash_L \rho, \sigma \quad p + q > r + s}{\phi_p, \psi_q \Vdash_{\mathsf{V}(L)} \rho_r, \sigma_s} \ (\mathsf{Super\text{-}mod})$$

We note that the resulting construction V is functorial only for SCS and $\mathsf{MLS}^{\mathsf{f}}$, respectively. This restriction is not too surprising because SCS^* is the Kleisli category of SCS with respect to the monad $\mathcal{K}$, which on domains is known to be the Smyth-powerdomain [3, Thm 6.2.14]. Having V functorial on SCS^* would

therefore amount to a combination of nondeterminism and probabilistic choice. It has become clear recently that this problem cannot have a simple solution because there is no distributive law between these two constructions. We refer the reader to [29, 36, 37] for a more detailed discussion.

To complete our construction we consider the condition $v(X) = 1$ for normal valuations. In L, $o[\![\phi]\!] = X$ if and only if $\Vdash_L \phi$ (if and only if ϕ is logically equivalent to $\top$ with respect to $\Vdash_L$). So $\Vdash_{\mathsf{V}(L)}$ restricts further to normal valuations by adding the rule:

$$\frac{}{\Vdash_{\mathsf{V}^1(L)} \top_q}\ (\mathsf{Norm})$$

All rules necessary to characterize $\mathsf{V}^1(X)$ are collected together in Figure 1.

We conclude by stating a result which is shown with very different methods than the ones employed in the present note, and which we cannot fully spell out for lack of space:

Theorem 3. *If the continuous sequent calculus L is decidable, then so is $V^1(L)$.*

$$\frac{\phi \Vdash_L \psi \qquad p > q}{\phi_p \Vdash \psi_q}\ (\mathsf{C}\Omega)$$

$$\frac{}{\bot_p \Vdash}\ (\mathsf{Str}') \qquad \frac{}{\Vdash \top_q}\ (\mathsf{Norm})$$

$$\frac{\phi \Vdash_L \rho \quad \psi \Vdash_L \rho \quad \phi, \psi \Vdash_L \sigma \quad p + q > r + s}{\phi_p, \psi_q \Vdash \rho_r, \sigma_s}\ (\mathsf{Sub\text{-}mod})$$

$$\frac{\phi \Vdash_L \rho \quad \phi \Vdash_L \sigma \quad \psi \Vdash_L \rho, \sigma \quad p + q > r + s}{\phi_p, \psi_q \Vdash \rho_r, \sigma_s}\ (\mathsf{Super\text{-}mod})$$

where $p, q, r, s \in \mathbb{Q} \cap (0, 1)$, $\phi, \psi, \rho, \sigma \in (L, \Vdash_L)$.
The entailment $\Vdash$ in the conclusions refers to the continuous sequent calculus $\overline{\mathsf{V}}^1(L)$.

Fig. 1. The proof rules for probabilistic domain logic

7 Conclusions and Further Work

The papers [22, 24, 23] and the present note confirm, in our opinion, that the category SCS^* offers a flexible and convenient universe of semantic spaces. As we have emphasized all along, one of its key features is its intimate relationship with (very standard!) logic via Stone duality. This allows us to describe spaces and constructions spatially, localically, and logically in a straightforward and elegant fashion.

Trying to establish the equivalence of logical and spatial domain constructions on the logical side has shown that this requires concepts and techniques

from Proof Theory such as cut elimination and interpolation, a connection which has hitherto — to the best of our knowledge — not been observed.

SCS^* strictly extends all common classes of algebraic and continuous domains, and contains classical spaces such as the unit interval in its Hausdorff topology. The probabilistic powerdomain shows that this extension is necessary, as there is no other suitably closed category available to us which accommodates this construction. The modularity axioms of our logical characterisation of the probabilistic powerdomain also demonstrate that the extension of domain logic to full (rather than intuitionistic) sequents is advantageous.

As a semantic universe, SCS^* takes the notion of a non-deterministic (rather than functional) computation as basic, which is, of course, reminiscent of traditional work in programming languages [10], but which has also more recently been found to be fundamental to exact real number computation [27]. This provides an exciting prospect for future work.

In previous work, [16, 14, 36], the probabilistic powerdomain has been characterised as a free cone over the space X. It is would be interesting to see if this characterization can be used to prove completeness of our axiomatization without referring to the spatial side at all. Such an approach was carried out successfully in [24] for the more "categorical" constructions on SCS^*.

Having laid the groundwork, it should now be possible to establish the precise connection to work in probabilistic verification. More speculatively, perhaps, one could also try to extend the present work so as to capture more accurately truly observable properties of probabilistic programs, that is, to model the Bayesian view of probability.

Acknowledgements

The research reported here was started when the first author visited the School of Computer Science of the University of Birmingham in the Summer of 2001, supported by a guest professorship of that department. We have also greatly profited from insightful comments by anonymous referees on this and an earlier version of the paper.

References

[1] S. Abramsky. The lazy lambda calculus. In D. Turner, editor, *Research Topics in Functional Programming*, pages 65–117. Addison Wesley, 1990. 217

[2] S. Abramsky. A domain equation for bisimulation. *Information and Computation*, 92:161–218, 1991. 217

[3] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Clarendon Press, 1994. 227

[4] M. Alvarez-Manilla. *Measure theoretic results for continuous valuations on partially ordered spaces.* PhD thesis, Imperial College, University of London, 2001. 217, 218

[5] M. Alvarez-Manilla, A. Edalat, and N. Saheb-Djahromi. An extension result for continuous valuations. *Journal of the London Mathematical Society*, 61:629–640, 2000. 218

[6] M. Alvarez-Manilla, A. Jung, and K. Keimel. Valuations on a stably compact space. In preparation. 218, 220

[7] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, to appear. 216

[8] Josee Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. In *Proceedings of the 12th IEEE Symposium on Logic in Computer Science*, pages 149–158, 1997. 216

[9] Josee Desharnais, Abbas Edalat, and Prakash Panangaden. A logical characterization of bisimulation for labeled markov processes. In *Logic in Computer Science*, pages 478–487, 1998. 216

[10] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976. 229

[11] A. Edalat. Dynamical systems, measures and fractals via domain theory. *Information and Computation*, 120(1):32–48, 1995. 216

[12] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer Verlag, 1980. 230

[13] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*. Cambridge University Press, 2002. Revised edition of [12], forthcoming. 217

[14] R. Heckmann. Spaces of valuations. In S. Andima, R. C. Flagg, G. Itzkowitz, P. Misra, Y. Kong, and R. Kopperman, editors, *Papers on General Topology and Applications: Eleventh Summer Conference at the University of Southern Maine*, volume 806 of *Annals of the New York Academy of Sciences*, pages 174–200, 1996. 229

[15] Reinhold Heckmann. Probabilistic power domains, information systems, and locales. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics VIII*, pages 410–437, 1994. In LNCS 802:1994. 219, 226

[16] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, Edinburgh, 1990. Also published as Technical Report No. CST-63-90. 216, 217, 218, 229

[17] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings of the 4th Annual Symposium on Logic in Computer Science*, pages 186–195. IEEE Computer Society Press, 1989. 216, 217, 218

[18] A. Jung, M. Kegelmann, and M. A. Moshier. Multi lingual sequent calculus and coherent spaces. In S. Brookes and M. Mislove, editors, *13th Conference on Mathematical Foundations of Programming Semantics*, volume 6 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1997. 18 pages. 217

[19] A. Jung, M. Kegelmann, and M. A. Moshier. Stably compact spaces and closed relations. In S. Brookes and M. Mislove, editors, *17th Conference on Mathematical Foundations of Programming Semantics*, volume 45 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 2001. 24 pages. 220

[20] A. Jung and Ph. Sünderhauf. On the duality of compact vs. open. In S. Andima, R. C. Flagg, G. Itzkowitz, P. Misra, Y. Kong, and R. Kopperman, editors, *Papers on General Topology and Applications: Eleventh Summer Conference at the*

University of Southern Maine, volume 806 of *Annals of the New York Academy of Sciences*, pages 214–230, 1996. 217

[21] A. Jung and R. Tix. The troublesome probabilistic powerdomain. In A. Edalat, A. Jung, K. Keimel, and M. Kwiatkowska, editors, *Proceedings of the Third Workshop on Computation and Approximation*, volume 13 of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers B.V., 1998. 23 pages. 217

[22] Achim Jung, Mathias Kegelmann, and M. Andrew Moshier. Multi lingual sequent calculus and coherent spaces. *Fundamenta Informaticae*, 37:369–412, 1999. 217, 219, 220, 221, 222, 225, 228

[23] Achim Jung, Matthias Kegelmann, and M. Andrew Moshier. Stably compact spaces and closed relations. In Stephen Brookes and Michael Mislove, editors, *Electronic Notes in Theoretical Computer Science*, volume 45. Elsevier Science Publishers, 2001. 228

[24] M. Kegelmann. Factorisation systems on domains. *Applied Categorical Structures*, 7(1–2):113–128, 1999. 217, 220, 222, 223, 228, 229

[25] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981. 218

[26] J. D. Lawson. Valuations on continuous lattices. In Rudolf-Eberhard Hoffmann, editor, *Continuous Lattices and Related Topics*, volume 27 of *Mathematik Arbeitspapiere*, pages 204–225. Universität Bremen, 1982. 218

[27] J.R. Longley. When is a functional program not a functional program? In *Proceedings of Fourth ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 1999. 229

[28] Annabelle McIver. A generalisation of stationary distributions, and probabilistic program algebra. In Stephen Brookes and Michael Mislove, editors, *Electronic Notes in Theoretical Computer Science*, volume 45. Elsevier Science Publishers, 2001. 216

[29] M. W. Mislove. Nondeterminism and probabilistic choice: Obeying the law. In *Proceedings 11th CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 350–364. Springer Verlag, 2000. 216, 228

[30] Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996. 216

[31] N. Saheb-Djahromi. CPO's of measures for nondeterminism. *Theoretical Computer Science*, 12:19–37, 1980. 218

[32] M. B. Smyth. Powerdomains and predicate transformers: a topological view. In J. Diaz, editor, *Automata, Languages and Programming*, volume 154 of *Lecture Notes in Computer Science*, pages 662–675. Springer Verlag, 1983. 217

[33] M. B. Smyth. Totally bounded spaces and compact ordered spaces as domains of computation. In G. M. Reed, A. W. Roscoe, and R. F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 207–229. Clarendon Press, 1991. 219

[34] M. B. Smyth. Stable compactification I. *Journal of the London Mathematical Society*, 45:321–340, 1992. 217

[35] M. B. Smyth. Topology. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science, vol. 1*, pages 641–761. Clarendon Press, 1992. 217, 219

[36] R. Tix. *Continuous D-Cones: Convexity and Powerdomain Constructions*. PhD thesis, Technische Universität Darmstadt, 1999. 216, 228, 229

[37] D. Varacca. The powerdomain of indexed valuations. In *17th Logic in Copmuter Science Conference*. IEEE Computer Society Press, 2002. 228

[38] S. J. Vickers. *Topology Via Logic*, volume 5 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1989. 217

Possible World Semantics for General Storage in Call-By-Value

Paul Blain Levy

PPS, Université Denis Diderot
Case 7014, 2 Place Jussieu, 75251 Paris Cedex 05, France
paul.levy@pps.jussieu.fr

Abstract. We describe a simple denotational semantics, using possible worlds, for a call-by-value language with ML-like storage facilities, allowing the storage of values of any type, and the generation of new storage cells. We first present a criticism of traditional Strachey semantics for such a language: that it requires us to specify what happens when we read non-existent cells. We then obtain our model by modifying the Strachey semantics to avoid this problem.
We describe our model in 3 stages: first no storage of functions or recursion (but allowing storage of cells), then we add recursion, and finally we allow storage of functions. We discuss similarities and differences between our model and Moggi's model of ground store. A significant difference is that our model does not use monadic decomposition of the function type.

1 Storage and Its Denotational Models

1.1 Overview

Many call-by-value (CBV) programming languages such as ML and Scheme provide a facility to store values in *cells*, i.e. memory locations. In ML, these cells are typed using `ref`: a cell storing values of type A is itself a value of type `ref` A. To date, besides recent work [1] blending operational and denotational semantics, there have been 3 ways of modelling such a CBV language denotationally:

- traditional Strachey-style semantics, used e.g. in [2]
- possible world semantics, used in [3, 4, 5] to model storage of ground values only
- game semantics [6].

In this paper, we argue that Strachey-style semantics, whilst very natural for a language with a fixed set of cells, is unnatural for a language in which new cells can be generated, because in the latter case it requires us to specify what happens when we read a non-existent cell, something that can never occur in reality. We modify Strachey semantics to avoid this problem, and obtain thereby a surprisingly simple possible world model for general store (not just ground store). The model is different from, and in some ways simpler than, the ground

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 232–246, 2002.
© Springer-Verlag Berlin Heidelberg 2002

store model of [3, 4]. One notable difference is that our model does not use Moggi's *monadic decomposition* of $A \to_{\mathbf{CBV}} B$ as $A \to TB$ [7], whereas the ground store model does.

For the purposes of exposition, we consider 3 levels of liberality in languages with storage.

1. Only ground values such as booleans and numbers can be stored.
2. As well as ground values, cells themselves can be stored.
3. Any value at all—including a function—can be stored. This is the case in ML and Scheme.

Languages of level 1 and 2 can also be classified according to whether they provide recursion. This division does not apply to languages of level 3, because recursion can be encoded using function storage, as noted by Landin (folklore).

The paper is organized as follows. We first present our criticism of Strachey semantics and give the basic ideas of the possible world semantics. After giving the syntax and big-semantics for the language, we present our model incrementally.

- We first model level 2 storage without recursion—here we can use sets instead of cpos.
- Then we model level 2 storage with recursion.
- Finally we model level 3 storage, i.e. the full language.

We compare with the ground store model and discuss some further directions.

1.2 From Strachey-Style to Possible World Semantics

For convenience of exposition we will consider a language with the following properties:

- it has level 2 storage and no recursion, so that we can work with sets rather than cpos.
- it distinguishes between a *value* $\Gamma \vdash^{\mathsf{v}} V : A$ and a *producer* $\Gamma \vdash^{\mathsf{p}} M : A$. The latter is an ordinary CBV term that can perform effects before producing an answer. (Moggi's monadic metalanguage would represent it as a term of type TA.) This explicit distinction at the level of judgements—which we call *fine-grain CBV*—makes it easier to describe the semantics.

We give a summary of the traditional Strachey semantics for such a language, where we write S for the set of states.

- A type A (and hence a context Γ) denotes a set, which we think of as the set of denotations of closed values of type A.
- A value $\Gamma \vdash^{\mathsf{v}} V : A$ denotes a function from $[\![\Gamma]\!]$ to $[\![A]\!]$.
- A producer $\Gamma \vdash^{\mathsf{p}} M : A$ denotes a function from $S \times [\![\Gamma]\!]$ to $S \times [\![A]\!]$.

A key question is how we are to interpret `ref` A. This is easy if the number of cells is fixed. If, for example, the language provides 3 boolean-storing cells, then `ref bool` will denote $\$3 = \{0, 1, 2\}$. Here, we use the notation $\$n$ for the set $\{0, \ldots, n-1\}$, the canonical set of size n.

But in languages such as ML and Scheme, new cells can be generated in the course of execution, and the state of the memory is given by two pieces of information:

- the *world*, which tells us how many cells there are of each type—we write $\mathcal{W}$ for the poset of worlds
- the *store*, which tells us what the cells contain—we write Sw for the set of stores in a given world w.

Thus the set S of states is given as $\sum_{w \in \mathcal{W}} Sw$. The Strachey-style semantics [2] for such a language interprets `ref` A by $\mathbb{N}$.

We claim, however, that this approach is problematic. For suppose w is a world in which there are 3 boolean-storing cells and s is a store in this world and M is the term

$$\texttt{x} : \texttt{ref bool} \ \vdash^{\mathsf{p}} \ \texttt{read x as y. produce y} \ : \texttt{bool}$$

What is $[\![M]\!](w, s)(\texttt{x} \mapsto 7)$ going to be in our semantics? It is quite arbitrary, because what $[\![M]\!](w, s)(\texttt{x} \mapsto 7)$ describes is absurd: the term M, executed in state (w, s), reads cell 7—which does not exist in world w—and returns the boolean that it finds there. This is operationally impossible precisely because the world can only grow bigger—if there were a "destroy cell" instruction, this situation could actually happen.

An obvious way to avoid this problem of non-existent cells is for $[\![M]\!]$ to take as arguments a state (w, s) and an environment that *makes sense in world* w. To set up such a semantics, the denotation of a type must depend on the world. For example, if w is a world where there are 3 boolean-storing cells, then $[\![\texttt{ref bool}]\!]w$ is $\$3$. So the above problem does not arise.

1.3 Denotation of Function Type

Recall that in the Strachey-style semantics, using $S = \sum_{w \in \mathcal{W}} Sw$, the semantics of the function type is given by

$$\begin{aligned} [\![A \to B]\!] &= S \to [\![A]\!] \to (S \times [\![B]\!]) \\ &\cong \prod_{w' \in \mathcal{W}} (Sw' \to [\![A]\!] \to \sum_{w'' \in \mathcal{W}} (Sw'' \times [\![B]\!])) \end{aligned}$$

This means that a value V of type $A \to B$ will be applied to a state (w', s') and operand U of type A, and then terminate in some state (w'', s'') with a result W of type B. But we know that if V is a w-value, then $w' \geqslant w$ and U is a w'-value,

and that $w'' \geqslant w'$ and W is a w''-value. We therefore modify the above equation as follows:

$$[\![A \to B]\!]w = \prod_{w' \geqslant w} (Sw' \to [\![A]\!]w' \to \sum_{w'' \geqslant w'} (Sw'' \times [\![B]\!]w'')) \tag{1}$$

In summary, this equation says that a w-value of type $A \to B$, when applied in a future state (w', s') to an operand (a w'-value of type A), will terminate in a state (w'', s'') even further in the future, returning a w''-value of type B.

1.4 Relating the Different Worlds

As we move from world w to the bigger world w', each w-value of type A in the environment becomes a w'-value of type A. In the syntax, the conversion from w-terms to w'-terms is just a trivial inclusion, but in the denotational semantics, we must explicitly provide a function from $[\![A]\!]w$ to $[\![A]\!]w'$, which we call $[\![A]\!]^w_{w'}$. We require

$$[\![A]\!]^w_w a = a \tag{2}$$

$$[\![A]\!]^w_{w''} a = [\![A]\!]^{w'}_{w''}([\![A]\!]^w_{w'} a) \quad \text{for } w \leqslant w' \leqslant w'' \tag{3}$$

In the terminology of category theory, A denotes a functor from the poset $\mathcal{W}$ (regarded as a category) to **Set**.

2 The Language

A *world* w is a finite multiset on types; i.e. a function from the set types of types to $\mathbb{N}$ such that the set $\mathsf{cells}\ w = \sum_{A \in \mathsf{types}} \$w_A$ is finite. We use worlds to formulate the syntax in Fig. 1. Notice that if $w \leqslant w'$ then every w-term is also a w'-term—this fact will be used implicitly in the big-step semantics.

A *syntactic w-store* π is a function associating to each cell $(A, l) \in \mathsf{cells}\ w$ a closed w-value of type A. By contrast we will use s to represent a *denotational-semantic* store—this distinction is important when we have function storage. A *syntactic state* is a pair w, π where π is a syntactic w-store. We use syntactic states to present big-step semantics in Fig. 2.

Definition 1 (observational equivalence). *Given two producers $\Gamma \vdash^{\mathsf{p}} M, N : A$, we say that $M \simeq N$ when for every ground context $\mathcal{C}[\cdot]$, i.e. context which is a producer of ground type* `bool`*, and for every syntactic state w, π and every n we have*

$$\exists w', \pi'(w, \pi, \mathcal{C}[M] \Downarrow w', \pi', n) \text{ iff } \exists w', \pi'(w, \pi, \mathcal{C}[N] \Downarrow w', \pi', n)$$

We similarly define $\simeq$ for values.

We list in Fig. 3 some basic equivalences that all the CBV models, including the Strachey semantics, validate.

Types $A ::= \texttt{bool} \mid 1 \mid A \times A \mid A \to A \mid \texttt{ref}\ A$

Rules for 1 are omitted, as it is analogous to $\times$.

Judgements $w|\Gamma \vdash^{\mathsf{v}} V : A$ $\quad w|\Gamma \vdash^{\mathsf{p}} M : A$ $\quad$ where w is a world.

In the special case $w = 0$, we write $\Gamma \vdash^{\mathsf{v}} V : A$ and $\Gamma \vdash^{\mathsf{p}} M : A$.

Terms

$$\frac{}{w|\Gamma, \mathtt{x} : A, \Gamma' \vdash^{\mathsf{v}} \mathtt{x} : A} \qquad \frac{w|\Gamma \vdash^{\mathsf{v}} V : A \quad w|\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{let}\ V\ \texttt{be}\ \mathtt{x}.\ M : B}$$

$$\frac{w|\Gamma \vdash^{\mathsf{v}} V : A}{w|\Gamma \vdash^{\mathsf{p}} \texttt{produce}\ V : A} \qquad \frac{w|\Gamma \vdash^{\mathsf{p}} M : A \quad w|\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} N : B}{w|\Gamma \vdash^{\mathsf{p}} M\ \texttt{to}\ \mathtt{x}.\ N : B}$$

$$\frac{}{w|\Gamma \vdash^{\mathsf{v}} \texttt{true} : \texttt{bool}} \qquad \frac{}{w|\Gamma \vdash^{\mathsf{v}} \texttt{false} : \texttt{bool}}$$

$$\frac{w|\Gamma \vdash^{\mathsf{v}} V : \texttt{bool} \quad w|\Gamma \vdash^{\mathsf{p}} M : B \quad w|\Gamma \vdash^{\mathsf{p}} M' : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{if}\ V\ \texttt{then}\ M\ \texttt{else}\ M' : B}$$

$$\frac{w|\Gamma \vdash^{\mathsf{v}} V : A \quad w|\Gamma \vdash^{\mathsf{v}} V' : A'}{w|\Gamma \vdash^{\mathsf{v}} (V, V') : A \times A'} \qquad \frac{w|\Gamma \vdash^{\mathsf{v}} V : A \times A' \quad w|\Gamma, \mathtt{x} : A, \mathtt{y} : A' \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{pm}\ V\ \texttt{as}\ (\mathtt{x}, \mathtt{y}).M : B}$$

$$\frac{w|\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{v}} \lambda \mathtt{x}.M : A \to B} \qquad \frac{w|\Gamma \vdash^{\mathsf{v}} V : A \quad w|\Gamma \vdash^{\mathsf{v}} W : A \to B}{w|\Gamma \vdash^{\mathsf{p}} V`W : B}$$

Terms For Divergence/Recursion

$$\frac{}{w|\Gamma \vdash^{\mathsf{p}} \texttt{diverge} : B} \qquad \frac{w|\Gamma, \mathtt{f} : A \to B, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{v}} \mu \mathtt{f} \lambda \mathtt{x}.M : A \to B}$$

Terms For Storage

$$\frac{}{w|\Gamma \vdash^{\mathsf{v}} \texttt{cell}_A l}\ (A, l) \in \mathsf{cells}\ w \qquad \frac{w|\Gamma \vdash^{\mathsf{v}} V : \texttt{ref}\ A \quad w|\Gamma \vdash^{\mathsf{v}} W : A \quad w|\Gamma \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{p}} V := W.\ M : B}$$

$$\frac{w|\Gamma \vdash^{\mathsf{v}} V : \texttt{ref}\ A \quad w|\Gamma, \mathtt{x} : A \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{read}\ V\ \texttt{as}\ \mathtt{x}.\ M : B} \qquad \frac{w|\Gamma \vdash^{\mathsf{v}} V : A \quad w|\Gamma, \mathtt{x} : \texttt{ref}\ A \vdash^{\mathsf{p}} M : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{new}\ \mathtt{x} := V.\ M : B}$$

$$\frac{w|\Gamma \vdash^{\mathsf{v}} V : \texttt{ref}\ A \quad w|\Gamma \vdash^{\mathsf{v}} V' : \texttt{ref}\ A \quad w|\Gamma \vdash^{\mathsf{p}} M : B \quad w|\Gamma \vdash^{\mathsf{p}} M' : B}{w|\Gamma \vdash^{\mathsf{p}} \texttt{if}\ V = V'\ \texttt{then}\ M\ \texttt{else}\ M' : B}$$

Here, we do not allow $V = V'$ to be a boolean value, because the operational semantics exploits the fact that values do not need to be evaluated.

Fig. 1. Terms of fine-grain CBV

The form of the big-step semantics is $w, \pi, M \Downarrow w', \pi', W$ where

- w, π is a syntactic state
- M is a closed w-producer
- w', π' is a syntactic state such that $w' \geqslant w$
- W is a closed w'-value of the same type as M.

$$\frac{w, \pi, M[V/\mathtt{x}] \Downarrow w', \pi', W}{w, \pi, \mathtt{let}\ V\ \mathtt{be}\ \mathtt{x}.\ M \Downarrow w', \pi', W}$$

$$\frac{}{w, \pi, \mathtt{produce}\ V \Downarrow w, \pi, V} \qquad \frac{w, \pi, M \Downarrow w', \pi', V \quad w', \pi', N[V/\mathtt{x}] \Downarrow w'', \pi'', W}{w, \pi, M\ \mathtt{to}\ \mathtt{x}.\ N \Downarrow w'', \pi'', W}$$

$$\frac{w, \pi, M[V/\mathtt{x}, V'/\mathtt{y}] \Downarrow w', \pi', W}{w, \pi, \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x}, \mathtt{y}).\ M \Downarrow w', \pi', W}$$

$$\frac{w, \pi, M[V/\mathtt{x}] \Downarrow w', \pi', W}{w, \pi, V`\lambda\mathtt{x}.M \Downarrow w', \pi', W}$$

$$\frac{w, \pi, \mathtt{diverge} \Downarrow w', \pi', W}{w, \pi, \mathtt{diverge} \Downarrow w', \pi', W} \qquad \frac{w, \pi, M[V/\mathtt{x}, \mu\mathtt{f}\lambda\mathtt{x}.M/\mathtt{f}] \Downarrow w', \pi', W}{w, \pi, V`\mu\mathtt{f}\lambda\mathtt{x}.M \Downarrow w', \pi', W}$$

$$\frac{w, \pi, M[V/\mathtt{x}] \Downarrow w', \pi', W}{w, \pi, \mathtt{read}\ \mathtt{cell}_A l\ \mathtt{as}\ \mathtt{x}.\ M \Downarrow w', \pi', W} \quad V \text{ is the contents of } A\text{-storing cell } l \text{ in } \pi$$

$$\frac{w, \pi', M \Downarrow w'', \pi'', W}{w, \pi, \mathtt{cell}_A l := V;\ M \Downarrow w'', \pi'', W} \quad \pi' \text{ is } \pi \text{ with } A\text{-storing cell } l \text{ assigned } V$$

$$\frac{w', \pi', M[\mathtt{cell}_A l/\mathtt{x}] \Downarrow w'', \pi'', W}{w, \pi, \mathtt{new}\ \mathtt{x} := V;\ M \Downarrow w'', \pi'', W} \quad (w', \pi') \text{ is } (w, \pi) \text{ extended with a cell } l \text{ storing } V$$

$$\frac{w, \pi, M \Downarrow w', \pi', W}{w, \pi, \mathtt{if}\ \mathtt{cell}_A l = \mathtt{cell}_A l\ \mathtt{then}\ M\ \mathtt{else}\ M' \Downarrow w', \pi', W}$$

$$\frac{w, \pi, M' \Downarrow w', \pi', W}{w, \pi, \mathtt{if}\ \mathtt{cell}_A l = \mathtt{cell}_A l'\ \mathtt{then}\ M\ \mathtt{else}\ M' \Downarrow w', \pi', W}\ (l \neq l')$$

Exploiting determinism, we say that $w\pi, M$ *diverges* when there is no w', π', V such that $w, \pi, M \Downarrow w', \pi', V$.

Fig. 2. Big-step semantics for fine-grain CBV with storage

We employ the *bound/unbound* convention: when, in an equation—such as the η-law $M = \lambda \mathtt{x}.(\mathtt{x}`M)$—the term $\Gamma \vdash^{\mathsf{c}} M : \underline{B}$ occurs both in the scope of an $\mathtt{x}$-binder and not in the scope of an $\mathtt{x}$-binder, we assume $\mathtt{x} \notin \Gamma$. We do not write the weakening explicitly.

$$
\begin{array}{lrcl}
(\beta) & \mathtt{let\ x\ be}\ V.\ M & = & M[V/\mathtt{x}] \\
(\beta) & \mathtt{if\ true\ then}\ M\ \mathtt{else}\ M' & = & M \\
(\beta) & \mathtt{if\ false\ then}\ M\ \mathtt{else}\ M' & = & M' \\
(\beta) & \mathtt{pm}\ (V, V')\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M & = & M[V/\mathtt{x}, V'/\mathtt{y}] \\
(\beta) & (\lambda \mathtt{x}.M)V & = & M[V/\mathtt{x}] \\
(\beta) & \mathtt{produce}\ V\ \mathtt{to\ x}.\ M & = & M[V/\mathtt{x}] \\
(\eta) & M[V/\mathtt{z}] & = & \mathtt{if}\ V\ \mathtt{then}\ M[\mathtt{true}/\mathtt{z}]\ \mathtt{else}\ M[\mathtt{false}/\mathtt{z}] \\
(\eta) & M[V/\mathtt{z}] & = & \mathtt{pm}\ V\ \mathtt{as}\ (\mathtt{x},\mathtt{y}).M[(\mathtt{x},\mathtt{y})/\mathtt{z}] \\
(\eta) & V & = & \lambda \mathtt{x}.(V\mathtt{x}) \\
(\eta) & M & = & M\ \mathtt{to\ x.\ produce\ x} \\
 & (P\ \mathtt{to\ x}.\ M)\ \mathtt{to\ y}.\ N & = & P\ \mathtt{to\ x}.\ (M\ \mathtt{to\ y}.\ N) \\
 & (V := W; M)\ \mathtt{to\ y}.\ N & = & V := W; (M\ \mathtt{to\ y}.\ N) \\
 & (\mathtt{read}\ V\ \mathtt{as\ x}.\ M)\ \mathtt{to\ y}.\ N & = & \mathtt{read}\ V\ \mathtt{as\ x}.\ (M\ \mathtt{to\ y}.\ N) \\
 & (\mathtt{new\ x} := V;\ M)\ \mathtt{to\ y}.\ N & = & \mathtt{new\ x} := V;\ (M\ \mathtt{to\ y}.\ N)
\end{array}
$$

Fig. 3. Basic CBV equivalences, using bound/unbound convention

3 Denotational Semantics without Divergence

In this section we exclude `diverge` and μ and storage of functions, so that we can model using sets rather than cpos. We say that a type D is a *data type* if values of type D can be stored. The types of the restricted language are given by

$$
\begin{array}{rcl}
D & ::= & \mathtt{bool} \mid D \times D \mid \mathtt{ref}\ D \\
A & ::= & D \mid \mathtt{bool} \mid A \times A \mid A \to A
\end{array}
$$

Proposition 1. *Let M be a w-producer and s a w-store in this restricted language. Then $w, \pi, M \Downarrow w', \pi', W$ for (clearly unique) w', π', W.*

This is proved by a standard Tait-style argument. We now present the denotational semantics for this restricted language. As we stated in the introduction, each type A in each world w denotes a set $[\![A]\!]w$. These sets are given by

$$
\begin{aligned}
Sw &= \prod_{(D,l) \in \mathsf{cells}\ w} [\![D]\!]w \\
[\![\mathtt{bool}]\!]w &= \{\mathsf{true}, \mathsf{false}\} \\
[\![A \times A']\!]w &= [\![A]\!]w \times [\![A']\!]w \\
[\![\mathtt{ref}\ A]\!]w &= \$w_A \\
[\![A \to B]\!]w &= \prod_{w' \geqslant w} (Sw' \to [\![A]\!]w' \to \sum_{w'' \geqslant w'} (Sw'' \times [\![B]\!]w''))
\end{aligned}
$$

The functions $[\![A]\!]^w_{w'}$ are given simply:

- $[\![\texttt{bool}]\!]^w_{w'}$ is the identity on $\{\mathsf{true}, \mathsf{false}\}$.
- $[\![A \times A']\!]^w_{w'}$ takes (a, a') to $([\![A]\!]^w_{w'} a, [\![A]\!]^w_{w'} a')$.
- $[\![\texttt{ref}\ D]\!]^w_{w'}$ is the inclusion from $\$w_D$ to $\$w'_D$.
- $[\![A \to B]\!]^w_{w'}$ takes a family $\{f_{w''}\}_{w'' \geqslant w}$ to the restricted family $\{f_{w''}\}_{w'' \geqslant w'}$.

It is easily verified that they satisfy (2)–(3). A context Γ is interpreted similarly.

A value $w_0|\Gamma \vdash^{\mathsf{v}} V : A$ will denote, for each world $w \geqslant w_0$, a function $[\![V]\!]w$ from $[\![\Gamma]\!]w$ to $[\![A]\!]w$. These functions are related: if $w_0 \leqslant w \leqslant w'$ then

$$\begin{tikzcd} {[\![\Gamma]\!]w} \arrow[r, "{[\![V]\!]w}"] \arrow[d, "{[\![\Gamma]\!]^w_{w'}}"'] & {[\![A]\!]w} \arrow[d, "{[\![A]\!]^w_{w'}}"] \\ {[\![\Gamma]\!]w'} \arrow[r, "{[\![V]\!]w'}"'] & {[\![A]\!]w'} \end{tikzcd} \qquad \text{must commute.} \tag{4}$$

Informally, (4) says that if we have an environment ρ of closed w-values, substitute into V and then regard the result as a closed w'-value, we obtain the same as if we regard ρ as an environment of closed w'-values and substitute it into V.

The special case that $w_0 = 0$, in which we have a value $\Gamma \vdash^{\mathsf{v}} V : A$, is interesting. In categorical terminology, V denotes a natural transformation from $[\![\Gamma]\!]$ to $[\![A]\!]$.

A producer $w_0|\Gamma \vdash^{\mathsf{p}} M : A$ denotes, for each $w \geqslant w_0$, a function $[\![M]\!]w$ from $Sw \times [\![\Gamma]\!]w$ to $\sum_{w' \geqslant w}(Sw' \times [\![A]\!]w')$. This is because in a given state (w, π) where $w \geqslant w_0$ and environment of w-values, it terminates in a state (w', π'), where $w' \geqslant w$, producing a w'-value. There is no required relationship between the functions $[\![M]\!]w$ for different w. The semantics of terms is straightforward.

Remark 1. According to the prescription above, the denotation of a closed value $w| \vdash^{\mathsf{v}} V : A$ is an element of

$$\{a \in \prod_{w' \geqslant w} [\![A]\!]w : [\![A]\!]^{w'}_{w''} a(w') = a(w'') \text{ when } w \leqslant w' \leqslant w''\}$$

There is an obvious bijection between this set and $[\![A]\!]w$. This shows that our thinking of $[\![A]\!]w$ as the set of denotations of closed w-values of type A, which pervades the informal parts of this paper, is in agreement with the technical development.

Remark 2. For each datatype D, the function $[\![-]\!]$ from the set of closed w-values of type D to the set $[\![D]\!]w$ is a bijection, by induction on D. Because of this, until Sect. 6, we neglect the distinction between syntactic and denotational-semantic store, and we write both as s.

Proposition 2 (soundness). *If* $w, s, M \Downarrow w', s', W$ *then* $[\![M]\!]ws = (w', s', [\![W]\!]w')$.

This is proved by straightforward induction.

Corollary 1. *(by Prop. 1) If M is a closed ground w-producer (i.e. producer of type* `bool`*) then $w, \pi, M \Downarrow w', \pi', n$ iff $[\![M]\!]ws = (w', \pi', n)$. Hence terms with the same denotation are observationally equivalent.*

4 Adding Recursion

In this section, we allow the `diverge` and recursion constructs, but we continue to prohibit function storage. We thus avoid Sw and $[\![A \to B]\!]w$ being mutually recursive.

In the denotational model, $[\![A]\!]w$ is a cpo rather than a set, although $[\![Dw]\!]$ (for a datatype D) and Sw will continue to be sets (or flat cpos). The functions $[\![A]\!]^w_{w'}$ and $[\![V]\!]w$ and $[\![M]\!]w$ are required to be continuous. In the language of category theory, a type denotes a functor from $\mathcal{W}$ to **Cpo** (the category of cpos and continuous functions), and a value $\Gamma \vdash^{\mathsf{v}} V : A$ again denotes a natural transformation.

The key semantic equation (1) must be modified for the possibility of divergence

$$[\![A \to_{\mathbf{CBV}} B]\!]w = \prod_{w' \geqslant w} (Sw' \to [\![A]\!]w' \to (\sum_{w'' \geqslant w'} (Sw'' \times [\![B]\!]w''))_\perp) \qquad (5)$$

This equation says that a w-value of type $A \to B$, when applied in a future-world store (w', π') to an operand (a w'-value of type A), will *either* diverge *or* terminate in state (w'', s'') returning a w''-value of type B.

Similarly a producer $w_0|\Gamma \vdash^{\mathsf{p}} M : A$ will now denote, in each world $w \geqslant w_0$, a continuous function $[\![M]\!]w$ from $Sw \times [\![\Gamma]\!]w$ to $(\sum_{w' \geqslant w}(Sw' \times [\![A]\!]w'))_\perp$. The lifting allows for the possibility of divergence. The interpretation of terms is straightforward.

Proposition 3 (soundness/adequacy).

1. *If $w, s, M \Downarrow w', s', W$ then $[\![M]\!]ws = (w', s', [\![W]\!]w')$.*
2. *If w, s, M diverges, then $[\![M]\!]ws = \perp$.*

Proof. (1) is straightforward. For (2), we define admissible relations $\leqslant^{\mathsf{v}}_{A\,w}$ between $[\![A]\!]w$ and closed w-values of type A, for which $a \leqslant^{\mathsf{v}}_{Aw} V$ and $w \leqslant x$ implies $([\![A]\!]^w_x)a \leqslant^{\mathsf{v}}_{Ax} V$, and $\perp$-containing admissible relations $\leqslant^{\mathsf{p}}_{A\,w}$ between $(\sum_{w' \geqslant w}(Sw' \times [\![A]\!]w'))_\perp$ and triples x, s, M (where $x \geqslant w$ and s is a x-store and M is a closed x-producer of type A). These are defined by mutual induction on types in the evident way. For data types D, we will have $d \leqslant^{\mathsf{v}}_{D,w} V$ iff $d = [\![V]\!]w$. We prove that for any producer $w|A_0, \ldots, A_{n-1} \vdash^{\mathsf{p}} M : A$, if $w \leqslant x$ and $s \in Sx$ and $a_i \leqslant^{\mathsf{v}}_{A_i x} W_i$ for $i = 0, \ldots, n-1$ then $[\![M]\!]xs\overrightarrow{a_i} \leqslant^{\mathsf{p}}_{Ax} x, s, M[\overrightarrow{W_i/\mathsf{x}_i}]$; and similarly for values. The required result is immediate.

Corollary 2. *If M is a closed ground w-producer then $w, \pi, M \Downarrow w', \pi', n$ iff $[\![M]\!]ws = \mathsf{lift}(w', \pi', n)$ and w, π, M diverges iff $[\![M]\!]ws = \perp$. Hence terms with the same denotation are observationally equivalent.*

5 Theory of Enriched-Compact Categories

We review some key results about solution of domain/predomain equations from [8, 9]. Whilst those papers work with the category $\mathbf{Cpo}^\perp$ of pointed cpos

and strict continuous functions, everything generalizes[1] to enriched-compact categories, as we now describe. All of this material is somewhat implicit in [11].

Definition 2. *An* enriched-compact category $\mathcal{C}$ *is a* **Cpo**-*enriched category with the following properties.*

- *Each hom-cpo* $\mathcal{C}(A, B)$ *has a least element* $\perp$.
- *Composition is bi-strict i.e.* $\perp; g = \perp = f; \perp$.
- $\mathcal{C}$ *has a zero object i.e. an object which is both initial and terminal. (Because of bi-strictness, just one of these properties is sufficient.)*
- *Writing* $\mathcal{C}^{\mathsf{ep}}$ *for the category of embedding-projection pairs in* $\mathcal{C}$*, we have that for every countable directed diagram* $D : \mathbb{D} \longrightarrow \mathcal{C}^{\mathsf{ep}}$ *has an* O-colimit *(necessarily unique up to unique isomorphism). We recall from [9] that an O-colimit for* D *is defined to be a cocone* $(V, \{(e_d, p_d)\}_{d \in \mathbb{D}})$ *from* D *in* $\mathcal{C}^{\mathsf{ep}}$ *satisfying*

$$\bigvee_{d \in \mathbb{D}} (p_d; e_d) = \mathsf{id}_V \tag{6}$$

Definition 3. *Let* F *be a locally continuous functor from* $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ *to* $\mathcal{C}$*. Then an* invariant *for* F *is an object* D *together with an isomorphism* $i : F(D, D) \cong D$*. It is a* minimal invariant *when the least fixed point of the continuous endofunction on* $\mathcal{C}(D, D)$ *taking* e *to* $i^{-1}; F(e, e); i$ *is the identity.*

Proposition 4. *Let* F *be a locally continuous functor from* $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ *to* $\mathcal{C}$*. Then* F *has a minimal invariant, and it is unique up to unique isomorphism.*

Proof. This is proved as in [8].

Definition 4. *1. A subcategory* $\mathcal{C}$ *of a category* $\mathcal{D}$ *is* lluf *[12] when* $\mathsf{ob}\ \mathcal{C} = \mathsf{ob}\ \mathcal{D}$.

2. If $\mathcal{B}$ *is a subcategory of* $\mathcal{D}$ *we write* $\mathcal{B} \bullet\!\!\rightarrow \mathcal{D}$ *for the category with the objects of* $\mathcal{B}$ *and the morphisms of* $\mathcal{D}$*, i.e. the unique category* $\mathcal{C}$ *such that*

$$\mathcal{B} \subset_{\mathsf{lluf}} \mathcal{C} \subset_{\mathsf{full}} \mathcal{D}$$

3. A lluf admissible subcategory $\mathcal{B}$ *of an enriched-compact category* $\mathcal{C}$ *is* embedding-complete *when it contains all the embeddings (and in particular the isomorphisms) in* $\mathcal{D}$.

Def. 4(3) is important because frequently we seek an isomorphism in a **Cpo**-enriched category $\mathcal{B}$ which is not enriched-compact (such as **Cpo**). So we look for an enriched-compact category $\mathcal{D}$ that contains $\mathcal{C}$ as an embedding-complete subcategory.

Proposition 5. *1. The category* $\mathbf{Cpo}^{\perp}$ *is enriched-compact.*

2. The category **Cpo** *is an embedding-complete subcategory of the enriched-compact category* **pCpo** *of cpos and partial continuous functions.*

[1] Another generalization is to the "rational categories" of [10], but they are for call-by-name.

3. *Any small product* $\prod_{i \in I} \mathcal{C}_i$ *of enriched-compact categories is enriched-compact. If* $\mathcal{B}_i \subset \mathcal{C}_i$ *is embedding-complete for all* $i \in I$*, then so is* $\prod_{i \in I} \mathcal{B}_i \subset \prod_{i \in I} \mathcal{C}_i$.
4. *Let* $\mathfrak{I}$ *be a small category and* $\mathcal{C}$ *be enriched-compact. Then the functor category* $[\mathfrak{I}, \mathcal{C}]$ *is enriched-compact.*
5. *Let* $\mathfrak{I}$ *be a small category. Then* $[\mathfrak{I}, \mathbf{Cpo}]$ *is an embedding-complete subcategory of the enriched-compact category* $[\mathfrak{I}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathfrak{I}, \mathbf{pCpo}]$.

Proof. (1)–(3) are standard.

(4) Given a countable directed diagram D in $[\mathfrak{I}, \mathcal{C}]$, set $(Vi, \{(e_{di}, p_{di})\}_{d \in \mathbb{D}})$ to be the O-colimit in $\mathcal{C}$ of Di and set $Vi \xrightarrow{Vf} Vj$ to be $\bigvee_{d \in \mathbb{D}} (p_{di}; D_{df}; e_{dj})$ for $i \xrightarrow{f} j$. The required properties are trivial.

(5) We construct the O-colimit of a countable directed diagram D in $[\mathfrak{I}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathfrak{I}, \mathbf{pCpo}]$ as in the previous case. We need to show that $Vi \xrightarrow{Vf} Vj$ is total for any $f : i \longrightarrow j$. Given $x \in Vw$, we know that

$$\bigvee_{d \in \mathbb{D}} (p_{dw}; e_{dw})x = x$$

Therefore, for sufficiently large d, x is in the domain of p_{dw}. Hence, for such d, x is in the domain of $p_{dw}; D_{df}; e_{dw'}$, because D_{df} and $e_{dw'}$ are total. So x is in the domain of $Vf = \bigvee_{d \in \mathbb{D}} (p_{di}; D_{df}; e_{dj})$ as required.

6 Storing Functions

We now want to model the full language. We want to provide a cpo Sw for each world w and a functor $\mathcal{W} \xrightarrow{[\![A]\!]w} \mathbf{Cpo}$ for each type A. Thus we seek an object (and isomorphism) in the category

$$\mathcal{C}_0 = \prod_{w \in \mathcal{W}} \mathbf{Cpo} \times \prod_{A \in \mathsf{types}} [\mathcal{W}, \mathbf{Cpo}]$$

By Prop. 5, this is an embedding-complete subcategory of the enriched-compact category

$$\mathcal{C} = \prod_{w \in \mathcal{W}} \mathbf{pCpo} \times \prod_{A \in \mathsf{types}} ([\mathcal{W}, \mathbf{Cpo}] \bullet\!\!\rightarrow [\mathcal{W}, \mathbf{pCpo}])$$

We define a locally continuous functor F from $\mathcal{C}^{\mathsf{op}} \times \mathcal{C}$ to $\mathcal{C}$ in Fig. 4; its minimal invariant is an object and isomorphism in $\mathcal{C}_0$—this is our semantics of types. Semantics of terms proceeds as in Sect. 4, with isomorphisms inserted where required.

Proposition 6 (soundness/adequacy).

1. *If* $w, \pi, M \Downarrow w', \pi', W$ *then* $[\![M]\!]ws = (w', \pi', [\![W]\!]w')$.

2. *If* w, π, M *diverges, then* $[\![M]\!]ws = \bot$.

Proof. (1) is straightforward induction. The proof of (2) is obtained from that of Prop. 3(2), using Pitts' techniques [8], which generalize to an arbitrary enriched-compact category.

Corollary 3. *If* M *is a closed ground* w*-producer then* $w, \pi, M \Downarrow w', \pi', n$ *iff* $[\![M]\!]ws = \mathsf{lift}(w', \pi', n)$ *and* w, π, M *diverges iff* $[\![M]\!]ws = \bot$. *Hence terms with the same denotation are observationally equivalent.*

Construction of $F : \mathcal{C}^{\mathsf{op}} \times \mathcal{C} \longrightarrow \mathcal{C}$
For objects D, E

$$
\begin{aligned}
F(D,E)_{Sw} &= \prod_{(A,l)\in \mathsf{cells}\ w} E_A \\
F(D,E)_{\mathtt{bool}w} &= \{\mathsf{true}, \mathsf{false}\} \\
F(D,E)_{\mathtt{bool}\,{}^{w}_{x}}\, b &= b \\
F(D,E)_{(A\times A')w} &= E_{Aw} \times E_{A'w} \\
F(D,E)_{A\times A'\,{}^{w}_{x}} &= (E_{A\,{}^{w}_{x}}\, c, E_{A'\,{}^{w}_{x}}\, c') \\
F(D,E)_{(\mathtt{ref}\ A)w} &= \$w_A \\
F(D,E)_{\mathtt{ref}\ A\ {}^{w}_{x}}\, i &= i \\
F(D,E)_{(A\to B)w} &= \prod_{w' \geqslant w} (D_{Sw'} \to D_{Aw'} \to (\sum_{w'' \geqslant w'} (E_{Sw''} \times E_{Bw''}))_\bot) \\
F(D,E)_{A\to B\,{}^{w}_{x}} &= \lambda x' s'.\, f x' s'
\end{aligned}
$$

For morphisms $D' \xrightarrow{h} D$ and $E \xrightarrow{k} E'$

$$
F(h,k)_{Sw}\, s = \begin{cases} ((A,l) \mapsto k_{Sw}\, s(A,l)) \\ \quad \text{if } k_{Sw}\, s(A,l) \text{ is defined for all } (A,l) \in \mathsf{cells}\ w \\ \text{undefined otherwise} \end{cases}
$$

$$
\begin{aligned}
F(h,k)_{\mathtt{bool}w}\, b &= b \\
F(h,k)_{(\mathtt{ref}\ A)w}\, i &= i
\end{aligned}
$$

$$
F(h,k)_{(A\times A')w}(c,c') = \begin{cases} (k_{Aw}\, c, k_{A'w}\, c') \\ \quad \text{if } k_{Aw}\, c \text{ and } k_{A'w}\, c' \text{ are defined} \\ \text{undefined otherwise} \end{cases}
$$

$$
F(h,k)_{(A\to B)w}\, f = \lambda w'.\lambda s'.\lambda a'. \begin{cases} \mathsf{lift}(w'', k_{Sw''}\, s'', k_{Bw''}\, b'') \\ \quad \text{if } h_{Sw'}\, s' \text{ and } h_{Aw'}\, a' \text{ are defined} \\ \quad \text{and } f w'(h_{Sw'}\, s')(h_{Aw'}\, a') = \mathsf{lift}(w'', s'', b'') \\ \quad \text{and } k_{Sw''}\, s'' \text{ and } k_{Bw''}\, a'' \text{ are defined} \\ \bot \text{ otherwise} \end{cases}
$$

Fig. 4. Construction of F

7 Monadic Decomposition and the Ground-Store Model

The set model of Sect. 3 gives us the following structure on the cartesian category $[\mathcal{W}, \mathbf{Set}]$:

$$
\begin{aligned}
(A \to_{\mathbf{CBV}} B)w &= \prod_{w' \geqslant w} (Sw' \to Aw' \to \sum_{w'' \geqslant w'} (Sw'' \times Bw'')) \\
(A \to_{\mathbf{CBV}} B)^w_x &= \lambda \mathtt{x}' s'.fx's' \\
TBw &= \prod_{w' \geqslant w} (Sw' \to \sum_{w'' \geqslant w'} (Sw'' \times Bw'')) \\
TB^w_x &= \lambda \mathtt{x}' s'.fx's'
\end{aligned}
$$

We know from Moggi's theory that, for any model of fine-grain CBV, when TB is set to be $1 \to_{\mathbf{CBV}} B$ as it is here, we can extend T to a strong monad, and then $A \to_{\mathbf{CBV}} B$ must be an exponential from A to TB. But the decomposition of $A \to_{\mathbf{CBV}} B$ as $A \to TB$ is hardly obvious here. It seems that a more natural categorical organization for our model is the "closed Freyd category" [13].

We recall the ground store model of [3], as generalized in [4], and see how it differs from ours. Let $\mathfrak{I}$ be the category of worlds and *injections*. Because we are dealing with ground store only, S is a functor from $\mathfrak{I}^{\mathsf{op}}$ to $\mathbf{Set}$: a store in a bigger world can always be restricted to a store in a smaller world.

The ground store model interprets values in the cartesian category $[\mathfrak{I}, \mathbf{Set}]$. This category has exponentials described as an end

$$(A \to B)w = \int_{w' \in (w/\mathfrak{I})} (Aw' \to Bw') \tag{7}$$

and a strong monad described using a coend

$$(TB)w = Sw \to \int^{w' \in (w/\mathfrak{I})} (Sw' \times Bw') \tag{8}$$

By monadic decomposition we obtain

$$(A \to_{\mathbf{CBV}} B)w = \int_{w' \in (w/\mathfrak{I})} (Aw' \to Sw' \to \int^{w'' \in (w'/\mathfrak{I})} (Sw'' \times Bw'')) \tag{9}$$

whose similarity to (1) is evident. Notice the importance of the contravariance of S for (8) to be covariant in w, and indeed for the coend to be meaningful. Once we can store cells, S is no longer contravariant: if $w \leqslant w'$, a w'-store s' cannot necessarily be restricted to a w-store, because some w-cell in s' might be storing a non-w-cell. Another difficulty is moving from sets to cpos, because although colimits of cpos exist [14], they are unwieldy.

An advantage of the ground store model over ours is that it validates the equivalences (employing the bound/unbound convention)

$$\mathtt{new}\ \mathtt{x} := V; M \simeq M$$

$$\mathtt{new}\ \mathtt{x} := V; \mathtt{new}\ \mathtt{y} := W; M \simeq \mathtt{new}\ \mathtt{y} := W; \mathtt{new}\ \mathtt{x} := V; M$$

We hope that our work will provide a starting-point for work on parametric models validating these and other equivalences.

8 Relationship with Call-By-Push-Value

Finally, we mention two links between our model and the *call-by-push-value* language of [15].

object language The model reflects the decomposition of $\rightarrow_{\mathbf{CBV}}$ into call-by-push-value given in [15].

metalanguage We want to use call-by-push-value as a metalanguage for the cpo equations of Sect. 6, in order to avoid having to construct the functor F in detail, and also to model storage combined with other effects [16].

We hope to treat these links in detail in future work. We also hope that working with call-by-push-value will help to establish connections with possible world models for call-by-name [17, 18, 19, 20], especially Ghica's model for pointers [21].

Acknowledgements

Thanks to Peter O'Hearn for discussion and advice.

References

[1] Ahmed, A., Appel, A., Virga, R.: A stratified semantics of general references embeddable in higher-order logic. In: Proceedings of IEEE Symposium on Logic in Computer Science, Copehagen, 2002. (2002) to appear 232

[2] Kelsey, R., Clinger, W., (Editors), J. R.: Revised^5 report on the algorithmic language Scheme. ACM SIGPLAN Notices **33** (1998) 26–76 232, 234

[3] Moggi, E.: An abstract view of programming languages. Technical Report ECS-LFCS-90-113, Dept. of Computer Science, Edinburgh Univ. (90) 232, 233, 244

[4] Plotkin, G. D., Power, A. J.: Notions of computation determine monads. In: Proceedings of Foundations of Software Science and Computation Structures, Grenoble, France (FoSSaCS '02). LNCS (2002) to appear 232, 233, 244

[5] Stark, I. D. B.: Names and Higher-Order Functions. PhD thesis, University of Cambridge (1994) 232

[6] Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1998) 232

[7] Moggi, E.: Notions of computation and monads. Information and Computation **93** (1991) 55–92 233

[8] Pitts, A. M.: Relational properties of domains. Information and Computation **127** (1996) 66–90 (A preliminary version of this work appeared as Cambridge Univ. Computer Laboratory Tech. Rept. No. 321, December 1993.) 240, 241, 243

[9] Smyth, M., Plotkin, G. D.: The category-theoretic solution of recursive domain equations. SIAM J. Computing **11** (1982) 240, 241

[10] Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF (extended abstract). In Hagiya, M., Mitchell, J. C., eds.: Theoretical Aspects of Computer Software. International Symposium TACS'94. Volume 789 of LNCS., Sendai, Japan, Springer-Verlag (1994) 1–15 241

[11] Stark, I.: A fully abstract domain model for the π-calculus. In: Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1996) 36–42 241

[12] Freyd, P. J.: Algebraically complete categories. In Carboni, A., et al., eds.: Proc. 1990 Como Category Theory Conference, Berlin, Springer-Verlag (1991) 95–104 Lecture Notes in Mathematics Vol. 1488 241

[13] Power, A. J., Thielecke, H.: Closed Freyd- and kappa-categories. In: Proc. ICALP '99. Volume 1644 of LNCS., Springer-Verlag, Berlin (1999) 625–634 244

[14] Jung, A.: Colimits in DCPO. 3-page manuscript, available by fax (1990) 244

[15] Levy, P. B.: Call-by-push-value: a subsuming paradigm (extended abstract). In Girard, J. Y., ed.: Typed Lambda-Calculi and Applications. Volume 1581 of LNCS., Springer (1999) 228–242 245

[16] Levy, P. B.: Call-by-push-value. PhD thesis, Queen Mary, University of London (2001) 245

[17] Odersky, M.: A functional theory of local names. In ACM, ed.: Proceedings of 21st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL), New York, NY, USA, ACM Press (1994) 48–59 245

[18] O'Hearn, P. W., Tennent, R. D.: Semantics of local variables. In Fourman, M. P., Johnstone, P. T., Pitts, A. M., eds.: Applications of Categories in Computer Science. Proceedings of the LMS Symposium, Durham July 1991, Cambridge University Press (1992) 217–238 245

[19] Oles, F. J.: A Category-Theoretic Approach to the Semantics of Programming Languages. Ph. D. dissertation, Syracuse University (1982) 245

[20] Reynolds, J. C.: The essence of Algol. In de Bakker, J. W., van Vliet, J. C., eds.: Algorithmic Languages, Amsterdam, North-Holland (1981) 345–372 245

[21] Ghica, D. R.: Semantics of dynamic variables in algol-like languages. Master's thesis, Queens' University, Kingston,Ontario (1997) 245

A Fully Abstract Relational Model of Syntactic Control of Interference

Guy McCusker

School of Cognitive and Computing Sciences, University of Sussex
Falmer, Brighton BN1 9QH, United Kingdom
guym@cogs.susx.ac.uk

Abstract. Using familiar constructions on the category of monoids, a fully abstract model of Basic SCI is constructed. Basic SCI is a version of Reynolds's higher-order imperative programming language Idealized Algol, restricted by means of a linear type system so that distinct identifiers are never aliases. The model given here is concretely the same as Reddy's *object spaces* model, so this work also shows that Reddy's model is fully abstract, which was not previously known.

Keywords: semantics, Algol-like languages, interference control, full abstraction, object spaces, monoids.

1 Introduction

For over 20 years there has been considerable interest among the semantics community in the study of *Algol-like languages*. Reynolds's seminal paper [11] pointed out that Algol 60 embodies an elegant and powerful combination of higher-order procedures and imperative programming, and began a strand of research which has generated a great deal of deep and innovative work. Much of this work was recently republished in a two-volume collection [7].

One theme of this research is that of *interference control*, which was also initiated by Reynolds [10]. When reasoning about higher-order programs, one often encounters the need to establish the non-interference of a pair of program phrases: if a (side-effecting) function is guaranteed not to alter variables which are used by its arguments, and vice versa, then more reasoning principles become available. Unfortunately, the common phenomenon of *aliasing* makes it difficult to detect whether two program phrases may interfere with one another: mere disjointness of the sets of variables they contain is not enough. However, Reynolds showed that if one restricts *all* procedure calls so that a procedure and its argument have no variables in common, aliasing is eliminated, and it follows that no procedure call suffers from interference with its argument. In modern terms, this restriction is the imposition of an affine type system on the λ-calculus part of Idealized Algol. The resulting programming language, which O'Hearn terms *Basic SCI*, can be extended in various ways to restore more programming power [6, 5], but is itself of interest as a minimal alias-free higher-order imperative programming language. (Other approaches to the control of

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 247–261, 2002.
© Springer-Verlag Berlin Heidelberg 2002

interference and aliasing have also been considered, including *islands* [3] and *regions* [12].)

This paper is a semantic study of the core language, Basic SCI. Using simple constructions in the category of monoids, we build a model of this language. This model turns out to be the same as an existing model due to Reddy [9], which was presented rather differently using coherence spaces. This *object spaces* model was an important precursor of the games-based models of imperative programming languages [2, 1] and the first model of a higher-order imperative language based on traces of observations rather than on state-transformers.

The first, rather minor, contribution of this paper is in showing that Reddy's model can be reconstructed so simply. We believe that our presentation is more direct and somewhat easier to work with, although it is perhaps less informative since it lacks some of the built-in structure of coherence which Reddy exploits.

The main result of this paper is that our model, and hence Reddy's, is not merely sound but *fully abstract*: it captures precisely the notion of behavioural equivalence in the language. Reddy's model was therefore the first example of a fully abstract semantics for a higher-order imperative language, though this was not known at the time; and it remains the only fully abstract model for an interference-controlled language that we are aware of. Its full abstraction is perhaps remarkable since it contains a great many undefinable elements. However, the definable elements do suffice to distinguish any two different elements of the model, and it is this which leads to full abstraction.

It is hoped that this work can be extended to encompass more powerful interference-controlled languages. The addition of *passive types*, whose elements are side-effect free and thus interfere with nothing, is a prime concern. Reddy showed how to extend his model in this direction. In doing so, full abstraction is lost, but of course, the full abstraction of the model of the core language was not known until now. The present work was in fact inspired by an ongoing attempt to model SCI using game-semantic techniques, conducted by the present author in conjunction with Wall [13], as part of a more general semantic study of interference control. We hope that this research will yield a fully abstract model of an extended language; but this remains to be seen.

2 Basic SCI

Basic SCI is the result of imposing an affine type system on Reynolds's Idealized Algol [11]. The types are given by the grammar

$$A \ ::= \texttt{comm} \mid \texttt{exp} \mid \texttt{var} \mid A \multimap A.$$

Here `comm` is the type of commands, `exp` is the type of natural-number-valued expressions, and `var` is the type of variables which may store natural numbers.

The terms of the language are as follows.

$$\begin{aligned} M ::= {} & x \mid \lambda x^A.M \mid MM \\ & \mid \ \texttt{skip} \mid M\,;M \mid \texttt{while}\ M\ \texttt{do}\ M \end{aligned}$$

$$
\begin{array}{ll}
\mid & M := M \mid !M \\
\mid & \texttt{succ}\ M \mid \texttt{pred}\ M \mid \texttt{ifzero}\ M\ M\ M \\
\mid & \texttt{new}\ x\ \texttt{in}\ M
\end{array}
$$

Here x ranges over an infinite collection of variables, and A ranges over types. We will often omit the type tag on abstractions when it will cause no confusion.

The type system is given by a collection of judgements of the form

$$x_1 : A_1, \ldots, x_n : A_n \vdash M : B$$

where the x_i are distinct variables, M is a term, and the A_i and B are types. We use Γ and Δ to range over contexts, that is, lists $x_1 : A_1, \ldots, x_n : A_n$ of variable-type pairs with all variables distinct. In the inductive definition which follows, it is assumed that all contexts are well-formed.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x^A.M : A \multimap B}$$

$$\frac{\Gamma \vdash M : A \multimap B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}$$

Note that in the last rule above, the assumption that Γ, Δ is a well-formed context implies that Γ and Δ have no variables in common.

$$\frac{}{\Gamma \vdash \texttt{skip} : \texttt{comm}} \qquad \frac{\Gamma \vdash M : \texttt{comm} \quad \Gamma \vdash N : A}{\Gamma \vdash M\,;N : A}\ A \in \{\texttt{comm}, \texttt{exp}, \texttt{var}\}$$

$$\frac{\Gamma \vdash M : \texttt{exp} \quad \Gamma \vdash N : \texttt{comm}}{\Gamma \vdash \texttt{while}\ M\ \texttt{do}\ N : \texttt{comm}}$$

$$\frac{\Gamma \vdash M : \texttt{var} \quad \Gamma \vdash N : \texttt{exp}}{\Gamma \vdash M := N : \texttt{comm}} \qquad \frac{\Gamma \vdash M : \texttt{var}}{\Gamma \vdash !M : \texttt{exp}}$$

$$\frac{\Gamma \vdash M : \texttt{exp}}{\Gamma \vdash \texttt{succ}\ M : \texttt{exp}} \qquad \frac{\Gamma \vdash M : \texttt{exp}}{\Gamma \vdash \texttt{pred}\ M : \texttt{exp}}$$

$$\frac{\Gamma \vdash M : \texttt{exp} \quad \Gamma \vdash N : A \quad \Gamma \vdash P : A}{\Gamma \vdash \texttt{ifzero}\ M\ N\ P : A}\ A \in \{\texttt{comm}, \texttt{exp}, \texttt{var}\}$$

$$\frac{\Gamma, x : \texttt{var} \vdash M : \texttt{comm}}{\Gamma \vdash \texttt{new}\ x\ \texttt{in}\ M : \texttt{comm}}$$

The operational semantics is given in terms of *stores* (also known as states). Given a context $\Gamma = x_1 : \texttt{var}, x_2 : \texttt{var}, \ldots, x_n : \texttt{var}$, a Γ-store σ is a function from the set $\{x_1, \ldots, x_n\}$ to natural numbers. We write $\sigma[x \mapsto n]$ to mean the

store which is identical to σ but maps x to n; this may be used to extend a Γ-store to a Γ, x-store, or merely to update x when x appears in Γ.

We give the operational semantics by means of a type-indexed family of relations. For each base type B, we define a relation of the form

$$\Gamma \vdash \sigma, M \Downarrow_B \sigma', V$$

where $\Gamma \vdash M : B$ and $\Gamma \vdash V : B$ are well-typed terms, Γ contains only `var`-typed variables, and σ and σ' are Γ-stores. The term V must be a *value*, that is, either `skip`, n or some $x \in \Gamma$.

For each function type $A \multimap B$, we define a relation of the form

$$\Gamma \vdash M \Downarrow_{A \multimap B} V$$

where again Γ contains only `var`-typed variables and M and V are well-typed terms of the appropriate type. Again V must be a value, that is, a term of the form $\lambda x.M'$. Note that there is no mention of store in the operational semantics of terms of higher type; this reflects the fact that terms of higher type do not affect and are not affected by the contents of the store until they are applied to arguments.

These relations are defined inductively. We just give a selection of the rules.

$$\frac{}{\Gamma \vdash \sigma, \mathtt{skip} \Downarrow_{\mathsf{comm}} \sigma, \mathtt{skip}}$$

$$\frac{\Gamma \vdash \sigma, M \Downarrow_{\mathsf{comm}} \sigma', \mathtt{skip} \quad \Gamma \vdash \sigma', N \Downarrow_B \sigma'', N'}{\Gamma \vdash \sigma, M\,;N \Downarrow_B \sigma'', N'}$$

$$\frac{\Gamma \vdash \sigma, N \Downarrow_{\mathsf{exp}} \sigma', n \quad \Gamma \vdash \sigma', M \Downarrow_{\mathsf{var}} \sigma'', x}{\Gamma \vdash \sigma, M := N \Downarrow_{\mathsf{exp}} \sigma''[x \mapsto n], \mathtt{skip}}$$

$$\frac{\Gamma \vdash \sigma, M \Downarrow_{\mathsf{var}} \sigma', x}{\Gamma \vdash \sigma, !M \Downarrow_{\mathsf{exp}} \sigma', n}\ \sigma'(x) = n$$

$$\frac{\Gamma \vdash \sigma, M \Downarrow_{\mathsf{exp}} \sigma', 0 \quad \Gamma \vdash \sigma', N \Downarrow_{\mathsf{comm}} \sigma'', \mathtt{skip} \quad \Gamma \vdash \sigma'', \mathtt{while}\ M\ \mathtt{do}\ N \Downarrow_{\mathsf{comm}} \sigma''', \mathtt{skip}}{\Gamma \vdash \sigma, \mathtt{while}\ M\ \mathtt{do}\ N \Downarrow_{\mathsf{comm}} \sigma''', \mathtt{skip}}$$

$$\frac{\Gamma \vdash \sigma, M \Downarrow_{\mathsf{exp}} \sigma', n+1}{\Gamma \vdash \sigma, \mathtt{while}\ M\ \mathtt{do}\ N \Downarrow_{\mathsf{comm}} \sigma', \mathtt{skip}}$$

$$\frac{\Gamma, x : \mathtt{var} \vdash \sigma[x \mapsto 0], M \Downarrow_{\mathsf{comm}} \sigma'[x \mapsto n], \mathtt{skip}}{\Gamma \vdash \sigma, \mathtt{new}\ x\ \mathtt{in}\ M \Downarrow_{\mathsf{comm}} \sigma', \mathtt{skip}}$$

$$\frac{}{\Gamma \vdash \lambda x.M \Downarrow_{A \multimap B} \lambda x.M}$$

$$\frac{\Gamma \vdash M \Downarrow_{A \multimap B} \lambda x.M' \quad \Gamma \vdash M'[N/x] \Downarrow_B V}{\Gamma \vdash MN \Downarrow_B V}\ B \text{ a function type}$$

$$\frac{\Gamma \vdash M \Downarrow_{A \multimap B} \lambda x.M' \quad \Gamma \vdash \sigma, M'[N/x] \Downarrow_B \sigma', V}{\Gamma \vdash \sigma, MN \Downarrow_B \sigma', V}\ B \text{ a base type}$$

Contextual equivalence We can define the notion of contextual equivalence in the usual way: given terms $\Gamma \vdash M, N : A$, we say that M and N are *contextually (or observationally) equivalent*, $M \cong N$, iff for all term-contexts $C[-]$ such that $\vdash C[M] : \texttt{comm}$ and $\vdash C[N] : \texttt{comm}$, $C[M] \Downarrow \texttt{skip} \iff C[N] \Downarrow \texttt{skip}$. (We omit mention of the unique store over no variables). As usual a term-context is a term with one or more occurrences of a "hole" written $-$, and $C[M]$ is the term resulting from replacing each occurrence of $-$ by M. We will often abbreviate the assertion $C[M] \Downarrow \texttt{skip}$ simply to $C[M]\Downarrow$.

3 A Categorical Model

In this section we define and explore the structure of the category which our model of Basic SCI will inhabit.

To build our model, we will be making use of the category **Mon** of monoids and homomorphisms, and exploiting the product, coproduct and powerset operations on monoids, and the notion of the free monoid over a set. For the sake of completeness, we review these constructions here.

First some notation. For a monoid A, we use e_A to denote the identity element, and write monoid multiplication as concatenation, or occasionally using the symbol $\cdot_A$. The underlying set of the monoid A is written as UA.

Free monoids Recall that for any set A, the *free monoid over* A is given by A^*, the monoid of strings over A, also known as the Kleene monoid over A.

The operation taking A to A^* is left-adjoint to the forgetful functor U : **Mon** $\to$ **Set**.

Products The category **Mon** has finite products. The product of monoids A and B is a monoid with underlying set $UA \times UB$, the Cartesian product of sets. The monoid operation is defined by

$$\langle a, b\rangle\langle a', b'\rangle = \langle a \cdot_A a', b \cdot_B b'\rangle.$$

The identity element is $\langle e_A, e_B\rangle$. Projection and pairing maps in **Mon** are given by the corresponding maps on the underlying sets. The terminal object is the one-element monoid.

Coproducts The category **Mon** also has finite coproducts. These are slightly awkward to define in general, and since we will not be making use of the general construction, we omit it here.

The special case of the coproduct of two free monoids is easy to define. Since the operation of building a free monoid from a set is left adjoint to the forgetful functor U, it preserves colimits and in particular coproducts. For sets A and B, the coproduct monoid $A^* + B^*$ is therefore given by $(A + B)^*$, the monoid of strings over the disjoint union of A and B.

The initial object is the one-element monoid.

Powerset The familiar powerset construction on **Set** lifts to **Mon** and retains much of its structure. Given a monoid A, define the monoid $\wp A$ as follows. Its underlying set is the powerset of UA, that is, the set of subsets of UA. Monoid multiplication is defined by

$$ST = \{x \cdot_A y \mid x \in S, y \in T\}$$

and the identity is the singleton set $\{e_A\}$.

We will exploit the fact that powerset is a *commutative monad* on **Mon**. In particular, we will make use of the Kleisli category $\mathbf{Mon}_\wp$. This category can be defined concretely as follows. Its objects are monoids, and a map from A to B is a monoid homomorphism from A to $\wp B$. The identity on A is the singleton map which takes each $a \in A$ to $\{a\}$. Morphisms are composed as follows: given maps $f : A \to B$ and $g : B \to C$, the composite $f\,;g : A \to C$ is defined by

$$(f;g)(a) = \{c \mid \exists b \in f(a).c \in g(b)\}.$$

The fact that the powerset monad is commutative means that the product structure on **Mon** lifts to a monoidal structure on $\mathbf{Mon}_\wp$ as follows. We define $A \otimes B$ to be the monoid $A \times B$. For the functorial action, we make use of the *double strength* map

$$\theta_{A,B} : \wp A \times \wp B \longrightarrow \wp(A \times B)$$

defined by

$$\theta_{A,B}(S,T) = \{\langle x, y\rangle \mid x \in S, y \in T\}.$$

This is a homomorphism of monoids. With this in place, given maps $f : A \to B$ and $g : C \to D$ in $\mathbf{Mon}_\wp$, we can define $f \otimes g : A \otimes C \to B \otimes D$ as the homomorphism $f \times g\,;\theta_{B,D}$. See for example [4] for more details on this construction.

The category we will use to model Basic SCI is $(\mathbf{Mon}_\wp)^{\mathrm{op}}$. This category can be seen as a category of "monoids and relations" of a certain kind, so we will call it **MonRel**. We will now briefly explore some of the structure that **MonRel** possesses.

Monoidal structure The monoidal structure on $\mathbf{Mon}_\wp$ described above is directly inherited by **MonRel**. Furthermore, since the unit I of the monoidal structure is given by the one-element monoid, which is also an initial object in **Mon**, I is in fact a terminal object in **MonRel**, so the category has an *affine* structure.

Exponentials Let A and B be any monoids, and C^* be the free monoid over some set C. Consider the following sequence of natural isomorphisms and definitional equalities.

$$\begin{aligned}
&\mathbf{MonRel}(A \otimes B, C^*)\\
&= \mathbf{Mon}(C^*, \wp(A \times B))\\
&\cong \mathbf{Set}(C, U\wp(A \times B))\\
&\cong \mathbf{Rel}(C, UA \times UB)\\
&\cong \mathbf{Rel}(UB \times C, UA))
\end{aligned}$$

Similarly we can show that

$$\mathbf{Rel}(UB \times C, UA)) \cong \mathbf{MonRel}(A, (UB \times C)^*).$$

The exponential $B \multimap C^*$ is therefore given by $(UB \times C)^*$. It is important to note that the free monoids are closed under this operation, so that we can form $A_1 \multimap (A_2 \multimap \ldots (A_n \multimap C^*))$ for any $A_1, \ldots, A_n$. That is to say, the free monoids form an *exponential ideal* in **MonRel**.

Products The coproduct in **Mon** is inherited by the Kleisli-category $\mathbf{Mon}_{\wp}$, and since **MonRel** is the opposite of this category, **MonRel** has finite products.

An alternative characterization We can also describe the category **MonRel** concretely, as follows. Objects are monoids, and maps $A \to B$ are relations R between the (underlying sets of) A and B, with the following properties:

homomorphism $e_A R e_B$, and if $a_1 R b_1$ and $a_2 R b_2$, then $a_1 a_2 R b_1 b_2$
identity reflection if $a R e_B$ then $a = e_A$
decomposition if $a R b_1 b_2$ then there exist a_1 and $a_2 \in A$ such that $a_i R b_i$ for $i = 1, 2$ and $a = a_1 a_2$.

Identities and composition are as usual for relations. Note that the property of "identity reflection" is merely the nullary case of the property of "decomposition".

It is routine to show that this definition yields a category isomorphic to $(\mathbf{Mon}_{\wp})^{\mathrm{op}}$. The action of the isomorphism is as follows. Given a map $A \to B$ in $(\mathbf{Mon}_{\wp})^{\mathrm{op}}$, that is to say, a homomorphism

$$f : B \longrightarrow \wp(A)$$

we can define a relation R_f between A and B as the set of pairs $\{(a, b) \mid a \in f(b)\}$.

The intuition behind our use of this category is that the objects we employ are monoids of "observations" that one makes of a program phrase. The monoid operation builds compound observations from simple ones. A map in the category tells us what input observations are required in order to produce a given output observation. The decomposition axiom above has something of a flavour of linearity or stability about it: it says that the only way to produce a compound observation is to produce the elements of that observation. This is made more explicit in Reddy's presentation which is based on coherence spaces.

4 Modelling Basic SCI

The categorical structure of **MonRel** developed above gives us enough to model the affine λ-calculus over base types which are interpreted as free monoids. We now flesh this out to complete the interpretation of Basic SCI by giving appropriate objects to interpret the base types and maps to interpret the constants of the language.

The idea behind our model of Basic SCI is that a program denotes a set of sequences of observable actions. Thus the types of **MonRel** will be interpreted as objects of the form A^* for a set A, that is, as free monoids.

$$\begin{aligned}[\![\mathtt{comm}]\!] &= 1^* \\ [\![\mathtt{exp}]\!] &= \mathbb{N}^* \\ [\![\mathtt{var}]\!] &= (\mathbb{N}+\mathbb{N})^*\end{aligned}$$

Here 1 denotes the one-element set, whose single element we will denote by $*$, $\mathbb{N}$ is the set of natural numbers, and $+$ denotes disjoint union of sets. The two copies of $\mathbb{N}$ used to interpret `var` correspond to the actions of reading a value from a variable and writing a value to a variable, so we will denote the elements of $\mathbb{N}+\mathbb{N}$ as $\mathsf{read}(n)$ and $\mathsf{write}(n)$.

The only observation we can make of a command is that the command terminates, so `comm` is interpreted using sequences over a one-element set. The basic observation one can make of a term of type `exp` is its value, so expressions denote sequences of natural numbers. For variables, one can observe the value stored in a variable, so there is an observation $\mathsf{read}(n)$ for each natural number n, and one can also observe that assigning the number n to a variable terminates, hence the actions $\mathsf{write}(n)$.

The interpretation of types of Basic SCI as objects of **MonRel** is completed by setting $[\![A \multimap B]\!] = [\![A]\!] \multimap [\![B]\!]$. The fact that objects of the form A^* form an exponential ideal in **MonRel** guarantees that the required exponentials exist. Unpacking the definition of exponential, we see that a basic observation that can be made of the function type $A \multimap B$ consists of a pair (s, b) where b is an observation from B and s is a sequence of observations from A. We will use the list-notation $[a_1, \ldots, a_n]$ to display such sequences.

Note that the semantics of a term of type $A \multimap B$ does not record the relative order of actions in A and B. For example, the interpretation of the type $\mathtt{comm} \multimap \mathtt{comm} \multimap \mathtt{comm}$ contains elements such as $([*,*],([*],*))$, which will belong to the denotation of the term

$$\lambda x^{\mathtt{comm}}.\lambda y^{\mathtt{comm}}.x; x; y$$

but also of

$$\lambda x^{\mathtt{comm}}.\lambda y^{\mathtt{comm}}.x; y; x.$$

This is only correct thanks to the non-interference property of the language: because x and y cannot interfere, the relative order of their execution is irrelevant.

A term $x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n \vdash M : B$ will be interpreted as a map

$$[\![x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n \vdash M : B]\!] : [\![A_1]\!] \otimes [\![A_2]\!] \otimes \cdots \otimes [\![A_n]\!] \to [\![B]\!].$$

Using the first definition of **MonRel**, such a map can be seen as a function taking an observation b from B as argument, and returning a set of tuples $(s_1, s_2, \ldots, s_n)$ where each s_i is a sequence of observations from A_i. This can be thought of as stipulating the actions that the environment must be prepared to perform in

order for the term to produce the action b. Note that the monoid $[\![B]\!]$ is always the free monoid over some set, so this map is uniquely determined by its action on singleton observations.

In order to define such maps concretely, we will write them as sets of tuples of the form $(s_1, \ldots, s_n, b)$ where b is a singleton observation. This notation accords with the concrete presentation of exponentials above.

We now define maps in **MonRel** to interpret the constants of Basic SCI. Recall that the product in **MonRel** of two free monoids A^* and B^* is given by $(A + B)^*$, where $+$ denotes disjoint union. We will use the notation fst and snd to tag the two components of this disjoint union; when a ternary product is needed, we use thd for the third tag.

$$\begin{aligned}
\mathsf{skip} : I \to [\![\mathtt{comm}]\!] &= \{(e_I, *)\} \\
\mathsf{seq}_A : [\![\mathtt{comm}]\!] \times A^* \to A^* &= \{([\mathsf{fst}(*), \mathsf{snd}(a)], a) \mid a \in A\} \\
\mathsf{read} : [\![\mathtt{var}]\!] \to [\![\mathbb{N}]\!] &= \{([\mathsf{read}(n)], n) \mid n \in \mathbb{N}\} \\
\mathsf{write} : [\![\mathtt{var}]\!] \times [\![\mathtt{exp}]\!] \to [\![\mathtt{comm}]\!] &= \{([\mathsf{snd}(n), \mathsf{fst}(\mathsf{write}(n))], *) \mid n \in \mathbb{N}\} \\
\mathsf{ifz} : [\![\mathtt{exp}]\!] \times A^* \times A^* \to A^* &= \{([\mathsf{fst}(0), \mathsf{snd}(a)], a) \mid a \in A\} \\
&\quad \cup \{([\mathsf{fst}(n), \mathsf{thd}(a)], a) \mid n \neq 0, a \in A\} \\
\mathsf{while} : [\![\mathtt{exp}]\!] \times [\![\mathtt{comm}]\!] \to [\![\mathtt{comm}]\!] &= \{([\mathsf{fst}(0), \mathsf{snd}(*), \mathsf{fst}(0), \mathsf{snd}(*), \ldots, \\
&\qquad \ldots, \mathsf{fst}(0), \mathsf{snd}(*), \mathsf{fst}(n)], *) \mid n \neq 0\}
\end{aligned}$$

Similar maps can be defined for the interpretation of the arithmetic constants.

The interpretation of the constructs of the basic imperative language can now be defined in the standard way. For example,

$$[\![M; N]\!] = \langle [\![M]\!], [\![N]\!] \rangle ; \mathsf{seq}$$

For the λ-calculus part of the language, we exploit the affine monoidal structure and the exponentials of the language. Again, these definitions are standard.

For variables:

$$[\![\Gamma, x : A \vdash x : A]\!] = \mathsf{proj} : [\![\Gamma]\!] \otimes [\![A]\!] \to [\![A]\!].$$

This projection map can be defined concretely as the set

$$\{(e_\Gamma, a, a) \mid a \in A\}$$

where e_Γ is the identity of the monoid $[\![\Gamma]\!]$.

For abstraction:

$$[\![\Gamma \vdash \lambda x^A.M : A \multimap B]\!] = \Lambda[\![M]\!] : [\![\Gamma]\!] \to ([\![A]\!] \multimap [\![B]\!])$$

where Λ denotes the natural isomorphism coming from the exponential structure.

For application:

$$[\![\Gamma, \Delta \vdash MN : B]\!] = ([\![M]\!] \otimes [\![N]\!]) ; \mathsf{ev} : [\![\Gamma]\!] \otimes [\![\Delta]\!] \to [\![B]\!]$$

where $\mathsf{ev} : (A \multimap B) \otimes B \to B$ is the counit of the exponential adjunction.

Finally, we give the semantics of the variable-allocation construct. First note that an element $s \in [\![\mathtt{var}]\!]$ consists of a sequence of $\mathsf{read}(-)$ and $\mathsf{write}(-)$ actions. We say that s *is a cell-trace* if the values carried by $\mathsf{read}(-)$ actions correspond to the values previously carried by $\mathsf{write}(-)$ actions in the obvious way: s is a cell-trace iff

- whenever $s = [\ldots, \mathsf{read}(n), \mathsf{read}(m), \ldots]$, $n = m$
- whenever $s = [\ldots, \mathsf{write}(n), \mathsf{read}(m), \ldots]$, $n = m$.

We can now define a map

$$\mathsf{new} : ([\![\mathtt{var}]\!] \multimap [\![\mathtt{comm}]\!]) \to [\![\mathtt{comm}]\!] = \{((s, *), *) \mid \mathsf{write}(0) \cdot s \text{ is a cell-trace}\}$$

and then

$$[\![\Gamma \vdash \mathtt{new}\ x\ \mathtt{in}\ M : \mathtt{comm}]\!] = [\![\lambda x.M]\!]\,;\mathsf{new}.$$

Ignoring all the structure in this semantics and considering the interpretation of a term as a set of tuples, our semantics is identical to that obtained by forgetting all structure in Reddy's semantics. We therefore have:

Lemma 1. *The semantics given above agrees with the object-space semantics of Reddy [9]: writing $[\![-]\!]_r$ for the Reddy semantics, we have that for any terms M and N of Basic SCI,*

$$[\![M]\!]_r = [\![N]\!]_r \iff [\![M]\!] = [\![N]\!].$$

We show the soundness of our semantics by means of a standard sequence of lemmas.

Lemma 2. *For any closed term M of type* $\mathtt{comm}$*, if $M\Downarrow$ then $[\![M]\!] = [\![\mathtt{skip}]\!]$.*

Proof. A straightforward but lengthy induction over the structure of derivations in the operational semantics. Very similar proofs can be found in Reddy's work [9] and in the work on game semantics of Algol-like languages [2].

Lemma 3. *For any closed term $M : \mathtt{comm}$, if $[\![M]\!] = [\![\mathtt{skip}]\!]$ then $M\Downarrow$.*

Proof. A Tait-Girard-Plotkin style computability argument is employed [8]. Similar arguments can be found in the works by Reddy and the game-semantics literature cited above.

Theorem 1 (Equational Soundness). *If $\Gamma \vdash M, N : A$ are terms such that $[\![M]\!] = [\![N]\!]$, then M and N are contextually equivalent.*

Proof. Since the semantics is compositional, for any context $C[-]$, we have $[\![C[M]]\!] = [\![C[N]]\!]$. By Lemmas 2 and 3, $C[M]\Downarrow$ iff $[\![C[M]]\!] = [\![\mathtt{skip}]\!]$ iff $[\![C[N]]\!] = [\![\mathtt{skip}]\!]$ iff $C[N]\Downarrow$ as required.

5 Full Abstraction

In this section we show the converse of our Equational Soundness theorem: Equational Completeness, which states that if two terms are contextually equivalent, then they have the same denotational semantics. In order to do so, we must study the definable elements of our model more closely, and eventually prove a partial definability result. Our proof will involve some programming in Basic SCI, and we will make use of some syntactic sugar to write down programs which we will not explicitly define. It is hoped that this causes no difficulties for the reader.

Let us first mention an interesting fact. If $C[-]$ is some context such that

$$C[\texttt{if}\ !x = 3\ \texttt{then skip else diverge}]\Downarrow,$$

then it is also the case that $C[x := 3]\Downarrow$. This inability of contexts to distinguish completely between reading and writing into variables is the main obstacle to overcome in our definability proof.

The following definition captures the relationship between sequences of observations which is at work in the above example.

Definition 1. *For any SCI type A, we define the* positive and negative read-write orders $\preceq^+$ *and* $\preceq^-$ *between elements of* $[\![A]\!]$ *as follows. We give only the definitions for singleton elements; the definitions are extended to sequences by requiring that the elements of the sequences are related pointwise.*

- *At type* `comm`*:*

$$* \preceq^+ * \wedge * \preceq^- *$$

- *At type* `exp`*:*

$$n \preceq^+ m \iff n = m \iff n \preceq^- m$$

- *At type* `var`*:*

$$\begin{aligned} a \preceq^+ a' &\iff (a = a') \vee \exists n.a = \mathsf{read}(n) \wedge a' = \mathsf{write}(n) \\ a \preceq^- a' &\iff a = a'. \end{aligned}$$

- *At type* $A \multimap B$*:*

$$\begin{aligned} (s,b) \preceq^+ (s',b') &\iff s \preceq^- s' \wedge b \preceq^+ b' \\ (s,b) \preceq^- (s',b') &\iff s \preceq^+ s' \wedge b \preceq^- b' \end{aligned}$$

In general, $s \preceq^+ t$ iff t can be obtained from s by replacing some occurrences of $\mathsf{read}(n)$ *actions in positive occurrences of the type* `var` *by the corresponding* $\mathsf{write}(n)$ *actions. The order $\preceq^-$ is the same but operates on negatively occurring actions.*

We also need a notion of state transition. Given an element $s \in [\![\texttt{var}]\!]$, we define the transitions $n \xrightarrow{s} n'$ where n and n' are natural numbers, as follows.

$$n \xrightarrow{[]} n \qquad n \xrightarrow{[\mathsf{read}(n)]} n \qquad n \xrightarrow{[\mathsf{write}(n')]} n'$$

$$\frac{n \xrightarrow{s} n' \quad n' \xrightarrow{s'} n''}{n \xrightarrow{ss'} n''}$$

We extend this to traces involving more than one `var` type as follows. Given a context $x_1 : \mathtt{var}, \ldots, x_n : \mathtt{var}$, an element $s = (s_1, \ldots, s_n) \in [\![\mathtt{var}]\!] \otimes \cdots \otimes [\![\mathtt{var}]\!]$, and states σ and σ' in variables $x_1, \ldots, x_n$, we write $\sigma \xrightarrow{s} \sigma'$ iff $\sigma(x_i) \xrightarrow{s_i} \sigma'(x_i)$ for each i.

We are now in a position to state our definability result.

Lemma 4. *Let A be any type of Basic SCI and let $a \in [\![A]\!]$ be any element of the monoid interpreting A. There exists a term*

$$x : A \vdash \mathsf{test}(a) : \mathtt{comm}$$

such that $(s, *) \in [\![\mathsf{test}(a)]\!]$ iff $a \preceq^- s$. There also exists a context $\Gamma = x_1 : \mathtt{var}, \ldots, x_n : \mathtt{var}$, Γ-stores $\mathsf{init}(a)$ and $\mathsf{final}(a)$, and a term

$$\Gamma \vdash \mathsf{produce}(a) : A$$

such that there exists $(s, a') \in [\![\mathsf{produce}(a)]\!]$ with $\mathsf{init}(a) \xrightarrow{s} \mathsf{final}(a)$ if and only if $a \preceq^+ a'$.

Proof. We will prove the two parts of this lemma simultaneously by induction on the type A. First note that any $a \in [\![A]\!]$ is a sequence of elements from a certain alphabet. Before beginning the main induction, we show that it suffices to consider the case when a is a singleton sequence. The cases when a is empty are trivial: $\mathsf{test}([]) = \mathtt{skip}$ and $\mathsf{produce}([])$ is any divergent term. If $a = [a_1, a_2, \ldots, a_n]$, then we can define $\mathsf{test}(a)$ as

$$\mathsf{test}([a_1])\,;\mathsf{test}([a_2])\,;\ldots;\mathsf{test}([a_n]).$$

For the $\mathsf{produce}$ part, suppose that $A = A_1 \multimap A_2 \multimap A_k \multimap B$ for some base type B, and that the context Γ contains all the variables needed to define the $\mathsf{produce}(a_i)$. For any store σ over variables $x_1, \ldots, x_n$, define $\mathsf{check}(\sigma)$ to be the term

```
if (!x_1 ≠ σ(x_1)) then diverge
else if (!x_2 ≠ σ(x_2)) then diverge
...
else if (!x_n ≠ σ(x_n)) then diverge
else skip
```

Define $\mathsf{set}(\sigma)$ to be $x_1 := \sigma(x_1)\,;\cdots;x_n := \sigma(x_n)$.

An appropriate term $\mathsf{produce}(a)$ can then be defined as follows.

$$
\begin{array}{l}
\Gamma, x : \mathtt{var} \vdash \lambda \boldsymbol{y_i}^{\boldsymbol{A_i}}.\, x := !x + 1\,; \\
\qquad\qquad \mathtt{if}\ (!x = 1)\ \mathtt{then}\ \mathsf{produce}(a_1) y_1 \ldots y_n \\
\qquad\qquad \mathtt{else\ if}\ (!x = 2)\ \mathtt{then}\ \mathsf{check}(\mathsf{final}(a_1))\,; \\
\qquad\qquad\qquad\qquad\qquad \mathsf{set}(\mathsf{init}(a_2))\,; \\
\qquad\qquad\qquad\qquad\qquad \mathsf{produce}(a_2) y_1 \ldots y_n \\
\qquad\qquad \ldots \\
\qquad\qquad \mathtt{else\ if}\ (!x = n)\ \mathtt{then}\ \mathsf{check}(\mathsf{final}(a_{n-1}))\,; \\
\qquad\qquad\qquad\qquad\qquad \mathsf{set}(\mathsf{init}(a_n))\,; \\
\qquad\qquad\qquad\qquad\qquad \mathsf{produce}(a_n) y_1 \ldots y_n \\
\qquad\qquad \mathtt{else\ diverge}
\end{array}
$$

The required initial state $\mathsf{init}(a)$ is $\mathsf{init}(a_1)[x \mapsto 0]$, and the final state $\mathsf{final}(a)$ is $\mathsf{final}(a_n)[x \mapsto n]$.

We now define $\mathsf{test}(a)$ and $\mathsf{produce}(a)$ for the case when a is a singleton, by induction on the structure of the type A.

For the type `comm`, we define

$$
\begin{aligned}
\mathsf{test}(*) &= x : \mathtt{comm} \vdash x : \mathtt{comm} \\
\mathsf{produce}(*) &= y : \mathtt{var} \vdash y := !y + 1 : \mathtt{comm} \\
\mathsf{init}(*) &= (y \mapsto 0) \\
\mathsf{final}(*) &= (y \mapsto 1)
\end{aligned}
$$

Note the way the initial and final states check that the command $\mathsf{produce}(*)$ is used exactly once.

The type `exp` is handled similarly:

$$
\begin{aligned}
\mathsf{test}(n) &= x : \mathtt{exp} \vdash \mathtt{if}\ (x = n)\ \mathtt{then\ skip\ else\ diverge} : \mathtt{comm} \\
\mathsf{produce}(n) &= y : \mathtt{var} \vdash y := !y + 1; n : \mathtt{exp} \\
\mathsf{init}(n) &= (y \mapsto 0) \\
\mathsf{final}(n) &= (y \mapsto 1)
\end{aligned}
$$

For `var`, there are two kinds of action to consider: those for reading and those for writing. For writing we define:

$$
\begin{aligned}
\mathsf{test}(\mathsf{write}(n)) &= x : \mathtt{var} \vdash x := n : \mathtt{comm} \\
\mathsf{produce}(\mathsf{write}(n)) &= x : \mathtt{var}, y : \mathtt{var} \vdash y := !y + 1; x : \mathtt{var} \\
\mathsf{init}(\mathsf{write}(n)) &= (x \mapsto n + 1, y \mapsto 0) \\
\mathsf{final}(\mathsf{write}(n)) &= (x \mapsto n, y \mapsto 1)
\end{aligned}
$$

For $\mathsf{produce}(\mathsf{write}(n))$, the variable y checks that exactly one use is made, and the variable x checks that the one use is a write-action assigning n to the variable.

Reading is handled similarly:

$$
\mathsf{test}(\mathsf{read}(n)) = x : \mathtt{var} \vdash \mathtt{if}\ (!x = n)\ \mathtt{then\ skip\ else\ diverge} : \mathtt{comm}
$$

$$\begin{aligned}\mathsf{produce}(\mathsf{read}(n)) &= x : \mathtt{var}, y : \mathtt{var} \vdash y := !y + 1; x : \mathtt{var}\\ \mathsf{init}(\mathsf{read}(n)) &= (x \mapsto n, y \mapsto 0)\\ \mathsf{final}(\mathsf{read}(n)) &= (x \mapsto n, y \mapsto 1)\end{aligned}$$

In $\mathsf{init}(\mathsf{read}(n))$, the variable x holds n so that if the expression $\mathsf{produce}(\mathsf{read}(n))$ is used for a read, the value n is returned. The variable x must also hold n finally, so $\mathsf{produce}(\mathsf{read}(n))$ cannot reach the state $\mathsf{final}(\mathsf{read}(n))$ if it is used to write a value other than n. However, it would admit a single $\mathsf{write}(n)$ action. This is the reason for introducing the $\preceq$ relation: if a term of our language can engage in a $\mathsf{read}(n)$ action, then it can also engage in $\mathsf{write}(n)$.

For a function type $A \multimap B$, the action we are dealing with has the form (s, b) where s is a sequence of actions from A and b is an action from B. We can now define

$$\begin{aligned}\mathsf{test}(s,b) &= x : A \multimap B \vdash \mathtt{new}\ x_1, \ldots, x_n\ \mathtt{in}\\ &\qquad\qquad \mathsf{set}(\mathsf{init}(s));\\ &\qquad\qquad (\lambda x^B.\mathsf{test}(b))(x\mathsf{produce}(s));\\ &\qquad\qquad \mathsf{check}(\mathsf{final}(s));\\ \mathsf{produce}(s,b) &= \lambda x^A.\mathsf{test}(s); \mathsf{produce}(b)\\ \mathsf{init}(s,b) &= \mathsf{init}(b)\\ \mathsf{final}(s,b) &= \mathsf{final}(b)\end{aligned}$$

where $x_1, \ldots, x_n$ are the variables used in $\mathsf{produce}(s)$.

The non-interference between function and argument allows us to define these terms very simply: for $\mathsf{test}(s, b)$ we supply the function x with an argument which will produce the sequence s, and check that the output from x is b. We must also check that the function x uses its argument in the appropriate, s-producing way, which is done by means of the $\mathsf{init}(s)$ and $\mathsf{final}(s)$ states. For $\mathsf{produce}(s, b)$ we simply test that the argument x is capable of producing s, and then produce b.

It is straightforward to check that these terms have the required properties.

The following lemma holds because the language we are considering is deterministic.

Lemma 5. *If M is any term of Basic SCI and $(s, t), (s', t') \in [\![M]\!]$ are such that $(s, t) \preceq^- (s', t')$ then $(s, t) = (s', t')$.*

Theorem 2 (Equational Completeness). *If $\Gamma \vdash M : A$ and $\Gamma \vdash N : A$ are terms of basic SCI such that $M \cong N$ then $[\![M]\!] = [\![N]\!]$.*

Proof. Consider any $(s, a) \in [\![M]\!]$. Supposing that $\Gamma = y_1 : A_1, \ldots, y_n : A_n$, we know that $(e_I, (s, a)) \in [\![\lambda \boldsymbol{y}.M]\!]$. The term $\mathsf{test}(s, a)$ from the definability lemma (Lemma 4) therefore has the property that $[\![(\lambda x.\mathsf{test}(s, a))(\lambda \boldsymbol{y}.M)]\!] = [\![\mathtt{skip}]\!]$ so we know that $(\lambda x.\mathsf{test}(s, a))(\lambda \boldsymbol{y}.M)\Downarrow$ by soundness. Since $M \cong N$, we must also have that $(\lambda x.\mathsf{test}(s, a))(\lambda \boldsymbol{y}.N)\Downarrow$ and hence $[\![(\lambda x.\mathsf{test}(s, a))(\lambda \boldsymbol{y}.N)]\!] = [\![\mathtt{skip}]\!]$.

This implies that there is some $(s', a') \in [\![N]\!]$ such that $((s', a'), *) \in \mathsf{test}(s, a)$. By the defining property of $\mathsf{test}(s, a)$, it is the case that $(s, a) \preceq^- (s', a')$.

Applying a symmetric argument, we can show that there is some $(s'', a'') \in [\![M]\!]$ such that $(s', a') \preceq^- (s'', a'')$.

Since both (s, a) and (s'', a'') are in $[\![M]\!]$ and since $(s, a) \preceq^- (s'', a'')$, the previous lemma tells us that $(s, a) = (s'', a'')$ and hence $(s, a) = (s', a')$. Thus $[\![M]\!] \subseteq [\![N]\!]$. We can argue symmetrically to show that $[\![N]\!] \subseteq [\![M]\!]$ and hence conclude that $[\![M]\!] = [\![N]\!]$.

Putting soundness and completeness together yields full abstraction.

Theorem 3 (Full Abstraction). *Terms M and N of Basic SCI are equivalent if and only if $[\![M]\!] = [\![N]\!]$.*

References

[1] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings, Thirteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–344. IEEE Computer Society Press, 1998. 248

[2] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In O'Hearn and Tennent [7], pages 297–329 of volume 2. 248, 256

[3] J. Hogg. Islands: Aliasing protection in object-oriented languages. In *Proceedings of the OOPSLA '91 Conference on Object-oriented Programming Systems, Languages and Applications*, pages 271–285, November 1991. 248

[4] B. Jacobs. Semantics of weakening and contraction. *Annals of Pure and Applied Logic*, 69:73–106, 1994. 252

[5] P. W. O'Hearn. Resource interpretations, bunched implications and the $\alpha - \lambda$-calculus. In J.-Y. Girard, editor, *Proceedings, Typed Lambda-Calculi and Applications, L'Aquila, Italy, April 1999*, volume 1581 of *LNCS*, pages 258–279. Springer-Verlag, 1999. 247

[6] P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. *Theoretical Computer Science*, 228(1–2):211–252, 1999. A preliminary version appeared in the proceedings of MFPS XI. 247

[7] P. W. O'Hearn and R. D. Tennent, editors. *Algol-like Languages*. Birkhaüser, 1997. 247, 261

[8] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977. 256

[9] U. S. Reddy. Global state considered unnecessary: Object-based semantics for interference-free imperative programs. *Lisp and Symbolic Computation*, 9(1), 1996. 248, 256

[10] J. C. Reynolds. Syntactic control of interference. In *Conf. Record 5th ACM Symposium on Principles of Programming Languages*, pages 39–46, 1978. 247

[11] J. C. Reynolds. The essence of Algol. In *Proceedings of the 1981 International Symposium on Algorithmic Languages*, pages 345–372. North-Holland, 1981. 247, 248

[12] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, February 1997. 248

[13] M. Wall and G. McCusker. A fully abstract game semantics of SCI. Draft, 2002. 248

Optimal Complexity Bounds for Positive LTL Games

Jerzy Marcinkowski* and Tomasz Truderung**

Institute of Computer Science, Wrocław University
jma@ii.uni.wroc.pl
tt@ii.uni.wroc.pl

Abstract. We prove two tight bounds on complexity of deciding graph games with winning conditions defined by formulas from fragments of LTL.
Our first result is that deciding $LTL_+(\Diamond, \wedge, \vee)$ games is in PSPACE. This is a tight bound: the problem is known to be PSPACE-hard even for the much weaker logic $LTL_+(\Diamond, \wedge)$. We use a method based on a notion of, as we call it, persistent strategy: we prove that in games with positive winning condition the opponent has a winning strategy if and only if he has a persistent winning strategy.
The best upper bound one can prove for our problem with the Büchi automata technique, is EXPSPACE. This means that we identify a natural fragment of LTL for which the algorithm resulting from the Büchi automata tool is one exponent worse than optimal.
As our second result we show that the problem is EXPSPACE-hard if the winning condition is from the logic $LTL_+(\Diamond, \bigcirc, \wedge, \vee)$. This solves an open problem from [AT01], where the authors use the Büchi automata technique to show an EXPSPACE algorithm deciding more general $LTL(\Diamond, \bigcirc, \wedge, \vee)$ games, but do not prove optimality of this upper bound

1 Introduction

LTL (linear temporal logic) is one of possible specification languages for correctness conditions in reactive systems verification [MP91]. Two sorts of decision problems arise in this context. One of them is **model checking**. We ask here, for a given transition graph $\mathcal{G}$ of a system, and for a formula φ of LTL, whether φ is valid on all possible computation paths in $\mathcal{G}$. This question is natural when a closed system is verified, by which we mean one whose future behavior only depends on its current state but not on any kind of environment. Model checking for LTL conditions is known to be PSPACE-complete [SC85] (combined complexity). Although, if $\Diamond$ and $\Box$ are the only modalities allowed in the formula then model-checking is NP-complete [SC85]. Other fragments of LTL with easy model-checking problem (in NP or even in P) are identified in [DS98].

* Partially supported by Polish KBN grant 2 PO3A 01818.
** Partially supported by Polish KBN grant 8T11C 04319.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 262–275, 2002.
© Springer-Verlag Berlin Heidelberg 2002

In this paper we are interested in the second kind of decision problems in this area, which is **deciding a game** with condition φ. The computation path here is a result of an infinite game played by two players $\mathcal{S}$ (as *System*) and $\mathcal{E}$ (as *Environment*) on some game graph $\mathcal{G}$. Each vertex of $\mathcal{G}$ is either existential, when $\mathcal{S}$ decides on the next move, or universal, when $\mathcal{E}$ is the one who moves. The goal of $\mathcal{S}$ is to make the formula φ valid on the computation path. This paradigm is being considered in the context of automated synthesis. The future behavior of the system depends here not only on its current state but also on the inputs supplied by some unpredictable environment. It is known that deciding which of the players has a winning strategy in such a graph game is doubly exponential for general LTL formula φ [PR89].

1.1 Previous Work

Positive results. A classical technique for deciding an LTL game is to transform the winning condition φ into a deterministic ω-automaton A_φ, so called generator of φ, which accepts an infinite path if and only if φ is true on this path. Then take $\mathcal{B} = \mathcal{G} \times A_\varphi$ as a new game (where $\mathcal{G}$ is the game graph under consideration). The type of the game $\mathcal{B}$ (Büchi, Rabin, etc.) is the same as the type of the generator A_φ. The winning condition on $\mathcal{B}$ is defined in such a way that the same player who had a winning strategy in the φ game on $\mathcal{G}$ has a winning strategy in the game on $\mathcal{B}$.

In [AT01] Alur and La Torre consider fragments of LTL which have deterministic generators being Büchi automata, and thus the resulting game is a Büchi game and the winning player has a memoryless strategy. It is easy to decide such a game: this can be done in a quadratic time with respect to the size (number of vertices) of the game graph [Tho95]. Alur and La Torre improve on this: they notice that one can decide a Büchi game in SPACE($d \log n$), where n is the size of the game graph and d is another parameter called the longest distance of the game graph. They carefully construct Büchi generators for different fragments of LTL, trying to keep the longest distance as small as possible. In this way they show that deciding $LTL(\Diamond, \wedge)$ games is in PSPACE and that the same problem for $LTL(\Diamond, \bigcirc, \wedge, \vee)$ (and thus also for $LTL(\Diamond, \wedge, \vee)$) is in EXPSPACE.

Lower bounds. It is known since [PR89] that the doubly exponential algorithm deciding general LTL games is optimal. In their study of the complexity of games with conditions from fragments of LTL [AT01] Alur and La Torre show the PSPACE lower bound for $LTL_+(\Diamond, \wedge)$ (this proof is very easy) and the EXPTIME lower bound for $LTL(\Diamond, \bigcirc, \wedge)$, and thus for $LTL(\Diamond, \bigcirc, \wedge, \vee)$.

1.2 Our Contribution

Lower bound for $LTL_+(\Diamond, \bigcirc, \wedge, \vee)$. In Section 5 we solve an open problem from [AT01] proving:

Theorem 1. *Deciding games with the winning condition in $LTL_+(\Diamond, \bigcirc, \wedge, \vee)$ is EXPSPACE-hard.*

This is an optimal result, and a surprisingly strong one: it turns out that the problem for the positive part $LTL_+(\Diamond, \circ, \wedge, \vee)$ is as hard as for its boolean closure $LTL(\Diamond, \circ, \wedge, \vee)$.

In our proof we use the fact that EXPSPACE can be viewed as a variant of alternating EXPTIME. The game graph is defined in such a way that in the first stage of a play the opponents, by turn, construct (or, as we say, *declare*) a sequence which is intended to be a computation of an alternating machine. Then, in the second stage, some way must be provided to detect all possible sorts of cheating against the legality of this computation. And this is where our main tool comes, which we call *the objection graph*. It appears that a formula of $LTL(\Diamond, \circ, \wedge, \vee)$ expressing the property *there are two equal patterns of length n on the path, both beginning with the the state p* requires the size exponential in n. But as we show, if we have two players declaring a sequence, and each of them can "raise an objection" by moving the play into the objection graph, then a small (polynomial-length) formula of $LTL(\Diamond, \circ, \wedge, \vee)$ is enough to detect equality of patterns of length n, as well as all the legality violations we need to detect. Since we wanted to keep the formula positive, we could only grant to $\mathcal{S}$ the ability of raising objections. This means that his cheats in the first stage could remain undetected. This is why we need to construct the first stage with some care.

Positive result for $LTL_+(\Diamond, \vee, \wedge)$. In Section 4 we prove:

Theorem 2. *Deciding games with the winning condition in $LTL_+(\Diamond, \vee, \wedge)$ is in PSPACE.*

Again, it follows from [AT01] that this result is optimal. $LTL_+(\Diamond, \vee, \wedge)$ may appear to be quite a simple logic but still it requires huge generators. Indeed, while studying $LTL(\Diamond, \vee, \wedge)$ the authors of [AT01] show that a deterministic generator for the formula $\Diamond((p_1 \vee \Diamond q_1) \wedge (p_2 \vee \Diamond q_2) \wedge \ldots (p_k \vee \Diamond q_k))$ of the logic $LTL_+(\Diamond, \vee, \wedge)$, requires exponential longest distance and doubly exponential size. This means that with their Büchi automata methodology no upper bound better than EXPSPACE can be achieved for $LTL_+(\Diamond, \vee, \wedge)$ games. And this, as we prove, is one exponent worse than optimal. The core of our technique is the notion of a *persistent strategy*[1] (see Definition 1). In Section 3 we prove that if $\mathcal{E}$ has a winning strategy in any positive game then he also has a persistent winning strategy. And deciding an $LTL_+(\Diamond, \vee, \wedge)$ game if $\mathcal{E}$ uses a persistent strategy is in PSPACE, as we show, in Section 4.

2 Preliminaries

Linear Temporal Logic. Let P be a given finite set of *atomic propositions*. *Linear temporal logic* (LTL) formulas are built according to the grammar:

$$\varphi ::= s \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \circ\varphi \mid \Diamond\varphi \mid \Box\varphi \mid \varphi \,\mathsf{U}\, \varphi,$$

[1] The notion of persistent strategy is a very natural one. We believe it can have other applications. That is why we would not be surprised to learn that it has been studied before. However, we are not currently aware of any reference to such a study.

where s is a *state predicate*, that is a boolean combination of atomic propositions. Temporal operators $\bigcirc, \Diamond, \Box, \mathsf{U}$ are usually read as *next*, *eventually*, *always*, and *until* respectively. LTL formulas are interpreted in the standard way on infinite sequences over the alphabet $\Sigma = 2^P$.

Fragments of LTL. We denote by $LTL_+(op_1, \ldots, op_k)$ the set of LTL formulas built from state predicates using only boolean and temporal connectives $op_1, \ldots, op_k$. Furthermore, following [AT01], we denote by $LTL(op_1, \ldots, op_k)$ the set of formulas obtained as boolean combinations of $LTL_+(op_1, \ldots, op_k)$.

Game Graphs. A two-player φ game on $\mathcal{G}$ is given by an LTL formula φ, called a *winning condition*[2], and a *game graph* $\mathcal{G} = (V, V_\forall, V_\exists, E, v_0, \delta)$ with the set of vertices V partitioned into $V_\forall$ and $V_\exists$, the set of edges $E \subseteq V \times V$, the initial vertex $v_0 \in V$, and a function $\delta : V \to 2^P$ which assigns to each vertex a set of atomic propositions. We say that p is true in v if $p \in \delta(v)$. Elements of $V_\forall$ are called *universal vertices*, and elements of $V_\exists$ are called *existential vertices*. To denote elements of V we will use letters $u, v, w, \ldots$

A *finite play* is a sequence $u_0 \ldots u_k \in V^*$ such that u_0 is the initial vertex, and $\langle u_{i-1}, u_i \rangle \in E$, for all $i \in \{1, \ldots, k\}$. Similarly, an *infinite play* is an infinite sequence $u_0 u_1 \ldots$ of elements from V such that u_0 is the initial vertex, and $\langle u_{i-1}, u_i \rangle \in E$, for all $i \geq 1$. To denote (finite or infinite) plays we will use letters $\mathbf{u}, \mathbf{v}, \mathbf{w}, \ldots$ During a game, two players $\mathcal{S}$ (the System) and $\mathcal{E}$ (the Environment) construct a sequence $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}, \ldots$ of finite plays. They begin with $\mathbf{v_0} = v_0$. If $\mathbf{v_i} = \mathbf{w}w$ for some $\mathbf{w}w$ then $\mathbf{v_{i+1}} = \mathbf{w}ww'$, where w' is selected by $\mathcal{S}$ if w is existential, and by $\mathcal{E}$ if w is universal. Let $\mathbf{v}$ be the infinite play which is the limit of $\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2}, \ldots$ Then $\mathcal{S}$ wins if $\mathbf{v} \models \varphi$. A *strategy* and a *winning strategy* for $\mathcal{S}$ (or $\mathcal{E}$) is defined in the standard way.

The problem of deciding $LTL(op_1, \ldots, op_k)$ (or $LTL_+(op_1, \ldots, op_k)$) *games* is a problem of deciding whether $\mathcal{S}$ has a winning strategy for a given game graph, and a winning condition given as an $LTL(op_1, \ldots, op_k)$ (or $LTL_+(op_1, \ldots, op_k)$) formula.

3 Positive Games and Persistent Strategies

Definition 1. *The strategy of the player $\mathcal{P}$ is* persistent *if for each play $v_1 v_2 \ldots v_k$ played by $\mathcal{P}$ according to this strategy, if $v_i = v_j$, for some $1 \leq i, j < k$, and v_i is a vertex where $\mathcal{P}$ is to move, then $v_{i+1} = v_{j+1}$.*

In other words, a strategy of the player $\mathcal{P}$ is persistent if, each time $\mathcal{P}$ decides on a move in some vertex v, he repeats the decision he made when v was visited for the first time.

One of the most well-studied kind of strategies are memoryless strategies: this means that the way the player behaves only depends on the vertex of the graph, not on the history of the game. Being persistent is a weaker property than being memoryless:

[2] In general many types of winning conditions are considered (see [Tho90]).

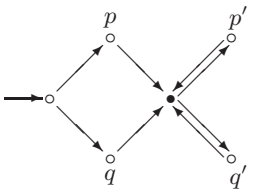


Fig. 1.

Example. Let $\mathcal{G}$ be a game graph with $V = \{u, u_p, u_q, u_{p'}, u_{q'}, v\}$ where u is the initial vertex and all vertices except v are existential (Fig. 1). The edges in E are: $\langle u, u_p\rangle, \langle u, u_q\rangle, \langle u_p, v\rangle, \langle u_q, v\rangle, \langle v, u_{p'}\rangle, \langle v, u_{q'}\rangle, \langle u_{p'}, v\rangle, \langle u_{q'}, v\rangle$. The variables p, q, p', q' are true in vertices $u_p, u_q, u_{p'}$ and $u_{q'}$ respectively. Let φ be the formula $\Diamond((p \wedge \Diamond p') \vee (q \wedge \Diamond q'))$. Then $\mathcal{E}$ does not have a memoryless winning strategy in the φ game on $\mathcal{G}$ but he does have a persistent winning strategy. □

As we are soon going to see the existence of a persistent winning strategy in the example above is not a coincidence.

Notations. For two plays $\mathbf{w}$ and $\mathbf{v}$ we will use the notation $\mathbf{w} \leq \mathbf{v}$ to say that $\mathbf{w}$ is a prefix of $\mathbf{v}$. Let $\mathbf{v}, \mathbf{w}$ be two plays, finite or not. Then by $\mathbf{w} \sqsubseteq \mathbf{v}$ we mean the expression "$\mathbf{w}$ is a subsequence of $\mathbf{v}$" (where *abc* is a subsequence of *adbdc*).

Definition 2. *We call a game* positive*, if for each two infinite plays* $\mathbf{w}$ *and* $\mathbf{v}$*, if* $\mathcal{S}$ *wins the play* $\mathbf{w}$ *and* $\mathbf{w} \sqsubseteq \mathbf{v}$ *then* $\mathcal{S}$ *wins also the play* $\mathbf{v}$*.*

It will be convenient in this section to see a strategy for $\mathcal{E}$ as a tree of all possible finite plays played according to this strategy. The following definition is consistent with the standard way of defining strategy:

Definition 3. *A* strategy for $\mathcal{E}$ *is a set* T *of finite plays such that:*

(i) $v_0 \in T$*, where* v_0 *is (the word consisting of) the initial vertex of* $\mathcal{G}$*;*
(ii) *if* $\mathbf{w} \in T$ *and* $\mathbf{v} \leq \mathbf{w}$ *is a nonempty prefix of* $\mathbf{w}$ *then* $\mathbf{v} \in T$*;*
(iii) *if* $\mathbf{w}w \in T$*, where* w *is an existential vertex of* $\mathcal{G}$*, then* $\mathbf{w}wv \in T$ *for each vertex* v *such that* $(w, v) \in E$*;*
(iv) *if* $\mathbf{w}w \in T$*, where* w *is a universal vertex of* $\mathcal{G}$*, then* $\mathbf{w}wv \in T$ *for exactly one vertex* v *such that* $(w, v) \in E$*.*

A strategy for $\mathcal{E}$, as defined above, has a natural structure of an infinite tree, and is *winning* if each infinite path of this tree is a play won by $\mathcal{E}$.

Lemma 1. *Let* T *be a winning strategy for* $\mathcal{E}$ *in some positive game. Let* T' *be a strategy for* $\mathcal{E}$ *with the property that for each* $\mathbf{w} \in T'$ *there exists* $\mathbf{v} \in T$ *such that* $\mathbf{w} \sqsubseteq \mathbf{v}$*. Then* T' *is also a winning strategy for* $\mathcal{E}$*.* □

The main result of this section is:

Theorem 3. *If $\mathcal{E}$ has a winning strategy in a some positive game on some graph $\mathcal{G}$ with n vertices, then he has a winning strategy which is persistent.*

The proof of the theorem will take the rest of this section. The following notation will be useful:

Definition 4. *Let T be a strategy for $\mathcal{E}$ and let v be a universal vertex. Then by T^v we denote the set of those $\mathbf{v} \in T$ which are of the form $\mathbf{v}'v$ for some $\mathbf{v}'$. Similarly, by T^{vw} we denote the set of those $\mathbf{v} \in T$ which are of the form $\mathbf{v}'vw$ for some $\mathbf{v}'$. By $T^v_{\mathbf{w}}$ (and $T^{vw}_{\mathbf{w}}$) we will denote the set of those $\mathbf{u} \in T^v$ ($\mathbf{u} \in T^{vw}$) for which $\mathbf{u} \geq \mathbf{w}$ holds.*

We will need a local version of the notion of a persistent strategy:

Definition 5. *Let T be a strategy for $\mathcal{E}$, and $\mathbf{v} \in T^v$ for some universal v. Let w be the (unique) vertex of V such that $\mathbf{v}w \in T$. Then T is $\mathbf{v}$-persistent if for each play $\mathbf{w} \in T^v_{\mathbf{v}}$ we have $\mathbf{w}w \in T$.*

The meaning of the definition is that T is $\mathbf{v}$-persistent for some $\mathbf{v} \in T^v$ if the decision about the way $\mathcal{E}$ plays in vertex v made at the moment after the play $\mathbf{v}$, will not be changed in the future. It is easy to see that a strategy T is persistent if and only if it is $\mathbf{v}$-persistent for each $\mathbf{v} \in T$ such that $\mathbf{v} \in T^v$ for some universal v.

To end the proof of Theorem 3, it will be enough to prove:

Lemma 2. *Let T be a winning strategy for $\mathcal{E}$. For each universal v and each $\mathbf{v} \in T^v$, there exists a winning strategy $\mathcal{T}(T, \mathbf{v})$ for $\mathcal{E}$ such that:*

1. *$\mathcal{T}(T, \mathbf{v})$ is $\mathbf{v}$-persistent;*
2. *if $\mathbf{v} \not\leq \mathbf{w}$ then $\mathbf{w} \in \mathcal{T}(T, \mathbf{v})$ if and only if $\mathbf{w} \in T$;*
3. *if $\mathbf{v} \leq \mathbf{u}$ and $\mathbf{u}uu' \in \mathcal{T}(T, \mathbf{v})$ for some universal $u \neq v$, then there exists $\mathbf{w}$ such that $\mathbf{v} \leq \mathbf{w}$ and $\mathbf{w}uu' \in T$.*

With the last lemma a persistent winning strategy for $\mathcal{E}$ can be constructed from any winning strategy for $\mathcal{E}$: going from the root of T down each path, replace T with $\mathcal{T}(T, \mathbf{v})$ each time a play $\mathbf{v} \in T^v$ is reached, where v is universal and v does not occur in $\mathbf{v}$ earlier than as its last symbol. This procedure converges to some winning strategy for $\mathcal{E}$, because on each path such a replacement will be done at most n times. By item 2 of the lemma two such replacements performed in $\leq$–incomparable points do not interfere. By item 3, if $\mathbf{u} \leq \mathbf{w}$ then $\mathcal{T}((\mathcal{T}(T, \mathbf{u})), \mathbf{w})$ remains $\mathbf{u}$-persistent, so the later replacements do not destroy the effect of the earlier.

Proof of Lemma 2. Let $\leq_v$ be the prefix ordering on T^v (so $\leq_v$ coincides on $T^v \times T^v$ with the relation $\leq$, the prefix ordering on the set of all finite plays).

There are 2 cases.

Case 1. There is a play $\mathbf{w} \in T^v_{\mathbf{v}}$ which is $\leq_v$ maximal.

It is easy to see that in this case we can put $\mathbf{s} \in \mathcal{T}(T, \mathbf{v})$ for $\mathbf{v} \not\leq \mathbf{s}$ such that $\mathbf{s} \in T$, and $\mathbf{vs} \in \mathcal{T}(T, \mathbf{v})$ for each $\mathbf{ws} \in T$. By Lemma 1, the obtained strategy is a winning strategy for $\mathcal{E}$.

Case 2. There is no such $\leq_v$ maximal play in $T^v_{\mathbf{v}}$.

This case is more complicated. We will need:

Definition 6. *In the situation of Case 2, let $\mathbf{u} \in T^v_{\mathbf{v}}$ be such that $\mathbf{u}v' \in T$. We will say that $\mathbf{u}$ is v'-dense if, for each $\mathbf{w} \in T^v_{\mathbf{u}}$, the set $T^{vv'}_{\mathbf{w}}$ is non-empty.*

It turns out that:

Lemma 3. *There exists $\mathbf{u}v' \in T$, where $\mathbf{u} \in T^v_{\mathbf{v}}$, such that $\mathbf{u}$ is v'-dense.*

Proof. Suppose there is no such $\mathbf{u}$. We define by induction a sequence $\mathbf{w}_1 \leq \mathbf{w}_2 \leq \mathbf{w}_3 \ldots$ of plays, and a sequence $w_1, w_2, \ldots$ of vertices. Let $\mathbf{w}_1 = \mathbf{v}$. Suppose that $\mathbf{w}_i \in T^v_{\mathbf{v}}$ is already defined, and let w_i be such that $\mathbf{w}_i w_i \in T$. We know that $\mathbf{w}_i$ is not w_i dense. This means that there exists $\mathbf{u} \in T^v_{\mathbf{w}_i}$ such that $T^{vw_i}_{\mathbf{u}}$ is empty. Define $\mathbf{w}_{i+1} = \mathbf{u}$. Notice that if $i > j$ then $T^{vw_j}_{\mathbf{w_i}}$ is empty (because $T^{vw_j}_{\mathbf{w}_i} \subseteq T^{vw_j}_{\mathbf{w}_{j+1}}$). Thus $w_i \neq w_j$. We get a contradiction since there are only finitely many elements of V. □

Once we have $\mathbf{w}$ which is w-dense for some w we are ready to construct $\mathcal{T}(T, \mathbf{v})$. Consider a play $\mathbf{u} \geq \mathbf{w}$. Then: $\mathbf{u} = \mathbf{w}\mathbf{v_1}v\mathbf{v_2}v \ldots v\mathbf{v_m}$ where v does not occur in $\mathbf{v_1}\mathbf{v_2} \ldots \mathbf{v_m}$. Let $\alpha(\mathbf{s})$ be $\mathbf{s}$ if the first symbol of $\mathbf{s}$ is w and the empty word otherwise. Define $\beta(\mathbf{u})$ as $\mathbf{v}\alpha(\mathbf{v_1}v)\alpha(\mathbf{v_2}v) \ldots \alpha(\mathbf{v_m})$.

Now: $\mathcal{T}_0(T, \mathbf{v}) = \{\mathbf{u} : \mathbf{v} \not\leq \mathbf{u} \wedge \mathbf{u} \in T\} \cup \{\beta(\mathbf{u}) : \mathbf{u} \geq \mathbf{w} \wedge \mathbf{u} \in T\}$.

$\mathcal{T}_0(T, \mathbf{v})$ is not yet a strategy: condition (iv) of Definition 3 may not hold in this tree: it is possible that some plays ending with a universal vertex different than v will have more than one direct successor there. One can prune the tree $\mathcal{T}_0(T, \mathbf{v})$ in any way, so the result satisfies Definition 3 (iv), and call the result $\mathcal{T}(T, \mathbf{v})$. With the use of Lemma 1 one can now verify that $\mathcal{T}(T, \mathbf{v})$ is a strategy as required by Definition 3 and by Lemma 2. This ends the proof of Lemma 2. □

4 Proof of Theorem 2

Notations. Let $n = |V|$ where V is the set of vertices of the game graph $\mathcal{G}$. By φ we always mean a formula of $LTL_+(\Diamond, \vee, \wedge)$ in this section.

Since φ is positive the following lemma holds:

Lemma 4. *For a given game graph $\mathcal{G}$ and a formula φ there exists $\rho(\mathbf{s})$ such that:*

1. *$\rho(\mathbf{s})$ is a positive boolean combination of expressions of the form $\mathbf{w} \sqsubseteq \mathbf{s}$, where $\mathbf{s}$ is the variable which is free in ρ, and each $\mathbf{w}$ is some fixed word of length not greater than l, where l is the $\Diamond$-depth of φ;*
2. *ρ and φ are equivalent in the sense that for each infinite play $\mathbf{v}$ it holds that $\rho(\mathbf{v})$ if and only if $\mathbf{v} \models \varphi$.*

Proof. Induction on l. □

By the last lemma, if $\mathbf{v}$ is some infinite play won by $\mathcal{S}$ then there exists a finite prefix $\mathbf{v}'$ of $\mathbf{v}$ such that, for every infinite play $\mathbf{w}$, if $\mathbf{v}' \leq \mathbf{w}$ then $\mathbf{w}$ is won by $\mathcal{S}$. In such a case we say that $\mathcal{S}$ *secures his win after the play* $\mathbf{v}'$. One can prove that if $\mathcal{S}$ has a winning strategy in the φ game on $\mathcal{G}$ then he can secure his win after a number of steps which is exponential with respect to the combined size of the instance. Our first conjecture was that the win of $\mathcal{S}$ can be secured in such a case already after polynomial number of steps. If true, this would give a straightforward way of proving Theorem 2. It would be enough to perform the mini-max search on the tree of all plays of polynomial depth, a procedure which is clearly in PSPACE. Our conjecture is, however, false:

Theorem 4. *There exist a formula φ and a graph $\mathcal{G}$ such that $\mathcal{S}$ has a winning strategy, but he is not always able to secure his win after a polynomial number of steps.*

Proof. Let M_0 be a game graph consisting of only one existential vertex v_0 where p_0 is true, and of one edge $E(v_0, v_0)$. Let φ_0 be the formula $\Diamond p_0$. We define φ_{k+1} as $\Diamond(p_{k+1} \wedge \Diamond\varphi_k \wedge (q_{k+1} \vee \Diamond r_{k+1}))$. Graph M_{k+1} (Fig. 2) consists of all the vertices and edges of graph M_k, of new existential vertices v_{k+1},w_{k+1} and z_{k+1} and a new universal vertex u_{k+1}. There are also new edges: from v_{k+1} and from w_{k+1} to the initial vertex of M_k, from each existential vertex of M_k to u_{k+1}, from u_{k+1} both to w_{k+1} and to z_{k+1}, and a loop from z_{k+1} to itself. The initial vertex of the new graph is v_{k+1}. The variables which are true in the vertices of M_k remain true in the same vertices in M_{k+1}. For the new vertices: p_{k+1} is true in v_{k+1} and in w_{k+1}, r_{k+1} is true in z_{k+1} and q_{k+1} is true in w_{k+1}.

Now, one can prove by induction on k that, for every k, $\mathcal{S}$ has a winning strategy in the φ_k game on M_k. Assume the claim is true for some k and consider the φ_{k+1} game on M_{k+1}. $\mathcal{S}$ moves from v_{k+1} to v_k and then uses his winning

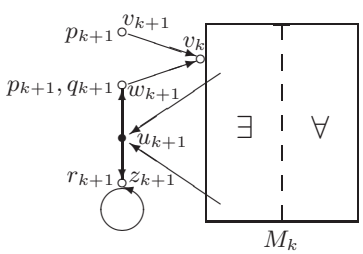


Fig. 2. Graph M_{k+1}

strategy in the φ_k game on M_k. Once he secures his win in the φ_k game on M_k he uses one of the new edges to leave M_k, and goes to u_{k+1}. Now $\mathcal{E}$ is to move. If he decides to go to z_{k+1} then the formula $\Diamond(p_{k+1} \wedge \Diamond\varphi_k \wedge \Diamond r_{k+1})$, which implies φ_{k+1}, is true on the constructed play. If $\mathcal{E}$ prefers to move to w_{k+1} instead of z_{k+1} then $\mathcal{S}$ enters M_k and once again uses his winning strategy in the φ_k game on M_k. Once he secures his win in this smaller game again, the formula $\Diamond(p_{k+1} \wedge \Diamond\varphi_k \wedge q_{k+1})$, which implies φ_{k+1}, holds true on the resulting game.

We also use induction on k in order to show that $\mathcal{E}$ can survive 2^k of steps before the win of $\mathcal{S}$ in the φ_k game on M_k is secured. Assume the claim is true for some k, and consider the situation for $k+1$. If $\mathcal{S}$ makes the step from the M_k part to u_{k+1} before he secures the win in the φ_k game there, then $\mathcal{E}$ can move to z_{k+1} and win. So $\mathcal{S}$ cannot enter u_{k+1} before 2^n moves are made. If now $\mathcal{E}$ moves from u_{k+1} to w_{k+1} then the only way to secure win for $\mathcal{S}$ is to move to v_k and win the φ_k game on M_k again, which again takes at least 2^n moves. □

But it turns out that in spite of Theorem 4 we are still able to find a way of restricting the search only to game trees of polynomial depth. Notice that the game under consideration is positive. So thanks to Theorem 3 we can assume that $\mathcal{E}$ is using a persistent strategy. To end the proof of Theorem 2 it is enough to prove:

Lemma 5. *If $\mathcal{S}$ has a winning strategy in a φ game on $\mathcal{G}$ and he plays against an opponent who uses a persistent strategy, then $\mathcal{S}$ can secure his win after a polynomial number of steps.*

4.1 Proof of Lemma 5

In this subsection we assume that $\mathcal{E}$ uses a persistent strategy.

Lemma 6. *Suppose $\mathbf{v} = v_1 v_2 \ldots v_m$ is a play such that v_m is an existential vertex and $\mathcal{S}$ has a winning strategy after $\mathbf{v}$ is played. Then there exists a play $\mathbf{v}u_1u_2\ldots u_k$, with k polynomial, such that:*

1. *if u_i is universal, for some $1 \leq i \leq k-1$, then $u_i = v_j$, for some $1 \leq j \leq m-1$ and $u_{i+1} = v_{j+1}$;*
2. *either the win of $\mathcal{S}$ is already secured after the play $\mathbf{v}u_1u_2\ldots u_k$ or u_k is a universal vertex which does not occur in $\mathbf{v}u_1u_2\ldots u_{k-1}$, and $\mathcal{S}$ has a winning strategy after the play $\mathbf{v}u_1u_2\ldots u_k$.*

Let us first show that Lemma 5 follows from 6. Notice that $\mathcal{E}$ has no opportunity between v_m and u_k in the lemma to make any decisions about the way the moves are being made. They are either made by $\mathcal{S}$, or are already determined, since the strategy of $\mathcal{E}$ is persistent. So, once the play $\mathbf{v}$ has been played, it is up to $\mathcal{S}$ if $\mathbf{v}u_1u_2\ldots u_k$ is played.

Notice also, that if $\mathbf{v} = v_1v_2\ldots v_m$ is a play such that v_m is a universal vertex and $\mathcal{S}$ has a winning strategy after $\mathbf{v}$ is played, then either $\mathcal{E}$ enters some existential vertex sooner than after n new steps, or he will enter a loop of

universal vertices, and then the win of $\mathcal{S}$ will be secured after at most nl new steps (where l is the $\Diamond$-depth of φ).

Hence Lemma 5 follows from Lemma 6 and from the fact that there are only less than n universal vertices in $\mathcal{G}$.

Proof of Lemma 6. If the play $\mathbf{v}$ is like in the lemma then one can clearly find a continuation of this play $\mathbf{v}v_{m+1}\dots v_{m'}$ such that:

1. for each $m+1 \le i \le m'-1$, if v_i is universal then $v_i = v_j$ for some $1 \le j \le m-1$ and $v_{i+1} = v_{j+1}$,
2. either the win of $\mathcal{S}$ is already secured after the play $\mathbf{v}v_{m+1}\dots v_{m'}$ or $v_{m'}$ is a universal vertex which does not occur in $\mathbf{v}v_{m+1}\dots v_{m'-1}$ and $\mathcal{S}$ has a winning strategy after the play $\mathbf{v}v_{m+1}\dots v_{m'}$.

Consider a directed graph $\mathcal{H}$ whose vertices are the elements of the sequence $v_{m+1}\dots v_{m'}$ and such that (w_1, w_2) is an edge of $\mathcal{H}$ if w_1 is existential and (w_1, w_2) is an edge of $\mathcal{G}$, or if w_1 is universal and the move from w_1 to w_2 was already chosen by $\mathcal{E}$ as a part of his persistent strategy (i.e. $w_1 = v_i$ and $w_2 = v_{i+1}$ for some $0 \le i < m$). Let $\sim$ be an equivalence on the vertices of $\mathcal{H}$ such that $w_1 \sim w_2$ if w_1 and w_2 are reachable from each other in $\mathcal{H}$. Let $\mathcal{H}_0$ be $\mathcal{H}/_\sim$. For two equivalence classes $[w_1]_\sim$ and $[w_2]_\sim$ in $\mathcal{H}_0$ define $[w_1]_\sim \succ [w_2]_\sim$ if $[w_1]_\sim \ne [w_2]_\sim$ and w_2 is reachable from w_1 in $\mathcal{H}$. Let now the sequence $w_1, w_2, \dots w_s$ be such that $w_1 = v_{m+1}$, and w_{i+1} is the first element of $v_{m+1}\dots v_{m'}$ which is right of w_i and such that $w_{i+1} \notin [w_i]_\sim$. Obviously $[w_{i+1}]_\sim \prec [w_i]_\sim$ and so $s \le n$. Now we construct the sequence $u_1, u_2 \dots u_k$: to do it, we first visit each element of $[w_1]_\sim$. Then we visit them again, and again, l times, where l is like in Lemma 4. This is possible since the elements of $[w_1]_\sim$ are reachable from each other. Then we go to $[w_2]_\sim$ and again visit each vertex of this class l times. Then we do the same for $[w_3]_\sim, \dots [w_s]_\sim$. We stop at $u_k = v_{m'}$. The resulting sequence $u_1, u_2 \dots u_k$ is obviously polynomially long. It is easy to see that if $\mathbf{w} \sqsubseteq v_0 v_1 \dots v_m v_{m+1} \dots v_{m'}$ holds, for some word $\mathbf{w}$ with $|\mathbf{w}| \le l$, then also $\mathbf{w} \sqsubseteq v_0 v_1 \dots v_m u_1 \dots u_k$ holds. Our claim follows now from Lemma 4. $\square$

5 Proof of Theorem 1

Suppose that M is a Turing machine which, for an input z of length n, uses exponential space, that is space bounded by 2^{n^k} for some integer k. We can assume, without the loss of generality, that the tape alphabet of M is $\{0, 1\}$, and that M has only one accepting configuration. In this configuration the state of M is q_f, and all the cells of the tape contain 0.

Let $z \in \{0, 1\}^*$ be the input word. Let $n = |z|$ and $N = n^k$. We will construct a game $(\mathcal{G}_z, \varphi_z)$ in which $\mathcal{S}$ has a winning strategy if and only if M *does not accept* z. It is easy to verify that this construction can be done in logarithmic space with respect to n.

The game graph $\mathcal{G}_z$, and the formula φ_z will be constructed in such a way that in order to keep φ_z false, $\mathcal{E}$ will need to *declare* in each *stage* s (from 0

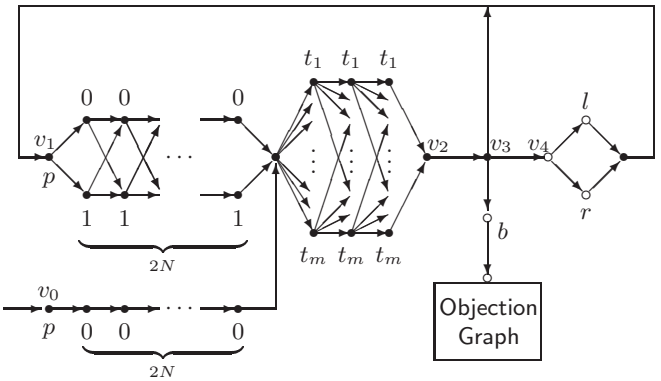


Fig. 3. Graph $\mathcal{G}_z$

up to $2^N - 1$) of the play a triple $\bar{a}(s), \bar{b}(s), \bar{c}(s)$ of configurations of M. This declaration will be understood as his claim that **l:** $\bar{b}(s)$ is reachable from $\bar{a}(s)$ in no more than $2^{(2^N - s+1)}$ computation steps of M, and **r:** $\bar{c}(s)$ is reachable from $\bar{b}(s)$ in such a number of steps. $\mathcal{E}$ will be also forced to declare $\bar{a}(0)$ as the initial configuration of M on z and $\bar{c}(0)$ as the unique accepting configuration.

At the end of each stage $\mathcal{S}$ will be allowed to say if he wishes to see the proof of **l** or the proof of **r**. If he decides on **l** then $\mathcal{E}$ will be supposed to declare $\bar{a}(s+1) = \bar{a}(s)$ and $\bar{c}(s+1) = \bar{b}(s)$. Analogously, if $\mathcal{S}$ decides on **r** after the stage s, then $\mathcal{E}$ will be supposed to declare $\bar{a}(s+1) = \bar{b}(s)$ and $\bar{c}(s+1) = \bar{c}(s)$. If $\mathcal{E}$ would like to cheat here, then finally, when the play reaches the objection graph, $\mathcal{S}$ will have the possibility of *raising objection*, and proving that he was cheated. Finally, φ_z will be written in such a way that the only chance for $\mathcal{E}$ to win will be either to declare $\bar{a}(2^N - 1)$ and $\bar{c}(2^N - 1)$ as equal, or such that $\bar{a}(2^N - 1)$ yields $\bar{c}(2^N - 1)$ in one computation step of M.

5.1 The Game Graph

Let $T = \{t_1, \ldots, t_m\} = \{0, 1\} \times (Q \cup \{-\})$, where Q is the set of states of M, and '$-$' is not an element of Q. Notice that $\bar{x} \in T^{2^N}$ can represent a configuration of M. In fact, $\bar{x}_0, \ldots \bar{x}_{2^N-1}$ represent values of tape cells. If the head of M is over the i-th cell containing y, and the state of M is q, then $\bar{x}_i = (y, q)$. For all the other cells $\bar{x}_i$ has the form $(y, -)$, where y is the content of cell i.

Graph $\mathcal{G}_z$ is shown in Fig. 3. Vertices are labeled by those atomic propositions which are true at them. Vertices labeled by $t_1, \ldots, t_m$ are placed in three columns in such a way that each vertex in the first and the second column is connected with every vertex in the next column. Solid circles represent universal vertices, whereas empty circles are existential. The definition of the objection graph will be given later. Notice that, whenever $\mathcal{E}$ is in the vertex v_1 labeled by p, he can choose any path of length $2N$ of vertices labeled by 0 or 1, thus he can choose any sequence $\bar{x} \in \{0, 1\}^{2N}$ which can be treated as a binary representation of a

pair (s, c), where $0 \leq s, c < 2^N$. In that case we say that $\mathcal{E}$ *declares* (s, c). The play begins in the vertex v_0, also labeled by p, where $\mathcal{E}$ has to declare $(0, 0)$.

Definition 7. $\mathcal{E}$ plays fair *if and only if the following conditions are satisfied:*

(i) *each time he is in v_1, he declares a pair (s, c) which is the immediate successor of (s', c') declared previously (i.e. $s = s'$ and $c = c' + 1$ if $c' < 2^N - 1$, and $s = s' + 1$ and $c = 0$ if $c' = 2^N - 1$),*
(ii) *each time he is in v_3, immediately after declaring (s, c) for $c < 2^N - 1$, he chooses v_1,*
(iii) *each time he is in v_3, immediately after declaring $(s, 2^N - 1)$ for $s < 2^N - 1$, he chooses v_4,*
(iv) *each time he is in v_3, immediately after declaring $(2^N - 1, 2^N - 1)$, he chooses the vertex labeled by b.*

As one can see, if $\mathcal{E}$ plays fair, then he declares each pair from $(0, 0)$ up to $(2^N - 1, 2^N - 1)$ in increasing order. After declaring $(2^N - 1, 2^N - 1)$, $\mathcal{E}$ terminates the play choosing the vertex labeled by b. Furthermore, $\mathcal{E}$, each time after declaring $(s, 2^N - 1)$, goes to vertex v_4, where $\mathcal{S}$ can choose between the vertices labeled by l or r.

Definition 8. *Suppose that $\mathcal{E}$ plays fair.*

Define as $a(s, i), b(s, i)$ and $c(s, i)$ the three elements of T which are labels of the vertices selected by $\mathcal{E}$ from the first, second and third column immediately after declaring (s, i). Let $\bar{a}(s) = \langle a(s, 0), \ldots, a(s, 2^N - 1)\rangle$, $\bar{b}(s) = \langle b(s, 0), \ldots, b(s, 2^N - 1)\rangle$, and $\bar{c}(s) = \langle c(s, 0), \ldots, c(s, 2^N - 1)\rangle$. We say that $\mathcal{E}$ declares configurations $\bar{a}(s), \bar{b}(s), \bar{c}(s)$ in stage s.

We say that $\mathcal{S}$ answers *$h \in \{l, r\}$* in stage *s if and only if he chooses vertex labeled by h immediately after $\mathcal{E}$ declares $(s, 2^N - 1)$. In that case we denote h by $h(s)$.*

It is easy to check that if $\mathcal{E}$ plays fair then, for each stage s, $\bar{a}(s), \bar{b}(s), \bar{c}(s)$ are well-defined, and for each stage $s < 2^N - 1$, also $h(s)$ is well-defined.

Definition 9. $\mathcal{E}$ plays according to M and z *if and only if he plays fair, and:*

(v) *$\bar{a}(0)$ corresponds to the initial configuration of M on the input z, and $\bar{c}(0)$ corresponds to the accepting configuration of M,*
(vi) *either $\bar{a}(2^N - 1) = \bar{c}(2^N - 1)$, or the configuration $\bar{a}(2^N - 1)$ yields the configuration $\bar{c}(2^N - 1)$ in one computation step of M,*
(vii) *for each stage $s \in \{0, \ldots, 2^N - 2\}$, if $h(s) = l$ then $\bar{a}(s + 1) = \bar{a}(s)$ and $\bar{c}(s + 1) = \bar{b}(s)$, and similarly, if $h(s) = r$ then $\bar{a}(s + 1) = \bar{b}(s)$ and $\bar{c}(s + 1) = \bar{c}(s)$.*

Lemma 7. *$\mathcal{E}$ is able to play according to M and z if and only if M accepts z.*

Proof. Rewrite the proof of the fact that EXPSPACE = AEXPTIME. □

We will call a formula γ of $LTL_+(\Diamond, \bigcirc, \vee, \wedge)$ *local* if it is small (polynomial) and has the form $\Diamond\gamma'$ where γ' is $\Diamond$-free. By local formulas we can express existence, on an infinite play, of some patterns of polynomial length. One can see that there exists a disjunction φ_1 of local formulas which is valid for exactly those plays which violate one of the conditions (i)-(vi) of Definitions 7 and 9.

Example. The subformula of φ_1 which holds if and only if condition (ii) of Definition 1 is violated could be as follows:

$$\Diamond\big(p \wedge (\bigcirc^{N+1}0 \vee \cdots \vee \bigcirc^{2N}0) \wedge (\bigcirc^{2N+7}(\neg p))\big),$$

where $\bigcirc^k 0$ stands for the sequence of operators $\bigcirc$ of length k followed by 0. $\square$

Things are more complicated with point (vii) of Definition 9: the formula written in the naive way would be exponentially big. That is because in this case we have to express some relation between two remote fragments of a play. To deal with this problem, we need some participation of $\mathcal{S}$. That is the point where the objection graph is used. In the next section we give the description of the objection graph, the definition of φ_2, and show that $\mathcal{S}$ can make φ_2 true if and only if point (vii) of Definition 9 is violated.

Now, we can define the winning condition of our game:

$$\varphi_z = \varphi_1 \vee \varphi_2.$$

The following lemma is a consequence of Lemma 7, and completes the proof of Theorem 1.

Lemma 8. *$\mathcal{S}$ has the winning strategy in game $(\mathcal{G}_z, \varphi_z)$ if and only if M does not accept z.*

5.2 Raising Objections

In this section we describe a mechanism which allows $\mathcal{S}$ to *raise an objection*, and consequently, to win the game whenever $\mathcal{E}$ violates point (vii) of Definition 9 for some pair of stages s and $s+1$. There are two symmetrical subcases: when $\mathcal{S}$ answers l in the stage s, and when $\mathcal{S}$ answers r in this stage, and so φ_2 will be a disjunction of two symmetrical formulas φ_l and φ_r. We will show how to write the first of them.

Once $\mathcal{S}$ enters the objection graph (Fig. 4) he first declares two numbers of length N. We will call the numbers s^1 and p^1. Then he declares three elements of T, call them a^1, b^1 and c^1, then again two numbers of length N which we call s^2 and p^2, and finally, before the play enters an infinite loop, he declares a^2, b^2 and c^2, again elements of T. One can easily write a local formula ρ expressing the fact that $p^1 = p^2$ and $s^1 + 1 = s^2$ but $a^1 \neq a^2$ or $b^1 \neq c^2$.

Assume that we have a formula ψ_q which is true in vertex v of an infinite play $\mathbf{v}$ if and only if the pattern of length $2N+4$ beginning in the direct successor of v is equal to the pattern of length $2N+4$ beginning in the direct successor of the vertex where q is true. We consider here two patterns to be equal if the same

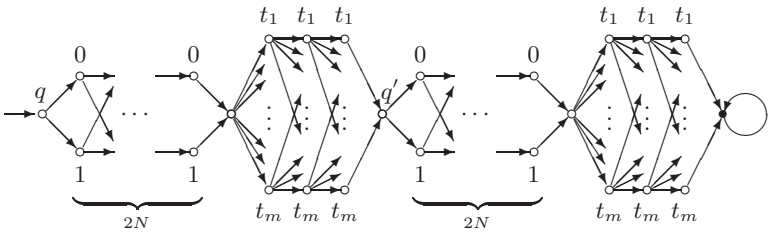


Fig. 4. The Objection Graph

atomic propositions are true in respective vertices of the patterns. Let $\psi_{q'}$ be like ψ_q but with q' instead of q. We can write φ_l as: $\rho \wedge \Diamond(p \wedge \psi_q \wedge \Diamond(l \wedge \Diamond(p \wedge \psi_{q'})))$.

Now, if indeed $\mathcal{E}$ violates point (vii) of Definition 9 in the way described in the beginning of this subsection, then the strategy for $\mathcal{S}$ is to find the number d of a position in the sequence where $\bar{a}(s)$ is not equal to $\bar{a}(s+1)$, or where $\bar{b}(s)$ is not equal to $\bar{c}(s+1)$, enter the objection graph, declare s as s^1, d as p^1, $a(s,d), b(s,d), c(s,d)$ as a^1, b^1 and c^1, then declare $s+1$ as s^2, again d as p^2 and finally $a(s+1,d), b(s+1,d), c(s+1,d)$ as a^2, b^2 and c^2.

It remains to define formula ψ_q:

$$\psi_q = \bigwedge_{i=1}^{2N+4} \psi_q^i,$$

where $\psi_q^i = \big(\bigcirc^i s_1 \wedge \Diamond(q \wedge \bigcirc^i s_1)\big) \vee \cdots \vee \big(\bigcirc^i s_l \wedge \Diamond(q \wedge \bigcirc^i s_l)\big)$, and $\{s_1, \ldots, s_l\} = T \cup \{0, 1\}$.

References

[AT01] R. Alur and S. La Torre, *Deterministic generators and games for LTL fragments*, Proceedings of LICS 2001, Springer Verlag, 2001, pp. 291–300. 262, 263, 264, 265

[DS98] S. Demri and P. Schnoebelen, *The complexity of propositional linear temporal logics in simple cases*, proceedings of STACS 1998, Springer Verlag, 1998, pp. 61–72. 262

[MP91] Z. Manna and A. Pnueli, *The temporal logic of reactive and concurent systems*, 1991. 262

[PR89] A. Pnueli and R. Rosner, *On the synthesis of a reactive module*, Proceedings of 16th ACM POPL, ACM Press, 1989, pp. 179–190. 263

[SC85] A. P. Sistla and E. M. Clarke, *The complexity of propositional temporal logics*, The Journal of ACM **32** (1985), no. 733, 733–749. 262

[Tho90] W. Thomas, *Automata on infinite objects*, Handbook of Theoretical Computer Science (J. van Leeuven, ed.), vol. B, Elsevier Science Publishers, 1990, pp. 133–186. 265

[Tho95] ———, *On the synthesis of strategies in infinite games*, Proceedings of STACS 1995, LNCS 900, Springer Verlag, 1995, pp. 1–13. 263

The Stuttering Principle Revisited: On the Expressiveness of Nested X and U Operators in the Logic LTL

Antonín Kučera* and Jan Strejček**

Faculty of Informatics, Masaryk University
Botanická 68a, CZ-602 00 Brno, Czech Republic
{tony,strejcek}@fi.muni.cz

Abstract. It is known that LTL formulae without the 'next' operator are invariant under the so-called *stutter-equivalence* of words. In this paper we extend this principle to general LTL formulae with given nesting depths of the 'next' and 'until' operators. This allows us to prove the semantical strictness of three natural hierarchies of LTL formulae, which are parametrized either by the nesting depth of just one of the two operators, or by both of them. As another interesting corollary we obtain an alternative characterization of LTL languages, which are exactly the regular languages closed under the generalized form of stutter equivalence. We also indicate how to tackle the state-space explosion problem with the help of presented results.

1 Introduction

Linear temporal logic (LTL) [Pnu77] is a popular formalism for specifying properties of (concurrent) programs. The syntax of LTL is given by the following abstract syntax equation:

$$\varphi \ ::= \ \mathtt{tt} \ \mid \ p \ \mid \ \neg\varphi \ \mid \ \varphi_1 \wedge \varphi_2 \ \mid \ \mathsf{X}\varphi \ \mid \ \varphi_1 \,\mathsf{U}\, \varphi_2$$

Here p ranges over a countable set $\Lambda = \{o, p, q, \ldots\}$ of *letters*. We also use $\mathsf{F}\varphi$ to abbreviate $\mathtt{tt}\,\mathsf{U}\,\varphi$, and $\mathsf{G}\varphi$ to abbreviate $\neg\mathsf{F}\neg\varphi$.

In this paper, we are mainly interested in theoretical aspects of LTL (though some remarks on a potential applicability of our results to model-checking with the logic LTL are mentioned in Section 4). To simplify our notation, we define the semantics of LTL in terms of languages over finite words (all of our results carry over to infinite words immediately). An *alphabet* is a finite set $\Sigma \subseteq \Lambda$. Let Σ be an alphabet and φ an LTL formula. Let $w \in \Sigma^*$ be a word over Σ. The length of w is denoted by $|w|$, and the individual letters of w are denoted by $w(0), w(1), \ldots, w(n-1)$, where $n = |w|$. Moreover, for every $0 \leq i < |w|$ we

* Supported by the Grant Agency of Czech Republic, grant No. 201/00/1023.
** Supported by the Grant Agency of Czech Republic, grant No. 201/00/0400, and by a grant FRVŠ No. 601/2002.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 276–291, 2002.
© Springer-Verlag Berlin Heidelberg 2002

denote by w_i the ith suffix of w, i.e., the word $w(i) \cdots w(|w|-1)$. Finally, for all $0 \leq i < |w|$ and $j \geq 1$ such that $i+j \leq |w|$ the symbol $w(i,j)$ denotes the subword of w of length j which starts with $w(i)$.

Remark 1. To simplify our notation, we adopt the following convention: whenever we refer to $w(i)$, w_i, or $w(i,j)$, we implicitly impose the condition that the object *exists*. For example, the condition '$w(4) = p$' should be read 'the length of w is at least 5 and $w(4) = p$'.

The *validity* of φ for $w \in \Sigma^*$ is defined as follows:

$$
\begin{array}{lll}
w \models \mathtt{tt} & & \\
w \models p & \text{iff} & p = w(0) \\
w \models \neg\varphi & \text{iff} & w \not\models \varphi \\
w \models \varphi_1 \wedge \varphi_2 & \text{iff} & w \models \varphi_1 \wedge w \models \varphi_2 \\
w \models \mathsf{X}\varphi & \text{iff} & w_1 \models \varphi \\
w \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} & \exists i \in \mathbb{N}_0 : w_i \models \varphi_2 \wedge \forall\, 0 \leq j < i : w_j \models \varphi_1
\end{array}
$$

For every alphabet Σ, every LTL formula φ defines the language $L_\varphi^\Sigma = \{w \in \Sigma^* \mid w \models \varphi\}$. From now on we omit the 'Σ' superscript in L_φ^Σ, because it is always clearly determined by the context.

It is well-known that languages definable by LTL formulae form a proper subclass of regular languages [Tho91]. More precisely, LTL languages are exactly the languages definable in first-order logic [Kam68] and thus exactly the languages recognizable by deterministic counter-free automata [MP71].

Since LTL contains just two modal connectives, a natural question is how they influence the expressive power of LTL. First, let us (inductively) define the *nesting depth* of the X and the U modality in a given LTL formula φ, denoted $X(\varphi)$ and $U(\varphi)$, respectively.

$$
\begin{array}{ll}
U(\mathtt{tt}) = 0 & X(\mathtt{tt}) = 0 \\
U(p) = 0 & X(p) = 0 \\
U(\varphi \wedge \psi) = \max\{U(\varphi), U(\psi)\} & X(\varphi \wedge \psi) = \max\{X(\varphi), X(\psi)\} \\
U(\mathsf{X}\varphi) = U(\varphi) & X(\mathsf{X}\varphi) = X(\varphi) + 1 \\
U(\varphi \,\mathsf{U}\, \psi) = \max\{U(\varphi), U(\psi)\} + 1 & X(\varphi \,\mathsf{U}\, \psi) = \max\{X(\varphi), X(\psi)\}
\end{array}
$$

Now we can introduce three natural hierarchies of LTL formulae. For all $m, n \in \mathbb{N}_0$ we define

$$
\begin{aligned}
\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n) &= \{\varphi \in \mathrm{LTL} \mid U(\varphi) \leq m \ \wedge \ X(\varphi) \leq n\} \\
\mathrm{LTL}(\mathsf{U}^m) &= \textstyle\bigcup_{i=0}^{\infty} \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^i) \\
\mathrm{LTL}(\mathsf{X}^n) &= \textstyle\bigcup_{i=0}^{\infty} \mathrm{LTL}(\mathsf{U}^i, \mathsf{X}^n)
\end{aligned}
$$

Hence, the $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchy takes into account the nesting depths of both modalities, while the $\mathrm{LTL}(\mathsf{U}^m)$ and $\mathrm{LTL}(\mathsf{X}^n)$ hierarchies 'count' just the nesting depth of U and X, respectively. Our work is motivated by basic questions about the presented hierarchies; in particular, the following problems seem to be among the most natural ones:

Question 1. Are those hierarchies semantically strict? That is, if we increase m or n just by one, do we always obtain a strictly more expressive fragment of LTL?

Question 2. If we take two classes A, B in the above hierarchies which are syntactically incomparable (for example, we can consider $\mathrm{LTL}(\mathsf{U}^4, \mathsf{X}^3)$ and $\mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^5)$, or $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X}^0)$ and $\mathrm{LTL}(\mathsf{U}^2)$), are they also semantically incomparable? That is, are there formulae $\varphi_A \in A$ and $\varphi_B \in B$ such that φ_A is not expressible in B and φ_B is not expressible in A?

Question 3. In the case of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchy, what is the semantical intersection of $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$? That is, what languages are expressible in both fragments?

We provide (positive) answers to Question 1 and Question 2. Here, the results about $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ hierarchy seem to be particularly interesting. As for Question 3, one is tempted to expect the following answer: The semantical intersection of $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ are exactly the languages expressible in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, where $m = \min\{m_1, m_2\}$ and $n = \min\{n_1, n_2\}$. Surprisingly, this answer turns out to be *incorrect*. For all $m \geq 1$, $n \geq 0$ we give an example of a language L which is definable both in $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$ and $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n+1})$, but not in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. It shows that the answer to Question 3 is not so easy as one might expect. In fact, Question 3 is left open as an interesting challenge directing our future work.

The results on Question 1 are closely related to the work of Etessami and Wilke [EW00] (see also [Wil99] for an overview of related results). They consider an until hierarchy of LTL formulae which is similar to our $\mathrm{LTL}(\mathsf{U}^m)$ hierarchy. The difference is that they treat the F operator 'explicitly', i.e., their U-depth counts just the nesting of the U-operator and ignores all occurrences of X and F (in our approach, $\mathsf{F}\varphi$ is just an abbreviation for $\mathtt{tt}\,\mathsf{U}\,\varphi$, and hence 'our' U-depth of $\mathsf{F}p$ is one and not zero). They prove the strictness of their until hierarchy in the following way: First, they design an appropriate Ehrenfeucht-Fraïssé game for LTL (the game is played on a pair of words) which in a sense characterizes those pairs of words which can be distinguished by an LTL formulae where the temporal operators are nested only to a certain depth. Then, for every k they construct a formula Fair_k with until depth k and prove that this particular formula cannot be equivalently expressed by any (other) formula with U-depth $k-1$ (here the previous results about the designed EF game are used). Since the formula Fair_k contains just one F operator (and many nested X and U operators), this proof carries over to our $\mathrm{LTL}(\mathsf{U}^m)$ hierarchy. In fact, [EW00] is in a sense 'stronger' result saying that one additional nesting level of U cannot be 'compensated' by arbitrarily-deep nesting of X and F. On the other hand, the proof does not allow to conclude that, e.g., $\mathrm{LTL}(\mathsf{U}^3, \mathsf{X}^0)$ contains a formula which is not expressible in $\mathrm{LTL}(\mathsf{U}^2)$ (because Fair_k contains the nested X modalities).

Our method for solving Questions 1 and 2 is different. Instead of designing appropriate Ehrenfeucht-Fraïssé games which could (possibly) characterize the membership to $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$, we formulate a general 'stuttering theorem' for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages. Roughly speaking, the theorem says that under cer-

tain 'local-periodicity' conditions (which depend on m and n) one can remove a given subword u from a given word w without influencing the (in)validity of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae (we say that u is (m,n)-*redundant* in w). This result can be seen as a generalization of the well-known form of stutter-invariance admitted by $\mathrm{LTL}(\mathsf{X}^0)$ formulae (a detailed discussion is postponed to Section 2). Thus, we obtain a simple (but surprisingly powerful) tool allowing to prove that a certain formula φ is *not* definable in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. The theorem is applied as follows: we choose a suitable alphabet Σ, consider the language L_φ, and find an appropriate $w \in L_\varphi$ and its subword u such that

- u is (m,n)-redundant in w;
- $w' \not\models \varphi$ where w' is obtained from w by deleting the subword u.

If we manage to do that, we can conclude that φ is not expressible in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$.

We use our stuttering theorem to answer Questions 1 and 2. Proofs are remarkably short (though it took us some time to find appropriate formulae which witness the presented claims). As another interesting corollary we obtain an alternative characterization of LTL languages which are exactly the regular languages closed under the generalized stutter equivalence of words. It is worth noting that some of the known results about LTL (like, e.g., the formula '$\mathsf{G}_2 p$' is not definable in LTL) admit a one-line proof if our general stuttering theorem is applied.

The paper is organized as follows. In Section 2 we formulate and prove the general stuttering theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages, together with some of its direct corollaries. In Section 3 we answer the Questions 1–3 in the above indicated way. In Section 4 we briefly discuss a potential applicability of our results to the problem of state-space explosion in the context of model-checking with LTL. Finally, in Section 5 we draw our conclusions and identify directions of future research.

2 A General Stuttering Theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$

In this section we formulate and prove the promised stuttering theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ languages. The definition is slightly technical and therefore we start with some intuition which aims to explain the underlying principles.

It is well-known that $\mathrm{LTL}(\mathsf{X}^0)$ formulae (i.e., formulae without the X operator) are *stutter invariant*. It means that one can safely delete *redundant letters* from words without influencing the (in)validity of $\mathrm{LTL}(\mathsf{X}^0)$ formulae (a letter $w(i)$ is redundant in w if $w(i) = w(i{+}1)$). Intuitively, it is not very surprising that this principle can be extended to $\mathrm{LTL}(\mathsf{X}^n)$ formulae (where $n \in \mathbb{N}_0$). We say that a letter $w(i)$ is n-*redundant* if $w(i) = w(i{+}j)$ for every $1 \leq j \leq n{+}1$. Now we could prove that $\mathrm{LTL}(\mathsf{X}^n)$ formulae are n-*stutter invariant* in the sense that deleting n-redundant letters from words does not influence (in)validity of $\mathrm{LTL}(\mathsf{X}^n)$ formulae (we do not provide an explicit proof here because this claim is

an immediate consequence of our general stuttering theorem; see also the 'pedagogical' remarks at the end of this section). Hence, LTL(X^n) languages are closed under deleting (as well as 'pumping') of n-redundant letters.

Since the notion of n-redundancy depends just on the X-depth of LTL formulae, one can also ask if there is another 'pumping principle' which depends mainly on the U-depth of LTL formulae; and indeed, there is one. In this case, we do not necessarily pump just individual letters, but whole subwords. To give some basic intuition, let us first consider the formula $\varphi \equiv (o \vee p) \mathsf{U}\, q$. Let $w \in \{o, p, q\}^*$ be a word such that $w \models \varphi$. We claim that if w is of the form $w = vuux$, where $v, u, x \in \Sigma^*$, then the word $w' = vux$ also satisfies φ. Our (general) arguments can be easier understood if they are traced down to the following example:

$$\begin{aligned} w &= \overbrace{ppp}^{v}\,\overbrace{oppqr}^{u}\,\overbrace{oppqr}^{u}\,\overbrace{orp}^{x} \\ w' &= \underbrace{ppp}_{v}\,\underbrace{oppqr}_{u}\,\underbrace{orp}_{x} \end{aligned}$$

Since $w \models \varphi$, there is w_i such that $w_i \models q$. Now we can distinguish three possibilities.

1. If $w(i)$ is within v, then deleting the first copy of u does not influence the validity of φ (in this case we could in fact delete the whole subword uux).
2. If $w(i)$ is within the second copy of u or within x, then the first copy of u can also be deleted without any problem.
3. If $w(i)$ is within the first copy of u then we can delete the *second* copy of u and the resulting word still satisfies φ.

The previous observation is actually valid for all LTL($\mathsf{U}^1, \mathsf{X}^0$) formulae. Moreover, one could prove (by induction on n) that for every $\varphi \in$ LTL($\mathsf{U}^n, \mathsf{X}^0$) and a word $w = vu^{n+1}x$ such that $w \models \varphi$ we have that $w' = vu^n x$ also models φ. However, we can do even better; there is one subtle point in the inductive argument which becomes apparent only when considering LTL($\mathsf{U}^n, \mathsf{X}^0$) formulae where $n \geq 2$. To illustrate this, let us take $\varphi \equiv (o \mathsf{U}\, p) \mathsf{U}\, (q \mathsf{U}\, r)$ and let w be a word of the form $w = vususux$ where $|s| = 1$. Hence, the subword us is repeated 'basically' twice after its first occurrence, but in the last copy we do not insist on the last letter (the missing 's'). We claim that if $w \models \varphi$, then also the word $w' = vusux$ models φ. Again, the reason can be well illustrated by an example:

$$\begin{aligned} w &= \overbrace{ppp}^{v}\,\overbrace{oppp}^{u}\,\overbrace{r}^{s}\,\overbrace{oppp}^{u}\,\overbrace{r}^{s}\,\overbrace{oppp}^{u}\,\overbrace{ooop}^{x} \\ w' &= \underbrace{ppp}_{v}\,\underbrace{oppp}_{u}\,\underbrace{r}_{s}\,\underbrace{oppp}_{u}\,\underbrace{ooop}_{x} \end{aligned}$$

Since $w \models \varphi$, there must be some w_i such that $w_i \models q \mathsf{U}\, r$. The most interesting situation is when $w(i)$ happens to be within the first copy of us. Actually, the 'worst' possibility is when $w(i)$ is the s (see the example above). As the U-depth of $q \mathsf{U}\, r$ is just one, we can rely on our previous observation; since $w_i = susux$,

we can surely remove the leading su subword. Thus, $sux \models q \mathsf{U} r$. In a similar way we can show that $ysux \models o \mathsf{U} p$ for each suffix y of vu (we know that $ysusux \models o \mathsf{U} p$ and hence we can again apply our previous observations). Now we can readily confirm that indeed $vusux \models \varphi$.

Increasing U-depth of $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ formulae allows to ignore more and more 'trailing letters'. More precisely, for any $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ formula φ we can 'ignore' the last $m-1$ letters in the repeated pattern.

Our general stuttering theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formulae combines both forms of stuttering (i.e., the 'letter stuttering' for the X operator, and the 'subword stuttering' for the U operator). In the next definition, the symbol u^ω (where $|u| \geq 1$) denotes the infinite word obtained by concatenating infinitely many copies of u.

Definition 1. *Let Σ be an alphabet and $w \in \Sigma^*$. A subword $w(i,j)$ is (m,n)-*redundant *in w iff the word $w(i+j, m \cdot j + n - m + 1)$ is a prefix of $w(i,j)^\omega$ (i.e., the subword $w(i,j)$ is repeated at least on the next $m \cdot j + n - m + 1$ letters).*

In the context of previous remarks, the above definition admits a good intuitive interpretation; the subword $w(i,j)$ has to be repeated 'basically' m times after its first occurrence (the $m \cdot j$ summand), but we can ignore the last $m-1$ letters. Since there can be n nested X operators, we must 'prolong' the repetition by n letters. Hence, the total number of letters by which we must prolong the repetition is $n - (m-1) = n - m + 1$. Before proving the stuttering theorem, we need to state one auxiliary lemma.

Lemma 1. *Let Σ be an alphabet, $m, n \in \mathbb{N}_0$, and $w \in \Sigma^*$. If a subword $w(i,j)$ is*

(i) (m,n)-redundant then it is also (m',n')-redundant for all $0 \leq n' \leq n$ and $0 \leq m' \leq m$.
(ii) $(m,n+1)$-redundant then the subword $w(i+1,j)$ is (m,n)-redundant.
(iii) $(m+1,n)$-redundant then the subword $w(i+k,j)$ is (m,n)-redundant for every $0 \leq k < j$.

Proof. (i) follows immediately as $j > 0$ implies $m' \cdot j + n' - m' + 1 \leq m \cdot j + n - m + 1$. (ii) is also simple—due to the $(m,n+1)$-redundancy of $w(i,j)$ we know that the subword is repeated at least on the next $m \cdot j + n - m + 2$ letters. Hence, the subword $w(i+1,j)$ is repeated at least on the next $m \cdot j + n - m + 1$ letters and thus it is (m,n)-redundant. A proof of (iii) is similar; if $w(i,j)$ is repeated on the next $(m+1) \cdot j + n - m$ letters, then the subword $w(i+k,j)$ (where $0 \leq k < j$) is repeated on the next $(m+1) \cdot j + n - m - k = m \cdot j + n - m + j - k$ letters, i.e., $w(i+k,j)$ is $(m, n+j-k-1)$-redundant. The (m,n)-redundancy of $w(i+k,j)$ follows from (i) and $k < j$. □

Definition 2. *Let Σ be an alphabet. For all $m, n \in \mathbb{N}_0$ we define the relation $\prec_{m,n} \subseteq \Sigma^* \times \Sigma^*$ as follows: $w \prec_{m,n} v$ iff v can be obtained from w be deleting some (m,n)-redundant subword. We say that $w, v \in \Sigma^*$ are (m,n)-stutter equivalent iff $w \approx_{m,n} v$, where $\approx_{m,n}$ is the least equivalence on Σ^* containing $\prec_{m,n}$. We say that a language $L \subseteq \Sigma^*$ is* (m,n)-stutter closed *if it is closed under $\approx_{m,n}$.*

Theorem 1 (stuttering theorem for $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$). *Let Σ be an alphabet, and let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ where $m, n \in \mathbb{N}_0$. The language L_φ is (m,n)-stutter closed.*

Proof. Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. It suffices to prove that for all $w, v \in \Sigma^*$ such that $w \prec_{m,n} v$ we have that $w \models \varphi \iff v \models \varphi$. We proceed by a simultaneous induction on m and n (we write $(m', n') < (m, n)$ iff $m' \leq m$ and $n' < n$, or $m' < m$ and $n' \leq n$).

Basic step: $m = 0$ and $n = 0$. Let $w, v \in \Sigma^*$ be words such that $w \prec_{0,0} v$. Let $w(i,j)$ be the $(0,0)$-redundant subword of w which has been deleted to obtain v. Since $\mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^0)$ formulae are just 'Boolean combinations' of letters and $\mathtt{tt}$, it suffices to show that $w(0) = v(0)$. If $i > 0$ then it is clearly the case. If $i = 0$, then $v(0) = w(j)$ and the $(0,0)$-redundancy of $w(0,j)$ implies that $w(j) = w(0)$.

Induction step: Let $m, n \in \mathbb{N}_0$, and let us assume (I.H.) that the theorem holds for all m', n' such that $(m', n') < (m, n)$. Let $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ and let $w, v \in \Sigma^*$ be words such that $w \prec_{m,n} v$. Let $w(i,j)$ be the (m,n)-redundant subword of w which has been deleted to obtain v. We distinguish four possibilities:

- $\varphi \in \mathrm{LTL}(\mathsf{U}^{m'}, \mathsf{X}^{n'})$ for some $(m', n') < (m, n)$. Since $w(i,j)$ is (m', n')-redundant by Lemma 1 (i), we can apply the induction hypothesis.
- $\varphi = \mathsf{X}\psi$. We need to prove that $w_1 \models \psi \iff v_1 \models \psi$. As ψ is an $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n-1})$ formula and $(m, n-1) < (m, n)$, the induction hypothesis implies that ψ cannot distinguish between words related by $\prec_{m,n-1}$. Hence, it suffices to show that $w_1 \prec_{m,n-1} v_1$. Let us consider the subword $w(i,j)$. If $i > 0$ then $w_1(i-1, j)$ is (m,n)-redundant and due to Lemma 1 (i) it is also $(m, n-1)$-redundant. Furthermore, v_1 can be obtained from w_1 by deleting the subword $w_1(i-1, j)$.
  If $i = 0$ then $w(0,j)$ is (m,n)-redundant. Lemma 1 (ii) implies that $w(1,j)$ is $(m, n-1)$-redundant. It means that the subword $w_1(0,j)$ is $(m, n-1)$-redundant. Furthermore, v_1 is obtained from w_1 by deleting $w_1(0,j)$.
- $\varphi = \psi \mathsf{U} \rho$. As the subformulae ψ, ρ belong to $\mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^n)$, they cannot (by induction hypotheses) distinguish between words related by $\prec_{m-1,n}$.
  Let $g : \{0, 1, \ldots, |w|-1\} \longrightarrow \{0, 1, \ldots, |v|-1\}$ be a function defined as follows.
  $$g(l) = \begin{cases} l, & l < i+j \\ l-j, & l \geq i+j \end{cases}$$
  We claim that for every $l < i+j$ we have that $w_l \prec_{m-1,n} v_{g(l)}$. Indeed, for $l < i$ it follows from Lemma 1 (i), and for $i \leq l < i+j$ it is due to Lemma 1 (iii). For every $l \geq i+j$ the words $v_{g(l)}$ and w_l are the same. Hence, for every $l < |w|$ we have that $w_l \models \psi \iff v_{g(l)} \models \psi$ and the same can be proven for ρ (in the same way).
  Now we show that if $w \models \psi \mathsf{U} \rho$ then also $v \models \psi \mathsf{U} \rho$. If $w \models \psi \mathsf{U} \rho$, there is $c \geq 0$ such that $w_c \models \rho$ and for every $d < c$ we have that $w_d \models \psi$.

Then $v_{g(c)} \models \rho$ (see above) and from the definition of g it follows that for every $d' < g(c)$ there is $d < c$ such that $g(d) = d'$. Thus, for every $d' < g(c)$ we have that $v_{g(d)} = v_{d'} \models \psi$. To sum up, we obtain that $v \models \psi \mathsf{U} \rho$.
Similarly, we also show that if $v \models \psi \mathsf{U} \rho$ then $w \models \psi \mathsf{U} \rho$. If $v \models \psi \mathsf{U} \rho$, there is $c \geq 0$ such that $v_c \models \rho$ and for every $d < c$ we have that $v_d \models \psi$. Let c' be the least number satisfying $g(c') = c$ (there is such a c' as the function g is surjective). Then $w_{c'} \models \rho$ (see above). From the definition of g we get that for every $d' < c'$ it holds that $g(d') < g(c') = c$ (otherwise we would obtain a contradiction with our choice of c'). Thus, $w_{d'} \models \psi$ and hence $w \models \psi \mathsf{U} \rho$.

– φ is a 'Boolean combination' of formulae of the previous cases. Formally, this case is handled by an 'embedded' induction on the structure of φ. The basic step (when φ is *not* of the form $\neg\psi$ or $\psi_1 \wedge \psi_2$) is covered by the previous cases. The induction step ($\varphi \equiv \neg\psi$ or $\varphi \equiv \psi_1 \wedge \psi_2$ where we assume that our theorem holds for ψ, ψ_1, ψ_2) follows immediately. □

A natural question suggested by Theorem 1 is whether it also holds vice versa, i.e., if every regular (m, n)-stutter closed language is definable by an $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ formula. We must reject this hypotheses, as shown by the following counterexample:

Example 1. Let $\Sigma = \{p, q, r\}$ and $\varphi = \neg r \wedge ((p \mathsf{U} q) \mathsf{U} r)$. It is easy to see that the language $L_\varphi = \{p, q\}^* qr\Sigma^*$ is $(1, 0)$-stutter closed. We prove that L_φ is not definable in $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$. To do that, it suffices to show that for any $\psi \in \mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ we have that $pqr \models \psi \iff pqpr \models \psi$ (observe that $pqr \in L_\varphi$ and $pqpr \notin L_\varphi$). There are three cases.

– $\psi \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^0)$. Since the validity of $\psi \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^0)$ formulae depends only on the first letter of a given word, we are done.
– $\psi = \psi_1 \mathsf{U} \psi_2$, where $\psi_1, \psi_2 \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^0)$. Let $S_1 = \{o \in \Sigma \mid o \models \psi_1\}$ and $S_2 = \{o \in \Sigma \mid o \models \psi_2\}$. For every $w \in \Sigma^*$ we have that $w \models \psi_1 \mathsf{U} \psi_2$ iff there exists $j \in \mathbb{N}_0$ such that $w(j) \in S_2$ and for all $0 \leq i < j$ we have that $w(i) \in S_1$. One can easily check that the words pqr, $pqpr$ cannot be distinguished by the just specified condition for any $S_1, S_2 \subseteq \Sigma$.
– ψ is a 'Boolean combination' of formulae of the previous cases. Here we argue by a (straightforward) structural induction. □

Nevertheless, we can easily prove the following alternative characterization of LTL languages:

Corollary 1. *A regular language L over Σ is definable in LTL iff L is (m, n)-stutter closed for some $m, n \in \mathbb{N}_0$.*

Proof. The '$\Longrightarrow$' direction follows immediately from Theorem 1. The other direction is a simple consequence of the fact that a regular language L is expressible in LTL iff the minimal deterministic automaton recognizing L is counter-free [Kam68, MP71]—one can easily argue that if the minimal deterministic

automaton for L is *not* counter-free, then L cannot be (m,n)-stutter closed for any $m,n \in \mathbb{N}_0$. (Indeed, if the minimal automaton is not counter-free then there are $u,w,v \in \Sigma^*$, $|w| \geq 1$, and $k \geq 2$ such that for each $i \in \mathbb{N}_0$ we have that $uw^{k \cdot i}v \in L$ and $uw^{k \cdot i-1}v \notin L$. Now suppose that L is (m,n)-stutter closed for some $m,n \in \mathbb{N}_0$ and let $d = (m+n+2)$. Since the first copy of w in $uw^{k \cdot d}v \in L$ is (m,n)-redundant, we have a contradiction with $uw^{k \cdot d-1}v \notin L$.) □

In the context of Corollary 1, the evidence provided by Example 1 cannot be seen as fully satisfactory—since every regular language L which is (m,n)-stutter closed for some $m,n \in \mathbb{N}_0$ is definable in LTL, there surely exist $m',n' \in \mathbb{N}_0$ such that L is definable in $\mathrm{LTL}(\mathsf{U}^{m'},\mathsf{X}^{n'})$. Due to Example 1 we know that the relationship among m,n and m',n' is not purely $m'=m$ and $n'=n$. Maybe the actual relationship is just slightly more complicated; and maybe there is no direct connection at all. Example 1 does not contradict any of the two hypothesis.

It is worth noting that even special cases of Theorem 1 can bring interesting consequences. For example, we already mentioned the well-known form of stuttering admitted by $\mathrm{LTL}(\mathsf{X}^0)$ formulae which can be generalized to $\mathrm{LTL}(\mathsf{X}^n)$ formulae. Formally, for each $w \in \Sigma^*$ and $n \in \mathbb{N}_0$ we define the *n-canonical form* of w, denoted $[w]_n$, which is the word obtained from w by deleting all n-redundant letters (see above). Two words $w,v \in \Sigma^*$ are *n-stutter equivalent* iff $[w]_n = [v]_n$. Observe that 0-stutter equivalence is exactly the well-known stutter equivalence of $\mathrm{LTL}(\mathsf{X}^0)$, and that n-stutter equivalence is subsumed by $\approx_{m,n}$ for each $m \in \mathbb{N}_0$ (indeed, it suffices to realize that $w(i,1)$ is an (m,n)-redundant subword iff $w(i)$ is an n-redundant letter). Hence, a direct corollary to Theorem 1 is

Corollary 2. *Let $\varphi \in \mathrm{LTL}(\mathsf{X}^n)$ where $n \in \mathbb{N}_0$. The language L_φ is closed under n-stutter equivalence.*

Of course, a direct proof of this corollary is a bit simpler than the proof of Theorem 1. However, it already brings interesting consequences.

Corollary 3. *The property $\mathsf{G}_2 p$ (which says 'at every even position is p') is not expressible in LTL.*

Proof. Suppose the converse. Let $\Sigma = \{p,q\}$. As $\mathsf{G}_2 p$ is expressible in LTL, there is $n \in \mathbb{N}_0$ and a formula $\varphi \in \mathrm{LTL}(\mathsf{X}^n)$ which is equivalent to $\mathsf{G}_2 p$. Since L_φ contains the word $p^{2n+2}q$ and the first occurrence of p is n-redundant, we obtain $p^{2n+1}q \in L_\varphi$ which is a contradiction. □

Another application for Corollary 2 will be given in Section 3. So, a direct proof of Corollary 2 might be of some use even in a basic course on LTL, because it is not much longer than a proof for 0-stutter equivalence (which is often included) and it brings interesting consequences 'for free'.

3 Answers for Questions 1, 2, and 3

Now we are ready to provide answers to Questions 1, 2, and 3 which were stated in Section 1 (though the Question 3 will be left open in fact). We start with a simple observation.

Lemma 2. *For each $n \geq 1$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^0, \mathsf{X}^n)$ which cannot be expressed in $\mathrm{LTL}(\mathsf{X}^{n-1})$.*

Proof. Let $\Sigma = \{p\}$ and $n \geq 1$. Consider the formula $\varphi \equiv \overbrace{\mathsf{XX}\cdots\mathsf{X}}^{n} p$. We show that L_φ is not closed under $(n{-}1)$-stutter equivalence (which suffices due to Corollary 2). It is easy; realize that $p^{n+1} \in L_\varphi$ and the first occurrence of p in this word is $(n{-}1)$-redundant. Since $p^n \notin L_\varphi$, we are done. □

A 'dual' fact is proven below (it is already non-trivial).

Lemma 3. *For each $m \geq 1$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ which cannot be expressed in $\mathrm{LTL}(\mathsf{U}^{m-1})$.*

Proof. Let $m \geq 1$ and let $\Sigma = \{q, p_1, \ldots, p_m\}$. We define a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^0)$ as follows:

$$\varphi = \mathsf{F}(p_1 \wedge \mathsf{F}(p_2 \wedge \ldots \wedge \mathsf{F}(p_{m-1} \wedge \mathsf{F}p_m) \ldots))$$

Let us fix an arbitrary $n \in \mathbb{N}_0$, and define a word $w \in \Sigma^*$ by

$$w = (q^{n+1}\, p_m\, p_{m-1} \cdots p_1)^m\, q^{n+1}$$

Clearly $w \models \varphi$ and the subword $w(0, n{+}1{+}m)$ is $(m{-}1, n)$-redundant. As the word w' obtained from w by removing $w(0, n{+}1{+}m)$ does not model φ, the language L_φ is not $(m{-}1, n)$-stutter closed. As it holds for every $n \in \mathbb{N}_0$, the formula φ is not expressible in $\mathrm{LTL}(\mathsf{U}^{m-1})$. □

The last technical lemma which is needed to formulate answers to Questions 1 and 2 follows.

Lemma 4. *For all $m, n \in \mathbb{N}_0$ there is a formula $\varphi \in \mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ which is expressible neither in $\mathrm{LTL}(\mathsf{U}^{m-1}, \mathsf{X}^n)$ (assuming $m \geq 1$), nor in $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^{n-1})$ (assuming $n \geq 1$).*

Proof. If $m = 0$ or $n = 0$, we can apply Lemma 2 or Lemma 3, respectively. Now let $m, n \geq 1$, and let $\Sigma = \{p_1, \ldots, p_k\}$ where $k = \max\{m, n{+}1\}$. We define formulae ψ and φ as follows:

$$\psi = \begin{cases} p_m \wedge \mathsf{X}^n p_{m-n} & \text{if } m > n \\ p_m \wedge \mathsf{X}^n p_{m+1} & \text{if } m \leq n \end{cases}$$

$$\varphi = \begin{cases} \mathsf{F}\psi & \text{if } m = 1 \\ \mathsf{F}(p_1 \wedge \mathsf{F}(p_2 \wedge \mathsf{F}(p_3 \wedge \ldots \wedge \mathsf{F}(p_{m-1} \wedge \mathsf{F}\psi) \ldots))) & \text{if } m > 1 \end{cases}$$

where X^l abbreviates $\overbrace{\mathsf{X}\mathsf{X}\ldots\mathsf{X}}^{l}$. The formula φ belongs to $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$. Let us consider the word w defined by

$$w = \begin{cases} (p_m\, p_{m-1} \ldots p_1)^m p_m\, p_{m-1} \ldots p_{m-n+1} & \text{if } m > n \\ (p_{n+1}\, p_n \ldots p_1)^{m+1} & \text{if } m = n \\ (p_{n+1}\, p_n \ldots p_1)^{m+1} p_{n+1}\, p_n \ldots p_{m+2} & \text{if } m < n \end{cases}$$

It is easy to check that $w \in L_\varphi$ and the subword $w(0,k)$ (where $k = \max\{m, n{+}1\}$) is $(m, n{-}1)$-redundant as well as $(m{-}1, n)$-redundant. As the word w' obtained from w by removing $w(0,k)$ does not satisfy φ, the language L_φ is neither $(m, n{-}1)$-stutter closed, nor $(m{-}1, n)$-stutter closed. □

The knowledge presented in the three lemmata above allows to conclude the following:

Corollary 4 (Answer to Question 1). *The* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$*,* $\mathrm{LTL}(\mathsf{U}^m)$*, and* $\mathrm{LTL}(\mathsf{X}^n)$ *hierarchies are strict.*

Corollary 5 (Answer to Question 2). *Let A and B be classes of* $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$*,* $\mathrm{LTL}(\mathsf{U}^m)$*, or* $\mathrm{LTL}(\mathsf{X}^n)$ *hierarchy (not necessarily of the same one) such that A is syntactically not included in B. Then there is a formula $\varphi \in A$ which cannot be expressed in B.*

Although we cannot provide a full answer to Question 3, we can at least reject the aforementioned 'natural' hypotheses (see Section 1).

Lemma 5 (About Question 3). *For all $m, n \in \mathbb{N}_0$ there is a language definable in* $\mathrm{LTL}(\mathsf{U}^{m+2}, \mathsf{X}^n)$ *as well as in* $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^{n+1})$ *which is not definable in* $\mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^n)$*.*

Proof. We start with the case when $m = n = 0$. Let $\Sigma = \{p, q\}$, and let $\psi_1 = \mathsf{F}(q \wedge (q\,\mathsf{U}\,\neg q))$ and $\psi_2 = \mathsf{F}(q \wedge \mathsf{X}\neg q)$. Note that $\psi_1 \in \mathrm{LTL}(\mathsf{U}^2, \mathsf{X}^0)$ and $\psi_2 \in \mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^1)$. Moreover, ψ_1 and ψ_2 are equivalent as they define the same language $L = \Sigma^* q (\Sigma \smallsetminus \{q\}) \Sigma^*$. This language is not definable in $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ as it is not $(1,0)$-stutter closed; for example, the word $w = pqpq \in L$ contains a $(1,0)$-redundant subword $w(0,2)$ but $w_2 = pq \notin L$.

The above example can be generalized to arbitrary m, n (using the designed formulae ψ_1, ψ_2). For given m, n we define formulae $\varphi_1 \in \mathrm{LTL}(\mathsf{U}^{m+2}, \mathsf{X}^n)$ and $\varphi_2 \in \mathrm{LTL}(\mathsf{U}^{m+1}, \mathsf{X}^{n+1})$, both defining the same language L over $\Sigma = \{q, p_1, \ldots, p_{m+1}\}$, and we give an example of a word $w \in L$ with an $(m+1, n)$-redundant subword such that w without this subword is not from L. We distinguish three cases.

- $m = n > 0$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{X}\mathsf{F}(p \wedge \mathsf{X}\mathsf{F}(p \wedge \mathsf{X}\mathsf{F}(p \wedge \ldots \wedge \mathsf{X}\mathsf{F}(p \wedge \psi_i) \ldots)))}^{m\text{-times}}$$

  The word $w = (pq)^{m+2} \in L$, $w(0,2)$ is $(m+1, n)$-redundant, and $w_2 = (pq)^{m+1} \notin L$.

– $m > n$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{XF}(q \wedge \mathsf{XF}(q \wedge \ldots \wedge \mathsf{XF}(q \wedge}^{n\text{-times}} \overbrace{\mathsf{F}(p_1 \wedge \mathsf{F}(p_2 \wedge \ldots \wedge \mathsf{F}(p_{m-n} \wedge}^{(m-n)\text{-times}} \psi_i) \ldots))) \ldots))$$

The word $w = (qp_{m-n}p_{m-n-1} \ldots p_1)^{m+1}q \in L$, $w(0, m-n+1)$ is $(m+1, n)$-redundant, and $w_{m-n+1} \notin L$.

– $m < n$. For $i \in \{1, 2\}$ we define

$$\varphi_i = \overbrace{\mathsf{F}(p_1 \wedge \mathsf{F}(p_2 \wedge \ldots \wedge \mathsf{F}(p_m \wedge}^{m\text{-times}} \overbrace{\mathsf{XX} \ldots \mathsf{X}}^{n} \psi_i) \ldots))$$

The word $w = (q^{n-m}p_{m+1}p_m \ldots p_1)^{m+2}q^{n-m} \in L$, $w(0, n+1)$ is $(m+1, n)$-redundant, and $w_{n+1} \notin L$. □

In fact, the previous lemma says that if we take two classes $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ which are syntactically incomparable and where $m_1, m_2 \geq 1$, then their semantical intersection is strictly greater than $\mathrm{LTL}(\mathsf{U}^m, \mathsf{X}^n)$ where $m = \min\{m_1, m_2\}$ and $n = \min\{n_1, n_2\}$. Moreover, it also says that if we try to minimize the nesting depths of X and U in a given formula φ (preserving the meaning of φ), there is generally no 'best' way how to do that.

4 A Note on Model-Checking with LTL

The aim of this section is to identify another (potential) application of Theorem 1 in the area of model-checking with the logic LTL. We show that the theorem can be used as a 'theoretical basis' for advanced state-space reduction techniques which might further improve the efficiency of LTL model-checking algorithms. The actual development of such techniques is a complicated problem beyond the scope of this paper; nevertheless, we can explain the basic principle, demonstate its potential power, and explicitly discuss the missing parts which must be completed to obtain a working implementation. The chosen level of presentation is semi-formal, and the content is primarily directed to a 'practically-oriented' reader.

The model-checking approach to formal verification (with the logic LTL) works according to the following abstract scheme:

– The verified system is formally described in a suitable modeling language whose underlying semantics associates a well-defined Kripke structure to the constructed model.
– Desired properties of the system are defined as a formula in the logic LTL. More precisely, one defines the properties which should be satisfied by all possible *runs* of the system, which formally correspond to certain maximal paths in the associated Kripke structure.
– It is shown that all runs satisfy the constructed LTL formula.

A principal difficulty is that the size of the associated Kripke structure is usually very large (this is known as the problem of state-space explosion). There are various strategies how to deal with this problem. For example, one can reduce the number of states by abstracting the code and/or the data of the system, use various 'compositional' techniques, or use restricted formalisms (like, e.g., pushdown automata) which allow for a kind of 'symbolic' model-checking where the explicit construction of the associated Kripke structure is not required. One of the most successful methods is *partial order reduction* (see, e.g., [CGP99]) which works for the LTL(X^0) fragment of LTL. It has been argued by Lamport [Lam83] that LTL(X^0) provides a sufficient expressive power for specifying correctness properties of software systems; one should avoid the use of the X operator because it imposes very strict requirements on 'scheduling' of transitions between states which can be hard to implement. Partial order reduction conveniently uses the *stutter invariance* of LTL(X^0) formulae in the sense of Corollary 2. Roughly speaking, the idea is as follows: if we are to decide the validity of a given LTL(X^0) formula for a given Kripke structure, we do not necessarily need to examine *all* runs; we can safely ignore those runs which are 0-stutter equivalent to already checked ones. To see how it works in practice, consider the following parallel programme consisting of two threads `A` and `B`.

```
x = 0;          procedure A()              procedure B()
cobegin           begin                      begin
  A; B;             for i=1 to 5 do            z = 2;
coend                 begin                    x = x + 7;
                        x = x + 1;             z = 2 * z;
                        x = x - 1;             z = z - 1;
                      end                    end
                  end
```

The underlying Kripke structure (see Fig. 1) models all possible interleavings between `A` and `B`. The states carry the information about variables and about the position of control in the two threads. The transitions correspond to individual instructions. In Fig. 1, we explicitly indicated the value of `x` in each state; the $\swarrow$ direction corresponds to instructions of `A`, and the $\searrow$ direction corresponds to instructions of `B`. Now imagine that we want to verify that `x` is always strictly less than 8 at every run (which is not true). It can be formally expressed by a formula $\mathsf{G}(\mathtt{x} < 8)$ where the predicate $\mathtt{x} < 8$ should be seen as a *letter* (in the sense of LTL semantics given in Section 1). Hence, to every run we can associate a word over the alphabet $\{\mathtt{x} < 8, \neg(\mathtt{x} < 8)\}$ and interpret our formula in the standard way. Since the values of all variables except for `x` are irrelevant, the instructions which do not modify the value of `x` always generate 0-redundant letters (while, for example, the instruction `x = x + 1` sometimes generates a redundant letter and sometimes not). Hence, many of the runs in Fig. 1 are in fact 0-stutter equivalent and hence one can safely 'ignore' many of them. Technically, a set of runs can be ignored by ignoring certain out-going transitions in certain states; and since we ignore some transitions, it can also happen that some states are not

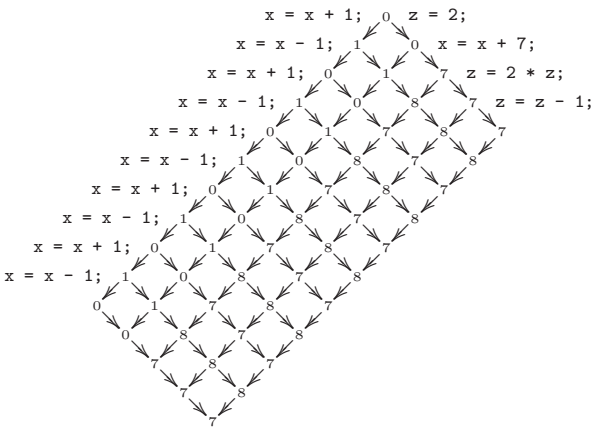


Fig. 1. The associated Kripke structure

visited at all—and thus we could in principle avoid their construction, keeping the Kripke structure smaller. The question is how to recognize those superfluous transitions and states. It does not make much sense to construct the whole Kripke structure and then try to reduce it; what we need is a method which can be applied *on-the-fly* while constructing the Kripke structure. *Partial-order reduction* (as described in [CGP99]) can do the job fairly well—if we apply it to the structure of Fig. 1 and the formula $\mathsf{G}(\mathtt{x} < 8)$, we obtain a 'pruned' structure of Fig. 2 (left)[1]. Now we come to the actual point of this section—since $\mathsf{G}(\mathtt{x} < 8)$ is an $\mathrm{LTL}(\mathsf{U}^1, \mathsf{X}^0)$ formula, we can also apply the principle of $(1, 0)$-stuttering which allows to 'ignore' even more runs in the Kripke structure of Fig. 1 (many of them are $(1, 0)$-stutter equivalent). One of possible results is shown in Fig. 2 (right)[2]; it clearly demonstrates the potential power of the new method. However, it is not clear if the method admits an on-the-fly implementation, which means that we cannot fully advocate its practical usability at the moment. This question is left open as another challenge.

To sum up, we believe that (m, n)-stuttering might be (potentially) used as the underlying principle for optimized model-checking in a similar fashion as 0-stuttering was used in the case of partial-order reduction. However, it can only be proven by designing a working and efficient on-the-fly reduction method, which is a non-trivial research problem on its own.

[1] The instructions which modify the variable `x` are treated as 'dangerous', i.e., as if they never produced redundant letters.

[2] To give a 'fair' comparison with partial-order reduction, the instructions which modify the variable `x` are again treated as 'dangerous'.

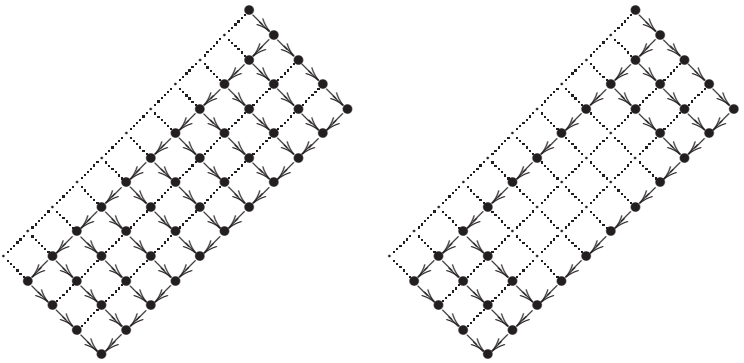


Fig. 2. The reduced Kripke structures

5 Conclusions

The main technical contribution of this paper is the general stuttering theorem presented in Section 2. With its help we were able to construct (short) proofs of other results. In particular, we gave an alternative characterization of LTL languages (which are exactly regular (m, n)-stutter closed languages), proved the strictness of the three hierarchies of LTL formulae introduced in Section 1, and we also showed several related facts about the relationship among the classes in the three hierarchies.

Some problems are left open. For example, the exact characterization of the semantical intersection of $\mathrm{LTL}(\mathsf{U}^{m_1}, \mathsf{X}^{n_1})$ and $\mathrm{LTL}(\mathsf{U}^{m_2}, \mathsf{X}^{n_2})$ classes (in the case when they are syntactically incomparable) surely deserves further attention. Moreover, we would be also interested if the potential applicability of Theorem 1 to model-checking (as indicated in Section 4) can really result in a practically usable state-space reduction method.

References

[CGP99] E. M. Clark, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999. 288, 289

[EW00] K. Etessami and T. Wilke. An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic. *Information and Computation*, 160:88–108, 2000. 278

[Kam68] H. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968. 277, 283

[Lam83] L. Lamport. What good is temporal logic? In *Proceedings of IFIP Congress on Information Processing*, pages 657–667, 1983. 288

[MP71] R. McNaughton and S. Papert. *Counter-Free Automata*. The MIT Press, 1971. 277, 283

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE Computer Society Press, 1977. 276

[Tho91] W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, B:135–192, 1991. 277

[Wil99] T. Wilke. Classifying discrete temporal properties. In *Proceedings of STACS'99*, volume 1563 of *LNCS*, pages 32–46. Springer, 1999. 278

Trading Probability for Fairness*

Marcin Jurdziński[1], Orna Kupferman[2], and Thomas A. Henzinger[1]

[1] EECS, University of California, Berkeley
[2] School of Computer Science and Engineering, Hebrew University

Abstract. Behavioral properties of open systems can be formalized as objectives in two-player games. Turn-based games model asynchronous interaction between the players (the system and its environment) by interleaving their moves. Concurrent games model synchronous interaction: the players always move simultaneously. Infinitary winning criteria are considered: Büchi, co-Büchi, and more general parity conditions. A generalization of determinacy for parity games to concurrent parity games demands probabilistic (mixed) strategies: either player 1 has a mixed strategy to win with probability 1 (almost-sure winning), or player 2 has a mixed strategy to win with positive probability.
This work provides efficient reductions of concurrent probabilistic Büchi and co-Büchi games to turn-based games with Büchi condition and parity winning condition with three priorities, respectively. From a theoretical point of view, the latter reduction shows that one can trade the probabilistic nature of almost-sure winning for a more general parity (fairness) condition. The reductions improve understanding of concurrent games and provide an alternative simple proof of determinacy of concurrent Büchi and co-Büchi games. From a practical point of view, the reductions turn solvers of turn-based games into solvers of concurrent probabilistic games. Thus improvements in the well-studied algorithms for the former carry over immediately to the latter. In particular, a recent improvement in the complexity of solving turn-based parity games yields an improvement in time complexity of solving concurrent probabilistic co-Büchi games from cubic to quadratic.

1 Introduction

In *formal verification*, a *closed system* is a system whose behavior is completely determined by the state of the system, while an *open system* is a system that interacts with its environment and whose behavior depends on this interaction [11]. While formal verification of closed systems uses models based on labeled transition systems, formal analysis of open systems, and the related problems of control and synthesis, use models based on *two-player games*, where one player represents the system, and the other player represents the environment [18, 19, 1, 7, 13, 14]. At each round of the game, player 1 (the system) and

* This research was supported in part by the Polish KBN grant 7-T11C-027-20, the AFOSR MURI grant F49620-00-1-0327, and the NSF Theory grant CCR-9988172.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 292–305, 2002.
© Springer-Verlag Berlin Heidelberg 2002

player 2 (the environment) choose moves, and the choices determine the next state of the game. Specifications of open systems can be expressed as objectives in such games, and deciding whether an open system satisfies a specification is reduced to deciding whether player 1 has a winning strategy in the game. The construction of winning strategies can also be used to *synthesize* correct systems and controllers from their specifications [18, 19].

In practice, games with finitary winning conditions, such as reachability and safety games play a prominent role. Games with infinitary winning conditions, such as Büchi, co-Büchi, and general parity games [21] are richer from the theoretical point of view. Apart from being a versatile tool in the theory of formal verification [21] they can also be used in practice to model liveness and fairness specifications [15].

While *turn-based* games have been heavily studied [15, 21, 12], *concurrent* games have been considered only recently [5, 4]. Modelling based on turn-based games assumes that interaction between the system and the environment is asynchronous and actions of the two players can be interleaved. Concurrent games are better suited for modeling synchronous interaction [2, 3]. In every round of a concurrent game the two players choose moves simultaneously and independently, and the pair of choices of both players determines the next state of the game. If a system exhibits a mix of synchronous and asynchronous interaction which depends on some external factors, one can attempt to reconcile the two by allowing in the game probabilistic moves assigning appropriate probabilities to each option.

Solving concurrent games requires new concepts and techniques when compared to turn-based games. For example, determinacy of turn-based parity games does not easily carry over to concurrent games. While deterministic and memoryless (pure) strategies suffice for turn-based games [9, 21, 23], probabilistic (mixed) strategies with possibly infinite memory are necessary for winning concurrent games [5, 4].

Theorem 1. [4] *In a concurrent parity game, in every vertex, either player 1 has a mixed strategy to win with probability 1, or player 2 has a mixed strategy to win with positive probability.*

We encourage the reader to refer to the papers by de Alfaro at al. [5] and de Alfaro and Henzinger [4] for small and lucid examples of games which exhibit some of the conceptual hurdles needed to be overcome in order to solve concurrent reachability, Büchi, and co-Büchi games.

This paper offers an alternative way to solve concurrent Büchi and co-Büchi games, by providing an efficient reduction of concurrent games to turn-based games. Specifically, we prove the following.

Theorem 2. *There are linear-time reductions from concurrent Büchi games to turn-based Büchi games, and from concurrent co-Büchi games to Parity(0,2) games.*[1]

From the theoretical point of view, interesting by-products of our proofs of the above fact are conceptually simple proofs of determinacy for concurrent Büchi and co-Büchi games that invoke the classical determinacy theorem for turn-based parity games [9, 21, 23]. On the practical side, our reductions turn solvers of non-probabilistic turn-based parity games into solvers of probabilistic concurrent games. Thus, improvements in the well-studied algorithms for the former [10, 15, 8, 20, 12, 22] will immediately carry over to the latter. In particular, a recent result [12] improving the complexity of parity games, together with our latter translation yields an improvement in the complexity of solving concurrent co-Büchi games from cubic [4] to quadratic.

A key novel technical concept behind the correctness proofs of our reductions is that of *witness functions* for concurrent Büchi and co-Büchi games, generalizing signature assignments [9, 23] and progress measures [12] from turn-based games to concurrent games. Witness functions label the states of a concurrent game with (tuples of) numbers so that certain local conditions on edges of the game graph are satisfied. A technical advantage of witness functions is that it suffices to check the local conditions on a set of vertices in order to conclude that the respective player has a winning strategy in an infinite game from every state in the set. As in the article of de Alfaro and Henzinger [4] the local conditions are expressed in terms of probability distributions of moves (mixed moves) each player can take from a vertex. For our reductions from concurrent to turn-based games we establish "finitary" characterizations of those conditions in terms of pure moves. Then we show that these finitary characterizations can be modeled by small sub-games in which the two players follow a certain "protocol" of choosing pure moves.

Due to lack of space, this extended abstract omits many proofs, some key technical auxiliary results, and generalizations of the main results to limit-sure winning and general parity winning conditions. A full version of this paper will deal with those issues in more detail.

2 Concurrent Probabilistic Games

For a finite set X, a *probability distribution* on X is a function $\xi : X \to [0,1]$ such that $\sum_{x \in X} \xi(x) = 1$. We denote the set of probability distributions on X by $\mathcal{D}(X)$. For a probability distribution $\xi \in \mathcal{D}(X)$ we define $||\xi||$, the *support* of ξ, by $||\xi|| = \{ x \in X \,:\, \xi(x) > 0 \}$.

A two-player *concurrent probabilistic game structure* $G = (V, A, A_1, A_2, \delta)$ consists of the following components.

[1] A Parity(0,2) winning condition consists of a partition of the state space into three sets P_0, P_1, and P_2, and the objective of player 1 is either to visit P_0 infinitely often, or visit P_2 infinitely often and P_1 only finitely often.

- A finite set V of vertices, and a finite set of actions A.
- Functions $A_1, A_2 : V \to 2^A$, such that for every vertex v, $A_1(v)$ and $A_2(v)$ are non-empty sets of actions available in vertex v to players 1 and 2, respectively.
- A probabilistic transition function $\delta : V \times A \times A \to \mathcal{D}(V)$, such that for every vertex v and actions $a \in A_1(v)$ and $b \in A_2(v)$, $\delta(v, a, b)$ is a probability distribution on the successor vertices.

At each step of the game, both players choose *moves* to proceed with. We consider two options here.

- *Pure action moves.* The set of moves is the set of actions $M = A$. The sets of moves available to players 1 and 2 in vertex v are $M_1(v) = A_1(v)$ and $M_2(v) = A_2(v)$, respectively.
- *Mixed (randomized) action moves.* The set of moves is the set of probability distributions on the set of actions $M = \mathcal{D}(A)$. The sets of moves available to players 1 and 2 in vertex v are $M_1(v) = \mathcal{D}\big(A_1(v)\big)$ and $M_2(v) = \mathcal{D}\big(A_2(v)\big)$, respectively. In this case we extend the transition function to $\delta : V \times M \times M \to \mathcal{D}(V)$, by $\delta(v, \alpha, \beta)(w) = \sum_{a \in A_1(v)} \sum_{b \in A_2(v)} \alpha(a) \cdot \beta(b) \cdot \delta(v, a, b)$.

We often write $\Pr_v^{\alpha,\beta}[w]$ for $\delta(v, \alpha, \beta)(w)$, and for a set $W \subseteq V$, we define $\Pr_v^{\alpha,\beta}[W] = \sum_{w \in W} \Pr_v^{\alpha,\beta}[w]$.

Thus, $\Pr_v^{\alpha,\beta}[w]$ is the probability that the successor vertex is w, given that the current vertex is v and the players chose to proceed with α and β. Similarly, $\Pr_v^{\alpha,\beta}[W]$ is the probability that the successor vertex is a member of W.

A concurrent probabilistic game is played in the following way. If v is the current vertex in a play then player 1 chooses a move $\alpha \in M_1(v)$, and simultaneously and independently player 2 chooses a move $\beta \in M_2(v)$. Then the play proceeds to a successor vertex w with probability $\Pr_v^{\alpha,\beta}[w]$.

A *path* in G is an infinite sequence $v_0, v_1, v_2, \ldots$ of vertices, such that for all $k \geq 0$, there are moves $\alpha \in M_1(v_k)$ and $\beta \in M_2(v_k)$, such that $\Pr_{v_k}^{\alpha,\beta}[v_{k+1}] > 0$. We denote by Ω the set of all paths.

We say that a concurrent game structure $G = (V, A, A_1, A_2, \delta)$ is:

- *Turn-based*, if for all $v \in V$, we have either $|A_1(v)| = 1$ or $|A_2(v)| = 1$; i.e., in every vertex only one player may have a non-trivial choice;
- *Deterministic*, if for all $v \in V$, $a \in A_1(v)$, and $b \in A_2(v)$, we have $||\delta(v, a, b)|| = 1$; i.e., in every move the next vertex is uniquely determined by the pure action moves chosen by the players. In this case we often write $\delta(v, a, b)$ for the unique $w \in V$, such that $\delta(v, a, b)(w) = 1$.

Strategies. A *strategy* for player 1 is a function $\pi_1 : V^+ \to M$, such that for a finite sequence $\overline{v} \in V^+$ of vertices, representing the history of the play so far, $\pi_1(\overline{v})$ is the next move to be chosen by player 1. A strategy must prescribe only available moves, i.e., $\pi_1(\overline{w} \cdot v) \in M_1(v)$, for all $\overline{w} \in V^*$, and $v \in V$. Strategies for player 2 are defined analogously. We write Π_1 and Π_2 for the sets of all strategies for players 1 and 2, respectively.

For an initial vertex v, and strategies $\pi_1 \in \Pi_1$ and $\pi_2 \in \Pi_2$, we define $\mathit{Outcome}(v, \pi_1, \pi_2) \subseteq \Omega$ to be the set of paths that can be followed when a play starts from vertex v and the players use the strategies π_1 and π_2. Formally, $v_0, v_1, v_2, \ldots \in \mathit{Outcome}(v, \pi_1, \pi_2)$ if $v_0 = v$, and for all $k \geq 0$, we have that $\delta(v_k, \alpha_k, \beta_k)(v_{k+1}) > 0$, where $\alpha_k = \pi_1(v_0, \ldots, v_k)$ and $\beta_k = \pi_2(v_0, \ldots, v_k)$.

Once a starting vertex v and strategies π_1 and π_2 for the two players have been chosen, the probabilities of events are uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of paths. For a vertex v, and an event $\mathcal{A} \subseteq \Omega$, we write $\Pr_v^{\pi_1,\pi_2}(\mathcal{A})$ for the probability that a path belongs to $\mathcal{A}$ when the game starts from v, and the players use the strategies π_1 and π_2.

Winning criteria. A *game* $\mathcal{G} = (G, \mathcal{W})$ consists of a game structure G and a *winning criterion* $\mathcal{W} \subseteq \Omega$ (for player 1). In this paper we consider the following winning criteria.

- *Büchi criterion.* For a set B of vertices, the Büchi criterion is defined by:

$$\text{Büchi}(B) = \{\, v_0, v_1, \cdots \in \Omega \;:\; \text{for infinitely many } k \geq 0, \text{ we have } v_k \in B \,\}.$$

- *Co-Büchi criterion.* For a set C of vertices, the co-Büchi criterion is defined by:

$$\text{Co-Büchi}(C) = \{\, v_0, v_1, \cdots \in \Omega \;:\; \text{for finitely many } k \geq 0, \text{ we have } v_k \in C \,\}.$$

- *Parity criterion.* Let $P = (P_0, P_1, \ldots, P_d)$ be a partition of the set of vertices. The parity criterion is defined by:

$$\text{Parity}(P) = \{\, \overline{v} \in \Omega \;:\; \min\big(\text{Inf}(\overline{v})\big) \text{ is even} \,\},$$

where for a path $\overline{v} = v_0, v_1, v_2, \cdots \in \Omega$, we define

$$\text{Inf}(\overline{v}) = \{\, i \in \mathbb{N} \;:\; \text{there are infinitely many } k \geq 0, \text{ such that } \; v_k \in P_i \,\}.$$

Note that a parity criterion $\text{Parity}(P_0, P_1)$ is equivalent to the Büchi criterion $\text{Büchi}(P_0)$, and a parity criterion $\text{Parity}(\emptyset, P_1, P_2)$ is equivalent to the co-Büchi criterion $\text{Co-Büchi}(P_1)$.

For uniformity we phrase all the results below in terms of parity games. By $C(0, j)$ we denote concurrent probabilistic parity games with a parity criterion $\text{Parity}(P_0, P_1, \ldots, P_j)$, and by $C(1, j)$ we denote concurrent probabilistic parity games with a parity criterion $\text{Parity}(\emptyset, P_1, P_2, \ldots, P_j)$. By $D(i, j)$ we denote $C(i, j)$ games with turn-based deterministic game structures. Thus, we write $C(0, 1)$ for concurrent probabilistic Büchi games, $C(1, 2)$ for concurrent probabilistic co-Büchi games, $D(0, 1)$ for turn-based deterministic Büchi games, etc.

Winning modes. Let $\mathcal{G} = (G, \mathcal{W})$ be a game. We say that a strategy $\pi_1 \in \Pi_1$ for player 1 is:

- a *sure winning* strategy for player 1 from vertex v in the game $\mathcal{G}(G, \mathcal{W})$, if for all $\pi_2 \in \Pi_2$, we have $\mathit{Outcome}(v, \pi_1, \pi_2) \subseteq \mathcal{W}$,

– an *almost-sure winning* strategy for player 1 from vertex v in the game $\mathcal{G}(G, \mathcal{W})$, if for all $\pi_2 \in \Pi_2$, we have $\Pr_v^{\pi_1,\pi_2}[\mathcal{W}] = 1$,
– a *positive-probability winning* strategy for player 1 from vertex v in the game $\mathcal{G}(G, \mathcal{W})$, if for all $\pi_2 \in \Pi_2$, we have $\Pr_v^{\pi_1,\pi_2}[\mathcal{W}] > 0$.

The same notions are defined similarly for player 2, with the set $\mathcal{W}$ in the winning condition replaced by $\Omega \setminus \mathcal{W}$.

For a class $\mathcal{C}$ of games, and a winning mode $\mu \in \{\mathrm{s}, \mathrm{a}, \mathrm{p}\}$, we write $\mathcal{C}_\mu$ for the class of games in which the goal of player 1 is to win with the mode μ, where "s" stands for sure win, "a" stands for almost-sure win, and "p" stands for positive-probability win. For example, $C(0,1)_\mathrm{a}$ are almost-sure win concurrent probabilistic Büchi games and $C(1,2)_\mathrm{p}$ are positive-probability win concurrent probabilistic co-Büchi games.

Solving games. The algorithmic problem of solving $\mathcal{C}_\mu$ games is the following: given a game $\mathcal{G}$ from class $\mathcal{C}$ and a vertex v in the game graph as the input, decide whether player 1 has a μ-winning strategy in game $\mathcal{G}$ from vertex v.

3 Witnesses for Turn-Based Deterministic Games

In order to prove that a strategy is winning for a player in a parity game, one needs to argue that all infinite plays consistent with the strategy are winning for the player. A technically convenient notion of a witness has been used in [9, 23, 12] to establish existence of a winning strategy by verifying only some finitary local conditions. We recall here the definitions and basic facts about witnesses (also called signature assignments [9, 23], or progress measures [12]) for a relevant special case $D(0,2)$ games; we leave it as an exercise to the reader to provide similar notions of witnesses for the even simpler case of $D(0,1)$ games.

For $n \in \mathbb{N}$, we write $[n]$ for the set $\{0, 1, 2, \ldots, n\}$, and $[n]_\infty$ for the set $\{0, 1, 2, \ldots, n, \infty\}$, where the element ∞ is bigger than all the others. Let $G = (V, A, A_1, A_2, \delta)$ be a game structure and let $\varphi : V \to [n]_\infty$. We define $\varphi_\infty = \{w \in V : \varphi(w) = \infty\}$, and for a vertex $v \in V$, we define $\varphi_{<v} = \{w \in V : \varphi(w) < \varphi(v)\}$, and $\varphi_{>v} = \{\, w \in V \,:\, \varphi(w) > \varphi(v) \,\}$.

Let $\mathcal{G} = \big(G, \mathrm{Parity}(P_0, P_1, P_2)\big)$ be a $D(0,2)$ game, where G is a concurrent game graph (V, A, A_1, A_2, δ), and $\delta : V \times A \times A \to V$.

Witness for player 1. For a function $\varphi : V \to [n]_\infty$, we say that a vertex $v \in V$ is *φ-progressive for player 1* if the following holds:

$$\begin{aligned} \exists a \in A_1(v).\, \forall b \in A_2(v).\ & \big(v \in P_0 \ \Rightarrow\ \delta(v,a,b) \notin \varphi_\infty\big) \wedge \\ & \big(v \in P_1 \ \Rightarrow\ \delta(v,a,b) \in \varphi_{<v}\big) \wedge \\ & \big(v \in P_2 \ \Rightarrow\ \delta(v,a,b) \notin \varphi_{>v}\big). \end{aligned} \tag{1}$$

We say that the function φ is a *(sure win) witness for player 1* if every vertex $v \in \varphi_{<\infty}$ is φ-progressive for player 2.

Witness for player 2. For a pair of functions $\psi = (\psi^0, \psi^2)$, such that $\psi^0 : V \to [n]_\infty$, and $\psi^2 : V \to [n]$, we say that a vertex $v \in V$ is *ψ-progressive for*

player 2 if the following holds:

$$\exists b \in A_2(v).\, \forall a \ \in A_1(v).\ \begin{array}{l} \big(v \in P_0 \ \Rightarrow\ \delta(v,a,b) \in \psi^0_{<v}\big) \wedge \\ \big(v \in P_1 \ \Rightarrow\ \delta(v,a,b) \notin \psi^0_{>v}\big) \wedge \\ \big(v \in P_2 \ \Rightarrow\ \delta(v,a,b) \in \psi_{<v}\big), \end{array} \tag{2}$$

where we define

$$\psi_{<v} = (\psi^0, \psi^2)_{<v} = \big\{\, w \in V \ :\ \big(\psi^0(w), \psi^2(w)\big) <_{\text{lex}} \big(\psi^0(v), \psi^2(v)\big) \,\big\},$$

and $<_{\text{lex}}$ is the lexicographic ordering. We define $\psi_{<\infty} = \psi^0_{<\infty}$. We say that the function ψ is a *(sure win) witness for player 2* if every vertex $v \in \psi_{<\infty}$ is ψ-progressive for player 2.

Lemma 1. *If φ is a witness for player 1 and ψ is a witness for player 2, then player 1 has a winning strategy from every vertex $v \in \varphi_{<\infty}$, and player 2 has a winning strategy from every vertex $v \in \psi_{<\infty}$.*

The following fact amounts to determinacy for turn-based parity games.

Theorem 3. [9, 23] *If $\mathcal{G}$ is a deterministic turn-based parity game, then there is a witness φ for player 1, and a witness ψ for player 2, such that $\varphi_{<\infty} \cup \psi_{<\infty} = V_{\mathcal{G}}$. Therefore, from every vertex one of the players has a winning strategy.*

In Section 4 we define witnesses for both players in concurrent almost-sure win Büchi games. Then in Section 5 we use them to give a reduction from concurrent probabilistic almost-sure win Büchi games to turn-based deterministic Büchi games. As a by-product we get the following as a corollary of Theorem 3.

Theorem 4. *If $\mathcal{G}$ is a $C(0,1)_a$ game, then there is a witness φ for player 1, and a witness ψ for player 2, such that $\varphi_{<\infty} \cup \psi_{<\infty} = V_{\mathcal{G}}$.*

In Section 6 we define witnesses for both players in concurrent almost-sure win co-Büchi games. Then in Section 7 we use them to give a reduction from concurrent probabilistic almost-sure win co-Büchi games to $D(0,2)$ games. As a by-product we get the following as a corollary of Theorem 3.

Theorem 5. *If $\mathcal{G}$ is a $C(1,2)_a$ game, then there is a witness φ for player 1, and a witness ψ for player 2, such that $\varphi_{<\infty} \cup \psi_{<\infty} = V_{\mathcal{G}}$.*

Note that Theorems 4 and 5 together with Lemma 1 imply Theorem 1, i.e., determinacy of concurrent Büchi and co-Büchi games.

4 Witnesses for Concurrent Büchi Games

Proving that a player in a concurrent parity game has a winning strategy, in particular for non-sure winning modes, is often quite involved. Instead of proving from first principles that certain strategies are winning for a player, we introduce, for various winning modes and criteria, the notions of witnesses, which are

functions that assign natural numbers to vertices so that the assignment satisfies certain "local" constraints. We then prove that a witness for a player gives rise to a winning strategy for him. Once we show that witnesses are sufficient conditions for existence of winning strategies, we only need to focus on constructing witnesses, which is easier than analyzing probabilities of sets of infinite probabilistic plays induced by strategies, since only local finitary constraints need to be verified.

Let $\mathcal{G} = \big(G, \mathrm{Parity}(P_0, P_1)\big)$ be a $C(0,1)$ game with $G = (V, A, A_1, A_2, \delta)$.

Witness for player 1. For a function $\varphi : V \to [n]_\infty$, we say that a vertex $v \in V$ is *φ-progressive for player 1* if the following holds:

$$\begin{aligned}\exists_{\varepsilon>0}.\ \exists_{\alpha\in\mathcal{D}(A_1(v))}.\ \forall_{\beta\in\mathcal{D}(A_2(v))}.\ \big(v \in P_0 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_\infty] = 0\big) \wedge \\ \big(v \in P_1 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_\infty] = 0 \ \wedge\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_{<v}] \geq \varepsilon\big).\end{aligned} \tag{3}$$

We say that the function φ is an *(almost-sure win) witness for player 1* if every vertex $v \in \varphi_{<\infty}$ is φ-progressive for player 1.

Lemma 2. *If $\varphi : V \to [n]_\infty$ is a witness for player 1 in the $C(0,1)_a$ game, then he has an (almost-sure) winning strategy from every vertex in $\varphi_{<\infty}$.*

Witness for player 2. For a function $\psi : V \to [n]_\infty$, we say that a vertex $v \in V$ is *ψ-progressive for player 2* if the following holds:

$$\begin{aligned}\forall_{\delta>1}.\ \exists_{\beta\in\mathcal{D}(A_2(v))}.\ \forall_{\alpha\in\mathcal{D}(A_1(v))}.\ \big(v \in P_0 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\psi_{<v}] > 0\big) \wedge \\ \big(v \in P_1 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\psi_{<v}] > 0 \ \vee\ \mathrm{Pr}_v^{\alpha,\beta}[\psi_{>v}] \leq 1/\delta\big).\end{aligned} \tag{4}$$

We say that the function $\psi : V \to [n]_\infty$ is a *(positive win) witness for player 2* if every vertex $v \in \psi_{<\infty}$is ψ-progressive for player 2.

Lemma 3. *If $\psi : V \to [n]_\infty$ is a witness for player 2 in the $C(0,1)_a$ game, then he has a (positive-probability) winning strategy from every vertex in $\psi_{<\infty}$.*

5 Translation of $C(0,1)_\mathrm{a}$ Games to $D(0,1)$ Games

The following "finitary" characterizations of vertices that are φ- and ψ-progressive for player 1 and player 2, respectively, are the key to our reduction of concurrent probabilistic to turn-based non-probabilistic games.

Action progressive vertices. Let $\varphi, \psi : V \to [n]_\infty$. We say that a vertex $v \in V$ is *action φ-progressive for player 1* if the following holds:

$$\begin{aligned}&\big(v \in P_0 \ \Rightarrow\ \exists_{a\in A_1(v)}.\ \forall_{b\in A_2(v)}.\ \mathrm{Pr}_v^{a,b}[\varphi_\infty] = 0\big) \wedge \\ &\big(v \in P_1 \ \Rightarrow\ \forall_{b\in A_2(v)}.\ \exists_{a\in A_1(v)}.\ \mathrm{Pr}_v^{a,b}[\varphi_{<v}] > 0 \wedge \\ &\qquad\qquad (\forall_{b'\in A_2(v)}.\ \mathrm{Pr}_v^{a,b'}[\varphi_\infty] = 0)\big).\end{aligned} \tag{5}$$

We say that a vertex $v \in V$ is *action ψ-progressive for player 2* if the following holds:

$$\begin{aligned}&\big(v \in P_0 \ \Rightarrow\ \forall_{a\in A_1(v)}.\ \exists_{b\in A_2(v)}.\ \mathrm{Pr}_v^{a,b}[\psi_{<v}] > 0\big) \wedge \\ &\big(v \in P_1 \ \Rightarrow\ \exists_{b\in A_2(v)}.\ \forall_{a\in A_1(v)}.\ \mathrm{Pr}_v^{a,b}[\psi_{>v}] = 0 \vee \\ &\qquad\qquad (\exists_{b'\in A_2(v)}.\ \mathrm{Pr}_v^{a,b'}[\psi_{<v}] > 0)\big).\end{aligned} \tag{6}$$

Lemma 4. *Let $\varphi, \psi : V \to [n]_\infty$. 1. If a vertex $v \in V$ is action φ-progressive for player 1, then it is φ-progressive for him. 2. If a vertex $v \in V$ is action ψ-progressive for player 2, then it is ψ-progressive for him.*

We give a reduction of $C(0,1)_{\mathrm{a}}$ games to $D(0,1)$ games. The idea of the reduction is to replace each concurrent transition in the concurrent game by a small turn-based game in which each player aims at satisfying the condition in the definition of his action progressive vertex.

Let $G = (V, A, A_1, A_2, \delta)$ be a concurrent probabilistic game structure, and let $(G, \mathrm{Parity}(P_0, P_1))_{\mathrm{a}}$ be a $C(0,1)_{\mathrm{a}}$ game. We define a $D(0,1)$ game $(G', (P'_0, P'_1))$ in the following way. The set of vertices of G' includes the set V of vertices of G. We describe the transition function of G' from every vertex $v \in V$, and from the extra vertices that a few-step play in G' from v to another vertex $w \in V$ can go through.

First, for every $v \in V$, $a \in A_1(v)$, $b \in A_2(v)$ we define one-step games $\mathcal{H}_0(v, a, b)$ and $\mathcal{H}_1(v, a, b)$ as follows: the unique initial vertex of game $\mathcal{H}_i(v, a, b)$ has priority i, and the following hold.

- In the initial vertex of game $\mathcal{H}_0(v, a, b)$ player 2 chooses a successor $w \in V$, such that $\delta(v, a, b)(w) > 0$;
- In the initial vertex of game $\mathcal{H}_1(v, a, b)$ player 1 chooses a successor $w \in V$, such that $\delta(v, a, b)(w) > 0$.

In the correctness proof of the reduction given below, the games $\mathcal{H}_0(v, a, b)$ and $\mathcal{H}_1(v, a, b)$ act as gadgets that allow:

- in game $\mathcal{H}_0(v, a, b)$: player 1 to "verify" the condition $\mathrm{Pr}_v^{a,b}[\varphi_\infty] = 0$, and player 2 to "verify" the condition $\mathrm{Pr}_v^{a,b}[\psi_{<v}] > 0$; and
- in game $\mathcal{H}_1(v, a, b)$: player 1 to "verify" the condition $\mathrm{Pr}_v^{a,b}[\varphi_{<v}] > 0$, and player 2 to "verify" the condition $\mathrm{Pr}_v^{a,b}[\psi_{>v}] = 0$.

Next, we define the transition function of $\mathcal{G}'$ from every $v \in V$. Note that this transition function is simply a translation of the formula (5) from the definition of an action progressive vertex into a "formula evaluation" game, where player 1 is the "existential" player, and player 2 is the "universal" player.

- If $v \in P_0$ then the following game is played:
  1. in vertex $v \in P_0$, player 1 chooses a successor (v, a), where $a \in A_1(v)$;
  2. in vertex (v, a), player 2 chooses a one-step game $\mathcal{H}_0(v, a, b)$, where $b \in A_2(v)$.
- If $v \in P_1$ then the following game is played:
  1. in vertex $v \in P_1$, player 2 chooses a successor (v, b), where $b \in A_2(v)$;
  2. in vertex (v, b), player 1 chooses a successor (v, b, a), where $a \in A_1(v)$;
  3. in vertex (v, b, a), player 2 chooses either: the one-step game $\mathcal{H}_1(v, a, b)$, or the successor $(v, b, a, *)$;
  4. in vertex $(v, b, a, *)$ player 2 chooses a one-step game $\mathcal{H}_0(v, a, b')$, where $b' \in A_2(v)$.

Clearly, the game graph G' is turn-based and deterministic. The set P'_0 contains P_0 and all the initial vertices of the one-step games $\mathcal{H}_0(v, a, b)$. All the other vertices belong to P'_1.

Theorem 6. *Let $\mathcal{G}$ be a $C(0,1)_a$ game. For every vertex $v \in V_{\mathcal{G}}$, player 1 has an (almost-sure) winning strategy from v in $\mathcal{G}$ if and only if player 1 has a (sure) winning strategy from v in the $D(0,1)$ game $\mathcal{G}'$.*

Proof idea: The idea of the proof is to argue that witnesses for either of the players in game $\mathcal{G}'$ give rise to witnesses for the same player in game $\mathcal{G}$. Then by the determinacy theorem for turn-based deterministic games (Theorem 3) and Lemma 3 we get Theorems 4 and 6. More precisely, it suffices to establish the following.

1. If $\varphi' : V' \to [n]_\infty$ is a witness for player 1 in the $D(0,1)$ game $\mathcal{G}'$ then the restriction φ of φ' to V is a witness for player 1 in the $C(0,1)_{\mathrm{a}}$ game $\mathcal{G}$.
2. If $\psi' : V' \to [n]_\infty$ is a witness for player 2 in the $D(0,1)$ game $\mathcal{G}'$ then the restriction ψ of ψ' to V is a witness for player 2 in the $C(0,1)_{\mathrm{a}}$ game $\mathcal{G}$.

Remark 1. Observe that the game graph G' we construct above contains vertices of the form (v, a, b), where v is a vertex of the original game graph G, and a and b are moves of players 1 and 2 in v, respectively. Thus formally, in order to claim that our reduction is linear, we need to assume, e.g., that the numbers of moves available to a player in every vertex are $O(1)$. The same applies to our reduction from Section 7.

6 Witnesses for Concurrent Co-Büchi Games

Let $\mathcal{G} = \big(G, \mathrm{Parity}(\emptyset, P_1, P_2)\big)$ be a $C(1,2)$ game, with $G = (V, A, A_1, A_2, \delta)$.

Witness for player 1. For a function $\varphi : V \to [n]_\infty$, we say that a vertex $v \in V$ is *φ-progressive for player 1* if the following holds:

$$\begin{aligned}&\exists_{\varepsilon>0}.\ \exists_{\alpha\in\mathcal{D}(A_1(v))}.\ \forall_{\beta\in\mathcal{D}(A_2(v))}.\\ &\quad\big(v \in P_1 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_\infty] = 0 \ \wedge\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_{<v}] \geq \varepsilon\big) \wedge\\ &\quad\big(v \in P_2 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_\infty] = 0 \ \wedge\ \mathrm{Pr}_v^{\alpha,\beta}[\varphi_{<v}] \geq \varepsilon \cdot \mathrm{Pr}_v^{\alpha,\beta}[\varphi_{>v}]\big).\end{aligned} \tag{7}$$

We say that the function φ is an *(almost-sure win) witness for player 1* if every vertex $v \in \varphi_{<\infty}$ is φ-progressive for player 1.

Lemma 5. *If $\varphi : V \to [n]_\infty$ is an (almost-sure win) witness for player 1, then he has an (almost-sure) winning strategy from every vertex in $\varphi_{<\infty}$.*

Witness for player 2. For a pair of functions $\psi = (\psi^0, \psi^2)$, such that $\psi^0 : V \to [n]_\infty$, and $\psi^2 : V \to [n]$, we say that a vertex $v \in V$ is *ψ-progressive for player 2* if the following holds:

$$\begin{aligned}&\exists_{m\in\mathbb{N}}.\ \forall_{\delta>1}.\ \exists_{\beta\in\mathcal{D}(A_2(v))}.\ \forall_{\alpha\in\mathcal{D}(A_1(v))}.\\ &\quad\big(v \in P_1 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\psi^0_{<v}] > 0 \ \vee\ \mathrm{Pr}_v^{\alpha,\beta}[\psi^0_{>v}] \leq 1/\delta\big) \ \wedge\\ &\quad\big(v \in P_2 \ \Rightarrow\ \mathrm{Pr}_v^{\alpha,\beta}[\psi^0_{<v}] > 0 \ \vee\\ &\qquad\qquad (\mathrm{Pr}_v^{\alpha,\beta}[\psi_{<v}] \geq 1/\delta^m \ \wedge\ \mathrm{Pr}_v^{\alpha,\beta}[\psi^0_{>v}] \leq (1/\delta) \cdot \mathrm{Pr}_v^{\alpha,\beta}\big[\psi_{<v}\big])\big).\end{aligned} \tag{8}$$

We say that the function $\psi : V \to [n]_\infty$ is a *(positive win) witness for player 2* if every vertex $v \in \psi_{<\infty}$ is ψ-progressive for player 2.

Lemma 6. *If $\psi = (\psi^0, \psi^2)$ is a (positive win) witness for player 2, then he has a (positive-probability) winning strategy from every vertex in $\psi_{<\infty}$.*

7 Translation of $C(1,2)_\mathrm{a}$ Games to $D(0,2)$ Games

The following "finitary" characterization of vertices that are φ-progressive for player 1 is the starting point of the idea behind our reduction of concurrent probabilistic co-Büchi games to turn-based non-probabilistic parity games.

Action progressive vertices. Let $\varphi : V \to [n]_\infty$. We say that a vertex $v \in V$ is *action φ-progressive for player 1* if the following holds:

$$\begin{aligned}
&\big(v \in P_1 \;\Rightarrow\; \forall_{b\in A_2(v)}.\, \exists_{a\in A_1(v)}.\; \mathrm{Pr}_v^{a,b}[\varphi_{<v}] > 0 \,\wedge \\
&\qquad\qquad\qquad\qquad (\forall_{b'\in A_2(v)}.\; \mathrm{Pr}_v^{a,b'}[\varphi_\infty] = 0)\big) \;\wedge \\
&\Big(v \in P_2 \;\Rightarrow\; \exists_{\emptyset\neq X\subseteq A_1(v)}.\; (\forall_{a\in X}.\, \forall_{b\in A_2(v)}.\; \mathrm{Pr}_v^{a,b}[\varphi_\infty] = 0) \;\wedge \\
&\qquad \big((\forall_{a\in X}.\, \forall_{b\in A_2(v)}.\, \mathrm{Pr}_v^{a,b}[\varphi_{>v}] = 0) \;\vee\; (\exists_{a'\in X}.\, \mathrm{Pr}_v^{a',b}[\varphi_{<v}] > 0)\big)\Big).
\end{aligned} \tag{9}$$

Lemma 7. *Let $\varphi : V \to [n]_\infty$. If a vertex $v \in V$ is action φ-progressive for player 1, then it is φ-progressive for him.*

We present a reduction of $C(1,2)_\mathrm{a}$ games to $D(0,2)$ games. Let G be a concurrent probabilistic game graph (V, A, A_1, A_2, δ), and let $\big(G, \mathrm{Parity}(\emptyset, P_1, P_2)\big)$ be a $C(1,2)_\mathrm{a}$ game. We define a $D(0,2)$ game (G', P'_0, P'_1, P'_2) in the following way. The set of vertices of G' includes the set V of vertices of G. We describe the transition function of G' from every vertex $v \in V$, and the extra vertices that a game from v can go through.

We are going to use one-step games $\mathcal{H}_0(v,a,b)$ and $\mathcal{H}_1(v,a,b)$ defined in Section 5. Moreover we define a very similar one-step game $\mathcal{H}_2(v,a,b)$, such that its unique initial vertex has priority 2, and in the unique initial vertex of $\mathcal{H}_2(v,a,b)$ player 2 chooses a successor $w \in V$, such that $\delta(v,a,b)(w) > 0$.

As in the case of Büchi games in Section 5, the one-step games $\mathcal{H}_i(v,a,b)$, for $i \in \{0,1,2\}$, serve as gadgets that allow the players to "verify" certain conditions occurring in the definition of an action progressive vertex.

Next, we define the transition relation from every $v \in V$.

- If $v \in P_1$ then the same game is played as for $v \in P_1$ in the reduction of $C(0,1)_\mathrm{a}$ games to $D(0,1)$ games described in Section 5.
- If $v \in P_2$ then the following game is played:
  1. in vertex $v \in P_2$, player 1 chooses a successor (v,a), where $a \in A_1(v)$;
  2. in vertex (v,a), player 2 chooses a successor (v,a,b), where $b \in A_2(v)$;
  3. in vertex (v,a,b), player 2 chooses either: the one-step game $\mathcal{H}_0(v,a,b)$, or the successor $(v,a,b,*)$;

4. in vertex $(v, a, b, *)$, player 1 chooses either: the one-step game $\mathcal{H}_2(v, a, b)$, or the successor (v, b);
5. in vertex (v, b), player 1 chooses a successor (v, b, a'), where $a' \in A_1(v)$;
6. in vertex (v, b, a'), player 2 chooses either: the one-step game $\mathcal{H}_1(v, a', b)$, or the vertex (v, a').

The vertices in V keep their priority, i.e., P_1' includes P_1, and P_2' includes P_2. All the other new vertices different from the initial vertices of games $\mathcal{H}_i(v, a, b)$ have priority 2.

Theorem 7. *Let $\mathcal{G}$ be a $C(1,2)_a$ game. For every vertex $v \in V_{\mathcal{G}}$, player 1 has an (almost-sure) winning strategy from v in $\mathcal{G}$ if and only if player 1 has a (sure) winning strategy from v in the $D(0,2)$ game $\mathcal{G}'$.*

Since $D(0,2)$ games can be solved in quadratic time [12] and $\mathcal{G}'$ is linear in $\mathcal{G}$, we have the following.

Theorem 8. *$C(1,2)_a$ games can be solved in quadratic time.*

8 Discussion

Algorithms for solving concurrent probabilistic games that have been known so far [5, 4] are fairly complicated. On the other hand, the problem of solving turn-based games has been heavily studied and there are many algorithms available [10, 15, 8, 20, 12, 22]. So, from a practical point of view, our reductions allow to directly apply this work, and future related work, to solving concurrent probabilistic games. We note that even though the proofs of correctness of our translations are involved, the translations themselves are fairly simple, so at a very low cost, one can turn a solver for turn-based parity games into a solver for almost-sure winning concurrent reachability, Büchi, and co-Büchi games.

We demonstrated the reductions for the reachability, Büchi, and co-Buchi winning criteria. For turn-based Büchi games, special cases are known to be solvable in linear time. This includes *weak* games [16], and games whose transitions form a tree with back edges [17]. Using our reductions, one can define classes of concurrent probabilistic games for which the game can be decided in linear time.

We conjecture that our translations can be generalized to all almost-sure concurrent parity games and to limit-sure winning [4].

Conjecture 1. There is a linear-time reduction from the problem of solving $C(1,d)$ games to the problem of solving $D(0,d)$ games.

Since $D(0,d)$ games can be solved in time $O(n^{\lfloor d/2 \rfloor + 1})$ [12], this would imply improving the asymptotic time complexity of solving almost-sure concurrent parity games with d priorities from $O(n^{d+1})$ [4] to $O(n^{\lfloor d/2 \rfloor + 1})$.

Finally, let us note that the ability to reduce concurrent games to turn-based games does not mean that concurrent games are a superfluous model. Concurrent

games are appropriate for modeling concurrent systems in which the underlying components interact synchronously [2, 3]. While our reductions are convenient for algorithmic analysis of such systems, the turn-based systems we construct no longer model the original system in any natural sense.

Acknowledgements

We thank Yuval Peres, Rupak Majumdar, and Luca de Alfaro for helpful discussions, and Philippe Darondeau for a suggestion to use probabilistic moves to model varying degrees of (a)synchrony.

References

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 15(1):73–132, 1993. 292

[2] L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. The control of synchronous systems. In C. Palamidessi, editor, *CONCUR 00: Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 458–473. Springer-Verlag, 2000. 293, 304

[3] L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang. The control of synchronous systems, part II. In K. G. Larsen and M. Nielsen, editors, *CONCUR 01: Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001. 293, 304

[4] Luca de Alfaro and Thomas A. Henzinger. Concurrent omega-regular games. In *Proceedings of the 15th Annual Symposium on Logic in Computer Science*, pages 141–154, Santa Barbara, California, 2000. IEEE Computer Society Press. 293, 294, 303

[5] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. In *Proc. 39th IEEE Symposium on Foundations of Computer Science*, October 1998. 293, 303

[6] Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. Technical Report UCB/ERL M98/33, University of California at Berkeley, 1998.

[7] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Proc. 38th IEEE Symp. on Foundations of Computer Science*, pages 100–109, Florida, October 1997. 292

[8] A. Browne, E. M. Clarke, S. Jha, D. E. Long, and W. Marrero. An improved algorithm for the evaluation of fixpoint expressions. *Theoretical Computer Science*, 178(1–2):237–255, May 1997. A preliminary version appeared in Proceedings of CAV'94, volume 818 of LNCS, Springer-Verlag. 294, 303

[9] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (Extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1–4 October 1991. IEEE Computer Society Press. 293, 294, 297, 298

[10] E. Allen Emerson and Chin-Laung Lei. Efficient model checking in fragments of the propositional mu-calculus (Extended abstract). In *Proceedings, Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 16–18 June 1986. IEEE. 294, 303

[11] D. Harel and A. Pnueli. On the development of reactive systems. In K. Apt, editor, *Logics and Models of Concurrent Systems*, volume F-13 of *NATO Advanced Summer Institutes*, pages 477–498. Springer-Verlag, 1985. 292

[12] Marcin Jurdziński. Small progress measures for solving parity games. In Horst Reichel and Sophie Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, February 2000. Springer-Verlag. 293, 294, 297, 303

[13] O. Kupferman, P. Madhusudan, P. S. Thiagarajan, and M. Y. Vardi. Open systems in reactive environments: Control and synthesis. In *Proc. 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 92–107. Springer-Verlag, 2000. 292

[14] O. Kupferman, M. Y. Vardi, and P. Wolper. Module checking. *Information and Computation*, 164:322–344, 2001. 292

[15] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993. 293, 294, 303

[16] D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *Proc. 13th International Colloquium on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*. Springer-Verlag, 1986. 303

[17] Damian Niwiński. Private communication, 1996. 303

[18] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, Austin, January 1989. 292, 293

[19] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *IEEE Transactions on Control Theory*, 77:81–98, 1989. 292, 293

[20] Helmut Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, September 1996. 294, 303

[21] Wolfgang Thomas. Languages, automata, and logic. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Language Theory*, volume III, pages 389–455. Springer-Verlag, 1996. 293, 294

[22] Jens Vöge and Marcin Jurdziński. A discrete strategy improvement algorithm for solving parity games (Extended abstract). In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215, Chicago, IL, USA, July 2000. Springer-Verlag. 294, 303

[23] Igor Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001. 293, 294, 297, 298

A Logic of Probability with Decidable Model-Checking

Danièle Beauquier[1], Alexander Rabinovich[2], and Anatol Slissenko[1]

[1] University Paris 12, France
{beauquier,slissenko}@univ-paris12.fr
[2] Tel-Aviv University, Israel
rabino@math.tau.ac.il

Abstract. A predicate logic of probability, close to logics of probability of Halpern and al., is introduced. Our main result concerns the following model-checking problem: deciding whether a given formula holds on the structure defined by a given Finite Probabilistic Process. We show that this model-checking problem is decidable for a rather large subclass of formulas of a second-order monadic logic of probability. We discuss also the decidability of satisfiability and compare our logic of probability with the probabilistic temporal logic $pCTL^*$.

1 Introduction

Logics with probabilities were considered in different contexts; on the one hand in artificial intelligence for reasoning about uncertainty in expert systems, and on the other hand for specification and verification of systems which exhibit some uncertainty such as fault-tolerant or randomized systems.

One can distinguish two families of logical approaches for reasoning about probabilities: (1) the first one extends the predicate logics (2) the second one extends temporal logics.

A fundamental contribution to the study of predicate logics of probability was done in [FHM90, Hal90], mainly motivated by the problems of artificial intelligence. The paper [Hal90] contains a good survey and analysis of previous works on predicate logics of probability. There has been very interesting works on extensions of predicate logic by probability quantifiers in model-theory community (see survey [Kei85]). The models considered there are different from thus considered in this paper and these works seems to be unrelated to our.

Most of the work related to the verification uses probabilistic extensions of temporal logics. The first applications of temporal logic to probabilistic systems were considered in studying which temporal properties are satisfied with probability 1 by systems modeled as finite Markov chains [LS82]. Later, papers [HJ94, ASB+95] introduced logics $pCTL$ and $pCTL^*$ that can express quantitative bounds on the probability of system evolutions. This approach is surveyed, for example in [Han94] and [CY95].

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 306–321, 2002.
© Springer-Verlag Berlin Heidelberg 2002

In this paper we are interested in the verification of probabilistic systems. However, unlike previous works on verification we take as a specification formalism a probabilistic extension of predicate logic. Predicate logics offer some advantages (over modal and temporal logics) due to their expressiveness and convenience for formalization of complicated properties. We follow the general setting of [FHM90, Hal90, AH94] to introduce a rather expressive predicate logic of probability.

Our main result is a description of a fragment of a second-order monadic logic of probability with the following decidable model-checking problem.
Model-checking problem: decide whether a given formula holds on the structure defined by a given finite Markov chain.

The paper is organized as follows. In Section 2 we give a general description of the logic we use and emphasize how it deviates from [Hal90, FHM90]. In Section 3 we show that monadic predicate logic of probability is undecidable. Section 4 contains our main result about a fragment of probabilistic monadic logic of order with decidable model-checking. Section 5 compares our logic with probabilistic temporal logic $pCTL^*$. We prove that the property "there is a moment at which Q holds with probability one" cannot be expressed in $pCTL^*$, but is directly formalized in our logic of probability. Finally, section 6 presents some further results and open questions.

2 Logic of Probability

We follow Halpern's presentation of logic of probability [Hal90]. There, arithmetic operations on probabilities are allowed, probabilities may be variables which are quantified. In our setting, we just compare probabilities with rational constants. However, we consider second order logics, when [Hal90] confines himself to first order ones.

We consider a language that consists of a collection Σ of predicate symbols of various arities. We also have a collection of predicate variables of various arities. Given formulas φ and ψ in the logic, we allow formulas of the form $\mathbf{Prob}_{>q}(\varphi)$ and $\mathbf{Prob}_{>q}(\varphi|\psi)$, where q is a rational number which can be read as "the probability of φ is greater than q" and "the probability of φ under the condition ψ is greater than q" respectively.

2.1 Syntax

More formally we define the syntax as follows. The vocabulary consists of a set of deterministic predicate symbols, a set of probabilistic predicate symbols, predicate variables and individual variables. We also assume that rational constants are in the vocabulary.
Formulas:

– *Atomic formulas* are of the form $R(x_1, \ldots, x_k)$, where R is a (deterministic or probabilistic) predicate symbol of arity k and $x_1, \ldots, x_k$ are individual variables; or of the form $Q(x_1, \ldots, x_k)$, where Q is a deterministic predicate variable of arity k and $x_1, \ldots, x_k$ are individual variables.

– If φ_1 and φ_2 are formulas then $(\varphi_1 \vee \varphi_2)$ and $\neg\varphi_1$ are formulas.

– If φ is a formula then $\exists\, x\, \varphi$ and $\exists\, Q\, \varphi$, where x is an individual variable and Q is a deterministic predicate variable, are formulas.

– If φ, ψ are formulas, and q is a rational number then $\mathbf{Prob}_{>q}(\varphi)$ and $\mathbf{Prob}_{>q}(\varphi \mid \psi)$ are formulas.

Conjunction $(\varphi_1 \wedge \varphi_2)$, implication $(\varphi_1 \rightarrow \varphi_2)$, universal quantification $\forall\alpha\ \varphi$ are defined as usual, using disjunction, negation and existential quantifier. Expressions like $\mathbf{Prob}_{<q}$, $\mathbf{Prob}_{\leq q}$, $\mathbf{Prob}_{\geq q}$, $\mathbf{Prob}_{=q}$ can be also defined in terms of $\mathbf{Prob}_{>p}$ using negation and modified bounds on probability in a syntactical manner. For example, we define $\mathbf{Prob}_{<p}(\varphi)$ as $\mathbf{Prob}_{>(1-p)}(\neg\varphi)$.

2.2 Semantics

First we recall some basic notions from probability theory.

A *measurable space* is a pair (Ω, Δ) consisting of a non empty set Ω and a σ-algebra Δ of its subsets that are called *measurable sets* and represent random events in probability context. A *σ-algebra* over Ω contains Ω and is closed under complementation and countable union. Adding to a measurable space a *probability measure* $\mu : \Delta \rightarrow [0,1]$ such that $\mu(\Omega) = 1$ and that is countably additive, we get a *probability space* (Ω, Δ, μ). Probabilistic predicates are interpreted as random predicates. Given a domain $\mathcal{U}$ and a probabilistic space (Ω, Δ, μ) a *random* (or *stochastic*) *predicate* P of arity k is a function from $\Omega \times \mathcal{U}^k$ to $Bool = \{\text{true, false}\}$ such that for any fixed $u_1, \ldots, u_k \in \mathcal{U}$ the set $\{\omega \in \Omega : P(\omega, u_1, \ldots, u_k)\}$ is measurable.

A *probabilistic structure* for the language described above is a tuple $(\langle\mathcal{U}, \delta\rangle, \langle\Omega, \Delta, \mu\rangle, \pi)$, where

– $\langle\mathcal{U}, \delta\rangle$ is a first-order structure with universe $\mathcal{U}$, and δ assigns a relation over $\mathcal{U}$ of the appropriate arity to each deterministic predicate symbol;

– $\langle\Omega, \Delta, \mu\rangle$ is a probabilistic space;

– π assigns to each probabilistic predicate symbol P of arity k a random predicate $\pi(P) : \Omega \times \mathcal{U}^k \rightarrow Bool$.

Define a *valuation* ν to be a function which assigns to each individual variable an element of $\mathcal{U}$, and to each deterministic predicate variable a *finite* relation over $\mathcal{U}$ of the appropriate arity ('finite' means that the set of tuples for which the deterministic predicate is true is finite).

Given a probabilistic structure $M = (\langle\mathcal{U}, \delta\rangle, \langle\Omega, \Delta, \mu\rangle, \pi)$, an element $\omega \in \Omega$ and a valuation ν, formally we define when a formula φ holds at ω in M under a valuation ν, written $M, \nu, \omega \models \varphi$, by the following inductive clauses:

(S1) $M, \nu, \omega \models R(x_1, \ldots, x_k)$ for a deterministic predicate symbol R of arity k and individual variables $x_1, \ldots, x_k$ iff $\delta(R)(\nu(x_1), \ldots, \nu(x_k))$ is true.

(S2) $M, \nu, \omega \models Q(x_1, \ldots, x_k)$ for a deterministic predicate variable Q of arity k iff $\nu(Q)(\nu(x_1), \ldots, \nu(x_k))$ is true.

(S3) $M, \nu, \omega \models P(x_1, \ldots, x_k)$ for a probabilistic predicate P of arity k iff $\pi(P)(\omega, \nu(x_1), \ldots, \nu(x_k))$ is true.

(S4) Quantifiers over individual variables and Boolean connectors are treated as usually.

(S5) Quantifiers over deterministic predicate variables are interpreted as quantifiers over deterministic predicate variables that range only over finite relations over $\mathcal{U}$.

(S6) $M, \nu, \omega \models \mathbf{Prob}_{>q}(\varphi)$ iff $\mu(\{\omega' \in \Omega : M, \nu, \omega' \models \varphi\}) > q$, that is iff the set of all ω' for which $M, \nu, \omega' \models \varphi$ holds has a measure greater than q.

(S7) $M, \nu, \omega \models \mathbf{Prob}_{>q}(\varphi|\psi)$ iff $\mu\{\omega' \in \Omega : M, \nu, \omega' \models (\varphi \wedge \psi)\} > q \cdot \mu\{\omega' \in \Omega : M, \nu, \omega' \models \psi\}$, i. e. the conditional probability of φ under ψ is $> q$.

Remark that (S6) is a particular case of (S7) when $\psi = true$.

The semantics is well defined only if the sets that appear in (S6) and (S7) are measurable. From now on we assume that

(**Countability Assumption**) *The domain $\mathcal{U}$ of probabilistic structures is countable.*

Proposition 1 *Under Countability Assumption the sets that appear in (S6) and (S7) are measurable, and the semantics is well defined.*

Proof. By induction on the structure of formulas. The only not quite straightforward step is quantification. For formula $\exists x\ \varphi$ we use that any σ-algebra is closed under countable union. For formula $\exists Q\ \varphi$ we use the fact that the set of finite predicates over a countable domain is countable and thus we can again use that any σ-algebra is closed under countable union. □

Proposition 2 *Suppose that two valuations ν_1 and ν_2 agree on the free variables of a formula φ. Then $M, \nu_1, \omega \models \varphi$ iff $M, \nu_2, \omega \models \varphi$.*

Proposition 3 *If all the occurrences of probabilistic predicates in a formula φ are in the scope of some operator* **Prob** *then $M, \nu, \omega_1 \models \varphi$ iff $M, \nu, \omega_2 \models \varphi$ for every $\omega_1, \omega_2 \in \Omega$. In particular, for any formula ψ we have $M, \nu, \omega_1 \models \mathbf{Prob}_{>q}(\psi)$ iff $M, \nu, \omega_2 \models \mathbf{Prob}_{>q}(\psi)$.*

3 Undecidability of Monadic Logic of Probability

The decidability of the probabilistic propositional logic follows from [FH94] where the decidability of a more general logic was proved. For first-order logic it is well-known that the satisfiability problem is decidable if the language has only unary predicates (Monadic Logic) and the satisfiability problem is undecidable even with one binary predicate [Hod93]. Many undecidability results for probabilistic logics can be found in [AH94] where this question was investigated in detail. It was shown in [AH94] that the satisfiability problem of their probabilistic logic even with one unary predicate is Σ_1^2 complete. However the logics considered there admit addition of probabilities or even multiplication of probabilities and quantifiers over reals and the methods of [AH94] are not applicable for our (much weaker) probabilistic logic.

In this section we prove (Theorem 1) that the satisfiability/validity problem for monadic logic of probability (that is a logic of probability where all predicates are monadic and the domain is $\mathbb{N}$) is undecidable. We reduce the satisfiability

problem for first-order predicate logic with one binary predicate to the satisfiability problem for monadic logic of probability.

First, we define a translation from the first-order formulas over a binary predicate to formulas of probabilistic logic with two unary predicates. Let R be a binary predicate symbol and ϕ be a formula in the signature $\{R\}$. Replace in ϕ every occurrence of $R(x, y)$ by $\mathbf{Prob}_{>0}(P(x) \wedge Q(y))$, where P and Q are unary predicate symbols. The resulting formula $\psi(P, Q)$ is called the *translation* of ϕ.

Proposition 4 *The formula $\phi(R)$ is satisfiable iff its translation $\psi(P, Q)$ is satisfiable.*

Proof. It is clear that if the translation of ϕ is satisfiable in a probabilistic structure M then ϕ is satisfiable in the structure $\langle |M|,\ R^* \rangle$, where $|M|$ is the universe of M and $R^*(a, b)$ holds iff $M, a, b \models \mathbf{Prob}_{>0}(P(x) \wedge Q(y))$.

Let $\mathcal{M}$ be a structure for a binary predicate name R where the interpretation of R is a relation R^* over a countable universe $\mathcal{U} = \{a_1, a_2, \ldots, a_n, \ldots\}$. Let us define a probabilistic structure M as follows. Take as a probabilistic space $\Omega = \mathcal{U}$ with a discrete distribution of probabilities $\mu(\{a_n\}) = 1/2^n$ for every n if Ω is infinite, and μ is uniform if Ω is finite.

For each $a_n \in \Omega$, set $\pi(P)(a_n, t) =$ true iff $t = a_n$ and set $\pi(Q)(a_n, t) =$ true iff $R^*(a_n, t)$.

Observe that for every $a, b \in \mathcal{U}$, $R^*(a, b)$ iff $M, a, b \models \mathbf{Prob}_{>0}(P(x) \wedge Q(y))$. Hence for every sentence ϕ in the signature $\{R\}$ and its translation ψ we have $\mathcal{M} \models \phi$ iff $M \models \psi$ In particular, if ϕ is satisfiable then its translation is satisfiable. □

From Proposition 4 we can deduce:

Theorem 1 *The satisfiability problem for monadic logic of probability is undecidable.*

We do not know the exact complexity for the satisfiability problem of monadic logic of probability, however we believe that it is much lower than Σ_1^2. We also have the following property:

Proposition 5 *There exists a satisfiable formula of monadic logic of probability with equality such that all its models have an infinite probabilistic space.*

Proof. There is a closed predicate formula $\phi(R)$ over a binary predicate R which is satisfiable only in structures where the universe is infinite. For example take for $\phi(R)$ the conjunction of the three properties, R is transitive, irreflexive and $\forall x\ \exists y\ R(x, y)$. Consider the formula $\psi(P, Q)$ obtained as above, replacing in $\phi(R)$ every occurrence of $R(x, y)$ by $\mathbf{Prob}_{>0}(P(x) \wedge Q(y))$. Consider the probabilistic monadic formula

$$\Psi(P, Q) = \psi(P, Q) \wedge \mathbf{Prob}_{=1}(\exists! x\ P(x)) \wedge \forall x\ \mathbf{Prob}_{>0}(P(x))$$

We claim that :

(1) $\Psi(P, Q)$ is satisfiable,

(2) Every model of $\Psi(P, Q)$ has an infinite probabilistic space.

In order to prove (1), consider the following model M. Take a countable infinite universe $\mathcal{U} = \{a_1, a_2, \ldots, a_n, \ldots\}$. Take as a probabilistic space $\Omega = \mathcal{U}$ with a discrete distribution of probabilities $\mu(\{a_n\}) = 1/2^n$ for every n.

For each $a_n \in \Omega$ set $\pi(P)(a_n, t) = true$ iff $t = \{a_n\}$ and $\pi(Q)(a_n, t) = true$ iff $t \in \{a_{n+1}, a_{n+2}, \ldots\}$. Then it is clear from the construction that M satisfies $\Psi(P, Q)$.

Here is the proof of (2). Suppose there is a structure M that is a model of $\Psi(P, Q)$ with a finite probabilistic space $\Omega = \{\omega_1, \ldots, \omega_k\}$. We can suppose that $\mu(\omega_i) > 0$ for $i = 1, \ldots, k$. Thus for $i = 1, \ldots, k$ there exists a unique $a_i \in \mathcal{U}$ such that $\pi(P)(\omega_i, a_i)$ because M satisfies $\mathbf{Prob}_{=1}(\exists! x\ P(x))$. Choose an element a in universe $\mathcal{U}$ different from all the a_i. Since M satisfies $\forall\, x\ \mathbf{Prob}_{>0}(P(x))$, there exists an $\omega \in \Omega$ such that $\pi(P)(\omega, a) = true$. A contradiction. □

4 Model-Checking for a Fragment of Logic of Probabilities

In this section we consider a logic of probability where all predicates are monadic and the domain is $\mathbb{N}$ with order. This logic is denoted *PMLO*. The probabilistic structures used in this section are defined by Finite Probabilistic Processes. We study the following model-checking problem: decide whether a given *PMLO*-formula φ holds on the structure defined by a given Finite Probabilistic Process. We introduce a rather large subclass $\mathcal{C}$ of formulas for which the model-checking problem is 'almost always decidable'.

Subsection 4.1 explains how Finite Probabilistic Processes define probabilistic structures. Subsection 4.2 introduces a class $\mathcal{C}$ of formulas with decidable model-checking problem.

4.1 Probabilistic Structures Defined by Finite Probabilistic Processes

Definition. A *Finite Probabilistic Process* is a finite labelled Markov chain [KS60] $M = (S, P, V, \mathcal{L})$, where S is a finite set of states, P is a transition probability matrix: $S^2 \to [0, 1]$ such that $P(i, j)$ is a rational number for all $(i, j) \in S^2$, $\sum_{j \in S} P(i, j) = 1$ for every $i \in S$, and $V : S \to 2^{\mathcal{L}}$ is a valuation function which assigns to each state a set of symbols from a finite set $\mathcal{L}$.

The pair (S, P) is called a *finite Markov chain.*

The following Lemma is a well known fact in the theory of matrices (see e. g. [Gan77],13.7.5, 13.7.1)

Lemma 1 *Let (S, P) be a finite Markov chain. There exists a positive natural number d* period of the Markov chain *such that the limits*

$$\lim_{m \to \infty} P^{r+dm} = P_r \quad (r = 0, 1, \ldots, d-1)$$

exist. Moreover if the elements of P are rational, then these limits are computable from P and the convergence to the limits is geometric, i. e. $|P^{r+dm}(i, j) -$

$P_r(i,j)| < a \cdot b^m$ when $m \geq m_0$ for some positive rationals a, $b < 1$ and natural m_0 also computable from P.

Given a Finite Probabilistic Process $M = (S, P, V, \mathcal{L})$ and a state s, we define a probabilistic structure M_s as follows:
Signature: a deterministic binary predicate $<$, and monadic probabilistic predicates Q for every label $Q \in \mathcal{L}$.
Interpretation:

- the universe of the structure M_s is the set $\mathbb{N}$ of natural numbers;
- $<$ is interpreted as the standard less relation over $\mathbb{N}$;
- probabilistic space (Ω, Δ, μ)(see [KSK66]) : $\Omega = sS^\omega$ is the set of all infinite sequences of states starting from s, Δ is the σ-algebra generated by the basic cylindric sets $D_u = uS^\omega$, for every $u \in sS^*$, and the probability measure μ is defined by $\mu(D_u) = \prod_{i=0,\ldots,n-1} P(s_i, s_{i+1})$ where $u = s_0 s_1 \ldots s_n$;
- interpretation of monadic probabilistic predicates:

for each $\omega = s_0 s_1 \ldots s_n \ldots \in \Omega$, for each $n \in \mathbb{N}$ we have $\pi(Q)(\omega, n)$ iff $Q \in V(s_n)$ (i.e. Q belongs to the label of state s_n). At this point, notice that for every integer n, the set $\{\omega \in \Omega : \pi(Q)(\omega, n)\}$ is μ-measurable since it is a finite union of basic cylinders.

Example. Let us consider a Call Establishment procedure in a simple telephone network where the capacity of simultaneous outgoing calls is less than the number of users. An abstraction of this procedure represents the behavior of a user where time is assumed to be discrete (Figure 1).

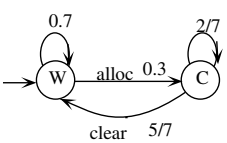


Fig. 1.

To simplify it is assumed that a user which is not connected is continuously attempting to get a connection (state $Wait$) and at each time moment he succeeds to be connected with probability 3/10. Moreover when the calling is established the duration of the call (state $Call$) follows a geometric distribution: at each time moment, the probability to finish the call has probability 5/7.
One can write a liveness property such that:

$$\varphi =_{df} \forall t \; Prob_{=1}(\exists \, t' > t \; Call(t') \mid Wait(t)) \tag{1}$$

which expresses that at every time, if the user is waiting for a connection, the probability that he will be served later is equal to one.

One can also express some probabilistic property concerning the time the user has to wait before being served:

$$\psi =_{df} \forall t \; Prob_{\geq 0.9}(\exists t' \; (t < t' \; \wedge \; t' < t + 3 \; \wedge \; Call(t')) \mid Wait(t)) \tag{2}$$

The set of labels here is equal to the set of states, and the label of a state is the state itself. One can prove that $M_{Wait} \models \varphi$ and $M_{Wait} \not\models \psi$.

4.2 A Fragment of Logic of Probability with Decidable Model-Checking

Recall that MLO denotes monadic second order logic of order over natural numbers and $WMLO$ denotes monadic second order logic of order over natural numbers where second-order quantification is over finite sets instead of arbitrary sets.

Below, when speaking about $WMLO$-formulas, we consider only $WMLO$-formulas without free second order variables. The predicate symbols of these formulas are interpreted as arbitrary sets. When we apply a **Prob** operator to such a formula we interpret all its predicate symbols as probabilistic ones.

Definition. A $PMLO$-formula φ belongs to the class $\mathcal{C}$ iff operators $\mathbf{Prob}_{>q}$ are not nested and are applied only to $WMLO$-formulas with at most one free individual variable. For example

$$\exists t \exists t' (t < t' \wedge \mathbf{Prob}_{>1/3}(P(t) \wedge \exists Q \forall t'' > t\ Q(t'')) \wedge \mathbf{Prob}_{>1/2}(\neg P(t'))) \quad (3)$$

where P is a probabilistic predicate and Q a deterministic one, belongs to $\mathcal{C}$.

The properties expressed in (1) and (2) are also in the class $\mathcal{C}$.

As one more example that needs a weak second order quantification we can mention the following property: the probability that a given probabilistic predicate has an even number of elements is greater than 0.9.

The main result of this subsection is Theorem 2 which, roughly speaking, says that it is decidable whether a given formula $\varphi \in \mathcal{C}$ holds in the structure defined by a given Finite Probabilistic Process M.

In order to express our decidability result about model checking, we need to introduce the notion of *parametrized* formula of logic of probability.

The set of *parametrized formulas* is defined similarly to the set of formulas except that operators $\mathbf{Prob}_{>q}$ with $q \in \mathbb{Q}$ are replaced by $\mathbf{Prob}_{>p}$, where p is a parameter name.

For example

$$\exists t \exists t' (t < t' \wedge \mathbf{Prob}_{>p_1}(P(t) \wedge\ \exists Q \forall t'' > t\ Q(t'')) \wedge \mathbf{Prob}_{>p_2}(\neg P(t')))$$

is a parametrized formula.

A formula φ is said to be *completely closed* if it is closed, and no probabilistic predicate is out of scope of an operator **Prob**. If φ is a completely closed formula, $M \models \varphi$ stands for $M, \omega \models \varphi$, that is well-defined and is independent from ω due to Proposition 3.

Let φ be a parametrized formula with parameters $p_1, \ldots, p_m$ and $\alpha_1, \ldots, \alpha_m$ be a sequence of rational values. We denote by $\varphi_{\alpha_1,\ldots,\alpha_m}$ the formula obtained by replacing in φ each parameter p_i by the value α_i. The set of parametrized completely closed formulas is defined exactly like the set of completely closed formulas. By abuse of terminology, we say that a parametrized formula φ belongs to $\mathcal{C}$ if all (or, equivalently, any of) its instances $\varphi_{\alpha_1,\ldots,\alpha_m}$ are in $\mathcal{C}$.

Theorem 2 *Given a Finite Probabilistic Process M, a state s_0 of M and a parametrized completely closed formula φ in the class $\mathcal{C}$ with m parameters, one can compute for each parameter p_i in φ a finite set P_i of rational values*

($i = 1, \ldots, m$), such that for each tuple $\alpha = (\alpha_1, \ldots, \alpha_m)$ where $\alpha_i \in \mathbb{Q} \setminus P_i$, $i = 1, \ldots, m$, one can decide whether (M, s_0) satisfies φ_α.

Remarks. 1. The complexity of our decision procedure is mainly determined by the complexity of decision procedure for MLO-formulas (that is non-elementary in the worst case).

2. In the definition of class $\mathcal{C}$ we allow to apply probabilistic operators only to formulas with one free individual variable. This is not essential restriction. The decidability result can be extended to the case when **Prob** is applied to formulas with many free individual variables. However the proof of the decidability of this extended fragment is more subtle and will be given in the full version of the paper.

3. The fact that we cannot treat some finite number of exceptional values seems to be essential from mathematical point of view. One cannot exclude that the model checking problem is undecidable for these exceptional values. However, for practical properties the values of probabilities can always be slightly changed without loss of its essential significance, and this permits to eliminate these exceptional values of probabilities.

4.3 Proof of Theorem 2

In the rest of this section the proof of Theorem 2 is given.

We introduce a notation: $\mathbb{N}_{\geq a} = \{n \in \mathbb{N} | \, n \geq a\}$ and we recall what are future and past $(W)MLO$-formulas.

Definition. A $(W)MLO$-formula $\varphi(x_0, X_1, X_2, ..., X_m)$ with only one free first-order variable x_0 is a *future formula* if for every $a \in \mathbb{N}$ and every m subsets $S_1, S_2, .., S_m$ of $\mathbb{N}$, the following holds:

$(\mathbb{N}, a, S_1, S_2, .., S_m) \models \varphi(x_0, X_1, X_2, ..., X_m)$

iff $(\mathbb{N}_{\geq a}, a, S'_1, S'_2, .., S'_m) \models \varphi(x_0, X_1, X_2, ..., X_m)$

where $S'_i = S_i \cap \mathbb{N}_{\geq a}$ for $i = 1, 2, ..., m$.

Past $(W)MLO$-formulas are defined in a symmetric way. Note that this is a semantic notion.

Theorem 4.1.7 [CY95] gives the following corollary that we will use:

Theorem 3 *Let $\varphi(t)$ be a future $(W)MLO$-formula with only one free variable and M be a Finite Probabilistic Process. One can compute for each state s of M, the probability f_s of the set of $\omega \in \Omega = sS^\omega$ that satisfy $\varphi(0)$.*

Recall that a set $S \subseteq \mathbb{N}$ is *ultimately periodic* if there are $h, d \in \mathbb{N}$ such that for all $n > h$, $n \in S$ iff $n + d \in S$.

Below, for simplicity, we will write $\mathbf{Prob}_{M_s}(\varphi(n))$ instead of $\mu\{\omega : M_s, n, \omega \models \varphi(t)\}$ for a Finite Probabilistic Process M, state s of M and $n \in \mathbb{N}$.

Lemma 2 *Let $M_1, \ldots, M_k$ be Finite Probabilistic Processes, s_i be a state of M_i ($1 \leq i \leq k$), $\varphi_1(t), \ldots, \varphi_k(t)$ be future $WMLO$-formulas with only one free variable t and $c_1, \ldots, c_k \in \mathbb{Q}$. For all (rational) values of p except a finite number of computable values, the set*

$$\{n \in \mathbb{N} : \sum_{1 \leq i \leq k} c_i \cdot \mathbf{Prob}_{M_{i,s_i}}(\varphi_i(n)) > p\}$$

is finite or ultimately periodic, and is computable.

Proof. We give a proof for $k = 1$. The general case is treated similarly. Let $\varphi(t)$ be a future $WMLO$-formula with only one free variable t. Using Theorem 3, one can compute for each state s of M, the probability f_s of the set of $\omega \in \Omega = sS^\omega$ that satisfy $\varphi(0)$. Let F be the column vector $(f_s)_{s \in S}$. Let P be the transition probability matrix of M. Let I be the row vector with elements all equal to zero except the element in place s_0 which is equal to 1. Vector I represents the initial probability distribution over states of M.

For a given n, the probability that (M_{s_0}, n) satisfies $\varphi(t)$ is equal to $I \cdot P^n \cdot F$.

So, we have to compute the set $\mathbb{N}_{\varphi,p}$ of integers n such that $I \cdot P^n \cdot F > p$. In the general case, P^n does not converge when $n \to \infty$.

Let d be the period of the Markov chain from Lemma 1. For each $r \in D = \{0, \ldots, d-1\}$ consider the set $\mathbb{N}_r = r + d\mathbb{N}$. For $n \in \mathbb{N}_r$, the product $I \cdot P^n \cdot F$ has a limit p_r when $n \to \infty$ (Lemma 1). Define $\mathcal{P} = \{p_0, p_1, \ldots, p_{d-1}\}$. Fix a value $p \in \mathbb{Q} \setminus \mathcal{P}$. Let D^+ be the set of integers r such that $p_r > p$, and D^- be the set of integers r such that $p_r < p$.

For $r \in D^-$, let $K_{r,p}$ be the set $\{n \in \mathbb{N}_r : I \cdot P^n \cdot F > p\}$. Note that $K_{r,p}$ is finite and computable from p. For $r \in D^+$, let $K'_{r,p}$ be the set $\{n \in \mathbb{N}_r : I \cdot P^n \cdot F \leq p\}$. Note that $K'_{r,p}$ is finite and computable from p. Thus for $p \in \mathbb{Q} \setminus \mathcal{P}$, the set $\mathbb{N}_{\varphi,p}$ is equal to the union $\bigcup_{r \in D^-} K_{r,p} \cup \bigcup_{r \in D^+} \mathbb{N}_r \setminus K'_{r,p}$, and $\mathbb{N}_{\varphi,p}$ is finite or ultimately periodic and is computable. □

Lemma 3 *Let $M_1, \ldots, M_k$ be Finite Probabilistic Processes, s_i be a state of M_i $(1 \leq i \leq k)$, $\varphi_1(t), \ldots, \varphi_k(t)$ be past $WMLO$-formulas with only one free variable t and $c_1, \ldots, c_k \in \mathbb{Q}$. For all (rational) values of p except a finite number of computable values, the set*

$$\{n \in \mathbb{N} : \sum_{1 \leq i \leq k} c_i \cdot \mathbf{Prob}_{M_{i,s_i}}(\varphi_i(n)) > p\}$$

is finite or ultimately periodic, and is computable.

Proof. We prove this lemma for $k = 1$. Let $\varphi(t)$ be a past $WMLO$-formula with only one free variable t. A structure S for such a formula φ is defined as an infinite word on the alphabet $\Sigma = 2^{\mathcal{L}}$ where $\mathcal{L}$ is the set of monadic symbols of $\varphi(t)$. The property defined by $\varphi(t)$ depends only on the prefix of size $t+1$ of a model. Thus [Büc60], there exists a finite complete deterministic automaton $\mathcal{A}$ on the alphabet Σ accepting a language of finite words $L(\mathcal{A})$ such that $S, n \models \varphi(t)$ iff the prefix of S of size $n+1$ belongs to $L(\mathcal{A})$.

Therefore, given the automaton $\mathcal{A}$ and the Finite Probabilistic Process M, we build a new Finite Probabilistic Process M', "product" of M and $\mathcal{A}$ in the following way:

States of M' are pairs (q, s) where q is a state of $\mathcal{A}$ and s is a state of M. There is a transition from (q, s) to (q', s') iff (q, σ, q') is a transition in $\mathcal{A}$, where σ is the valuation of s in M and the probability of this transition is the same as the probability of (s, s') in M.

At last, the set of labels $\mathcal{L}'$ of M' is reduced to one symbol F and the valuation of (q, s) is $\{F\}$ if q is a final state in $\mathcal{A}$, and $\emptyset$ otherwise.

It is clear that :
$M_{s_0}, n \models \mathbf{Prob}_{>p}(\varphi(t))$ iff $M'_{(q_0,s_0)}, n \models \mathbf{Prob}_{>p}(F(t))$
where q_0 is the initial state of $\mathcal{A}$ and F is the monadic probabilistic symbol defined by $\mathcal{L}'$.

Since $F(t)$ is a future $WMLO$-formula, using Lemma 2 we get the result. □

Lemma 4 *Let M be a Finite Probabilistic Process, s_0 be a state of M, $\varphi(t)$ and $\psi(t)$ be $WMLO$-formulas with only one free variable t. For all rational values of p, except a finite computable set $\mathcal{P}$, the sets*
(1) $\mathbb{N}_{\varphi,p} =_{df} \{n \in \mathbb{N} | M_{s_0}, n \models \mathbf{Prob}_{>p}(\varphi(t))\}$,
(2) $\mathbb{N}_{\varphi,\psi,p} =_{df} \{n \in \mathbb{N} | M_{s_0}, n \models \mathbf{Prob}_{>p}(\varphi(t)|\psi(t))\}$
are finite or ultimately periodic, and are computable.

Proof. (1) Let $\varphi(t)$ be a $WMLO$-formula with only one free variable t. Such a formula $\varphi(t)$ is equivalent (Lemma 9.3.2 in [GHR94]) to a finite disjunction of mutually exclusive formulas $\varphi_i(t)$ of the form $(\alpha_i(t) \wedge \beta_i(t))$, where $\alpha_i(t)$ are past formulas and $\beta_i(t)$ are future formulas. Moreover the $\alpha_i(t)$ and $\beta_i(t)$ are computable from formula $\varphi(t)$.

For each state s_j of M we introduce a new probabilistic predicate S_j, and add S_j in the valuation of s_j. Let M' be the new Finite Probabilistic Process obtained in this way.

The following equalities hold:
$\mathbf{Prob}_{M_s}(\varphi(n)) = \mathbf{Prob}_{M_s}(\bigvee_i \varphi_i(n)) = \sum_{i\in I} \mathbf{Prob}_{M_s}(\varphi_i(n))$
$= \sum_{i\in I} \mathbf{Prob}_{M_s}(\alpha_i(n) \wedge \beta_i(n))$
$= \sum_{i\in I} \mathbf{Prob}_{M'_s}(\bigvee_{j\in J}((\alpha_i(n) \wedge S_j(n)) \wedge (\beta_i(n) \wedge S_j(n)))$
$= \sum_{i\in I} \sum_{j\in J} \mathbf{Prob}_{M'_s}((\alpha_i(n) \wedge S_j(n)) \wedge (\beta_i(n) \wedge S_j(n)))$
$= \sum_{i\in I} \sum_{j\in J} \mathbf{Prob}_{M'_s}\big(\ \alpha_i(n) \wedge S_j(n)\ \big) \cdot \mathbf{Prob}_{M'_s}\big(\ \beta_i(n) \wedge S_j(n) \mid \alpha_i(n) \wedge S_j(n)\ \big)$
$= \sum_{i\in I} \sum_{j\in J} \big(\ \mathbf{Prob}_{M'_s}(\alpha_i(n) \wedge S_j(n)) \cdot \mathbf{Prob}_{M_{s_j}} \beta_i(0)\ \big)$.

We can compute the rational constants $\mathbf{Prob}_{M_{s_j}} \beta_i(0)$ using Theorem 3 and then we apply Lemma 3 to finish the proof.

The proof of (2) can be reduced to the proof of (1). □

Proof.(*of Theorem 2*) For each $i = 1, \ldots, m$, let ψ_i be the subformula of ϕ of the form $\mathbf{Prob}_{>p_i} \varphi_i(t_i)$. One can compute using Lemma 4 a finite set of probabilities P_i such that for each value $\alpha_i \in \mathbb{Q} \setminus P_i$, the set $R_{\alpha_i} = \{n : M_{s_0}, n \models \mathbf{Prob}_{>\alpha_i} \varphi_i(t_i)\}$, is computable and is finite or ultimately periodic.

For each $i = 1, \ldots, m$, each value $\alpha_i \in \mathbb{Q} \setminus P_i$ each subformula ψ_i of ϕ, of the form $\mathbf{Prob}_{>p_i} \varphi_i(t_i)$, one can compute, using Lemma 4 the set $R_{\alpha_i} = \{n : M_{s_0}, n \models \mathbf{Prob}_{>\alpha_i} \varphi_i(t_i)\}$, and this set is an ultimately periodic set. There exists a first-order MLO-formula $\theta_{\alpha_i}(X)$ which characterizes R_{α_i}, i. e. R_{α_i} is the unique predicate that satisfies $\theta_{\alpha_i}(X)$. For example if R_{α_i} is the set of even integers, then $\theta_{\alpha_i}(X)$ will be "$X(0) \wedge \forall t(X(t) \leftrightarrow X(t+2))$".

Introduce new monadic predicate names N_{α_i}. Let Ψ_α be the formula obtained from φ_α by replacing $\mathbf{Prob}_{>\alpha_i} \varphi_i(t_i)$ by $N_{\alpha_i}(t_i)$. Consider now the MLO-formula $\Psi'_\alpha = \big(\ (\bigwedge_{0\le i\le m} \theta_{\alpha_i}(N_{\alpha_i})) \rightarrow \Psi_\alpha\ \big)$. Clearly, (M, s) satisfies φ_α iff the

MLO-formula Ψ'_α is valid. Since the validity problem for MLO is decidable, it follows that the problem whether (M_{s_0}) satisfies φ_α is decidable. □

5 Comparison with Probabilistic Temporal Logic *pCTL**

The logic $pCTL^*$ is one of the most widespread among probabilistic temporal logics [ASB+95]. The relationship between our logic and $pCTL^*$ is rather complex. The semantics for logic of probability is defined over arbitrary probabilistic structures, however $pCTL^*$ is defined only for Finite Probabilistic Processes. Moreover, unlike logic of probability, the truth value of $pCTL^*$ formula depends not only on the probabilistic structure defined by a Finite Probabilistic Process but also on the 'branching structure' of this process. Hence, there is no meaning preserving translation from $pCTL^*$ to *monadic* logic of probability. We also show below that even on the class of models restricted to Finite Probabilistic Processes no $pCTL^*$ formula is equivalent to the probabilistic formula $\exists t\ \mathbf{Prob}_{\geq 1} Q(t)$, where Q is a probabilistic predicate symbol.

Let us recall the syntax and the semantics of the logic $pCTL^*$ as defined in [ASB+95]. Formulas are evaluated on a probabilistic structure associated to a Finite Probabilistic Process $(S, P, V, \mathcal{L})$.

There are two types of formulas in $pCTL^*$: state formulas (which are true or false in a specific state) and path formulas (which are true or false along a specific path).

Syntax. State formulas are defined by the following syntax:

1. each a in $\mathcal{L}$ is a state formula
2. If f_1 and f_2 are state formulas, then so are $\neg f_1$, $f_1 \vee f_2$
3. If g is a path formula, then $Pr_{<q}(g)$, $Pr_{>q}(g)$ are state formulas for every rational number q.

Path formulas are defined by the following syntax:

1. A state formula is a path formula
2. If g_1 and g_2 are path formulas, then so are $\neg g_1$, $g_1 \vee g_2$
3. If g_1 and g_2 are path formulas, then so are Xg_1, $g_1 U g_2$.

(X and U are respectively the $Next$ and $Until$ temporal operators).

Semantics. Given a Finite Probabilistic Process $M = (S, P, V, \mathcal{L})$ state formulas and path formulas are interpreted as defined below. Formulas f_1 and f_2 are state formulas and g_1 and g_2 are path formulas. Let s be a state, and Π be an arbitrary infinite path in M. Satisfaction of a state formula is defined with respect to s and satisfaction of a path formula with respect to Π. For each integer $k \geq 0$, we denote by Π^k the path obtained from Π when removing the first k states (thus $\Pi^0 = \Pi$) and by $[\Pi]_k$ the kth state of Π.

- $M, s \models Q$ iff $a \in V(Q)$,
- $M, s \models \neg f_1$ iff $M, s \not\models f_1$, $M, s \models f_1 \vee f_2$ iff $M, s \models f_1$ or $M, s \models f_2$,
- $M, s \models \mathbf{Prob}_{>q}(g_1)$ iff $\mu\{\sigma \in sS^\omega | M, \sigma \models g_1\} > q$,

  $M, s \models \mathbf{Prob}_{<q}(g_1)$ is defined in a similar way,
- $M, \Pi \models f_1$ iff $[\Pi]_0 \models f_1$,

• $M, \Pi \models \neg g_1$ iff $M, \Pi \not\models g_1$, $M, \Pi \models g_1 \vee g_2$ iff $M, \Pi \models g_1$ or $M, \Pi \models g_2$,
• $M, \Pi \models X g_1$ iff $M, \Pi^1 \models g_1$,
• $M, \Pi \models g_1 U g_2$ iff there exists $k \geq 0$ such that $M, \Pi^k \models g_2$ and for all $0 \leq j < k$, $M, \Pi^j \models g_1$.

Below we give an example that illustrate differences between the logic of probabilities and $pCTL^*$. Consider the Finite Probabilistic Processes K and L shown on Figure 2 below. Let φ be the following pCTL* formula

$\mathbf{Prob}_{=1}\big(\, X(\mathbf{Prob}_{=\frac{1}{2}}(X\ P) \wedge \mathbf{Prob}_{=\frac{1}{2}}(X\ Q))\,\big)$.

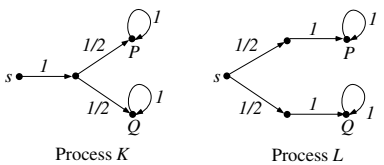


Fig. 2.

Note that $K, s \models \varphi$ but $L, s \not\models \varphi$. However, the probabilistic structures K_s and L_s are the same. Hence, unlike the truth value of logic of probability, the truth value of $pCTL^*$ formula depends not only on the probabilistic structure defined by the Finite Probabilistic Process but also on the 'branching structure' of this process. Therefore there is no direct, meaning preserving translation from $pCTL^*$ to *monadic* logic of probability.

In the rest of this section we show that even on the class of models restricted to Finite Probabilistic Processes no $pCTL^*$ formula is equivalent to the probabilistic formula $\exists t\ \mathbf{Prob}_{\geq 1} Q(t)$ where Q is a probabilistic predicate symbol. More precisely,

Theorem 4 *Let $\varphi = \exists t\ \mathbf{Prob}_{\geq 1} Q(t)$ where Q is a probabilistic predicate symbol. There is no $pCTL^*$ formula ψ such that for every Finite Probabilistic Process M and every state s of M one has $M_s \models \varphi$ iff $M, s \models \psi$.*

Consider the Finite Probabilistic Processes $\mathcal{K}_{m,n}$ and $\mathcal{K}_m$ for $m \geq 1$ and $n \geq 1$ as shown in Figure 3. Edges (i, j) are labeled by probabilities $P(i, j)$. Process $\mathcal{K}_m$ contains only one state (state s_m) labeled by the probabilistic predicate Q, other states have empty labels and process $\mathcal{K}_{m,n}$ contains only two states (states s_m and t_n) labeled by the probabilistic predicate Q. Let us call Π_m the unique infinite path starting in s in $\mathcal{K}_m$.

Lemma 5 *(1) For every $pCTL^*$ path formula g, there exists an integer $r \geq 1$ such that for every $m \geq r$, $\mathcal{K}_m, \Pi_m \models g$ iff $\mathcal{K}_r, \Pi_r \models g$.*

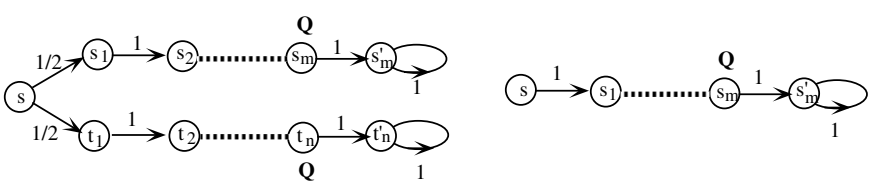


Fig. 3. $\mathcal{K}_{m,n}$ and $\mathcal{K}_m$

(2) For every pCTL state formula f, there exists an integer $r \geq 1$ such that for every $m, n \geq r$, $\mathcal{K}_{m,n}, s \models f$ iff $\mathcal{K}_{r,r}, s \models f$.*

Proof. The proof is by induction on the complexity of g and f. □

Finally we are ready to prove Theorem 4.

Proof of Theorem 4. Let us suppose that such a $pCTL^*$ formula ψ exists. Using Lemma 5, there exists an integer $r \geq 1$ such that for every $m, n \geq r$, $\mathcal{K}_{m,n}, s \models \psi$ iff $\mathcal{K}_{r,r}, s \models \psi$. That contradicts the fact that $\mathcal{K}_{m,n}, s \models \varphi$ iff $m = n$. □

6 Conclusion and Further Results

Our main result is a description of a fragment of a second-order monadic logic of probability with decidable model-checking. An important and difficult open question is whether one can prove the decidability of model-checking for all values of probabilities, without exceptions. Another open question is to consider other domains such as the real domain or the tree domain instead of the set of integers. It would be of great interest in specification of real-time uncertain systems. Below some extensions of our results are described.

A. Probabilities 0 and 1. Probabilities 0 and 1 play an important role in many questions related to specification and verification. Some probability logics, e. g. [LS82], consider only probabilistic operators $\mathbf{Prob}_{=0}$ and $\mathbf{Prob}_{=1}$. Theorem 2 can be strengthened as follows

Theorem 5 *Given a Finite Probabilistic Process M, a state s_0 of M and a parametrized completely closed formula φ in the class $\mathcal{C}$ with m parameters, one can compute for each parameter p_i in φ a finite set P_i of rational values, such that $0, 1 \notin P_i$ and for each tuple $\alpha = (\alpha_1, \ldots, \alpha_m)$ where $\alpha_i \in \mathbb{Q} \setminus P_i$ for $i = 1, \ldots, m$ one can decide whether (M_s) satisfies φ_α.*

In particular, we obtain the following corollary

Corollary 1 *Given a Finite Probabilistic Process M, a state s_0 of M and a completely closed formula φ in the class $\mathcal{C}$ with all probability operators only of the form $\mathbf{Prob}_{=0}$ and $\mathbf{Prob}_{=1}$. It is decidable whether (M_s) satisfies φ.*

B. Many variables inside Prob. In the definition of class $\mathcal{C}$ we allow to apply probabilistic operators only to formulas with one free individual variable. This is not essential restriction. The results of section 4 can be extended to the case when **Prob** is applied to formulas with many free individual variables. However the proof of the decidability of this extended fragment is more subtle and will be given in the full version of the paper.

C. On nesting. In class $\mathcal{C}$ we disallow nesting of **Prob** operators. Below we sketch how the decidability result can be extended to formulas with nested **Prob**. The main step in the proof of Theorem 2 shows that over a probabilistic structure M_s described by a Finite Probabilistic Process, the formula

$\mathbf{Prob}_{>q}(\varphi(t))$ defines the set $S_q = \{n : M_s, n \models \mathbf{Prob}_{>q}(\varphi(t))\}$ which is ultimately periodic for all but finitely many q; the latter we call *exceptional* values and their complement *good* values. Now consider a nested formula of the form $\mathbf{Prob}_{>p_1}(\ldots \mathbf{Prob}_{>p_2}(\varphi)\ldots)$ with parameters p_1 and p_2. We can find a finite set of exceptional values for the innermost **Prob**. For each good value q_2 we can compute an ultimately periodic set which is definable also by a $WMLO$-formula and replace $\mathbf{Prob}_{>q_2}(\varphi)$ by this $WMLO$-formula. After the replacement we obtain an unnested formula ψ of the form $\mathbf{Prob}_{>p_1}(\ldots)$. Now we can proceed and find for ψ a finite set of exceptional values of p_1 (for fixed q_2). If q_2 is a good value for $\mathbf{Prob}_{>p_2}(\varphi)$ and q_1 is a good value for the corresponding ψ then we can compute the truth value of the formula $\mathbf{Prob}_{>q_1}(\ldots \mathbf{Prob}_{>q_2}(\varphi)\ldots)$. Thus the whole set of exceptional values of (p_1, p_2) may be infinite but it is 'very sparse', in particular it is nowhere dense.

References

[AH94] M. Abadi and J. Halpern. Decidability and expressiveness for first-order logic of probability. *Information and Computation*, 112(1):1–36, 1994. 307, 309

[ASB+95] A. Aziz, V. Singhal, F. Balarin, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In *Computer Aided Verification. Proceeding of CAV'95*, pages 155–165. Springer Verlag, 1995. Lect. Notes in Comput. Sci., vol. 939. 306, 317

[Büc60] J. R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik u. Grundlag. Math.*, (6):66–92, 1960. 315

[CY95] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42:857–907, 1995. 306, 314

[FH94] R. Fagin and J. Halpern. Reasoning about knowledge and probability. *J. of the Assoc. Comput. Mach.*, 41(2):340–367, 1994. 309

[FHM90] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87:1,2:78–128, 1990. 306, 307

[Gan77] F. R. (Feliks Ruvimovich) Gantmakher. *The Theory of Matrices*. Chelsea Pub. Co., New York, 1977. 311

[GHR94] D. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Clarendon Press, Oxford, 1994. 316

[Hal90] J. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46:311–350, 1990. 306, 307

[Han94] H. A. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1994. Series: "Real Time Safety Critical System", vol. 1. 306

[HJ94] H. A. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6(5):512–535, 1994. 306

[Hod93] W. A. Hodges . *Model Theory*. Cambridge University Press, Cambridge, 1993. 309

[KS60] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. D Van Nostad Co., Inc., Princeton, N. J., 1960. 311

[KSK66] J. G. Kemeny, J. L. Snell, and A. W. Knapp. *Denumerable Markov Chains*. D Van Nostad Co., Inc., Princeton, N. J., 1966. 312

[Kei85] H. J. Keisler. Probability quantifiers. In J. Barwise and S. S.Feferman, editors, *Model Theoretic Logics*, pages 509–556. Springer, 1985. 306

[LS82] D. Lehmann and S. Shelah. Reasoning about time and chance. *Information and Control*, 53(3):165–198, 1982. 306, 319

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 131–191. North-Holland, 1990.

Solving Pushdown Games with a Σ_3 Winning Condition

Thierry Cachat, Jacques Duparc, and Wolfgang Thomas

Lehrstuhl für Informatik VII, RWTH, D-52056 Aachen
{cachat,duparc,thomas}@informatik.rwth-aachen.de
Fax: (49) 241-80-22215

Abstract We study infinite two-player games over pushdown graphs with a winning condition that refers explicitly to the infinity of the game graph: A play is won by player 0 if some vertex is visited infinity often during the play. We show that the set of winning plays is a proper Σ_3-set in the Borel hierarchy, thus transcending the Boolean closure of Σ_2-sets which arises with the standard automata theoretic winning conditions (such as the Muller, Rabin, or parity condition). We also show that this Σ_3-game over pushdown graphs can be solved effectively (by a computation of the winning region of player 0 and his memoryless winning strategy). This seems to be a first example of an effectively solvable game beyond the second level of the Borel hierarchy.

1 Introduction

The theory of infinite two-person games, originally developed in descriptive set theory, has found enormous interest in recent years also in theoretical computer science. Whereas in the framework of set theory, the mere existence of winning strategies is the central question, the applications in computer science are concerned with algorithmic aspects. In the past ten years, this development led to interesting connections with the verification and automatic synthesis of reactive programs (see, e.g., [13, 16]). It turned out that central problems in the verification of state-based systems can be studied in the game theoretical framework (an example is the model-checking problem for the modal μ-calculus), and that the construction of discrete controllers can be viewed as the synthesis of winning strategies in certain infinite games.

The standard setting of these applications are the finite-state games. Here one deals with a finite game graph where each vertex is associated to one of the two players (called 0 and 1). A play is an infinite sequence of vertices which arises when a token is moved through the graph, where in each step the token is moved by the player to whom the current vertex is associated. The winning condition (say for player 0) is given by an automata theoretic acceptance condition applied to plays. A prominent example is the Muller condition which is specified by a family $\mathcal{F}$ of vertex sets and which requires that the vertices visited infinitely often in the considered play form a set in $\mathcal{F}$. The core result on finite-state games is the Büchi-Landweber Theorem ([2]). It says that for a game on a finite graph

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 322–336, 2002.
© Springer-Verlag Berlin Heidelberg 2002

with Muller winning condition one can compute the "winning region" of player 0 (i.e., the set of vertices from which player 0 has a winning strategy) and that the corresponding winning strategies are executable by finite automata. Many more results have been shown, in particular on the so-called parity games, where even memoryless strategies suffice [5, 14].

The Muller and parity winning conditions (as well as related ones like Rabin and Streett conditions) define sets of plays which are located at a very low level of the Borel hierarchy, namely in $B(\boldsymbol{\Sigma}_2)$, the Boolean closure of the Borel class $\boldsymbol{\Sigma}_2$. This restriction to winning conditions of low set theoretical complexity is justified by two reasons: First, most winning conditions which are motivated by practical applications (safety, liveness, assume-guarantee properties, fairness, etc.), and Boolean combinations thereof, all define sets in $B(\boldsymbol{\Sigma}_2)$. Secondly, by Büchi's and McNaughton's results on the transformation of monadic second-order logic formulas into deterministic Muller automata, any winning condition which is formalizable in linear time temporal logic or in monadic second-order logic (S1S) over infinite strings defines a $B(\boldsymbol{\Sigma}_2)$-set. (One transforms a logical formula φ into an equivalent deterministic Muller automaton, say with transition graph G_φ, and proceeds from a game graph G and a winning condition defined by φ to $G \times G_\varphi$ as game graph equipped with the Muller winning condition applied to the second components of vertices.) In this connection, Büchi claims in [3, p. 1173] as a general thesis that any set of ω-sequences with an "honestly finite presentation" (by some form of "finite-state recursion") belongs to $B(\boldsymbol{\Sigma}_2)$.

Recently, the Büchi-Landweber Theorem was extended to infinite game graphs, and in particular to the transition graphs of pushdown automata [10, 11, 16]. For example, it was shown by Walukiewicz [16] that parity games over pushdown graphs can be solved effectively. But the restriction to the parity condition is now only justifiable by pragmatic aspects, and it is well conceivable that higher levels of the Borel hierarchy are reachable by natural winning conditions exploiting the infinity of pushdown transition graphs.

In the present paper we propose such a winning condition, by the requirement that (in a winning play) there should be one vertex occurring infinitely often. Syntactically, this is formulated as a condition on a play ρ using a $\boldsymbol{\Sigma}_3$-prefix of unbounded quantifiers:

> "there is a vertex v such that for all time instances t there is $t' > t$ such that v is visited at t' in the play ρ under consideration"

In Section 3 below we show that for a suitable deterministic pushdown automaton the corresponding set of winning plays forms indeed a $\boldsymbol{\Sigma}_3$-complete set in the Borel hierarchy. The completeness proof needs some prerequisites of set theory, in particular on continuous reductions and the Wadge game [15]. In Section 2, these preparations are collected.

In Section 4 we show that the $\boldsymbol{\Sigma}_3$-winning condition does not prohibit an algorithmic solution of the corresponding games. Building on the approach of [4] for Büchi games over pushdown graphs, we present an algorithm to decide whether a given vertex of a pushdown transition graph is in the winning region of player 0; and from this, also a memoryless winning strategy can be extracted.

In the final section we discuss some related acceptance conditions (studied in ongoing work) which involve a specified set F of vertices and requires that some $v \in F$ is visited infinitely.

The main result of this paper may be considered as a first tiny step in a far-reaching proposal of Büchi ([3, p.1171-72]). He considers constructive game presentations by "state-recursions", as they arise in automata theoretic games, and he asks to extend the construction of winning strategies in the form of "recursions" (i.e., algorithmic procedures) from the case of $B(\boldsymbol{\Sigma}_2)$-games to appropriate games on arbitrary levels of the Borel hierarchy.

2 Borel Hierarchy and Wadge Game

Given a finite alphabet Σ, we consider the set Σ^ω of all infinite words over Σ as a topological space by equipping it with the Cantor topology, where the open sets are those of the form $W \cdot \Sigma^\omega$ for some set $W \subseteq \Sigma^*$ of finite words. The *finite Borel Hierarchy* is a sequence $\boldsymbol{\Sigma}_1, \boldsymbol{\Pi}_1, \boldsymbol{\Sigma}_2, \boldsymbol{\Pi}_2, \ldots$ of classes of ω-languages over Σ, inductively defined by:

- $\boldsymbol{\Sigma}_1 = \{\textit{Open sets}\} \;=\; \{W \cdot \Sigma^\omega : \; W \subseteq \Sigma^*\}$
- $\boldsymbol{\Pi}_n = \left\{ A^{\complement} : \; A \in \boldsymbol{\Sigma}_n \right\} \quad (\text{for } n \geq 1)$
- $\boldsymbol{\Sigma}_{n+1} = \left\{ \bigcup_{i \in \mathbb{N}} A_i : \; \forall i \in \mathbb{N} \;\; A_i \in \boldsymbol{\Pi}_n \right\} \quad (\text{for } n \geq 0)$

Let $B(\boldsymbol{\Sigma}_n)$ be the class of Boolean combinations of $\boldsymbol{\Sigma}_n$-sets. The Borel classes are arranged in the form

$$\begin{tikzcd}[column sep=small]
\boldsymbol{\Sigma}_1 \arrow[dr] & & \boldsymbol{\Sigma}_2 \arrow[dr] & & \\
& B(\boldsymbol{\Sigma}_1) \arrow[ur] \arrow[dr] & & B(\boldsymbol{\Sigma}_2) \arrow[ur] \arrow[dr] & \cdots \\
\boldsymbol{\Pi}_1 \arrow[ur] & & \boldsymbol{\Pi}_2 \arrow[ur] & &
\end{tikzcd}$$

where each arrow denotes strict inclusion. A set that is in $\boldsymbol{\Sigma}_k$ but not in $\boldsymbol{\Pi}_k$ is called a *true-*$\boldsymbol{\Sigma}_k$-set. (For background see e.g. [9].)

Recall that a function $\phi : \; \Sigma_A^\omega \longrightarrow \Sigma_B^\omega$ is continuous if every inverse image of an open set is open. In other words, for any $W_B \subseteq \Sigma_B^*$ there exists some $W_A \subseteq \Sigma_A^*$ such that $x \in W_A \cdot \Sigma^\omega \iff \phi(x) \in W_B \cdot \Sigma^\omega$. Now, given $A \subseteq \Sigma_A^\omega$ and $B \subseteq \Sigma_B^\omega$, we say A continuously reduces to B (denoted $A \leq_W B$ since originally studied by Wadge [15]) if there is a continuous mapping ϕ such that $x \in A \iff \phi(x) \in B$. This ordering should be regarded as a measure of topological complexity. Intuitively $A \leq_W B$ means that A is less complicated than B with regard to the topological structure.

One among many properties about this ordering is that for each integer n, if A is $\boldsymbol{\Sigma}_n$-*complete* (i.e. both $A \in \boldsymbol{\Sigma}_n$ and $B \leq_W A$ holds for all $B \in \boldsymbol{\Sigma}_n$), then A is a true $\boldsymbol{\Sigma}_n$-set, which means it does not belong to $\boldsymbol{\Pi}_n$.

The main device in working with this measure of complexity 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 1 (Wadge game) *Given $A \subseteq \Sigma_A^\omega$, $B \subseteq \Sigma_B^\omega$, $\mathbf{W}(A\ ,\ B)$ is an infinite two player game between players I and II where players take turns, I plays letters in Σ_A, and II plays finite words over the alphabet Σ_B. At the end of an infinite play (in ω moves), I has produced an ω-sequence $x \in \Sigma_A^\omega$ of letters and II has produced an ω-sequence of finite words which concatenated give rise to a finite or ω-word $y \in \Sigma_B^* \cup \Sigma_B^\omega$. The winning condition on the resulting play, denoted here $x\hat{}y$, is the following:*

$$\text{II wins the play } x\hat{}y \iff_{def} y \text{ is infinite } \wedge\ (x \in A \longleftrightarrow y \in B)$$

Proposition 2 ([15]) *II has a winning strategy in $\mathbf{W}(A\ ,\ B) \iff A \leq_W B$.*

Example 3 Consider the set $\mathbb{J}$ of all infinite words over the alphabet $\{0,1\}$ that have infinitely many 0. We show that $\mathbb{J}$ is $\boldsymbol{\Pi}_2$-complete.
To verify $\mathbb{J} \in \boldsymbol{\Pi}_2$ we note that the complement $\mathbb{J}^c$ belongs to $\boldsymbol{\Sigma}_2$:

$$x \notin \mathbb{J} \iff x \in \bigcup_{n\in\mathbb{N}} \{0,1\}^n \cdot 1^\omega\ .$$

Note that $\{0,1\}^n \cdot 1^\omega$ is the complement of $\{0,1\}^n \cdot \{0,1\}^* \cdot 0 \cdot \{0,1\}^\omega$ and hence a $\boldsymbol{\Pi}_1$-set, whence $\bigcup_{n\in\mathbb{N}}\{0,1\}^n \cdot 1^\omega$ is a $\boldsymbol{\Sigma}_2$-set. To show $\boldsymbol{\Pi}_2$-completeness, let A be any set in $\boldsymbol{\Pi}_2$, $A = \bigcap_{n\in\mathbb{N}} W_n \cdot \Sigma^\omega$, with $W_n \subseteq \Sigma^*$. We describe a winning strategy for player II in the game $\mathbf{W}(A\ ,\ \mathbb{J})$:
Set $i := 0$,
do
  if I's current position u does not have any prefix in W_i,
  then play the letter 1, i remains the same,
  else play the letter 0, $i := i + 1$,
od

Clearly, this strategy is winning for II since it induces an infinite word y that contains infinitely many 0 if and only if the infinite word x played by I belongs to each and every open set $W_n \cdot \Sigma^\omega$; hence $x \in A \iff y \in \mathbb{J}$.

3 Pushdown Automata with a $\boldsymbol{\Sigma_3}$-Acceptance Condition

We consider *deterministic* pushdown automata of the form $\mathcal{P} = (\Sigma,\ \Gamma,\ Q,\ \delta,\ q_i)$, where Σ is the finite input alphabet, Γ is the finite stack alphabet, Q is the set of control states, q_i is the initial state, and δ is the partial transition function from $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ to $Q \times \Gamma^*$ with the usual restriction on choice between ε-move and Σ-moves (for all $q \in Q$ and $\alpha \in \Gamma$, either $\delta(q,\varepsilon,\alpha)$ is undefined and $\forall a \in \Sigma\ \ \delta(q,a,\alpha)$ is defined, or $\delta(q,\varepsilon,\alpha)$ is defined and $\forall a \in \Sigma\ \ \delta(q,a,\alpha)$ is undefined). A configuration (or "global state") is a pair $(q,w) \in Q \times \Gamma^*$, often written as the word qw, consisting of control state q and stack content w.

Given $a \in \Sigma \cup \{\varepsilon\}$; $q, q' \in Q$; $\mu, \nu \in \Gamma^*$; $\alpha \in \Gamma$; we write $a:\ (q, \alpha\cdot\mu) \underset{\mathcal{P}}{\vdash\!\!-\!\!-} (q', \nu\cdot\mu)$ if $\delta(q,a,\alpha) = (q',\nu)$. Finally we denote the transitive closure of $\underset{\mathcal{P}}{\vdash\!\!-\!\!-}$ by $\overset{*}{\underset{\mathcal{P}}{\vdash\!\!-\!\!-}}$.

So $u:\ (q,\nu)\overset{*}{\underset{\mathcal{P}}{\vdash}}(q',\nu')$ holds if the input word u leads $\mathcal{P}$ from the configuration (q,ν) to (q',ν').

Let us equip these pushdown automata with the following acceptance condition: $\mathcal{P}$ accepts $x \in \Sigma^\omega$ iff

$$\exists q \in Q\ \exists \mu \in \Gamma^*\ \forall n\ \exists\ m > n \quad x \restriction m:\ (q_i,\bot)\overset{*}{\underset{\mathcal{P}}{\vdash}}(q,\mu)\ ,$$

(where $x \restriction m$ is the initial segment of x up to position m) and let $L(\mathcal{P})$ be the set of words $x \in \Sigma^\omega$ accepted by $\mathcal{P}$. To say it in words, x is accepted if there is a configuration that occurs infinitely many times while reading x. Or, considering the fact both Q and Γ are finite, a word x is accepted by $\mathcal{P}$ iff, while reading x, for some n the stack content goes back infinitely many times to a word of length n.

By its very definition, it is easy to see that $L(\mathcal{P})$ belongs to $\boldsymbol{\Sigma}_3$: let $A_{q,\mu,n}$ denote the set of finite words u of length precisely n such that, after reading u (from the initial configuration), $\mathcal{P}$ is in configuration (q,μ). We have

$$L(\mathcal{P}) = \underbrace{\bigcup_{\substack{q\in Q\\ \mu\in\Gamma^*}} \underbrace{\bigcap_{n\in\mathbb{N}} \underbrace{\bigcup_{k\in\mathbb{N}} \underbrace{A_{q,\mu,n+k}\cdot\Sigma^\omega}_{\in\boldsymbol{\Sigma}_1\cap\boldsymbol{\Pi}_1}}_{\in\boldsymbol{\Sigma}_1}}_{\in\boldsymbol{\Pi}_2}}_{\in\boldsymbol{\Sigma}_3}$$

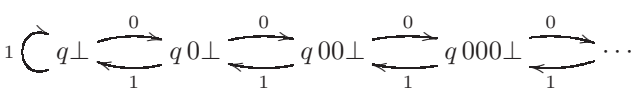
Let us verify that this representation cannot be improved w.r.t. nesting of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Pi}$.

Proposition 4 *There exists a DPDA $\mathcal{P}$ such that $L(\mathcal{P})$ is $\boldsymbol{\Sigma}_3$-complete.*

Proof of proposition 4: We consider a DPDA $\mathcal{P}$ which adds a "0" on top of the stack when it reads a 0, and when it reads 1 it deletes one letter, unless the stack is already empty, in which case it does nothing. Formally, let $\mathcal{P} = (\Sigma,\ \Gamma,\ Q,\ \delta,\ q)$ be the DPDA defined by $\Sigma = \{0,1\}$, $\Gamma = \{\bot, 0\}$, $Q = \{q\}$, and δ fixed as follows:

- $\delta(q,0,\bot) = (q, 0\cdot\bot)$
- $\delta(q,1,\bot) = (q,\bot)$
- $\delta(q,0,0) = (q, 0\cdot 0)$
- $\delta(q,1,0) = (q,\epsilon)$

The figure shows the configuration graph of $\mathcal{P}$:



In order to prove that $L(\mathcal{P})$ is $\boldsymbol{\Sigma}_3$-complete, we need to show that for any $A \in \boldsymbol{\Sigma}_3$ the relation $A \leq_W L(\mathcal{P})$ holds. For this purpose, let A be a subset of Σ^ω such that $A = \bigcup_{n\in\mathbb{N}} A_n$ where each A_n belongs to $\boldsymbol{\Pi}_2$. Let $\mathbb{J}$ be the $\boldsymbol{\Pi}_2$-complete set defined above in Example 3. For each n, let σ_n be a winning strategy for II in the game $\mathbf{W}(A_n\ ,\ \mathbb{J})$. Let also $\phi:\ \mathbb{N} \longrightarrow \mathbb{N}\times\mathbb{N}$ be any bijection that satisfies $\phi(k) = (n,m) \Longrightarrow m \leq k$. We describe a winning strategy for II in $\mathbf{W}(A\ ,\ L(\mathcal{P}))$. We write $x_0, x_1, x_2, \ldots$ for the letters chosen by I and $y_0, y_1, y_2, \ldots$ for the finite words chosen by II.
Assume $\phi(k) = (n,m)$. Then player II's k^{th} move y_k is defined as follows:

- if $\sigma_n(x_0, x_1, \ldots, x_m)$ contains the letter 0, then y_k is the *shortest* sequence of 0 or 1 such that

$$y_0 \cdot y_1 \cdot \cdots \cdot y_k :\ (q, \bot) \overset{*}{\underset{\mathcal{P}}{\vdash\!\!\!-}} (q, 0^n \cdot \bot)$$

- if $\sigma_n(x_0, x_1, \ldots, x_m)$ does not contain 0, then $y_k = 0$

This strategy is well defined since $m \leq k$ always holds, therefore $(x_0, x_1, \ldots, x_m)$ is a subsequence of $(x_0, x_1, \ldots, x_k)$. This strategy is winning for II because, if I and II have respectively played x and y, we can we verify that $x \in A$ iff $y \in L(\mathcal{P})$ as follows:

If $x \in A$ then $x \in A_n$ for some n. Since σ_n is winning for II in $\mathbf{W}(A_n\ ,\ \mathbb{J})$, there exist infinitely many m such that $\sigma_n(x_0, x_1, \ldots, x_m)$ contains 0. Therefore, by construction, the word $0^n \cdot \bot$ appears infinitely many times as stack content. Thus $y \in L(\mathcal{P})$.

If $x \notin A$ then $x \notin A_n$ holds for any n. Since σ_n is winning for II in $\mathbf{W}(A_n\ ,\ \mathbb{J})$, there exist only finitely many m such that $\sigma_n(x_0, x_1, \ldots, x_m)$ contains 0. So, for each integer n let k_n be the smallest integer such that

$$\forall k \geq k_n \forall i \leq n\ \ \forall m \in \mathbb{N}\ \Big(\phi(k) = (i,m)\ \Longrightarrow\ \sigma_i(x_0, x_1, \ldots, x_m) \text{ contains no } 0\Big).$$

By construction, after $k+n$ moves, no word $0^i \cdot \bot$ for any $i \leq n$ will appear as stack content. This shows that any configuration of $\mathcal{P}$ occurs only finitely many times, hence $y \notin L(\mathcal{P})$. □

In the next section we are more interested in the set $\mathcal{R}(\mathcal{P})$ of successful runs of $\mathcal{P}$ than in $L(\mathcal{P})$. Let us note that also $\mathcal{R}(\mathcal{P})$ is a true $\boldsymbol{\Sigma}_3$-set:

Proposition 5 *Let $\mathcal{P}$ be as in the preceding proposition. Then the set $\mathcal{R}(\mathcal{P}) \subseteq (Q \cdot \Gamma^*)^\omega$ of accepting runs of $\mathcal{P}$ is $\boldsymbol{\Sigma}_3$-complete.*

Proof of proposition 5: It is easy to see that $\mathcal{R}(\mathcal{P}) \in \boldsymbol{\Sigma}_3$; see the explanation of the acceptance condition in the introduction. In order to verify $\boldsymbol{\Sigma}_3$-completeness, consider the function $\phi : \Sigma^\omega \longrightarrow (Q \cdot \Gamma^*)^\omega$ which associates to $x \in \Sigma^\omega$ the $\mathcal{P}$-run ρ_x on x. Obviously, ϕ is continuous (one does not even need the Wadge game to verify this), and we have $x \in L(\mathcal{P}) \Longleftrightarrow \rho_x \in \mathcal{R}(\mathcal{P})$. Thus $L(\mathcal{P}) \leq_W \mathcal{R}(\mathcal{P})$. □

It should be noted that for nondeterministic pushdown automata the situation is much different: As shown by Finkel [6, 7], nondeterministic pushdown automata equipped with the Büchi acceptance condition can recognize Borel-sets of any finite rank and even non-Borel sets.

4 Effective Solvability

4.1 Outline

In the present section we use pushdown automata for the specification of infinite games (between two players 0 and 1) rather than for the definition of ω-languages. The acceptance condition considered in the previous section is now employed as a winning condition for Player 0. Our aim is to show that for any such pushdown game one can compute the winning region of Player 0 (the set of those configurations from which Player 0 can force a win) and, moreover, a positional winning strategy.

Let us first introduce the game-theoretic setting. A pushdown game graph is specified by a variant of the pushdown automata considered in the previous section, which we call *pushdown game systems*. The input alphabet Σ and the initial state q_0 are canceled, but a partition $Q = Q_0 \uplus Q_1$ of the state set Q into sets Q_0, Q_1 is introduced. Note that by the deletion of Σ the transitions become unlabeled, and thus there is not a deterministic transition function any more but a transition relation: A *pushdown game system* (PDS) is of the form $\mathrm{P} = (\Gamma, Q_0, Q_1, \Delta)$, where Γ is the finite stack alphabet, $Q = Q_0 \uplus Q_1$ the finite state set, and $\Delta \subseteq Q \times \Gamma \times Q \times \Gamma^*$ the finite transition relation. Of course, given a pushdown game system one may obtain a normal DPDA by introducing an initial state and a sufficiently large input alphabet Σ, which would allow to regain a deterministic (partial) transition function.

A pushdown game system P determines a *pushdown game graph* $G_{\mathrm{P}} = (V, E)$ with vertex set $V = Q\Gamma^*$ and the edge set E consisting of the pairs $(p\gamma\mu, q\nu\mu) \in V \times V$ such that $(p, \gamma, q, \nu) \in \Delta$. Define $V_0 = Q_0\Gamma^*$ and $V_1 = Q_1\Gamma^*$. A *play* over (V, E) from $v \in V$ is a sequence $u_0, u_1, u_2, \cdots$ built up by the two players 0,1 as follows: We have $u_0 = v$; given $u_i \in V_0$, Player 0 chooses u_{i+1} such that $(u_i, u_{i+1}) \in E$, and given $u_i \in V_1$, Player 1 chooses u_{i+1} with $(u_i, u_{i+1}) \in E$. The play is won by Player 0 iff

$$\text{there is a configuration from } V \text{ that appears infinitely often in the play,} \quad (1)$$

equivalently, iff for some length n a configuration of length n is visited infinitely often. Our aim is to compute the set W_0 of winning positions of Player 0: the positions from which he can win whatever Player 1 does.

As a preparatory step, we recall the definition of winning regions of somewhat simpler games: reachability games, where Player 0 has to reach a configuration of a given "target set" T just *once* in order to win, and Büchi games where Player 0 has to ensure that infinitely often configurations in T are visited. We recall the corresponding definitions (see, e.g., [13]) which rely on the fact that

our game graphs are of bounded degree. Given a set $T \subseteq V$, the 0-attractor of T is the set of configurations from which Player 0 can force the play to reach T. It is inductively defined by:

$$\begin{aligned} Attr_0^0(T) &= T \ , \\ Attr_0^{i+1}(T) &= Attr_0^i(T) \cup \big\{u \in V_0 \mid \exists v, (u,v) \in E,\ v \in Attr_0^i(T)\big\} \\ &\quad \cup \big\{u \in V_1 \mid \forall v, (u,v) \in E \Rightarrow v \in Attr_0^i(T)\big\} \ , \\ Attr_0(T) &= \textstyle\bigcup_{i \in \mathbb{N}} Attr_0^i(T) \ . \end{aligned}$$

Here $Attr_0^i(T)$ is the set of configurations from which Player 0 can force a visit in T in at most i steps. If we slightly modify the definition, we get $Attr_0^+(T)$: the set of configurations from which Player 0 can force the play to reach T in *at least* one move, whatever Player 1 does.

$$\begin{aligned} X_0(T) &= \emptyset \ , \\ X_{i+1}(T) &= X_i(T) \cup \{u \in V_0 \mid \exists v, (u,v) \in E,\ v \in T \cup X_i(T)\} \\ &\quad \cup \{u \in V_1 \mid |u| > 1,\ \forall v, (u,v) \in E \Rightarrow v \in T \cup X_i(T)\} \ , \\ Attr_0^+(T) &= \textstyle\bigcup_{i \geqslant 0} X_i(T) \ . \end{aligned}$$

For technical reasons concerning the definition of $Attr_0^+(T)$, it is convenient to allow deadlocks by the empty stack in the game graph and to declare here Player 1 as the winner of any play terminating with empty stack.

We are now able to define $\text{Büchi}_0(T)$, the set of those configurations from which Player 0 can force to reach T infinitely many times (to win the "Büchi game for T"):

$$\begin{aligned} \text{Büchi}_0^0(T) &= V \ , \\ \text{Büchi}_0^{i+1}(T) &= Attr_0^+(\text{Büchi}_0^i(T) \cap T) \ , \\ \text{Büchi}_0(T) &= \textstyle\bigcap_{i \in \mathbb{N}} \text{Büchi}_0^i(T) \ . \end{aligned}$$

We note $\Gamma^{\leqslant M}$ the language $\{\epsilon\} \cup \Gamma^1 \cup \cdots \cup \Gamma^M$. The effective solution of pushdown games with winning condition (1) is based on the following straightforward representation of the winning region W_0 of player 0:

Proposition 6 *Over a game graph induced by a pushdown game system, the winning region W_0 of Player 0 w.r.t. winning condition (1) is* $W_0 = \bigcup_{M>0} \textit{Büchi}_0(Q\Gamma^{\leqslant M})$.

Let us refine this into an algorithmic description of W_0. In [4] it is shown that if the set T is regular (the configurations of the pushdown game graph are considered as words), then one can compute a finite automaton recognizing $Attr_0(T)$, respectively $Attr_0^+(T)$, which hence are again regular. Using the regularity of $Attr_0^+(T)$ one can compute a finite automaton recognizing $\text{Büchi}_0(T)$. Of course $\Gamma^{\leqslant M}$ is regular for $M \geqslant 0$, so $\text{Büchi}_0(Q\Gamma^{\leqslant M})$ can be computed. To compute the set W_0 of Proposition 6, we finally have to overcome the problem that W_0 is an infinite union. We shall prove that

$$W_0 = Attr_0(\text{Büchi}_0(Q\Gamma^{\leqslant N})\Gamma^*)$$

where $N = 1 + |\Gamma||Q| \max\{|\nu| - 1 \mid (p, \gamma, q, \nu) \in \Delta\}$. The idea is that if Player 1 can make the stack increase by more than N letters, then he can make it increase indefinitely (without returning to previous stack contents an unbounded number of times) and thus wins.

4.2 Details

We first recall the constructions of [4]. Given a regular set T of configurations, it is recognized by a finite automaton A_T over the alphabet $Q \uplus \Gamma$. Then a finite construction, originally presented in [1] in the framework of alternating pushdown systems, transforms A_T into $\mathrm{A}_{Attr(T)}$, an *alternating* finite automaton that recognizes $Attr_0(T)$. The state space remains the same during the construction, the algorithm just adds new transitions. By an obvious modification of the algorithm, it is possible to construct a finite automaton $\mathrm{A}_{Attr+(T)}$, recognizing $Attr_0^+(T)$.

We describe here the format of these automata and explain how to use them for the construction of an automaton recognizing $\text{Büchi}_0(T)$. The automata to recognize sets of configurations are alternating finite word automata with a special convention about initial states: Given a PDS $\mathrm{P} = (\Gamma, Q_0, Q_1, \Delta)$, a P-*automaton* is a finite automaton $\mathrm{A} = (P, \Gamma, \longrightarrow, Q, F)$, where $P \supseteq Q$ is its finite set of states, $\longrightarrow \subseteq P \times (\Gamma \cup \{\epsilon\}) \times 2^P$ the set of transitions, $Q \subseteq P$ the set of initial states (note that these are the control locations of P), and $F \subseteq P$ a set of final states. A transition $r \xrightarrow{\gamma} S$ indicates a move from state r via letter $\gamma \in \Gamma$ simultaneously to all states of S, i.e. by a universal branching of runs. Existential branchings are captured by nondeterminism. (So, a transition like $r \xrightarrow{\gamma} (r_1 \wedge r_2) \vee (r_3 \wedge r_4)$ is represented here by *two* transitions $r \xrightarrow{\gamma} \{r_1, r_2\}$ and $r \xrightarrow{\gamma} \{r_3, r_4\}$.) For each $p \in P$ and $w \in \Gamma^*$, the automaton A accepts a configuration $pw \in Q\Gamma^*$ iff there exists a successful A-run on w from the initial state p. Successful runs are defined in the standard way, using computation trees for the representation of simultaneously active states; the acceptance condition requires that some computation tree exists which at every leaf ends in a final state. By $q \xrightarrow{w}_* S$ we indicate that such a computation tree exists on input qw such that its leaf states form the set S.

Let us explain the transformation of a P-automaton A recognizing T into a P-automaton recognizing $\text{Büchi}_0(T)$. We consider the case $T = Q\Gamma^{\leqslant M}$ for a given number M and set

$$Y_0^M = Q\Gamma^{\leqslant M} \ , \ Y_{i+1}^M = Attr_0^+(Y_i^M) \cap Q\Gamma^{\leqslant M} \ , \text{ and } \ Y_\infty^M = \bigcap_{i \geqslant 0} Y_i^M \ .$$

Then $\text{Büchi}_0(Q\Gamma^{\leqslant M}) = Attr_0(Y_\infty^M)$.

In the sequel the relation E is written in infix-notation with the symbol "$\hookrightarrow$": so we have $(u, v) \in E \iff u \hookrightarrow v$ and also $(p, \gamma, q, \nu) \in \Delta \iff p\gamma \hookrightarrow q\nu$. Consider the PDS $\mathrm{P} = (\Gamma, Q_0, Q_1, \Delta)$ with $Q = Q_0 \cup Q_1$. The construction of the automaton recognizing Y_∞^M starts with a P-automaton B_0 which recognizes $Q\Gamma^{\leqslant M}$: its state set is $Q \cup \{f_0, \cdots, f_M\}$, with transitions $f_i \xrightarrow{\Gamma} f_{i+1}$ for $i < M$,

each f_i being a final state, and the states of $Q \cup \{f_0\}$ are merged into a unique state named f_0, *i.e.*, f_0 is initial.

In stages or "generations" $i = 1, 2, 3, \cdots$ new copies of Q are added. We write (q, i) or short q^i for the copy of a node $q \in Q$ added in stage i. So the state space will by a subset of $(Q \times \mathbb{N}) \cup \{f_0, \ldots, f_M\}$ (where $q^0 = f_0$ for all $q \in Q$). We write Q^i for the set $Q \times \{i\}$. Two auxiliary operations are needed which refer to this indexing by stages:

Definition 7 *For a finite set $S \subseteq (Q \times \mathbb{N}) \cup \{f_0, \ldots, f_M\}$ let*

$$\phi(S) = \{q^i \mid q^{i+1} \in S\} \ \cup \ (S \cap \{f_0, \cdots, f_M\}) \ ,$$

with the convention that q^0 is f_0 for all q.

Definition 8 *For $i > 0$ and a set $S \subseteq (Q \times [1, i]) \cup \{f_0, \cdots, f_M\}$, let*

$$\pi^i(S) = \{q^i \mid \exists i \geqslant k > 0, q^k \in S\} \cup (S \cap \{f_0, \cdots, f_M\}) \ .$$

This is the projection of the set S on the generation i (except for $\{f_0, \cdots, f_M\}$).

Algorithm 9 *To compute an automaton recognizing Y_∞^M*
Input: *PDS* $\mathrm{P} = (\Gamma, Q_0, Q_1, \Delta)$ *and* $M > 0$
Output: *a* P*-automaton* C *that recognizes* Y_∞^M

Initialization: Set $\mathrm{C} := \mathrm{B}_0$ *recognizing* $Q\Gamma^{\leqslant M} = Z_0$*, with states* q^0 *(for* $q \in Q$*) and* $f_0, \ldots, f_M$*, where for all* $q \in Q$*,* q^0 *is set to be* f_0*. (Recall that for all* $\gamma \in \Gamma, f_i \xrightarrow{\gamma} f_{i+1}$*, and the* f_i*'s are the final states.)*
$i := 0$.

repeat
$i := i + 1$ *(i is number of the current generation)*
Add the states q^i*, for each* $q \in Q$*, using them as initial states.*
Add an ϵ*-transition from* q^i *to* q^{i-1} *for each* $q \in Q$
{*obtain an automaton still recognizing* Z_{i-1} }
Add new transitions to C *by the saturation procedure presented in [4]:*
repeat
(Player 0) if $p \in Q_0$, $p\gamma \hookrightarrow q\mu \in \Delta$ *and* $q^i \xrightarrow{\mu}_* S$ *in the current automaton,* then *add a new transition* $p^i \xrightarrow{\gamma} S$.
(Player 1) if $p \in Q_1$, $\{p\gamma \hookrightarrow q_1\mu_1, \cdots, p\gamma \hookrightarrow q_n\mu_n\}$ *are* all *the* Δ*-rules (game moves) starting from* $p\gamma$ *and* $\forall k$, $q_k^i \xrightarrow{\mu_k}_* S_k$ *in the current automaton,* then *add a new transition* $p^i \xrightarrow{\gamma} \bigcup_k S_k$.
until *no new transition can be added*
{ *the obtained automaton recognizes* $Attr_0(Z_{i-1})$ }
remove the ϵ*-transitions.*
{ *obtain* B'_i *recognizing* $Attr_0^+(Z_{i-1}) = Z'_i$ }
replace each transition $q^i \xrightarrow{\gamma} S$ *by* $q^i \xrightarrow{\gamma} \pi^i(S)$.
{ *obtain* B''_i *recognizing* $Z''_i \subseteq Z'_i$ }
replace each transition $q^i \xrightarrow{\gamma} S$ *by* $q^i \xrightarrow{\gamma} S \cup \{f_0\}$

{ *obtain* B_i *recognizing* $Z_i'' \cap Q\Gamma^{\leqslant M} = Z_i$, *we have* $\bigcap_{i\geq 0} Y_i^M \subseteq Z_i$ }
set $\mathrm{C} := \mathrm{B}_i$, *finishing generation number* i
until $i > 1$ *and* $\forall p, \gamma :\ p^i \xrightarrow{\gamma} S \iff p^{i-1} \xrightarrow{\gamma} \phi(S)$.

Note that we can erase the q^{i-1}'s and their transitions as soon as the generation i is done. To compare successive generations we have the following property.

Proposition 10 *In Algorithm 9, for all* $\nu \in \Gamma^*, q \in Q, i \geqslant 1$ *we have*

$$q^{i+1} \xrightarrow{\nu}_* S \Rightarrow q^i \xrightarrow{\nu}_* \phi(S)$$

The proofs of this proposition and of the following theorem are similar to the corresponding claims in [4]; for completeness they are given in the appendix. Note that because of the projection π, the transitions $q^i \xrightarrow{\nu}_* S$ verify $S \subseteq (Q \times \{i\}) \cup \{f_0, \cdots, f_M\}$. Note also that no new transition from the states $f_0, \cdots, f_M$ is added.

Theorem 11 *The automaton* C *constructed in Algorithm 9 recognizes* Y_∞^M.

It remains to eliminate the quantification on M implicit in $\bigcup_{M>0} \mathrm{Büchi}_0(Q\Gamma^{\leqslant M})$, by choosing a sufficiently large bound for M. We introduce an ordering relation which permits to compare transitions.

Definition 12 *For any* $S, S' \subseteq Q^i \cup \{f_0, \cdots, f_M\}$,

$$S \sqsubseteq S' \ \Leftrightarrow \begin{cases} S \cap Q^i \ \subseteq \ S' \cap Q^i \ \textit{and} \\ \max(\{j \mid f_j \in S\} \cup \{-1\}) \leqslant \max(\{j \mid f_j \in S'\} \cup \{-1\}) \end{cases}$$

The idea is that in case $S \sqsubseteq S'$, one recognizes "more" after a transition $q^i \longrightarrow S$ than after a transition $q^i \longrightarrow S'$. To compare transitions $q^i \longrightarrow S$ and $q^j \longrightarrow S'$, with $i < j$, one considers $\pi_j(S)$ and S' with respect to $\sqsubseteq$. The index j of $f_j \in S$ measures the possibility for Player 1 to increase the length of the stack, and possibly win.

Proposition 13 *In the automaton* C *constructed by Algorithm 9, assume that for a transition* $q^i \xrightarrow{\gamma} S$ *we have* $S \sqsubseteq S'$ *for each transition* $q^i \xrightarrow{\gamma} S'$ *from the same state. If* $\ell = \max\{j \mid f_j \in S\} \geqslant 0$, *then from the configuration* $q\gamma$, *Player 1 can reach a configuration where the length of the stack is at least* ℓ.

Proof of proposition 13: Induction on the number of transitions constructed by the algorithm. Note that the projection π^i does not change the value of ℓ. If $\ell = 0$, the property is trivially true. □

We consider now $N = 1 + |\Gamma||Q| \max\{|\mu| - 1 \mid \exists\, p\gamma \hookrightarrow q\mu \in \Delta\}$. The rightmost factor is the maximal number of letters that can be added to the stack in one move.

Proposition 14 *In the automaton* C *constructed by Algorithm 9, assume again that for a transition* $q^i \xrightarrow{\gamma} S$ *we have* $S \sqsubseteq S'$ *for each transition* $q^i \xrightarrow{\gamma} S'$ *from the same state. If* $\ell = \max\{j \mid f_j \in S\} \geqslant N$, *then from configuration* $q\gamma$, *Player 1 can win the game by increasing the stack indefinitely.*

Proof of proposition 14: According to the previous proposition, Player 1 can ensure the stack increases by at least ℓ letters. Using a classical pumping argument (see e.g. [8]), there exists $(q, \alpha) \in Q \times \Gamma$ such that, during this process, two different configurations $q\alpha\nu$ and $q\alpha\xi\nu$ are met ($\nu \in \Gamma^*, \xi \in \Gamma^+$), and the letters of ν and ξ are not scanned (nor changed) any more in the stack after these configurations. This proves that continuing from $q\alpha\xi\nu$, Player 1 can force the stack to increase indefinitely. This shows that a configuration in $q\gamma\Gamma^*$ cannot be in the winning region W_0 of player 0. □

It follows from the proposition that in C we can eliminate transitions $q^i \longrightarrow S$, such that $f_j \in S, j > N$.

Corollary 15 *For all $M \geqslant N$, $Y_\infty^N \cdot \Gamma^* = Y_\infty^M \cdot \Gamma^*$.*

Proof of corollary 15: The inclusion from left to right is clear. For the other inclusion, the automaton recognizing $Y_\infty^M \Gamma^*$ "contains" that of $Y_\infty^N \Gamma^*$. It has possibly some other transitions $q^i \longrightarrow S$, with $f_j \in S, j > N$, which verify the hypotheses of Proposition 14. Those transitions do not permit to accept a configuration in W_0, *i.e.*, no winning play from such a configuration is possible. But clearly $Y_\infty^M \Gamma^* \subseteq \bigcup_{M>0} \text{Büchi}_0(Q\Gamma^{\leqslant M}) \subseteq W_0$ (a play from Y_∞^M is also possible from $Y_\infty^M \Gamma^*$. See Proposition 6). □

Theorem 16 *Given a pushdown game system, one can compute a finite automaton recognizing the winning region*

$$W_0 = Attr_0(Y_\infty^N \Gamma^*)$$

of Player 0 w.r.t. the $\boldsymbol{\Sigma}_3$-winning condition (1).

Proof of theorem 16: Clearly $Attr_0(Y_\infty^N \Gamma^*) \subseteq W_0$. Proposition 6 states that

$$W_0 = \bigcup_{M>0} \text{Büchi}_0(Q\Gamma^{\leqslant M}) \ ,$$

which is, by the preceding porposition,

$$\bigcup_{M>0} Attr_0(Y_\infty^M) \subseteq \bigcup_{M>0} Attr_0(Y_\infty^M \Gamma^*) \subseteq Attr_0(Y_\infty^N \Gamma^*) \ .$$

□

The construction of an automaton recognizing $W_0 = Attr_0(Y_\infty^N \Gamma^*)$ works as follows: one uses Algorithm 9 with $M = N$. The resulting automaton C recognizes Y_∞^N. Now one merges the states f_k to a unique final state f, and one adds a transition $f \xrightarrow{\gamma} f$ for all $\gamma \in \Gamma$, in order to obtain an automaton which recognizes $Y_\infty^N \Gamma^*$. To recognize $Attr_0(Y_\infty^N \Gamma^*)$ we just need another application of the saturation procedure as it appears in Algorithm 9, which finally results in an (alternating) automaton C′ which recognizes W_0.

Following the constructions of [4], it is easy to extract a (positional) winning strategy for player 0 on the set W_0. The choice of an appropriate transition from a game graph vertex $qw \in W_0$ is done by analyzing an accepting run of the automaton C' on the input qw. For the details we have to refer to [4].

5 Discussion and Concluding Remarks

The $\boldsymbol{\Sigma}_3$-acceptance condition considered above was introduced as an example, illustrating the possibility to reach higher levels of the Borel hierarchy than $B(\boldsymbol{\Sigma}_2)$. For applications in ω-language theory a more general form is appropriate, referring to a set $F \subseteq Q\Gamma^*$: Call a DPDA-run ρ accepting if

$$\exists w \in F \ \forall i \ \exists j \geqslant i \ \rho(j) = w \ . \tag{2}$$

If F is finite, then this condition is equivalent to

$$\forall i \ \exists j \geqslant i \ \exists w \in F \ \rho(j) = w \ , \tag{3}$$

i.e., to the usual Büchi acceptance condition. In order to define an interesting class of ω-languages including true $\boldsymbol{\Sigma}_3$-sets, it is necessary to combine the acceptance conditions (2) and (3). Note that condition (2) alone does not allow to simulate condition (3): For example, the ω-language over $\{0, 1, \$\}$ which contains:
– all ω-words over $\{0, 1\}$, and
– the ω-words $u\$u^{\mathrm{R}}x$ with $u \in \{0, 1\}^*$ and arbitrary $x \in \{0, 1, \$\}^*$
is recognizable by a DPDA with the Büchi acceptance condition (3) but not definable with acceptance condition (2).

How can one reach even higher levels of the Borel hierarchy than just $\boldsymbol{\Sigma}_3$? A natural idea is to require infinitely many configurations, each of them being visited infinitely often, as accepting condition:

$$\forall j \ \exists q\mu \in Q\Gamma^j\Gamma^* \ \forall n \ \exists \ m > n \quad x \restriction m : \ (q_i, \bot) \overset{*}{\underset{\mathcal{P}}{\vdash}} (q, \mu). \tag{4}$$

Remarkably, this condition comes down to a $\boldsymbol{\Sigma}_3$ condition: it is logically equivalent to the conjunction of our $\boldsymbol{\Sigma}_3$ condition (one configuration is visited infinitely often) and the condition that the stack growth is unbounded:

$$\exists q\mu \in Q\Gamma^* \ \forall n \ \exists \ r, s, t > n \ \exists q'\mu' \in Q\Gamma^s$$
$$\left(x \restriction r : \ (q_i, \bot) \overset{*}{\underset{\mathcal{P}}{\vdash}} (q, \mu) \quad \wedge \quad x \restriction t : \ (q_i, \bot) \overset{*}{\underset{\mathcal{P}}{\vdash}} (q', \mu') \right) \ .$$

Let us modify (4) by moving slightly the occurrence of control state in the formula:

$$\forall j \ \forall q \in Q \ \exists \mu \in \Gamma^j\Gamma^* \ \forall n \ \exists \ m > n \quad x \restriction m : \ (q_i, \bot) \overset{*}{\underset{\mathcal{P}}{\vdash}} (q, \mu). \tag{5}$$

In other words, if we call q-configuration the words of the form $q\mu \in \{q\}\Gamma^*$, we deal with the condition:

for all state q there exists infinitely many
q-configurations that are visited infinitely often

This can be shown to be a $\boldsymbol{\Pi}_4$-acceptance condition which does not collapse to $\boldsymbol{\Sigma}_3$: it leads to true Π_4 sets. The same holds for the closely related condition

there exists some state q such that there exists infinitely
many q-configurations that are visited infinitely often

or even for this very same condition with a fixed state q.

Acknowledgment

We thank the referees for useful remarks.

References

[1] A. Bouajjani, J. Esparza, and O. Maler, Reachability analysis of pushdown automata: Application to model-checking, CONCUR '97, LNCS 1243, pp 135-150, 1997. 330

[2] J. R. Büchi, Landweber L. H., *Solving sequential conditions by finite-state strategy.* Transactions of the American Mathematical Society vol. 138 (1969) 295–311. 322

[3] J. R. Büchi, State-strategies for games in $F_{\sigma\delta} \cap G_{\delta\sigma}$ J. Symbolic Logic 48 (1983), no. 4, 1171–1198. 323, 324

[4] T. Cachat, Symbolic strategy synthesis for games on pushdown graphs, in: ICALP'02, Springer LNCS (to appear). `http://www-i7.informatik.rwth-aachen.de/~cachat/` 323, 329, 330, 331, 332, 334

[5] E. A. Emerson and C. S. Jutla, *Tree automata, mu-calculus and determinacy*, FoCS '91, IEEE Computer Society Press (1991), pp. 368–377. 323

[6] O. Finkel, Topological properties of omega context-free languages, *Theoret. Comput. Sci.* 262 (2001), no. 1-2, 669–697. 328

[7] O. Finkel, Wadge hierarchy of omega context-free languages, *Theoret. Comput. Sci.* 269 (2001), no. 1-2, 283–315. 328

[8] J. E. Hopcroft and J. D. Ullman, *Formal Languages and their relation to automata*, Addison-Wesley, 1969. 333

[9] A. S. Kechris, *Classical descriptive set theory*, Graduate texts in mathematics, vol 156, Springer Verlag (1994). 324

[10] O. Kupferman and M. Y. Vardi, An Automata-Theoretic Approach to Reasoning about Infinite-State Systems, CAV 2000, LNCS 1855, 2000. 323

[11] S. Seibert, *Effektive Strategiekonstruktionen für Gale-Stewart-Spiele auf Transitionsgraphen*, Technical Report 9611, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Germany, July 1996. 323

[12] C. Stirling, *Modal and Temporal Properties of Processes*, Springer (Texts in Computer Science), 2001.

[13] W. Thomas, On the synthesis of strategies in infinite games, STACS '95, LNCS 900, pp. 1–13, 1995. 322, 328

[14] W. Thomas, Languages, automata, and logic, in *Handbook of Formal Language Theory* (G. Rozenberg, A. Salomaa, Eds.), Vol 3, Springer-Verlag, Berlin 1997, pp. 389–455. 323

[15] W. W. Wadge *Reducibility and Determinateness on the Baire Space* Ph.D. Thesis, University of California, Berkeley, 1984. 323, 324, 325

[16] I. Walukiewicz, Pushdown processes: games and model checking, CAV '96, LNCS 1102, pp 62-74, 1996. Full version in Information and Computation 157, 2000. 322, 323

Partial Fixed-Point Logic on Infinite Structures

Stephan Kreutzer

LuFG Mathematische Grundlagen der Informatik, RWTH Aachen
kreutzer@informatik.rwth-aachen.de

Abstract. We consider an alternative semantics for partial fixed-point logic (PFP). To define the fixed point of a formula in this semantics, the sequence of stages induced by the formula is considered. As soon as this sequence becomes cyclic, the set of elements contained in every stage of the cycle is taken as the fixed point. It is shown that on finite structures, this fixed-point semantics and the standard semantics for PFP as considered in finite model theory are equivalent, although arguably the formalisation of properties might even become simpler and more intuitive. Contrary to the standard PFP semantics which is only defined on finite structures the new semantics generalises easily to infinite structures and transfinite inductions. In this generality we compare - in terms of expressive power - partial with other known fixed-point logics. The main result of the paper is that on arbitrary structures, PFP is strictly more expressive than inflationary fixed-point logic (IFP). A separation of these logics on finite structures would prove PTIME different from PSPACE.

1 Introduction

Logics extending first-order logic by fixed-point constructs are well studied in finite model theory. Introduced in the early eighties, it soon became clear that there are tight connections between the various forms of fixed-point logics and such important complexity classes as polynomial time and space. This relationship is made precise in the results by Immerman [Imm86] and Vardi [Var82] that, on finite ordered structures, *least fixed-point logic* (LFP) provides a logical characterisation of polynomial time computations in the sense that a class of finite ordered structures is decidable in polynomial time if, and only if, it is definable in LFP. Other complexity classes such as polynomial or logarithmic space can also be characterised in this way, using different fixed-point logics. Since the discovery of these results, fixed-point logics play a fundamental role in finite model theory, arguably even more important than first-order logic itself. We give precise definitions of these logics in Section 2. See [EF99] for an extensive study of fixed-point logics on finite structures. A survey that also treats infinite structures can be found in [DG02].

The best known of these logics is least fixed-point logic (LFP), which extends first-order logic (FO) by an operator to form least fixed-points of positive formulae (which define monotone operators.) But there are other fixed-point logics. Besides fragments of LFP, such as transitive closure logic and existential

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 337–351, 2002.
© Springer-Verlag Berlin Heidelberg 2002

or stratified fixed-point logic, which all have in common that they form fixed points of monotone operators, there are also fixed-point logics that allow the use of non-monotone operators. One such logic is the *inflationary fixed-point logic* (IFP), which allows the definition of inflationary fixed points of arbitrary formulae. It is the simplest logic allowing non-monotone operators, as it is still equivalent to LFP (see [GS86, Kre02].)

As mentioned above, on finite ordered structures, LFP and IFP capture PTIME. To characterise complexity classes above PTIME, like PSPACE for instance, a more liberal notion of fixed points has to be used. One such logic that is likely to be more expressive than IFP is *partial fixed-point logic*, where there are no restrictions on the formulae used within the fixed point operator. Thus it is no longer guaranteed that the sequence of stages induced by such a formula reaches a fixed point. However, if it does, this fixed point is taken as the semantics of the formula. Otherwise, i.e. if the sequence does not become stationary, the result is defined as being empty.

It has been shown by Abiteboul and Vianu [AV91a] that partial fixed-point logic provides a precise characterisation of PSPACE on finite ordered structures. Thus, showing that there are properties of finite ordered structures definable in PFP but not in IFP would yield a separation of polynomial time and space. However, on unordered structures, neither IFP nor PFP can express all of PTIME. For instance, it is easy to see that it cannot be decided in PFP whether a finite set is of even cardinality, a problem that from a complexity point of view is extremely simple. It is therefore remarkable that a separation of PTIME and PSPACE follows even from a separation of IFP and PFP on arbitrary finite structures, not necessarily being ordered. This result is due to Abiteboul and Vianu [AV91b]. See also [Daw93].

Theorem. PTIME = PSPACE *if, and only if,* IFP = PFP.

There are also fixed-point logics capturing the complexity classes NP and EXPTIME, namely *non-deterministic* and *alternating non-inflationary fixed-point logic* (see [AVV97].) For these logics, similar theorems as above have been shown. Thus, the most important questions in complexity theory, the separation of complexity classes, have direct analogues in logic, namely in the comparison of the expressive power of various fixed-point logics. A profound understanding of the nature and limits of the various kinds of fixed-point operators is therefore important and necessary. In this line of research, the main contribution of this paper is to introduce a semantics for partial fixed-point logic that is equivalent to the standard semantics on finite structures, but contrary to the standard semantics, is also well defined on infinite structures. On infinite structures, we will then be able to compare partial and inflationary fixed-point logic and show that there are properties definable in PFP which are not definable in IFP. Thus, IFP is strictly contained in PFP. We also argue that the alternative semantics for PFP allows a more intuitive formulation of queries than the standard semantics.

2 Preliminaries

In this section we present the basic definitions for the explorations in the later sections. Let τ be a signature and $\mathfrak{A} := (A, \tau)$ be a τ-structure with universe A. Let $\varphi(R, \overline{x})$ be a first-order formula with free variables $\overline{x}$ and a free relation symbol R not occurring in τ. The formula φ defines an operator

$$\begin{aligned} F_\varphi : \mathcal{P}(A) &\longrightarrow \mathcal{P}(A) \\ R &\longmapsto \{\overline{a} : (\mathfrak{A}, R) \models \varphi[\overline{a}]\}. \end{aligned}$$

A fixed point of the operator F_φ is any set R such that $F_\varphi(R) = R$. Clearly, as φ is arbitrary, the corresponding operator F_φ need not to have any fixed points at all. For instance, the formula $\varphi(R, \overline{x}) := \neg\forall\overline{y}\, R\overline{y}$ defines the operator F_φ mapping any set $R \subsetneqq A^k$ to A^k and the set A^k itself to the empty set. Thus F_φ has no fixed points. However, if the class of admissible formulae is restricted, the existence of fixed points can be guaranteed. One such restriction is to require that the formulae are positive in the fixed-point variable. As positiveness implies monotonicity, an operator F_φ defined by a positive formula φ always has fixed points, in fact even a least fixed point $\mathbf{lfp}(F_\varphi) := \bigcap\{P : F_\varphi(P) = P\}$. This forms the basis of the most common fixed-point logic, the least fixed-point logic.

To obtain more general logics, i.e. logics allowing non-monotone operators also, one has to consider suitable semantics to guarantee the existence of meaningful fixed-points. The simplest such logic is the *inflationary fixed-point logic*.

Definition 2.1 (Inflationary Fixed-Point Logic). *Inflationary fixed-point logic* (IFP) is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \overline{x})$ is a formula with free first-order variables $\overline{x} := x_1, \ldots, x_k$ and a free second-order variable R of arity k, then

$$\psi := [\mathbf{ifp}_{R,\overline{x}}\, \varphi](\overline{t})$$

is also a formula, where $\overline{t}$ is a tuple of terms of the same length as $\overline{x}$. The free variables of ψ are the variables occurring in $\overline{t}$ and the free variables of φ other than $\overline{x}$.

Let $\mathfrak{A}$ be a structure with universe A providing an interpretation of the free variables of φ other than $\overline{x}$. Consider the following sequence of sets induced by φ on $\mathfrak{A}$.

$$\begin{aligned} R^0 &:= \emptyset \\ R^{\alpha+1} &:= R^\alpha \cup F_\varphi(R^\alpha) \\ R^\lambda &:= \bigcup_{\beta<\lambda} R^\beta \qquad \text{for limit ordinals } \lambda. \end{aligned}$$

The sets R^α are called the *stages* of the induction on φ and $\mathfrak{A}$. Clearly the sequence of stages is increasing and thus leads to a fixed point R^∞. For any tuple $\overline{a} \in A$,

$$\mathfrak{A} \models [\mathbf{ifp}_{R,\overline{x}}\, \varphi](\overline{a}) \text{ if, and only if, } \overline{a} \in R^\infty.$$

As usual, we also allow simultaneous fixed-point formulae, i.e. formulae of the form $\psi(\overline{x}) := [\mathbf{ifp}\ R_i : S](\overline{x})$, where

$$S := \begin{cases} R_1\overline{x}_1 \leftarrow \varphi_1(R_1, \ldots, R_k, \overline{x}_1) \\ \qquad\vdots \\ R_k\overline{x}_k \leftarrow \varphi_k(R_1, \ldots, R_k, \overline{x}_k) \end{cases}$$

is a system of formulae. Each formula φ_i in S induces an operator $F_{\varphi_i} : \mathrm{Pow}(A)^{r_1} \times \cdots \times \mathrm{Pow}(A)^{r_k} \to \mathrm{Pow}(A)^{r_i}$, taking sets $R_1, \ldots, R_k$ of appropriate arity to the set $\{\overline{a} : (\mathfrak{A}, R_1, \ldots, R_k) \models \varphi_i[\overline{a}]\}$, where the r_i denote the arities of the relations R_i. The stages of an induction on such a system S of formulae are now k-tuples of sets defined by

$$\begin{aligned} R_i^0 &:= \emptyset \\ R_i^{\alpha+1} &:= R_i^\alpha \cup F_{\varphi_i}(R_1^\alpha, \ldots, R_k^\alpha) \\ R_i^\lambda &:= \bigcup_{\beta<\lambda} R_i^\beta \qquad \text{for limit ordinals } \lambda. \end{aligned}$$

The formula ψ is true for a tuple $\overline{a}$ of elements interpreting the variables $\overline{x}$ if, and only if, $\overline{a} \in R_i^\infty$, where R_i^∞ denotes the i-th component of the simultaneous fixed point of the system S. Simultaneous inductions can easily be eliminated in favour of simple inductions by increasing the arity of the involved fixed-point variables (See [EF99].)

Proposition 2.2. *Any formula in* IFP *with simultaneous inductions is equivalent to a formula without simultaneous inductions.*

Nevertheless, formulae making use of simultaneous inductions are often much simpler to read than the equivalent simple formulae and we will extensively use simultaneous inductions in the sequel.

3 Partial Fixed-Point Logic

In this section we introduce *partial fixed-point logic*, which in some sense is the most general fixed-point extension of first-order logic. We first define the syntax, which is the same as for IFP, except that we write **pfp** for the fixed-point operator.

Definition 3.1 (Partial Fixed-Point Logic - Syntax). Partial fixed-point logic (PFP) *is defined as the extension of first-order logic by the following formula building rule. If $\varphi(R, \overline{x})$ is a formula with free first-order variables $\overline{x} := x_1, \ldots, x_k$ and a free second-order variable R of arity k, then*

$$\psi := [\mathbf{pfp}_{R,\overline{x}}\, \varphi](\overline{t})$$

is also a formula, where $\overline{t}$ is a tuple of terms of the same length as $\overline{x}$. The free variables of ψ are the variables occurring in $\overline{t}$ and the free variables of φ other than $\overline{x}$.

Having defined the syntax, we now turn to the definition of the semantics. We first present the standard definition of partial fixed-point semantics as common in finite model theory.

Definition 3.2 (Finite Model Semantics). *Let $\psi := [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{t})$ be a formula and let $\mathfrak{A}$ be a finite structure with universe A providing an interpretation of the free variables of φ other than $\overline{x}$. Consider the following sequence of stages induced by φ on $\mathfrak{A}$.*

$$R^0 := \emptyset$$
$$R^{\alpha+1} := F_\varphi(R^\alpha)$$

As there are no restrictions on φ, this sequence need not reach a fixed point. In this case, ψ is equivalent on $\mathfrak{A}$ to false. Otherwise, if the sequence becomes stationary and reaches a fixed point R^∞, then for any tuple $\overline{a} \in A$,

$$\mathfrak{A} \models [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{a}) \text{ if, and only if, } \overline{a} \in R^\infty.$$

Again we allow simultaneous inductions and as with IFP these can always be eliminated in favour of simple inductions. This semantics for PFP is standard in finite model theory and the basis of the results mentioned in the introduction. However, actually writing a formula in this logic is sometimes unnecessarily complicated. This is demonstrated by an example for *modal* partial fixed-point logic. The example is taken from [DK] where also more on modal partial fixed-point logic can be found.

We briefly recall the definition of modal logic and its extension by partial fixed-point operators. Modal logics are interpreted on *transition systems*, also called *Kripke structures*, which are edge and node labelled graphs. The labels of the edges come from a set $\mathcal{A}$ of actions, whereas the nodes are labelled by sets of propositions from a set $\mathcal{P}$.

Modal logic (ML) is built up from atomic propositions $p \in \mathcal{P}$ using boolean connectives $\wedge$, $\vee$, and $\neg$ and the so-called *next-modalities* $\langle a \rangle$, $[a]$ for each $a \in \mathcal{A}$. Formulae $\varphi \in$ ML are evaluated at a particular node in a transition system. We write $\mathcal{K}, v \models \varphi$ if φ holds at the node v in the transition system $\mathcal{K} := (V, (E_a)_{a\in\mathcal{A}}, (p)_{p\in\mathcal{P}})$. The semantics of ML-formulae is as usual with $\mathcal{K}, v \models p$, for $p \in \mathcal{P}$, if $v \in p^{\mathcal{K}}$, $\mathcal{K}, v \models \langle a \rangle\varphi$ if there is an a-successor u of v such that $\mathcal{K}, u \models \varphi$ and, dually, $\mathcal{K}, v \models [a]\varphi$ if for all a-successors u of v, $\mathcal{K}, u \models \varphi$. Now modal partial fixed-point logic (MPC) is defined analogously to PFP, i.e. formulae $\psi := [\mathbf{pfp}\ P : \varphi(P)]$ are allowed defining the set of elements in the partial fixed point of φ.

Consider the following problem, known as the *unary trace- or language equivalence problem*. It is defined as the problem of deciding whether two given finite automata over an unary alphabet accept the same language. This is formalised as follows. The input is a directed, rooted graph. The root is labelled by w and is not reachable from any other node in the graph. Further, there are disjoint subgraphs rooted at successors of the root. In each subgraph some nodes are marked as final states, e.g. coloured by a colour f, whereas the other nodes are

not coloured at all. Two subgraphs rooted at successors of the root are *trace equivalent*, if for each $n < \omega$, whenever in one of the graphs there is a path of length n from the root to a final state such a path also exists in the other.

We aim at defining in MPC the class $\mathcal{C}$ of structures as above such that all subgraphs rooted at successors of the root are trace equivalent. A simple idea to formalise this is the following. Consider the formula ψ defined as

$$\psi := [\mathbf{pfp}\ Z : \begin{array}{l} X \leftarrow (f \wedge \neg Y) \vee \Diamond X \\ Y \leftarrow f \\ Z \leftarrow (w \wedge \Diamond X \wedge \Diamond \neg X) \vee Z \end{array}].$$

In the first stage, X contains all final states, i.e. those labelled by f. In the successive stages, those elements are selected, which have a successor in X. Thus, the stage X^n contains exactly those elements from which there is a path of length $n-1$ to a node labelled by f. The variable Y is only used to ensure that the nodes labelled by f are added to X only once at the beginning, so that the induction is not started over and over again. Now, the root of the structure is added to Z if, for some n, in one subgraph there is a path of length n from its root to a final state but not in the other. Obviously, once the root is added to Z, it stays in forever. Thus, ψ is true at the root if, and only if, the subgraphs rooted at its successors are not trace equivalent. However, if at least one of the sub-structures is cyclic, the induction on X never becomes stationary and thus, by definition, the fixed point is empty. To rescue the formula, we have to think about some way to guarantee that the induction process becomes stationary although the only information we are interested in, namely whether the root eventually occurs in Z, is independent of this.

This suggest a different way to define partial fixed-point inductions. Consider the sequence of induction stages defined by ψ. Obviously, this sequence must eventually become cyclic. Now consider the set of elements that occur in all stages of this cycle and take this as the defined fixed point[1]. Applying this idea to the example above, we get that the fixed point of X becomes empty (unless there are self loops), the fixed point of Y contains all final states, and the fixed point of Z contains the root just in case there are two successors of it which are not trace equivalent. Thus, $\neg\psi$ is true in $\mathcal{K}, v$ if, and only if, $\mathcal{K}, v \in \mathcal{C}$. This motivates an alternative semantics for partial fixed-point logic based on these ideas.

Besides this problem of formalising properties, the standard semantics for PFP has the disadvantage that it does not generalise to infinite structures. For instance, as the sequence of stages induced by PFP-formulae is not necessarily increasing, it makes no sense to define limit stages as the union of the previous stages as in IFP. Therefore, so far partial fixed-point logic has only been considered on finite structures.

The drawback of this is that it also restricts the possibilities to study PFP and its properties and to compare it to other logics to finite structures. As

[1] Note that this set does not necessarily has to be a fixed point. Nevertheless we use this name to keep consistent with the other fixed-point logics.

mentioned in the introduction, the relationship between the various fixed-point logics is closely related to important complexity theoretical questions and thus a profound understanding of what the logics can and can not do is necessary and important. To achieve a better understanding of the logics, their properties on infinite structures might prove useful for the study on finite structures also. This is the second motivation for considering an alternative semantics for PFP, namely to give a semantics that generalises to infinite structures and transfinite inductions.

We are now ready to formally define a general semantics for partial fixed-point logic.

Definition 3.3 (General Semantics). *Let* $\psi := [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{t})$ *be a formula and let* $\mathfrak{A}$ *be a structure with universe* A *providing an interpretation of the free variables of* φ *other than* $\overline{x}$*. Consider the following sequence of stages induced by* φ *on* $\mathfrak{A}$*.*

$$\begin{aligned} R^0 &:= \emptyset \\ R^{\alpha+1} &:= F_\varphi(R^\alpha) \\ R^\lambda &:= \mathrm{final}((R^\alpha)_{\alpha<\lambda}) \qquad \textit{for limit ordinals } \lambda, \end{aligned}$$

where $\mathrm{final}((R^\alpha)_{\alpha<\lambda})$ *denotes the set of elements* $\overline{a}$ *such that there is a* $\beta < \lambda$ *and for all* $\beta < \gamma < \lambda$, $\overline{a} \in R^\gamma$*.*

Obviously, the sequence $(R^\alpha)_{\alpha\in\mathrm{Ord}}$ *must eventually become cyclic. Let* $\beta_1 < \beta_2$ *be minimal such that* $R^{\beta_1} = R^{\beta_2}$*. Then, for any tuple* $\overline{a} \in A$*,*

$$\mathfrak{A} \models [\mathbf{pfp}_{R,\overline{x}}\,\varphi](\overline{a}) \textit{ if, and only if, } \overline{a} \in R^\gamma \textit{ for all } \beta_1 \leq \gamma < \beta_2.$$

We also allow simultaneous inductions and again the proof that this does not increase the expressive power is straight forward.

Theorem 3.4. *Any formula in* PFP *under the general semantics with simultaneous inductions is equivalent to a formula without simultaneous inductions.*

According to the definition, the fixed point of a formula φ is defined as the set of elements which occur in every stage of the first cycle in the sequence of stages induced by φ. Note that this is not equivalent to saying that the fixed point consists of those elements $\overline{a}$ such that there is a stage β and $\overline{a}$ occurs in all stages greater than β. For instance, consider a structure $\mathfrak{A} := (\{0,1,2,3\})$ and the formula defining an operator taking $\emptyset \mapsto \{0,1\}$, $\{0,1\} \mapsto \{0,2\}$ and $\{0,2\} \mapsto \{0,1\}$. Further, it takes $\{0\} \mapsto \{2\}$ and $\{2\}$ to itself. Now consider the induction stages $(R^\alpha)_{\alpha\in\mathrm{Ord}}$ induced by this operator. Clearly, for all $0 < n < \omega$, $R^n = \{0,1\}$ if n is odd and $R^n = \{0,2\}$ if n is even. Thus, the partial fixed point as defined above is $\{0\}$. However, $R^\omega = \{0\}$ and for all $\alpha > \omega$, $R^\alpha = \{2\}$. Thus, defining the fixed point as the set of elements which are contained in all stages greater than some β yields a different set than the partial fixed point as defined above.

We now prove that in the restriction to finite structures both semantics, i.e. the semantics in Definition 3.2 and 3.3 are equivalent.

Notation. *To distinguish between the two semantics, we denote* PFP *under the finite model semantics as* $\mathrm{PFP}_{\mathrm{fin}}$ *and write the operator as* $\mathbf{pfp}^f$. *We write* $\mathrm{PFP}_{\mathrm{gen}}$ *and* $\mathbf{pfp}^g$ *whenever we speak about the general semantics. Further, if* φ *is any formula in* PFP, *we write* $\mathit{fin}(\varphi)$ *to denote the formula under the finite model semantics and* $\mathit{gen}(\varphi)$ *for the general semantics.*

We first prove a technical lemma that establishes the main step for the proof of the theorem below.

Lemma 3.5. *Let* $\varphi(R,\overline{x})$ *be a formula in* $\mathrm{PFP}_{\mathrm{gen}}$ *and* $\mathfrak{A}$ *be a structure. There is a formula* $\text{fixed-point}_\varphi(R,\overline{x})$ *depending on* φ *such that for any stage* R^α *of the induction on* φ *and* $\mathfrak{A}$ *and all* $\overline{a} \in A$,

$$(\mathfrak{A}, R^\alpha) \models \text{fixed-point}_\varphi[\overline{a}] \qquad \textit{iff} \qquad \begin{array}{l}\textit{there are } \beta < \gamma \leq \alpha \textit{ such that } (R^\xi)_{\beta \leq \xi \leq \gamma} \\ \textit{is a cycle, i.e. } R^\beta = R^\gamma, \textit{ and } \overline{a} \in \varphi^\infty.\end{array}$$

Further, if $\mathfrak{A}$ *is finite and* $\varphi \in \mathrm{PFP}_{\mathrm{fin}}$, *then* $\mathit{fin}(\text{fixed-point}_\varphi) \equiv \mathit{gen}(\text{fixed-point}_\varphi)$, *i.e. the result of* $\text{fixed-point}_\varphi$ *under the finite model and the general semantics is the same.*

Proof. Consider the formula $\textit{fixed-point}_\varphi(R,\overline{x}) := [\mathbf{pfp}\ Q_2 : S](\overline{x})$, where S is defined as

$$S := \begin{cases} Q\overline{x} \ \leftarrow \varphi(Q,\overline{x}) \\ Q_1\overline{x} \leftarrow (Q_1 = \emptyset \wedge Q = R \wedge R\overline{x}) \vee Q_1\overline{x} \\ Q_2\overline{x} \leftarrow Q_2\overline{x} \vee (Q_1 \neq \emptyset \wedge Q = R \wedge \\ \qquad\qquad [\mathbf{pfp}\ Z' : \begin{array}{l} Z \ \leftarrow (Z = \emptyset \wedge \varphi(R,\overline{x})) \vee (Z = R \wedge R\overline{x}) \vee \\ Z' \leftarrow (Z' = \emptyset \wedge Q_1\overline{x}) \vee (Z' \neq \emptyset \wedge Z'\overline{x} \wedge Z\overline{x}) \end{array} \begin{array}{l} \\ (Z \neq \emptyset \wedge Z \neq R \wedge \varphi(Z,\overline{x})) \end{array}](\overline{x})) \end{cases}$$

In the course of the induction on S, the variable Q runs through the stages of φ. The first time where $Q = R$, i.e. the stage R is reached, Q_1 is initialised to R. If there is another stage in the induction on Q such that $Q = R$, i.e. if the induction on φ becomes cyclic the first time, Q_2 gets all elements which are contained in all stages between the two occurrences of R. Thus, the fixed point Q_2^∞ contains exactly the elements of the fixed point of φ. □

We are now ready to prove the equivalence of the two partial fixed-point semantics defined above.

Theorem 3.6. *On finite structures,* $\mathrm{PFP}_{\mathrm{fin}}$ *and* $\mathrm{PFP}_{\mathrm{gen}}$ *are equivalent, i.e. for every* PFP*-formula under the finite model semantics there is an equivalent* PFP*-formula under the general semantics and vice versa.*

Proof. The forth direction follows easily by induction on the structure of the formula. In the main step, let $\psi := [\mathbf{pfp}^f_{R,\overline{x}}\, \varphi(R,\overline{x})](\overline{t})$ be a formula in $\mathrm{PFP}_{\mathrm{fin}}$. It is equivalent to the formula

$$\psi^g := [\mathbf{pfp}^g\ Q : \begin{array}{l} R\overline{x} \leftarrow \varphi^g(R,\overline{x}) \\ Q\overline{x} \leftarrow \forall\overline{x}(\varphi^g(R,\overline{x}) \leftrightarrow R\overline{x}) \wedge R\overline{x}. \end{array}](\overline{t})$$

where φ^g is a $\mathrm{PFP}_{\mathrm{gen}}$-formula equivalent to φ. By induction, such a formula always exists. Assume first that a fixed point of φ is reached on a structure $\mathfrak{A}$. In this case, both semantics are equivalent for trivial reasons and thus $\psi \equiv \psi^g$. Now assume that the fixed point of φ does not exist. Then at no stage $\forall \overline{x}(\varphi^g(R, \overline{x}) \leftrightarrow R\overline{x})$ becomes true and thus ψ^g defines the empty set.

The other direction is also proved by induction on the structure of the formulae. In the main step, assume that $\psi := [\mathbf{pfp}^g_{R,\overline{x}}\, \varphi(R, \overline{x})](\overline{t})$ is a formula under the general semantics. By induction, φ is equivalent to a formula φ^f in $\mathrm{PFP}_{\mathrm{fin}}$. Then, $[\mathbf{pfp}^g_{R,\overline{x}}\, \varphi^g(R, \overline{x})]\overline{t}$ is equivalent to

$$\psi^f := [\mathbf{pfp}^f\ Q : \begin{array}{l} R\overline{x} \leftarrow \varphi^f(R, \overline{x}) \\ Q\overline{x} \leftarrow \textit{fixed-point}_{(\varphi^f)}(R, \overline{x}) \end{array}]\overline{t}$$

By Lemma 3.5, the formula $\textit{fixed-point}_{(\varphi^f)}(R)$ can be chosen from $\mathrm{PFP}_{\mathrm{fin}}$. Thus, as $\varphi^f \in \mathrm{PFP}_{\mathrm{fin}}$, we get that ψ^f is itself a formula in $\mathrm{PFP}_{\mathrm{fin}}$. The equivalence of ψ^f and ψ is an immediate consequence of Lemma 3.5. □

The theorem allows us to transfer the results on $\mathrm{PFP}_{\mathrm{fin}}$ mentioned in the introduction, in particular the theorems by Abiteboul, Vianu, Immerman, and Vardi to $\mathrm{PFP}_{\mathrm{gen}}$. Thus, we immediately get the following corollary.

Corollary 3.7.

(i) $\mathrm{PFP}_{\mathrm{gen}}$ *has* PSPACE *data-complexity and captures* PSPACE *on ordered structures.*
(ii) $\mathrm{PFP}_{\mathrm{gen}} = \mathrm{IFP}$ *on finite structures if, and only if,* PTIME = PSPACE.
(iii) On finite structures, every $\mathrm{PFP}_{\mathrm{gen}}$ *formula is equivalent to a formula with only one application of a fixed-point operator.*

Proof. The corollary follows immediately from the fact that every $\mathrm{PFP}_{\mathrm{fin}}$ formula is equivalent to one with only one fixed-point operator and that the translation of $\mathrm{PFP}_{\mathrm{fin}}$-formulae to $\mathrm{PFP}_{\mathrm{gen}}$-formulae as presented in the proof of Theorem 3.6 does not increase the number of fixed-point operators. □

Using a diagonalisation argument as in Section 4 below, it is clear that for any fixed-point logic like LFP, IFP, or PFP, the alternation or the nesting depth hierarchy must be strict on arbitrary structures, i.e. allowing the nesting of fixed-point operators or the alternation of fixed-point operators and negation must strictly increase the expressive power. Thus, Part (*iii*) of the preceding corollary fails on infinite structures. We close the section by establishing a negation normal form for $\mathrm{PFP}_{\mathrm{gen}}$ formulae. Thus, the alternation of fixed points and negation does not provide more expressive power than just nesting fixed-points.

Theorem 3.8. *Every* $\mathrm{PFP}_{\mathrm{gen}}$ *formula is equivalent to one where negation occurs only in front of atoms.*

Proof. The proof follows easily using the formula defined in Lemma 3.5. However, we present a general proof for this that also works for IFP and shows that for

these logics the concept of negated fixed points does not add anything to the expressive power.

Let $\psi(\overline{t}) := \neg[\mathbf{pfp}_{R,\overline{x}}\varphi(R,\overline{x})](\overline{t})$ be a formula in PFP. Obviously, it is equivalent to the formula

$$\psi'(\overline{t}) := \exists 0 \exists 1\ [\mathbf{pfp}\ Q : \begin{array}{l} P\overline{x}y \leftarrow y = 1 \vee (y = 0 \wedge [\mathbf{pfp}_{R,\overline{x}}\varphi](\overline{x})) \\ Q\overline{x}\ \leftarrow P \neq \emptyset \wedge \neg P\overline{x}0 \end{array}](\overline{t}),$$

where $0, 1$ are variables not occurring in φ. The theorem now follows immediately by induction on the structure of the formulae. □

As discussed above, this implies that nesting fixed points strictly increases the expressive power, i.e. nested fixed points can not be eliminated in favour of a single fixed point.

4 Separating Partial and Inflationary Fixed-Point Logic

In this section we prove the main result of this paper, the separation of $\mathrm{PFP}_{\mathrm{gen}}$ and IFP. As we are not considering the finite model semantics anymore, we simply write PFP and **pfp** instead of $\mathrm{PFP}_{\mathrm{gen}}$ and $\mathbf{pfp}^g$.

We first present a class of structures called *acceptable* (See [Mos74, Chapter 5].) These structures are particularly well suited to be used with diagonalisation arguments.

4.1 Acceptable Structures

Definition 4.1. *Let A be an infinite set. A* coding scheme on *A is a triple $(\mathcal{N}, \leq, <>)$, for some $\mathcal{N} \subseteq A$, where the structure $(\mathcal{N}, \leq)$ is isomorphic to $(\omega, \leq)$ and $<>$ is an injective map of $\bigcup_{n<\omega} A^n$ into A.*

With each coding scheme we associate the following decoding relations and functions:

(i) $seq(x)$ which is true for x if, and only if, x is the code of some sequence $x_1, \ldots, x_n$.
(ii) $lh(x) = n$ if x is the code of a sequence of length n and otherwise, i.e. if $\neg\mathrm{seq}(x)$, $lh(x) = 0$.
(iii) $q(x, i) = x_i$ if $x =< x_1, \ldots, x_l >$ and $l \geq i$. Otherwise $q(x, i) = 0$. We write $(x)_i = a$ for $q(x, i) = a$.

Here, the numbers $0, 1, \ldots$ refer to the corresponding elements in $\mathcal{N}$.

An elementary coding scheme $\mathcal{C}$ on a structure $\mathfrak{A}$ *is a coding scheme on its universe where the relations $\mathcal{N}, \leq$, seq, lh, and q are elementary, i.e., first-order definable.*

A structure $\mathfrak{A}$ admitting an elementary coding scheme is called acceptable. *We call $\mathfrak{A}$* quasi-acceptable *if there exists an acceptable expansion $\mathfrak{A}'$ of $\mathfrak{A}$ by a finite set of* PFP*-definable relations.*

Observe that quasi-acceptable structures are those which admit an PFP-definable coding scheme, i.e., one where the relations $<$, seq, lh, and q are PFP-definable. See [Mos74, Chapter 5] for more on elementary and inductive coding schemes.

4.2 Coding and Diagonalisation

We show now how formulae can be encoded by elements of acceptable structures. For the rest of this section let $\mathfrak{A}$ be an acceptable τ-structure, where $\tau := \tau_{\text{rel}} \dot\cup \tau_{\text{const}}$ is the disjoint union of a finite set $\tau_{\text{rel}} := \{P_1, \ldots, P_l\}$ of relation symbols and a finite set $\tau_{\text{const}} := \{c_1, \ldots, c_m\}$ of constant symbols. W.l.o.g. we assume that no fixed-point variable is bound twice in the same formula and that the involved fixed-point variables R_i are numbered from 1 to the number k of fixed-point operators occurring in the formula such that for no $i < j \leq k$, φ_i is a sub-formula of φ_j, where φ_i and φ_j are the formulae defining the fixed point inductions on R_i and R_j respectively. Further, we assume that all formulae are of the form $[\mathbf{ifp}_{R_1,\overline{x}_1}\, \varphi_1](\overline{x}_1)$. We also assume that all fixed-point operators are of the form $[\mathbf{ifp}_{R,\overline{x}}\, R\overline{x} \vee \varphi(R, \overline{x})]$, i.e. the operators are syntactically made inflationary. Finally, we assume that if $\psi := [\mathbf{ifp}_{R,x_{i_1},\ldots,x_{i_k}}\, \varphi]$ occurs as a sub-formula of a formula χ, then the sub-formulae of φ may use atoms in which R occurs only in the form $Rx_{i_1}, \ldots, x_{i_k}$. It is clear that any IFP-formula can be brought into this form.

The actual encoding of formulae is based on a function $||\varphi||$ taking formulae or terms in IFP$[\tau]$ to elements of $\mathcal{N}$. The function is inductively defined as follows.

$$
\begin{array}{rcll}
||c_i|| & := & < \mathbf{c}, i > & c_i \in \tau_{\text{const}} \\
||x_i|| & := & < \mathbf{var}, i > & \\
||P_i\overline{a}|| & := & < \mathbf{rel}, i, < ||\overline{a}|| >> & P_i \in \tau_{\text{Rel}} \\
||\varphi_1 \vee \varphi_2|| & := & < \mathbf{or}, ||\varphi_1||, ||\varphi_2|| > & \\
||\neg\varphi|| & := & < \mathbf{neg}, ||\varphi|| > & \\
||R_i\overline{a}|| & := & < \mathbf{fp\text{-}var}, i, < ||\overline{a}|| >> & \text{for fixed-point variables } R_i \\
||\,[\mathbf{ifp}_{R_i,\overline{x}}\, \varphi](\overline{a})|| & := & < \mathbf{fp\text{-}op}, i, < ||\overline{a}|| >>, &
\end{array}
$$

where $\mathbf{c}, \mathbf{var}, \ldots$ denote arbitrary but fixed and distinct elements of $\mathcal{N}$. Here $< ||\overline{a}|| >$ is an abbreviation for $< ||a_1||, \ldots, ||a_k|| >$ where k is the arity of $\overline{a}$. In this encoding of formulae, sub-formulae involving fixed-point variables are only coded by the number of the involved fixed-point variable but no code of the formula defining it is stored. The next definition deals with this.

Definition 4.2. *Let φ be a formula in* IFP$[\tau]$ *and let the fixed-point operators occurring in it be* $[\mathbf{ifp}_{R_1,\overline{x}_1}\, \varphi_1]$, $\ldots$, $[\mathbf{ifp}_{R_n,\overline{x}_n}\, \varphi_n]$. *The formulae φ_i, for $1 \leq i \leq n$, are called the* defining formulae *of φ and each individual φ_i is called the* defining formula *of the fixed-point variable R_i.*

The function code taking formulae to their codes in $\mathcal{N}$ is defined as

$$
\begin{array}{rcl}
\mathit{code}: \text{IFP}[\tau] & \longrightarrow & \mathcal{N} \\
\varphi & \longmapsto & < ||\varphi_1||, \ldots, ||\varphi_k|| >,
\end{array}
$$

where $\varphi_1, \ldots, \varphi_k$ *are the defining formulae of* φ.

Below, we will use encodings of formulae to show that there are relations on acceptable structures which are PFP but not IFP-definable. We first fix some notation that will be used in the sequel.

Definition 4.3. *Let* $\varphi(\overline{x})$ *be a formula with free variables* $\overline{x}$*, where* $\overline{x} := x_{i_1}, \ldots, x_{i_k}$ *for some* k*. The* code a of a sequence matches φ*, if* $lh(a) \geq \max\{i_j : 1 \leq j \leq k\}$.

We write $a \models \varphi$*, if* a *matches* φ *and* φ *is true in* $\mathfrak{A}$ *under the variable assignment*

$$\beta : x_i \longmapsto \begin{cases} (a)_i & \text{for all } 1 \leq i \leq \mathrm{lh}(x) \\ 0 & \text{otherwise.} \end{cases}$$

If c *is the code of* φ *we also write* $a \models c$ *for* $a \models \varphi$.

We state the following lemma whose proof is technical but not very difficult.

Lemma 4.4. *There is a* PFP*-formula* formula(x) *that is true for all* c *which are valid codes of* IFP*-formulae.*

4.3 Separating Inflationary and Partial Fixed-Point Logic

In this section we show that partial fixed-point logic is strictly more expressive than inflationary fixed-point logic. The result uses the methods introduced in the sections above.

Definition 4.5. *The relation* $\textsc{Sat}_{\mathrm{IFP}} \subseteq A^2$ *is defined as*

$$\textsc{Sat}_{\mathrm{IFP}} := \{(c, a) : c \text{ is the code of an } \mathrm{IFP}[\tau]\text{-formula } \varphi \text{ and } \varphi \models c\}.$$

Clearly, $\textsc{Sat}_{\mathrm{IFP}}$ is not IFP-definable.

Lemma 4.6. $\textsc{Sat}_{\mathrm{IFP}}$ *is not definable in* IFP.

Proof. Suppose, $\textsc{Sat}_{\mathrm{IFP}}$ were definable in IFP. Then the relation $R(x) := \neg Sat(x, < x >)$ would be definable in IFP as well, by a formula $\varphi(x)$ say. Let c be the code of φ. Thus, as φ defines R, for all x, $R(x) \iff Sat(c, < x >)$ but, by definition of R, for all x, $R(x) \iff \neg Sat(x, < x >)$. For $x = c$ we get a contradiction. □

We show now that $\textsc{Sat}_{\mathrm{IFP}}$ is definable in PFP by inductively defining a ternary relation $R(c, i, a) \subseteq A^3$ such that $(c, i, a) \in R$ if, and only if, c is the code of a formula $\varphi \in \mathrm{IFP}[\tau]$ with defining formulae $\varphi_1, \ldots, \varphi_k$, i is an element of $\{1, \ldots, k\}$, and a is the code of a variable assignment matching the free variables in φ such that

$$(\mathfrak{A}, stage(c, 1), \ldots, stage(c, k)), a \models \varphi_i,$$

i.e. φ_i is true under the variable assignment a if all free fixed-point variables R_j are interpreted by the sets $stage(c,j)$ defined as $stage(c,j) := \{\overline{a} : (c,j,a) \in R,$ where a is the code of $\overline{a}\}$.

This relation will be built up by a partial fixed-point induction such that the following invariance property is preserved:

Invariance Property 4.7.

- *For all c, i, a, if $(c,i,a) \in R$ then c is the code of a formula $\varphi \in \mathrm{IFP}[\tau]$, with defining formulae $\varphi_1, \ldots, \varphi_k$, i is an element of $\{1, \ldots, k\}$, and a is the code of a variable assignment matching the free variables in φ such that*

$$(\mathfrak{A}, \mathrm{stage}(c,1), \ldots, \mathrm{stage}(c,k)), a \models \varphi_i,$$

  i.e. φ_i is true under the variable assignment a where all free fixed-point variables R_j are interpreted by the sets $\mathrm{stage}(c,j)$.
- *At each stage α of the induction on R, and all i and c as above, the set* $\mathrm{stage}(c,i)$ *occurs as a stage of the induction on φ_i where all free fixed-point variables R_j of φ_i are interpreted by* $\mathrm{stage}(c,j)$.

Before presenting a formula defining R we introduce some auxiliary formulae *first-order* and *fpr*. The formula *first-order*(R,c,i,a) assumes that the invariance property in 4.7 is satisfied by R. In this case, it defines the set of all (c,i,a) such that $a \models \varphi_i$, under the assumption that all free fixed-point variables R_j are interpreted by $stage(c,j)$ and for all sub-formulae of φ_i of the form $[\mathbf{ifp}_{R_j,\overline{x}_j}\, \varphi_j]$ the fixed point defined by this formula is $stage(c,j)$. Obviously, these assumptions are too optimistic for all i, as the second assumption will generally be true only for some, but not for all i. This formula will be used in a formula defining the relation R described above and there it will be guaranteed that *first-order* will only be "called" for values of i for which both assumptions are satisfied. In the following, we treat variables $t, t_1, \ldots$ as boolean variables, i.e. the only values they can take are 0 and 1, and we use expressions like $t = t_1 \vee t_2$ with the obvious semantics. We also use notation like "$c \widehat{=} \varphi_{c_1} \vee \varphi_{c_2}$" which means that c is the code of a formula $\varphi := \varphi_1 \vee \varphi_2$ and c_1, c_2 are the codes of the sub-formulae.

$$
\begin{aligned}
&\textit{first-order}(c,i,a) := \\
&\quad [\mathbf{pfp}_{Q,c,a,t}\ \text{“}c \widehat{=} \exists x_j \varphi_{c'}\text{”} \wedge ((\exists a'\, Qc'a'1 \wedge \forall i\, ((a)_i = (a')_i \vee i = j) \wedge t = 1) \vee \\
&\qquad\qquad\qquad\qquad (\forall a'\, (\forall i\, ((a)_i = (a')_i \vee i = j) \rightarrow Qc'a'0) \wedge t' = 0)) \vee \\
&\qquad\qquad \text{“}c \widehat{=} \varphi_{c_1} \vee \varphi_{c_2}\text{”} \wedge (\exists t_1 \exists t_2 (Qc_1 a\ t_1 \wedge Qc_2 a\ t_2 \wedge t = t_1 \vee t_2) \vee \\
&\qquad\qquad \text{“}c \widehat{=} \neg \varphi_{c'}\text{”} \wedge (\exists t'\, Qc'at' \wedge t = \neg t') \vee \\
&\qquad\qquad \text{“}c \widehat{=} P_i x_{i_1} \ldots x_{i_k}\text{”} \wedge (t \leftrightarrow P_i(a)_{i_1} \ldots (a)_{i_k}) \vee \\
&\qquad\qquad \text{“}c \widehat{=} R_i \overline{x}\text{”} \wedge (t \leftrightarrow Rcia) \vee \\
&\qquad\qquad \text{“}c \widehat{=} [\mathbf{ifp}_{R_i,\overline{x}}\, \varphi_i]\text{”} \wedge (t \leftrightarrow Rcia) \\
&\quad](c_i, a, 1)
\end{aligned}
$$

The correctness of the construction is proved in the following lemma.

Lemma 4.8. *Let R be a ternary relation satisfying the invariance property in 4.7. Then for all c, i, a, such that c is the code of a formula φ with defining sub-formulae $\varphi_1, \ldots, \varphi_k$ and $i \in \{1, \ldots, k\}$,*

$$(\mathfrak{A}, R) \models \text{first-order}(c, i, a) \quad \textit{if, and only if,} \quad a \models \varphi_i,$$

where all free fixed-point variables R_j and all sub-formulae of the form $[\mathbf{ifp}_{R_j, \overline{x}_j} \varphi_j]$ are interpreted by the sets $\text{stage}(R, j)$.

Proof. The lemma is proved by induction on the structure of φ. As this is a standard argument, we do not give the full proof here but refer to [Mos74, Chapter 5] for details. We demonstrate the idea behind the formula by proving the case for existential quantification. Suppose c is the code of a formula $\exists x_j \varphi_{c'}$ and c' is the code of $\varphi_{c'}$. Then "$c \widehat{=} \exists x_j\, \varphi_{c'}$" is satisfied and the formula checks whether there is (the code a' of) a variable assignment satisfying $\varphi_{c'}$, i.e. $(c', a', 1) \in Q$, such that a and a' agree on all variables except x_j. By induction, if there is such an a', then $a' \models \varphi'$ and thus $a \models \varphi$. In this case t is required to be 1. Otherwise, i.e. if there is no such a', $a \not\models \varphi$ and thus $t = 0$.

Note also how the truth of sub-formulae involving fixed points is directly read from the relation R. □

We also need a formula $\mathit{fpr}(R, c, i)$ that is true for c and i if $\mathit{stage}(c, i)$ is the fixed point of the induction on φ_i where all free fixed-point variables R_j of φ_i are interpreted by $\mathit{stage}(c, j)$.

$$\mathit{fpr}(R, c, i) := \forall a(\mathit{first\text{-}order}(R, c, i, a) \rightarrow R(c, i, a)).$$

Clearly, under the same assumptions as in Lemma 4.8, $(\mathfrak{A}, R) \models \mathit{fpr}(c, i)$ if, and only if, $\mathit{stage}(c, i)$ is the fixed-point of φ_i. We are now ready to define the main formula.

$$\begin{aligned}
&\mathit{compute}(c, a) := \\
&\quad [\mathbf{pfp}_{R,c,i,a}\, (\exists l \in \{1, \ldots, \text{lh}(c)\} \forall l < j \leq k\ \mathit{fpr}(R, c, j) \wedge \neg \mathit{fpr}(R, c, l) \wedge \\
&\qquad\qquad ((i = l \wedge \mathit{first\text{-}order}(c, i, a)) \vee (i < l \wedge \mathit{Rciat})) \wedge \mathit{formula}(c)) \vee \\
&\qquad\qquad (\forall l \in \{1, \ldots, \text{lh}(c)\}\ \mathit{fpr}(R, c, j)) \wedge \mathit{Rcia} \\
&\quad](c, 1, a).
\end{aligned}$$

The formula $\mathit{formula}(c)$ has been defined in Lemma 4.4 above. Recall the way formulae φ are coded by $c :=< ||\varphi_1||, \ldots, ||\varphi_k|| >$. The formula *compute* first defines the unique l such that the fixed points of all formulae φ_j with $j > l$ are already computed in R but the induction on φ_l has not yet reached its fixed point. For this l, the formula $\mathit{first\text{-}order}(c, l, a)$ is evaluated, i.e the next stage of the induction on φ_j is computed. Further, all triples (c, j, a) such that $j < l$ are kept in R, i.e. the current stages of the induction on φ_j with $j < l$ are left untouched. On the other hand, all triples (c, j, a) for $j > l$ are removed from R, i.e. the fixed-point induction on the formulae φ_j, which might depend on R_l, are set back to the empty set.

Thus, in the end there will be no such l as all fixed points are already computed. In this case the relation R is left untouched and thus the fixed point of *compute* has been reached. This proves the following lemma.

Lemma 4.9. $\textsc{Sat}_{\text{IFP}}$ *is definable in* PFP.

The proof of the following theorem and its corollary is now immediate.

Theorem 4.10. PFP *is more expressive than* IFP *on acceptable structures.*

Corollary 4.11. PFP *is more expressive than* IFP *on all structures in which an acceptable structure is* PFP*-interpretable.*

Among the structures in which an acceptable structure is PFP-interpretable are $(\omega, <)$ and $(\mathbb{R}, <, +)$ and all expansions of it, e.g. the ordered field of reals. Examples of structures not interpretable in an acceptable structure are structures over the empty signature or a signature containing constant symbols only, but also the real line $(\mathbb{R}, <)$.

References

[AV91a] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991. 338

[AV91b] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. of the 23rd ACM Symp. on the Theory of Computing*, 1991. 338

[AVV97] S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. *Journal of the ACM*, 44(1):30–56, 1997. An extended abstract appeared in the Proc. 7th IEEE Symp. on Structure in Complexity Theory, 1992. 338

[Daw93] A. Dawar. *Feasible Computation Through Model Theory*. PhD thesis, University of Pennsylvania, 1993. 338

[DG02] A. Dawar and Y. Gurevich. Fixed-point logics. *Bulletin of Symbolic Logic*, 8(1):65–88, 2002. 337

[DK] A. Dawar and S. Kreutzer. Partial and Alternating Fixed Points in Modal Logic. Unpublished. 341

[EF99] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 2nd edition, 1999. 337, 340

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986. 338

[Imm86] N. Immerman. Relational queries computable in polynomial time. *Information and Control*, 68:86–104, 1986. Extended abstract in Proc. 14th ACML Symp. on Theory of Computing, pages 147-152, 1982. 337

[Kre02] S. Kreutzer. Expressive equivalence of least and inflationary fixed-point logic. Proc. of the 17th Symp. on Logic in Computer Science (LICS), 2002. 338

[Mos74] Y. N. Moschovakis. *Elementary Induction on Abstract Structures*. North Holland, 1974. ISBN 0 7204 2280 9. 346, 347, 350

[Var82] M. Vardi. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

On the Variable Hierarchy of the Modal μ-Calculus

Dietmar Berwanger[1], Erich Grädel[1], and Giacomo Lenzi[2]

[1] Mathematische Grundlagen der Informatik
RWTH Aachen, D-52056 Aachen
{berwanger,graedel}@informatik.rwth-aachen.de
[2] Dipartimento di Matematica
Università di Pisa, via Buonarroti 2, I-56127 Pisa
lenzi@mail.dm.unipi.it

Abstract. We investigate the structure of the modal μ-calculus L_μ with respect to the question of how many different fixed point variables are necessary to define a given property. Most of the logics commonly used in verification, such as CTL, LTL, CTL*, PDL, etc. can in fact be embedded into the two-variable fragment of the μ-calculus. It is also known that the two-variable fragment can express properties that occur at arbitrarily high levels of the alternation hierarchy. However, it is an open problem whether the variable hierarchy is strict.
Here we study this problem with a game-based approach and establish the strictness of the hierarchy for the case of existential (i.e., $\Box$-free) formulae. It is known that these characterize precisely the L_μ-definable properties that are closed under extensions. We also relate the strictness of the variable hierarchy to the question whether the finite variable fragments satisfy the existential preservation theorem.
Keywords: modal μ-calculus, games, descriptive complexity

1 Introduction

The modal μ-calculus L_μ extends propositional multi-modal logic with operators for forming least and greatest fixed points. This logic has been extensively studied for a number of reasons. In terms of expressive power, it subsumes a variety of modal and temporal logics used in verification, in particular LTL, CTL, CTL*, PDL and also many logics applied in other areas of computer science, for instance description logics. On the other hand, L_μ has a rich theory, and is well-behaved under model-theoretic and algorithmic aspects.

One of the most important open problems concerning the μ-calculus is the complexity of the model checking problem: Is there a polynomial-time algorithm that, given a formula $\psi \in L_\mu$ and a finite Kripke structure $\mathcal{K}$, computes the set of nodes v such that $\mathcal{K}, v \models \psi$. Like most evaluation problems for logical systems, the model checking problem for L_μ can be reformulated as the strategy problem for appropriate evaluation games. The games associated with fixed point logics are *parity games*, which are infinite games where each position is assigned

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 352–366, 2002.
© Springer-Verlag Berlin Heidelberg 2002

a natural number, called its *priority*, and the winner of an infinite play is determined according to whether the least priority seen infinitely often during the play is even or odd. It is open whether winning sets and winning strategies for parity games can be computed in polynomial time. The best algorithms known today are polynomial in the size of the game, but exponential with respect to the number of priorities. Competitive model checking algorithms for the modal μ-calculus work by solving the strategy problem for the associated parity game (see, e.g., [11]).

The number of priorities, the main source of difficulty for solving parity games, is tightly related to the *alternation depth* of L_μ-formulae, i.e., the number of (genuine) alternations between least and greatest fixed points. The correspondence goes both ways: The model checking problem for a formula ψ with alternation depth d on a finite transition system $\mathcal{K}$ translates to the strategy problem of a parity game of size $O(|\psi| \cdot \|\mathcal{K}\|)$ and with at most $d+1$ priorities. Conversely, for any $d \in \mathbb{N}$, there exists an L_μ-formula W^d of alternation depth d that defines the winning positions of Player 0 in any parity game with d priorities.

It has been shown by Bradfield [6] that the alternation hierarchy of the μ-calculus is strict. Variants of this result have also been proven by Lenzi [13] and Arnold [2]. In fact the parity game formulae W^d witness the strictness of the alternation hierarchy.

Theorem 1 (Bradfield). *For any number $d > 0$, the formula W^d is not equivalent to any L_μ-formulae of alternation depth $d-1$.*

Fortunately, most specification properties used in practical applications require very small alternation depths. Indeed, many popular sublogics of L_μ, such as LTL, CTL, PDL, and CTL* can be embedded into the first or second alternation level of L_μ. However, an interesting counterexample to this pattern is Parikh's game logic GL [15]. To exploit its full reasoning power, GL should be interpreted over neighbourhood structures, which are more general than Kripke structures, with accessibility relations between *sets* of states. But GL is also a very interesting logic on Kripke structures. It looks very similar to PDL, but is used to reason about games rather than programs. For instance, a GL-formula of form $\langle g \rangle \psi$ expresses that Player 0 has a strategy in the game g to achieve an outcome where ψ holds. Games are constructed from atomic games by similar operations as the program operations of PDL, namely union, composition, and iteration, and in addition a dualization operation, mapping g to the dual game g^d, in which the roles of the two players are switched. The intertwining of role switches and iteration leads to unexpected expressive power, high complexity, and unbounded alternation levels. Indeed it has been shown by Berwanger [4] that GL (on Kripke structures) intersects non-trivially with all levels of the alternation hierarchy. In fact GL is strong enough to reason about parity games, in the sense that GL contains, for every d, a formula that is equivalent to W^d. It is currently still open whether GL (on Kripke structures) has the same expressive power as the μ-calculus.

While the strictness of the alternation hierarchy does not help us to separate GL and L_μ, recent investigations have brought to light another interesting hierarchy inside the μ-calculus, namely the *variable hierarchy*. By re-using fixed point variables several times it is possible to write many L_μ-formulae, even with very high nesting and alternation depth, using only very few variables. For any k, we denote by $L_\mu[k]$ the fragment of L_μ consisting of those formulae that make use of at most k distinct fixed-point variables. It turns out that most of the common sublogics of L_μ that are used in verification can be embedded into $L_\mu[2]$, the two-variable fragment of the μ-calculus. In particular this is the case for GL (on Kripke structures) and hence for all logics subsumed by GL, including CTL, LTL, CTL*, PDL, and Δ-PDL (see [16]).

The problem we are investigating in this paper is whether the variable hierarchy is strict. We conjecture that for every $k \geq 1$ there exist formulae in $L_\mu[k]$ that are not equivalent to any formula in $L_\mu[k-1]$. This can easily be shown for $k = 1$ and $k = 2$ (see Section 2) but it is currently unknown whether $L_\mu[2] = L_\mu$. Clearly, separating $L_\mu[2]$ from L_μ would also separate GL from L_μ,

The question whether the variable hierarchy is strict is meaningful and interesting not only for the μ-calculus itself, but also for relevant fragments. More precisely, given any set of formulae $L \subseteq L_\mu$ the variable hierarchy problem for L is the question whether there exist for every $k \geq 1$ formulae in $L[k] := L \cap L_\mu[k]$ that are not equivalent to any formula in $L[k-1]$.

In this paper we answer the question affirmatively for the $\Box$-*free fragment* of the μ-calculus. A formula $\psi \in L_\mu$ is called $\Box$-free or existential if it can be built from atoms and negated atoms by means of $\wedge, \vee$, existential modalities $\langle a \rangle$ and least and greatest fixed point. It is not difficult to see that every $\Box$-free formula is closed under extensions. This means that whenever $\mathcal{K}, v \models \psi$ and $\mathcal{K} \subseteq \mathcal{K}'$, then also $\mathcal{K}', v \models \psi$. The $\Box$-free fragment is important because of the preservation theorem that has recently been established by D'Agostino and Hollenberg [7] which shows that also the converse holds.

Theorem 2. *A formula $\psi \in L_\mu$ is closed under extensions if and only if it is equivalent to a $\Box$-free formula in L_μ.*

We describe here a sequence of rather simple $\Box$-free formulae $\psi^{(k)} \in L_\mu[k]$ and prove that, for every k, $\psi^{(k)}$ is not equivalent to any $\Box$-free formula with less than k variables. Intuitively, the formulae $\psi^{(k)}$ express that the models contain a substructure that is bisimilar to a kind of k-clique with labeled edges (see Section 4 for precise definitions). These formulae use only conjunctions, existential modalities, and greatest fixed points. In particular they are alternation free.

We conjecture that the formulae $\psi^{(k)}$ witness in fact the strictness of the variable hierarchy for the full μ-calculus, not just for its existential fragment. This conjecture is related to the question whether the existential preservation theorem holds for the bounded variable fragments $L_\mu[k]$ of the μ-calculus.

Our analysis suggests that the variable hierarchy is 'orthogonal' to the alternation hierarchy. We have proved [4] that already in $L_\mu[2]$ one can define properties on arbitrary levels of the alternation hierarchy. Further, while the

alternation hierarchy of L_μ is of importance for the complexity, at least for the currently known model checking algorithms (which are exponential in the number of alternations), this does not seem to be the case for the variable hierarchy.

Theorem 3 (Berwanger). *The model checking problem for $L_\mu[2]$ can be solved in polynomial time iff this is the case for the full μ-calculus.*

Finally, the clique formulae by which we proved the strictness of the hierarchy for existential formulae are pure ν-formulae, and thus on the lowest level of the alternation hierarchy.

Here is an overview over the contents of this paper. In Section 2 we define the μ-calculus, explain parity games and introduce the variable hierarchy. In Section 3 we discuss the $\Box$-free fragment of L_μ and prove a technical lemma on strategy trees for this fragment. In Section 4 we define the formulae that will witness the strictness of the variable hierarchy for the $\Box$-free fragment of L_μ. Finally, in Section 5 we prove our hierarchy theorem.

2 The μ-Calculus

Fix a set ACT of actions and a set PROP of atomic propositions. A transition system or Kripke structure for ACT and PROP is a structure $\mathcal{K}$ with universe V (whose elements are called states), binary relations $E_a \subseteq V \times V$ for each $a \in$ ACT, and monadic relations $p \subseteq V$ for each atomic proposition $p \in$ PROP (we do not distinguish notationally between atomic propositions and their interpretations).

Syntax of L_μ. For a set ACT of actions, a set PROP of atomic propositions, and a set VAR of variables, the formulae of L_μ are defined by the grammar

$$\varphi ::= \mathit{false} \mid \mathit{true} \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $p \in$ PROP, $a \in$ ACT, and $X \in$ VAR.

Semantics of L_μ. Formulae of L_μ are evaluated on transition systems at a particular state. Given a sentence ψ and a transition system $\mathcal{K}$ with state v, we write $\mathcal{K}, v \models \psi$ to denote that ψ holds in $\mathcal{K}$ at state v. The set of states $v \in V$ such that $\mathcal{K}, v \models \psi$ is denoted by $[\![\psi]\!]^{\mathcal{K}}$. We omit the definition of $[\![\psi]\!]^{\mathcal{K}}$ for the obvious cases. For the modal operators,

$$[\![\langle a \rangle \psi]\!]^{\mathcal{K}} := \{v : \text{ there exists a state } w \text{ such that } (v, w) \in E_a \text{ and } w \in [\![\psi]\!]^{\mathcal{K}}\}$$
$$[\![[a]\psi]\!]^{\mathcal{K}} := \{v : \text{ for all } w \text{ such that } (v, w) \in E_a, \text{ we have } w \in [\![\psi]\!]^{\mathcal{K}}\}.$$

To understand the semantics of fixed point formulae, note that a formula $\psi(X)$ with a propositional variable X defines on every transition system $\mathcal{K}$ (with state set V, and with interpretations for free variables other than X occurring in ψ) an operator $\psi^{\mathcal{K}} : \mathcal{P}(V) \to \mathcal{P}(V)$ assigning to every set $X \subseteq V$ the set $\psi^{\mathcal{K}}(X) := [\![\psi]\!]^{\mathcal{K},X} = \{v \in V : (\mathcal{K}, X), v \models \psi\}$. As X occurs only positively in ψ, the operator $\psi^{\mathcal{K}}$ is *monotone* for every $\mathcal{K}$, and therefore, by a well-known

theorem due to Knaster and Tarski, has a least fixed point $\mathrm{lfp}(\psi^{\mathcal{K}})$ and a greatest fixed point $\mathrm{gfp}(\psi^{\mathcal{K}})$. Now we put

$$[\![\mu X.\psi]\!]^{\mathcal{K}} := \mathrm{lfp}(\psi^{\mathcal{K}}) \text{ and } [\![\nu X.\psi]\!]^{\mathcal{K}} := \mathrm{gfp}(\psi^{\mathcal{K}}).$$

Model checking games. The semantics of L_μ can also be described in terms of *parity games*. Such a game is given by a transition system $\mathcal{G} = (V, V_0, E, \Omega)$, where V is a set of positions with a designated subset V_0, $E \subseteq V \times V$ is a transition relation, and $\Omega : V \to \mathbb{N}$ assigns to every position a *priority*. A *play* of $\mathcal{G}$ is a path $v_0, v_1, \ldots$ formed by the two players starting from a given position v_0. If the current position v belongs to V_0, Player 0 chooses a move $(v, w) \in E$ and the play proceeds from w. Otherwise, her opponent, Player 1, chooses the move. When no moves are available at the current position, the player who has to choose loses. In case this never occurs the play goes on infinitely and the winner is established by looking at the sequence $\Omega(v_0), \Omega(v_1), \ldots$ If the least priority appearing infinitely often in this sequence is even, Player 0 wins the play, otherwise Player 1 wins.

Let $V_1 := V \setminus V_0$ be the set of positions where Player 1 moves. A *strategy* for Player i in $\mathcal{G}$ is a partial function $f : V^*V_i \to V$ which indicates for an initial play $v_0, v_1, \ldots, v_r$ up to some position $v_r \in V_i$ a possible prolongation w, so that $(v_r, w) \in E$. If Player i wins every play where he moves according to f, we say that f is a *winning* strategy. A strategy that does not depend on the history of the play but only on the current position is called a *positional* strategy. The Forgetful Determinacy Theorem for parity games [8] states that these games are always determined (i.e., from each position one of the players has a winning strategy) and, in fact, positional strategies always suffice.

Theorem 4 (Forgetful Determinacy). *In any parity game the set of positions can be partitioned into two sets W_0 and W_1 such that Player* 0 *has a positional winning strategy on W_0 and Player* 1 *has a positional winning strategy on W_1.*

Given a transition system $\mathcal{K}, v_0$ and a L_μ-sentence ψ, the model checking game $\mathcal{G}(\mathcal{K}, \psi)$ is a parity game associated with the problem whether $\mathcal{K}, v_0 \models \psi$. There are several, essentially equivalent, ways to define this game. In the more transparent one, positions are pairs (v, φ) where φ is any (not necessarily closed) subformula of ψ, and it is assumed that every variable is bound at most once by a fixed-point definition (see, e.g., [5, 17]). For certain technical reasons, and since we want to re-use variables several times, we use here the slightly less intuitive variant (more familiar from automata theory [8, 12]), which, instead of subformulae, uses their closure, that is, the sentence obtained by replacing, recursively, every free occurence of a variable by its binding definition.

Definition 5. *The* closure $\mathrm{cl}(\psi)$ *of a sentence $\psi \in L_\mu$ is the smallest set of sentences so that $\psi \in \mathrm{cl}(\psi)$ and*

(1) if $\varphi_1 \vee \varphi_2 \in \mathrm{cl}(\psi)$ or $\varphi_1 \wedge \varphi_2 \in \mathrm{cl}(\psi)$ then $\{\varphi_1, \varphi_2\} \subseteq \mathrm{cl}(\psi)$;

(2) if $\langle a \rangle\varphi \in \mathrm{cl}(\psi)$ *or* $[a]\varphi \in \mathrm{cl}(\psi)$ *then* $\varphi \in \mathrm{cl}(\psi)$*;*
(3) for λ *denoting either* μ *or* ν*, if* $\lambda X.\varphi(X) \in \mathrm{cl}(\psi)$ *then* $\varphi(\lambda X.\varphi(X)) \in \mathrm{cl}(\psi)$*.*

Then the positions in the game $\mathcal{G}(\mathcal{K}, \psi)$ are pairs (v, φ) of states $v \in V$ and sentences $\varphi \in \mathrm{cl}(\psi)$. Player 0 moves from the positions $(v, \varphi_1 \vee \varphi_2)$, $(v, \langle a \rangle\varphi)$, (v, p) with $v \notin p$, and $(v, \neg p)$ with $v \in p$. All plays start at (v_0, ψ) and the transitions in E are such that

- no moves are possible from (v, α) where α is atomic or negated atomic;
- from $(v, \varphi_1 \vee \varphi_2)$ or $(v, \varphi_1 \wedge \varphi_2)$ transitions lead to (v, φ_1) and (v, φ_2);
- from $(v, \langle a \rangle\varphi)$ or $(v, [a\,]\varphi)$ there are transitions to all positions (w, φ) where w is an a-successor of v.
- from $(v, \lambda X.\varphi(X))$ there is a transition to $(v, \varphi(\lambda X.\varphi(X))$;

Thus, a play proceeds along the paths in $\mathcal{K}$ and in the syntax tree of ψ, until it hits a fixed point variable (which is a leaf in the syntax tree). There the play resumes with the binding definition of the variable. We call this event the *regeneration* of a variable.

By repeatedly regenerating variables, it may happen that neither Player 0 nor Player 1, here called Verifier and Falsifier, ever gets stuck. To decide the winner of such plays, priorities have to be defined appropriately. The intuition is that, to establish the truth of a μ-formula, Verifier should regenerate it only finitely often whereas ν-formulae can be regenerated infinitely often. Of course the difficulty may be that μ- and ν-formulae are deeply nested and there are several fixed-point formulae that are regenerated infinitely often during a play. But it can be shown that among these, there is always an outermost one, which determines the winner: if it is a ν-formula Verifier wins, if it is a μ-formula, Falsifier wins. Hence, the priority labelling assigns even priorities to positions $(v, \nu X.\varphi)$ and odd priorities to positions $(v, \mu X.\varphi)$. Further, priorities respect dependencies. If $\nu Y.\varphi$ depends on $\mu X.\eta$ then priorities of positions $(v, \nu Y.\varphi)$ are higher than those of positions $(w, \mu X.\eta)$. The remaining positions receive priorities that are higher than those associated with fixed-point formulae. For details (which are not needed in this paper) see, e.g., [5, 17].

Theorem 6. *Verifier has a winning strategy in the model checking game* $\mathcal{G}(\mathcal{K}, \psi)$ *from position* (u, ψ) *iff* $\mathcal{K}, u \models \psi$*.*

As a modal logic, the μ-calculus distinguishes between transitions systems only up to behavioral equivalence, captured by the notion of bisimulation.

Definition 7. *A* bisimulation *between two transition systems* $\mathcal{K}$ *and* $\mathcal{K}'$ *is a relation* $Z \subseteq V \times V'$ *between the domains of* $\mathcal{K}$ *and* $\mathcal{K}'$*, respecting the atomic propositions* $p \in \textsc{prop}$ *in the sense that* $\mathcal{K}, v \models p$ *iff* $\mathcal{K}', v' \models p$*, for* $(v, v') \in Z$*, and satisfying the following back and forth conditions.*

Forth: *for all* $(v, v') \in Z$*,* $a \in \textsc{act}$ *and every* w *such that* $(v, w) \in E_a$*, there exists a* w' *such that* $(v', w') \in E'_a$ *and* $(w, w') \in Z$*.*

Back: *for all* $(v, v') \in Z$*,* $a \in \textsc{act}$ *and every* w' *such that* $(v', w') \in E'_a$*, there exists a* w *such that* $(v, w) \in E_a$ *and* $(w, w') \in Z$*.*

Two transition systems $\mathcal{K}, u$ and $\mathcal{K}', u'$ are *bisimilar*, if there is a bisimulation Z between them with $(u, u') \in Z$.

An important model theoretic feature of modal logics is the *tree model property*, the fact that every satisfiable formula is satisfiable in a tree. This is a straightforward consequence of bisimulation invariance, since $\mathcal{K}, u$ is bisimilar to its tree unravelling.

Definition 8. *The* unravelling $\mathcal{T}(\mathcal{K}, u)$ *of a Kripke structure* $\mathcal{K}$ *from node* u *is the tree of all paths through* $\mathcal{K}$ *that start at* u*. More formally,*

- *the domain of* $\mathcal{T}(\mathcal{K}, u)$ *is the set* $V^{\mathcal{T}}$ *of all sequences* $\overline{v} = v_0 a_1 v_1 a_2 \cdots v_{r-1} a_r v_r$ *where* $v_i \in V$, $a_i \in \text{ACT}$, *such that* $v_0 = u$ *and* $(v_{i-1}, v_i) \in E_{a_i}$*;*
- *an atomic proposition* p *is true at* $v_0 a_1 v_1 a_2 \ldots v_{r-1} a_r v_r$ *in* $\mathcal{T}(\mathcal{K}, u)$ *iff it is true at* v_r *in* $\mathcal{K}$*;*
- *for all actions* a, $E_a^{\mathcal{T}}$ *contains the pairs* $(\overline{v}, \overline{v}av)$ *in* $V^{\mathcal{T}} \times V^{\mathcal{T}}$.

Obviously, the natural projection $\pi \colon \mathcal{T}(\mathcal{K}, u) \to \mathcal{K}, u$ which sends every sequence $\overline{v} = v_0 a_1 v_1 a_2 \ldots v_{r-1} a_r v_r \in V^{\mathcal{T}}$ to its last node v_r defines a bisimulation between $\mathcal{T}(\mathcal{K}, u)$ and $\mathcal{K}, u$.

The variable hierarchy.

Definition 9. *For any* $k \in \mathbb{N}$*, the* k-variable fragment $L_\mu[k]$ *of the* μ*-calculus is the set of formulae* $\psi \in L_\mu$ *that contain at most* k *distinct variables.*

The first three levels of this hierarchy are very easy to separate.

Proposition 10. $L_\mu[0] \subsetneq L_\mu[1] \subsetneq L_\mu[2]$.

With one fixed point variable, only alternation free formulae can be written. Though this suffices to state non-local properties beyond the expressive power of plain modal logic, e.g., $\nu X.\langle a\ \rangle X$ (the model has an infinite a-path), $L_\mu[1]$ remains below $L_\mu[2]$ which contains formulae with genuine fixed point alternation. For example, $\mu X.\nu Y.\langle a\ \rangle X \vee \langle b \rangle Y$ (there is an $\{a, b\}$-path with infinitely many b's) is strictly on the second level of the L_μ alternation hierarchy. Moreover, hard sentences of any level of the alternation hierarchy can be expressed with two variables, as it was shown in [4].

Simultaneous fixed points. There is a variant of L_μ that admits simultaneous fixed points of several formulae. This does not increase the expressive power but often allows for more modular and easier to read formalizations. The mechanism for building simultaneous fixed point formulae is the following: Given formulae $\varphi_1, \ldots, \varphi_k$ and variables $X_1, \ldots, X_k$

$$S := \begin{cases} X_1 \leftarrow \varphi_1 \\ \quad\vdots \\ X_k \leftarrow \varphi_k \end{cases}$$

is called a *system of rules*, which can be used to build the formulae $(\mu X_i : S)$ and $(\nu X_i : S)$.

Semantics: On every Kripke structure $\mathcal{K}$, the system S defines an operator $S^{\mathcal{K}}$ mapping a k-tuple $\overline{X} = (X_1, \ldots, X_k)$ of sets of states to $S_1^{\mathcal{K}}(\overline{X}), \ldots, S_k^{\mathcal{K}}(\overline{X})$ with

$$S_i^{\mathcal{K}}(\overline{X}) := [\![\varphi_i]\!]^{(\mathcal{K},\overline{X})}.$$

As $S^{\mathcal{K}}$ is monotone, there exist the least and greatest fixed points $\mathrm{lfp}(S) = (X_1^{\mu}, \ldots, X_k^{\mu})$ and $\mathrm{gfp}(S) = (X_1^{\nu} \ldots, X_k^{\nu})$. Now set

$$[\![(\mu X_i : S)]\!]^{\mathcal{K}} := X_i^{\mu} \text{ and } [\![(\nu X_i : S)]\!]^{\mathcal{K}} := X_i^{\nu}.$$

Examples of simultaneous fixed point formulae will be given in Section 4. It is known that simultaneous least fixed points can be eliminated in favor of nested individual fixed points (see, e.g., [3, page 27]). Indeed,

$$\big(\mu X : X \leftarrow \psi(X,Y),\ Y \leftarrow \varphi(X,Y)\big) \equiv \mu X.\psi(X, \mu Y.\varphi(X,Y)),$$

and this equivalence generalizes to larger systems in the obvious way. Note that the translation does neither increase the number of variables nor the alternation depth of the formula.

Proposition 11. *Every formula in L_μ with simultaneous fixed points can be translated into an equivalent formula with the same number of fixed point variables and the same alternation depth.*

3 The $\Box$-Free Fragment

We write $\mathcal{K} \subseteq \mathcal{K}'$ to denote that $\mathcal{K}'$ is an extension of $\mathcal{K}$ (or equivalently, that $\mathcal{K}$ is a substructure of $\mathcal{K}'$). A formula ψ is *closed under extensions* if, whenever $\mathcal{K} \models \psi$ and $\mathcal{K} \subseteq \mathcal{K}'$, then also $\mathcal{K}' \models \psi$. In most logics there is a natural notion of *existential formulae* (i.e., formulae where all first-order quantifiers are existential) and it is obvious that existential formulae are closed under extensions. In many cases, also the converse holds. For instance, it is a classical result in model theory, due to Tarski [19] and Łos [14], that every first-order sentence which is closed under extensions is equivalent to an existential sentence. Results of this kind are called *existential preservation theorems*, or also *Łos-Tarski Theorems* (often stated in the dual form, in terms of universal formulae and closure under substructures).

It should be pointed out that there are many scenarios where the Łos-Tarski Theorem fails. In particular this is the case for the k-variable fragments of first-order logic, for every $k \geq 2$ [1, 9], and for first-order logic on finite structures [18].

In the μ-calculus $\langle a \rangle$-operators correspond to existential quantifiers, and $[a]$-operators to universal ones. Therefore, the existential formulae are those in the $\Box$-free fragment $L_\mu^{\Diamond}$, which is defined by the grammar

$$\varphi ::= \mathit{false} \mid \mathit{true} \mid p \mid \neg p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid \mu X.\varphi \mid \nu X.\varphi.$$

We denote the k-variable fragment of this logic by $L_\mu^{\Diamond}[k]$. It has recently been shown that the existential preservation theorem does hold for the μ-calculus [7].

The proof makes use of the μ-automata of Janin and Walukiewicz [10] and it is not clear whether it carries over to the k-variable fragments of L_μ. It turns out that this question is related to the strictness of the variable hierarchy and we discuss it again at the end of this paper.

One technically useful property of fixed point formulae is guardedness. In terms of games this guarantees that plays cannot get hung on a state of the transition system while the second component scans the syntax tree forever.

Definition 12. *An L_μ-sentence ψ is* guarded *if each path in the syntax tree of ψ from a fixed point definition $\lambda X.\varphi$ to an occurrence of X passes through a modality.*

In [12], Kupferman, Vardi, and Wolper give a procedure to transform every L_μ-sentence into an equivalent guarded formula. This procedure does not increase the number of variables and preserves $\Box$-free formulae.

Proposition 13. *Every formula in $L_\mu^\Diamond[k]$ is equivalent to a guarded formula in $L_\mu^\Diamond[k]$.*

We conclude this section with an observation on strategy trees for evaluation games of $\Box$-free formulae. Let $\mathcal{K}, u$ be a transition system and ψ a guarded $L_\mu^\Diamond$-formula. For every strategy f of Verifier in the game $\mathcal{G}(\mathcal{K}, \psi)$, we define a tree $\mathcal{T}_f$, called the *strategy tree* for f, as follows.

The root of $\mathcal{T}_f$ is the initial position (u, ψ). Besides this, the domain of $\mathcal{T}_f$ comprises all initial plays according to f that end with a modal move:

$$\pi = (u, \psi), \ldots, (v, \langle a \rangle \varphi), (w, \varphi).$$

We call the elements $\pi \in \mathcal{T}_f$ *initial segments* of the plays against f. Let $\mathrm{node}(\pi)$ denote the last node w in $\mathcal{K}$ visited by π, and $\mathrm{path}(\pi)$ the sequence of all actions occurring in π. The edges of $\mathcal{T}_f$ are labelled by actions $a \in \textsc{act}$. The a-successors of a segment π are those segments which prolong π by an a-action, i.e., those $\pi' \in \mathcal{T}_f$ with $\pi \leq \pi'$ and $\mathrm{path}(\pi') = \mathrm{path}(\pi) \cdot a$.

Observe that while Verifier moves according to its fixed strategy f in $\mathcal{G}(\mathcal{K}, \psi)$, Falsifier can lead the play to any initial segment $\pi \in \mathcal{T}_f$, by means of some strategy which is not necessarily positional. Further, he can prolong the initial play π to any successor segment π' by adding to his strategy a collection of rules that do not involve any move in $\mathcal{K}$. We call this collection of rules a *local strategy*.

Hence, every edge (π, π') in $\mathcal{T}_f$ corresponds to a local strategy of Falsifier and every (maximal) path in $\mathcal{T}_f$ corresponds to a complete play in $\mathcal{G}(\mathcal{K}, \psi)$ which Falsifier can enforce by composing the local strategies along its edges.

Lemma 14. *If f is a winning strategy for Verifier in $\mathcal{G}(\mathcal{K}, \psi)$, then $\mathcal{T}_f \models \psi$.*

Proof. We can replicate f as a Verifier strategy for $\mathcal{G}(\mathcal{T}_f, \psi)$ in such a way that the resulting plays correspond to plays against f in the original game $\mathcal{G}(\mathcal{K}, \psi)$.

To accomplish this, at every disjunction $(\pi, \varphi_1 \vee \varphi_2)$ with $\mathrm{node}(\pi) = v$, Verifier applies the advice $f(v, \varphi_1 \vee \varphi_2) = (v, \varphi_i)$ and moves to (π, φ_i). This ensures

that for any reachable modal position $(\pi, \langle a \rangle \varphi)$ there is a play $\pi \ldots (v, \langle a \rangle \varphi)$ in $\mathcal{G}(\mathcal{K}, \psi)$, consistent with f, which Verifier then prolongs to a next initial segment π' by moving to the position $(w, \varphi) = f(v, \langle a \rangle \varphi)$. Note that in $\mathcal{T}_f$, the segment π' is an a-successor of π. Hence, in $\mathcal{G}(\mathcal{T}_f, \psi)$ Verifier can move from $(\pi, \langle a \rangle \varphi)$ to (π', φ) and proceed.

A play according to this strategy can end only at positions (π, true) in which case Verifier wins. Otherwise, if the play is infinite, the sequence of initial plays $\pi_1 \leq \pi_2 \leq \cdots \leq \pi_n \leq \ldots$ visited on $\mathcal{T}_f$ converges to an infinite play π_∞ of $\mathcal{G}(\mathcal{K}, \psi)$. Clearly, π_∞ is consistent with f and thus winning. On the other hand, in the considered play of $\mathcal{G}(\mathcal{T}_f, \psi)$ the sequence of occurring subformulae is the same as in π_∞, so Verifier wins in this case as well.

4 The Clique Formulae

We now define the formulae that witness the strictness of the variable hierarchy in $L_\mu^\Diamond$. The formulae $\psi^{(k)}$ do not contain propositional atoms and have actions ij, for all $i, j = 0, \ldots, k-1$.

Definition 15. *For any $k \in \mathbb{N}$, let $\psi^{(k)} := \nu X_0 \,.\, S$ where S is the system of rules*

$$X_i \leftarrow \bigwedge_{j=0}^{k-1} \langle ij \rangle X_j$$

for $i = 0, \ldots, k-1$.

Obviously, $\psi^{(k)}$ belongs to $L_\mu^\Diamond[k]$ and we will prove that no formula in $L_\mu^\Diamond[k-1]$ is equivalent to $\psi^{(k)}$.

As a model of this formula, consider the transition system $\mathcal{C}^k$ with nodes $\{0, \ldots, k-1\}$ and with transition relations $E_{ij} = \{(i, j)\}$, that is, the k-clique with edge labels that indicate the source and the destination of every edge. Clearly, $\mathcal{C}^k, 0 \models \psi^{(k)}$.

To describe the whole class of models, let $\mathcal{T}_i^k$ be the tree unravelling of $\mathcal{C}^k$ from node i. Further, let $\psi_i^{(k)}$ be the formula $(\nu X_i : S)$.

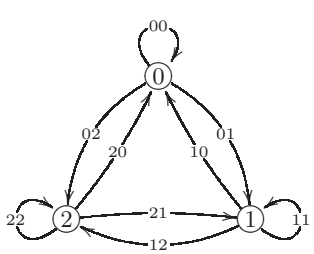


Fig. 1. The transition system $\mathcal{C}^3$

Lemma 16. *For every $i < k$, $\mathcal{T}_i^k \models \psi_i^{(k)}$. In fact, for any tree $\mathcal{T}$ we have that $\mathcal{T} \models \psi_i^{(k)}$ if and only if $\mathcal{T}$ contains a substructure (with the same root) that is isomorphic to $\mathcal{T}_i^k$.*

Hence, $\psi^{(k)}$ just expresses that its models are extensions of (the unravelling of) the clique $\mathcal{C}^k$. What makes this formula hard? To understand this, consider the following simpler variant of $\psi^{(k)}$ which only takes care of the destination of an action and which needs only one variable: $\varphi^{(k)} := \nu X. \bigvee_{i<k} \bigwedge_{j<k} \langle ij \rangle X$. Obviously $\psi^{(k)} \models \varphi^{(k)}$. The models of $\varphi^{(k)}$ are still extensions of the clique, up to renaming of edges. However, the naming of edges makes a crucial difference. Note that $\psi^{(k)}$ requires of its models to contain all paths of $\mathcal{C}^k$, and along these paths, the destination of an action is the source of the next one. We call such a pair of actions ij and jm matching. Intuitively, a formula needs memory to store the last label to be able to check whether all actions along paths are matching. In $\psi^{(k)}$ this is the role of the the k fixed point variables. Instead, $\varphi^{(k)}$ fails to distinguish matching from non matching paths, thus being unable to establish whether all matching paths are present. This leads $\varphi^{(k)}$ to accept structures that $\psi^{(k)}$ would reject.

5 The Hierarchy Theorem

We are now ready to prove our hierarchy theorem. Towards this goal we first analyse strategies in the game $\mathcal{G}(\mathcal{C}^k, \psi)$ for any guarded, $\Box$-free formula $\psi \equiv \psi^{(k)}$. Although Falsifier loses this game, he nevertheless has control over important aspects, notably over which node of $\mathcal{C}^k$ will be reached next.

The full branching power. Fix a winning strategy f for the game $\mathcal{G}(\mathcal{C}^k, \psi)$ and consider any initial segment

$$\pi = (0, \psi), \ldots, (i, \langle ij \rangle \varphi), (j, \varphi) \in \mathcal{T}_f$$

of a play against f. Recall that in the strategy tree $\mathcal{T}_f$, a segment π' is a (jm)-successor of π, if Falsifier has a local strategy in $\mathcal{G}(\mathcal{C}^k, \psi)$ to prolong the play from π to π' through a (jm)-action, so that $\mathrm{path}(\pi') = \mathrm{path}(\pi)(jm)$.

Definition 17. *We say that Falsifier has* full branching power *from $\pi \in \mathcal{T}_f$ if for every $m < k$, he has a strategy to lead the play from π to a successor segment π' with $\mathrm{node}(\pi') = m$, so that Falsifier again has full branching power from π'. More formally, this means that*

$$\mathcal{T}_f, \pi \models \nu X. \bigwedge_{m<k} \bigvee_{j<k} \langle jm \rangle X,$$

or, using the notation of the previous section, $\mathcal{T}_f, \pi \models \varphi^{(k)}$.

In other words, whenever the play enters a new node i, i.e, at the completion of an initial segment, Verifier must allow Falsifier to turn the play towards a successor along any action ij he chooses. Falsifier does not need to commit himself

to more than one modal move at each initial segment. We call a strategy that maintains the full branching power of Falsifier a *full branching strategy.* Observe that in $\mathcal{G}(\mathcal{C}^k, \psi)$ Falsifier may also have strategies that are not full branching. For example, when $\psi \equiv \psi^{(k)} \wedge \langle 00 \rangle \mathit{true}$, Falsifier may choose the conjunct $\langle 00 \rangle \mathit{true}$ which would of course scotch his branching power.

The construction of a full branching strategy in the game $\mathcal{G}(\mathcal{C}^k, \psi)$ against f can again be viewed as a game, played on $\mathcal{T}_f$. At any position π, Challenger selects a node $m < k$, and then Pathfinder moves from π to some successor π' with $\mathrm{node}(\pi') = m$. Challenger wins if Pathfinder cannot move, i.e., if he can force the play to remain finite, otherwise Pathfinder wins. With every move Pathfinder reveals a local Falsifier strategy to prolong π through the action chosen by Challenger. Composing these local strategies yields the desired full branching strategy for Falsifier. Thus, Falsifier has full branching power in the game $\mathcal{G}(\mathcal{C}^k, \psi)$ iff Pathfinder has a winning strategy in the game on $\mathcal{T}_f$.

Lemma 18. *For every guarded, $\Box$-free formula $\psi \equiv \psi^k$ and against every winning strategy f of Verifier, Falsifier has full branching power from the initial position in $\mathcal{G}(\mathcal{C}^k, \psi)$*

Proof. By Lemma 14 we know that $\mathcal{T}_f \models \psi^{(k)}$. Hence $\mathcal{T}_f \models \varphi^{(k)}$ which proves the full branching property.

Non-ambiguous formulae. We call a formula ψ *non-ambiguous* on the transition system $\mathcal{K}$, if for every subformula $\eta \in \mathrm{cl}(\psi)$ with $\eta \neq \mathit{true}$, there exists at most one node v of $\mathcal{K}$ such that $\mathcal{K}, v \models \eta$.

Lemma 19. *Any formula $\psi \in L_\mu^\Diamond$ equivalent to $\psi^{(k)}$ can be transformed, without increasing the number of variables, into an equivalent formula $\psi' \in L_\mu^\Diamond$ that is non-ambiguous on $\mathcal{C}^k$.*

Proof. We eliminate the subformulae that hold at more than one clique node. Assume that $\mathcal{C}^k, j_1 \models \eta$ and $\mathcal{C}^k, j_2 \models \eta$ for some $j_1 \neq j_2$ and some $\eta \in \mathrm{cl}(\psi)$, and let ψ' be the formula obtained by replacing every occurrence of η in ψ by *true*. We claim that $\psi' \equiv \psi$. By the tree model property of L_μ it suffices to establish this on trees.

It is obvious that ψ implies ψ'. For the converse, consider a tree model $\mathcal{T}$ of ψ'. We can partition the universe T of $\mathcal{T}$ according to the label of the incoming transition: $T = T_0 \dot\cup \cdots \dot\cup T_{k-1}$ such that the root of $\mathcal{T}$ belongs to T_0 and every node whose incoming edge is labelled ij belongs to T_j. Next, we define the extension $\mathcal{T}'$ of $\mathcal{T}$ obtained by adding at every node $v \in T_i$ the unravelling $\mathcal{T}_j^k$ of $\mathcal{C}^k$ from the node $j := j_1$ if $i \neq j_1$ and $j := j_2$ otherwise. In this way, every subtree of $\mathcal{T}'$ rooted at a node $v \in \mathcal{T}$ extends a clique unravelling $\mathcal{T}_j^k$, where η holds. Since η is $\Box$-free and hence closed under extensions, it follows that also $\mathcal{T}'_v$, the subtree of $\mathcal{T}'$ rooted at v, is a model of η. Moreover, if f_j is a winning strategy for Verifier in $\mathcal{G}(\mathcal{T}_j^k, \eta)$ it will also be a winning strategy in $\mathcal{G}(\mathcal{T}'_v, \eta)$.

By means of this, we can extend any winning strategy f of Verifier in $\mathcal{G}(\mathcal{T}, \psi')$ to a strategy in $\mathcal{G}(\mathcal{T}', \psi)$ as follows. At every position (v, φ) where $v \in \mathcal{T}$ and

$\varphi \neq \eta$ choose according to f. As Falsifier cannot move in the tree, the play will stay on nodes of $\mathcal{T}$ unless a position (v, η) is reached. When this occurs, Verifier drops f and proceeds with the strategy f_j which is winning in $\mathcal{G}(\mathcal{T}_j^k, \eta)$ and thus in $\mathcal{G}(\mathcal{T}'_v, \eta)$. In that way, every play of $\mathcal{G}(\mathcal{T}', \psi)$ is won by Verifier which means that $\mathcal{T}' \models \psi$ or, equivalently, $\mathcal{T}' \models \psi^{(k)}$.

Observe that in any game on $\psi^{(k)}$ modal moves, i.e., moves of the form $(v, \varphi) \to (v', \varphi')$ where $v \neq v'$, are possible only if $v \in T_i$, and v' is an ij-successor of v, for some pair ij. In particular, $\mathcal{G}(\mathcal{T}', \psi^{(k)})$ does not allow any moves between any node $v \in T_i$ of $\mathcal{T}$ and their jj'-successors $v' \in \mathcal{T}' \backslash \mathcal{T}$. Accordingly, $\mathcal{G}(\mathcal{T}', \psi^{(k)})$ is restricted to positions (v, φ) with $v \in \mathcal{T}$ and therefore just the same game as $\mathcal{G}(\mathcal{T}, \psi^{(k)})$. Since Verifier wins this game we obtain $\mathcal{T} \models \psi^{(k)}$ and can conclude that $\psi' \equiv \psi$.

We are now ready for the final step.

Theorem 20. *No $\Box$-free formula in $L_\mu[k-1]$ is equivalent to $\psi^{(k)}$.*

Proof. Towards a contradiction, suppose that $\psi \in L_\mu^\Diamond[k-1]$ is equivalent to $\psi^{(k)}$. Without loss of generality, we can assume that ψ is guarded and non-ambiguous on $\mathcal{C}^k$.

Fix a winning strategy f for Verifier in the game $\mathcal{G}(\mathcal{C}^k, \psi)$. Such a strategy must exist since $\mathcal{C}^k \models \psi$. We construct a full branching strategy g for Falsifier and prove that it forces the play $f\hat{\ }g$ (defined by the two strategies f and g) to be finite. But this is absurd since a full branching strategy necessarily leads to an infinite play.

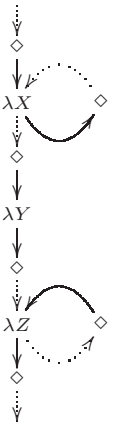
For every initial play in $\mathcal{G}(\mathcal{C}^k, \psi)$ we define a function $S : \{X_1, \ldots, X_{k-1}\} \to \{0, \ldots, k-1\}$, mapping each variable X_i to the node j at which it has last been opened (or to 0 if it has not been opened yet). Intuitively, the strategy of Falsifier is to always force the play towards a node that is *not* in the range of S. At the initial position, set $S(X_i) = 0$ for all X_i. As the game proceeds, S is changed only at positions of form $(j, \lambda X_i \varphi)$ at which it is updated by the rule $S(X_i) := j$.

The strategy g for Falsifier is defined as a concatenation of local strategies. At the initial position, and after any initial segment of the play, Falsifier selects a node $m \leq k$ that is not in the range of S (which must exist as the range of S has size at most $k-1$) and plays according to a local strategy g_m by which he forces the game to a segment π' with $\mathrm{node}(\pi') = m$. There he again selects a value m' not in the range of the current S, and continues with a local strategy $g_{m'}$ forcing the play towards node m'.

Since ψ is non-ambiguous on $\mathcal{C}^k$ we already know that for every fixed point formula $\lambda X_\ell.\varphi \in \mathrm{cl}(\psi)$ there is at most one node j such that the position $(j, \lambda X_\ell.\varphi)$ appears in plays won by Verifier. We claim that in the play $f\hat{\ }g$ any such position appears at most twice, which means that the play is finite.

To prove this, let π be the minimal initial segment of the play $f\hat{\ }g$ in which the position $(j, \lambda X_\ell.\varphi)$ occurs. At this position, S is updated by the rule $S(X_\ell) := j$. Since ψ is guarded, the play must go through a modality before it can reach this position again. The segment π ends with a move from $(j, \langle jm \rangle \eta)$ to (m, η), with $m = \mathrm{node}(\pi)$. We distinguish two cases.

(1) If $m \neq j$, then the position $(j, \lambda X.\varphi)$ will not occur anymore in the play $f\hat{}g$. Indeed at the end of the segment π Falsifier selects a local strategy $g_{m'}$ with $m' \neq j$ (as j is now in the range of S) and keeps the play away from node j until j has been removed from the range of S. But this only happens when a new fixed point definition with the same variable X_ℓ appears on the play, after which the regeneration of $(j, \lambda X_\ell.\varphi)$ is impossible anyway.



(2) If $m = j$ then it is possible to hit position $(j, \lambda X_\ell.\varphi)$ in the following segment. (For instance, it could be the case that $\eta = \lambda X_\ell.\varphi$). However, this can happen only once, since after this regeneration the play must again go through a modality before hitting position $(j, \lambda X_\ell.\varphi)$ a third time. But this is impossible because at position (m, η), Falsifier has decided that the game will now go to a node $m' \neq j$ and stay away from node j until j is removed from the range of S.

Corollary 21. *The variable hierarchy for $L_\mu^\Diamond$ is strict.*

6 Conclusion

We have established that the variable hierarchy is strict in the $\Box$-free fragment of the μ-calculus. A more ambitious goal, of course, is to determine whether the variable hierarchy is strict for the full μ-calculus. We believe that this is the case, and conjecture that the strictness is witessed by the formulae $\psi^{(k)}$.

Conjecture 22. For every $k \geq 1$, the formula $\psi^{(k)}$ is not equivalent to any formula in $L_\mu[k-1]$.

This conjecture is related to the question whether the existential preservation theorem holds for the bounded-variable fragments $L_\mu[k]$. Indeed, if any formula equivalent to ψ^k can be translated into a $\Box$-free formula without increasing the number of variables, then the above conjecture holds as a consequence of

Corollary 23. *If the existential preservation theorem holds for $L_\mu[k]$, then no formula of $L_\mu[k]$ is equivalent to $\psi^{(k+1)}$.*

As mentioned in the second section, the proof of the existential preservation theorem for L_μ in [7] is based on the μ-automata of Janin and Walukiewicz [10]. Every formula of L_μ can be translated to an equivalent μ-automaton. These automata differ from the more common alternating tree automata used, e.g., in [12], and there is no direct, inductive, translation from formulae to μ-automata. The advantage of this detour is that any μ-automaton that is closed under extensions can be transformed in a relatively straightforward way to an equivalent automaton whose transition function is existential. This modified automaton can then be translated back to a $\Box$-free formula of the μ-calculus. However, the construction of the μ-automaton involves a powerset construction and it is not clear whether the number of variables can be preserved. In fact it might well be the case that the existential preservation theorem fails for the bounded variable fragments of L_μ, as it happens for other logics, for instance first-order logic [1, 9].

References

[1] H. Andréka, J. van Benthem, and I. Németi, *Modal languages and bounded fragments of predicate logic*, Journal of Philosophical Logic, 27 (1998), 217–274. 359, 365

[2] A. Arnold, *The mu-calculus alternation-depth is strict on binary trees*, RAIRO Informatique Théorique et Applications, 33 (1999), 329–339. 353

[3] A. Arnold and D. Niwiński, *Rudiments of μ-calculus*, North Holland, 2001. 359

[4] D. Berwanger, *Game logic is strong enough for parity games*, Studia Logica. Special issue on Game Logic and Game Algebra, (2002). 353, 354, 358

[5] D. Berwanger and E. Grädel, *Games and model checking for guarded logics*, in Proceedings of LPAR 2001, Lecture Notes in Computer Science Nr. 2250, Springer, 2001, 70–84. 356, 357

[6] J. Bradfield, *The modal μ-calculus alternation hierarchy is strict*, Theoretical Computer Science, 195 (1998), 133–153. 353

[7] G. d'Agostino and M. Hollenberg, *Logical questions concerning the μ-calculus: interpolation, Lyndon, and Łos-Tarski*, Journal of Symbolic Logic, 65 (2000), 310–332. 354, 359, 365

[8] A. Emerson and C. Jutla, *Tree automata, mu-calculus and determinacy*, in Proc. 32nd IEEE Symp. on Foundations of Computer Science, 1991, 368–377. 356

[9] E. Grädel and E. Rosen, *Preservation theorems for two-variable logic*, Mathematical Logic Quarterly, 45 (1999), 315–325. 359, 365

[10] D. Janin and I. Walukiewicz, *Automata for the modal μ-calculus and related results*, in Proceedings of MFCS 95, Lecture Notes in Computer Science Nr. 969, Springer-Verlag, 1995, 552–562. 360, 365

[11] M. Jurdziński, *Small progress measures for solving parity games*, in STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings, vol. 1770 of Lecture Notes in Computer Science, Springer, 2000, 290–301. 353

[12] O. Kupferman, M. Vardi, and P. Wolper, *An automata-theoretic approach to branching-time model checking*, Journal of the ACM, 47 (2000), 312–360. 356, 360, 365

[13] G. Lenzi, *A hierarchy theorem for the mu-calculus*, in Proceedings of the 23rd International Colloquium on Automata, Languages and Programming, ICALP '96, F. Meyer auf der Heide and B. Monien, eds., vol. 1099 of Lecture Notes in Computer Science, Springer-Verlag, July 1996, 87–97. 353

[14] J. Łos, *On the extending of models (I)*, Fundamenta Mathematicae, 42 (1955), 38–54. 359

[15] R. Parikh, *The logic of games and its applications*, Annals of Discrete Mathematics, 24 (1985), 111–140. 353

[16] M. Pauly, *Logic for Social Software*, PhD thesis, University of Amsterdam, 2001. 354

[17] C. Stirling, *Bisimulation, model checking and other games*. Notes for the Mathfit instructional meeting on games and computation. Edinburgh, 1997. 356, 357

[18] W. W. Tait, *A counterexample to a conjecture of Scott and Suppes*, Journal of Symbolic Logic, 24 (1959), 15–16. 359

[19] A. Tarski, *Contributions to the theory of models I, II*, Indagationes Mathematicae, 16 (1954), 572–588. 359

Implicit Computational Complexity for Higher Type Functionals (Extended Abstract)

Daniel Leivant*

Computer Science Department, Indiana University
Bloomington, IN 47405
leivant@cs.indiana.edu

Abstract. In previous works we argued that second order logic with comprehension restricted to positive formulas can be viewed as the core of Feasible Mathematics. Indeed, the equational programs over strings that are provable in this logic compute precisely the poly-time computable functions.
Here we investigate the provable functionals of this logic, and show that they are precisely Cook and Urquhart's basic feasible functionals, BFF. This further confirms the stability of BFF as a notion of computational feasibility in higher type.
Using a formula-as-type morphism, we also show that BFF consists precisely of the functionals that are lambda representable in F_2 restricted to positive type arguments (and trivially augmented with basic constructors and destructors).

1 Introduction: Feasibility in Higher Type

Computable higher type functionals have been studied for about a century, for several intertwined reasons. One of the first to explicitly consider feasibility of functionals was Robert Constable, who in [5] introduced a machine model for functionals, and considered the definability of the functionals computable therein in a certain function algebra.[1] Mehlhorn [24] refined Constable's algebraic approach by lifting to second order types the characterization given by Cobham [4] of the class FP of functions computable in polynomial time. A corresponding machine model was defined by Kapron and Cook in [15], and shown to be equivalent to Mehlhorn's class.

Another thread in the evolution of the subject was concerned with functional interpretation of proofs in Buss's Bounded Arithmetic. In [1] Buss introduced a system IS_2^1 of arithmetic, and showed that its definable functions form precisely FP. In [2] Buss considered the intuitionistic variant of IS_2^1, and defined a complex functional interpretation which yields a poly-time instantiation theorem for the system. This approach was substantially refined and simplified by Cook and

* Research partially supported by NSF grant CCR-0105651.
[1] See [3] for a correction.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 367–381, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Urquhart in [7, 8], where they defined a system PV^ω, based on the typed lambda calculus, and which supports a functional interpretation of IS_1^2, analogous to Gödel's functional interpretation of first order arithmetic [12]. In [16] Cook and Kapron showed that the second order functionals defined in PV^ω, dubbed Basic Feasible Functional (BFF), are precisely the same as the functionals defined in Mehlhorn's system, viz. the same as the functionals computable by the machine model of [15].

It is not immediately clear that BFF should be admitted as a canonical delineation of the feasible second order functionals. Indeed, Cook exhibited in [6] a functional L that might be considered feasible, and yet falls outside the class BFF_2 of second order functionals in BFF. Cook stated three conditions that any proposed definition of type 2 feasibility must satisfy, and those are in fact satisfied by BFF_2 appropriately augmented with L. However, Seth showed [27] that when two additional and quite natural conditions are imposed, then BFF_2 emerges as the only admissible notion of feasibility for second order functionals. Nonetheless, it is useful to lift doubts about the robustness of BFF_2, and more generally of the class BFF of functionals in *all* finite type defined by terms of PV^ω, by providing additional natural characterizations, notably ones that are not tied umbilically to explicit resource restrictions, as are all characterizations above.

Frameworks for characterizing computational complexity classes without any reference to resources have been developed over the last dozen odd years, jointly referred to as implicit computational complexity. Included are. among others, ramified functional programs, ramified first order proof systems, higher order logics with restricted set-existence, structural restrictions on applicative terms and proofs, and modal and linear type systems and proof systems. Such formalisms are particularly attractive for delineating notions of feasibility in higher type, because they are based on concepts that do not refer directly to functions and computations, whence they lift seamlessly to higher type computing.

One implicit characterization of BFF_2 was proposed in [14], where a ramified imperative programming language of loop programs is presented, dubbed Type 2 Inflationary Tiered Loop Programs (ITLP_2), which yields exactly BFF_2. The imperative framework is appealing from an expository viewpoint, as well as for implementations. On the downside, the framework is not conducive to characterizing feasibility in order > 2, nor does it have natural links to proofs systems, as do characterizations by typed functional programs (via Curry-Howard style morphisms). Moreover, the formalism of [14] is based on a principle of "inflationary tiers", which intertwines tiers with an explicit bounding of resources, not significantly different from the use of Cobham's bounded recurrence in PV^ω, thereby defeating the very rationale of ramification and similar implicit characterizations of computational complexity.

We present here proof theoretic and applicative characterizations of feasibility in higher type, which are not only machine independent, but resource independent. We show that every functional in BFF is provable, in a natural sense, in second order logic with positive comprehension. Using a formula-as-type morphism that merges the ideas of [17] and [22], we show that the functions provable

as above are definable in the polymorphic lambda calculus with positive type arguments, as defined in [23]. We finally close the circle and show that the functionals definable as above are in BFF.

2 Functional Programs over Free Algebras

2.1 Lambda Calculus with Recurrence

Let $C = (\mathbf{c}_1 \dots \mathbf{c}_k)$ be a list of function identifiers, with each $\mathbf{c}_i$ assigned an arity $\underline{arity}(\mathbf{c}_i) = r_i \geqslant 0$. We refer to these as the *constructors*, and to the closed terms inductively generated from C the *ground (C-) terms.* We write $\mathbb{A}(C)$ for the free term algebra generated from C.[2]

We say that the *arity of* $\mathbb{A}(C)$ is $\underline{arity}(C) =_{\mathrm{df}} \max_i r_i$. A non-trivial term algebra of arity 1 with at least two constructors of arity 1 is a *word algebra.* For example, the algebra $\mathbb{N} = \mathbb{A}(\mathbf{0}^0, \mathbf{s}^1)$ is isomorphic to the natural numbers, and the word algebra $\mathbb{W} = \mathbb{A}(\boldsymbol{\varepsilon}^0, \mathbf{0}^1, \mathbf{1}^1)$ is essentially $\{0,1\}^*$; e.g. $\mathbf{0}(\mathbf{1}(\mathbf{1}(\boldsymbol{\varepsilon})))$ can be identified with 011. We posit that function application, *for first order functions,* associates to the right, allowing us to abbreviate the term above as $\mathbf{001}\boldsymbol{\varepsilon}$. We write V_0 for the vocabulary consisting of these three constructors.

We consider the simply-typed lambda calculus, $\boldsymbol{\lambda}_1$, with pairing. The *types* are generated from a base type o using the binary type operations $\rightarrow$ and $\times$. We omit parentheses when in no danger of ambiguity, modulo the proviso that $\times$ binds stronger than $\rightarrow$, and then that $\rightarrow$ and $\times$ associate to the right. For example, $o \rightarrow o \times o \rightarrow o$ abbreviates $o \rightarrow ((o \times o) \rightarrow o)$. We call a type *positive* if it is free of $\rightarrow$.

Each type τ is assigned an order $\underline{order}(\tau) \geqslant 0$ as usual: $\underline{order}(o) = 0$, $\underline{order}(\sigma \rightarrow \tau) = \max[1 + \underline{order}(\sigma), \underline{order}(\tau)]$, and $\underline{order}(\sigma_0 \times \sigma_1) = \max_i[\underline{order}(\sigma_i)]$.

For each type τ we posit an unbounded stock of *variables* of type τ; we superscript variables with their type, when convenient. Terms are generated from the variables using λ-abstraction, type-correct application, pairing (written $\langle E_0, E_1 \rangle$), and type-correct projection (written $\pi_i E$, $i = 0$ or 1). The corresponding types are defined as usual. We write tuples $\langle E_0, \dots, E_m \rangle$ to stand for tuples obtained by iterating pairing, i.e. $\langle E_0, \langle E_1, \cdots, \langle E_{m-1}, E_m \rangle \cdots \rangle \rangle$. The computational rules are β-reduction and projection-reduction. We write $E \Rightarrow E'$ (and say that E converts to E') if E' arises by replacing in E a subterm F by its reductum. We write $\rightleftharpoons$ for the reflexive-symmetric-transitive closure of $\Rightarrow$.

Given C as above, the *basic typed lambda calculus over* C, $\boldsymbol{\lambda}_1(C)$, is the extension of $\boldsymbol{\lambda}_1$ with the constructors of C as constants, where $\mathbf{c}_i$ is assigned type o if $r_i = 0$, and type $\alpha_i =_{\mathrm{df}} o^{r_i} \rightarrow o$ otherwise.[3] Thus the ground C-terms

[2] Note that to avoid an empty algebra, we must have $\underline{arity}(\mathbf{c}_i) = 0$ for some $\mathbf{c}_i \in C$.

[3] The notation $\alpha^r \rightarrow \beta$ can be interchangeably be read as an abbreviation for $(\alpha \times \cdots \times \alpha) \rightarrow \beta$, and for $\alpha \rightarrow (\alpha \rightarrow \cdots (\alpha \rightarrow \beta) \cdots)$. The former convention is closer to practice, the latter dispense with product types.

are precisely the closed λ-terms of type o. We consider $\mathbf{c}_i$ as constants rather than variables because they play a special role in natural extensions of $\boldsymbol{\lambda}_1(C)$.

One such extension is Gödel's system $\mathbf{T}$ of primitive-recursive functionals, which we denote by $\boldsymbol{\lambda}_1\mathsf{R}(C)$. For each type τ we have here a constant R_τ, of type $\alpha_1[\tau]\to\cdots\to\alpha_k[\tau]\to o\to\tau$, where $\alpha_i[\tau]=_{\mathrm{df}}\tau^{r_i}\to\tau$. The reduction rules of $\boldsymbol{\lambda}_1$ are extended with R-reductions:

$$\mathsf{R}_\tau E_1\cdots E_k(\mathbf{c}_i F_1\cdots F_{r_i})\Rightarrow_R E_i G_1\cdots G_{r_i}$$
$$\text{where}\qquad G_j=_{\mathrm{df}}\mathsf{R}_\tau E_1\cdots E_k F_j$$

For example, for $C=(\mathbf{0},\mathbf{s})$ the term $A=_{\mathrm{df}}\mathsf{R}_o\mathbf{0}\mathbf{s}$ defines the addition function; $D=_{\mathrm{df}}\lambda x.\,Axx$ defines doubling; $\mathsf{R}_o(\mathbf{s}\mathbf{0})D$ defines base-2 exponentiation; $P=_{\mathrm{df}}\pi_0\mathsf{R}_{o\times o}\langle\mathbf{0},\mathbf{0}\rangle S$, where $S=_{\mathrm{df}}\lambda x^{o\times o}\langle\pi_0 x,\mathbf{s}\mathbf{p}_0 x\rangle$, defines the predecessor function; and $C=_{\mathrm{df}}\lambda x.\,\mathsf{R}_o xP$ defines the cut-off subtraction function. Also, for each type τ $\lambda x^o y^\tau z^\tau.\,\mathsf{R}_\tau xy(\lambda uz)$ defines the conditional function $\texttt{if}\ x=0\ \texttt{then}\ y\ \texttt{else}\ z$. Similarly, for $C=(\boldsymbol{\varepsilon}^0,\mathbf{0}^1,\mathbf{1}^1)$, $\lambda x^0\,\mathsf{R}_o x\mathbf{0}\mathbf{1}$ defines the concatenation (append) function.

2.2 Bounded Recurrence

In his seminal [13] Grzegorczyk gave his famous classification of primitive recursive functions, closing each class under the schema of *bounded recursion,* i.e. the schema that admits a function f if the functions g_0, g_s and j are admitted, and[4]

$$\begin{aligned} f(\mathbf{0},\boldsymbol{y}) &= g_0(\boldsymbol{y})\\ f(\mathbf{s}x,\boldsymbol{y}) &= g_s(x,\boldsymbol{y},f(x,\boldsymbol{y}))\\ f(x,\boldsymbol{y}) &< j(x,\boldsymbol{y}) \end{aligned}$$

Cobham [4] showed that the functions over $\mathbb{N}$ computable in polynomial time can be characterized by admitting initial functions that yield values of size polynomial in the input's size, and then closing under bounded recurrence on words. That is, a function over $\mathbb{W}$ is in FP iff it is definable from the constructors of $\mathbb{W}$, and the square-size function $\Box(w)=_{\mathrm{df}}1^{|w|^2}\boldsymbol{\varepsilon}$, using explicit definitions and the following schema (BR) of bounded recurrence:[5]

$$\begin{aligned} f(\boldsymbol{\varepsilon},\boldsymbol{y}) &= g_\epsilon(\boldsymbol{y})\\ f(\mathbf{i}x,\boldsymbol{y}) &= g_i(x,\boldsymbol{y},f(x,\boldsymbol{y}))\quad (i=0,1)\\ |f(x,\boldsymbol{y})| &< |j(x,\boldsymbol{y})| \end{aligned}$$

[4] This "doctrine of size" for function definition is strikingly similar to Zermelo's doctrine of size for taming the comprehension principle of naive set theory: the naive admission of set definition by arbitrary description, $\{x\mid P(x)\}$ is replaced by the Separation Schema, which only admits $\{x\in S\mid P(x)\}$, S an already defined set.

[5] Cobham's phrased this schema as "bounded recursion on notations", and insisted on working with natural numbers. This was in accord with the early focus of mathematical logic on number systems, and the exclusive reference to numeric computing in traditional Recursion Theory. It seems to this author that force of habit can no longer excuse the twisting of symbolic computing to artificially fit into an irrelevant mold.

There is no loss of generality in assuming that the vector $\boldsymbol{y}$ of arguments is a singleton, since longer vectors can be symbolically concatenated (using some separator symbol, with the expanded alphabet recoded over $\{0,1\}^*$); component-extraction is then trivially definable using bounded recurrence. The generic statement of (BR) for arbitrary word algebras is similar.

We use the following alternative rendition (BR') of bounded recurrence:

$$\begin{aligned} f(\boldsymbol{\varepsilon}, y) &= g_\epsilon(y) \\ f(\mathbf{i}x, y) &= g_i(x, y, f(x, y)) \restriction J(x, y)) \qquad (i = 0, 1) \end{aligned}$$

Here $u \restriction v$ is the truncation of u to the length of v, e.g. $0010\epsilon \restriction 01\epsilon = 00\epsilon$, and $0110\epsilon \restriction 11111\epsilon = 0110\epsilon$.

Lemma 1. *For every word algebra $\mathbb{A}(C)$, the schema (BR') C) is equivalent, modulo linear time computing, to (BR). That is, if $\mathcal{C}$ is a class of functionals whose functions are closed under linear time, then each instance of (BR) can be effectively converted to an instance of (BR'), and vice versa.*

Proof. We give the proof for $\mathbb{W}$. If f is defined from g_ϵ, g_0, g_1 and j by (BC), then f is defined from g_ϵ, g_0, g_1 and J by (BC'), where $J(x, y) =_{\mathrm{df}} \max[j(\mathbf{0}x, y), j(\mathbf{1}x, y)]$. Conversely, if f is defined from g_ϵ, g_0, g_1 and J by (BC'), then f is defined from g_ϵ, g_0, g_1 and j by (BC), where $j(x, y) =$ if $x = \boldsymbol{\varepsilon}$ then $g_0(x, y)$ else $J(p(x), y)$, where p is the predecessor function. □

2.3 Simultaneous Monotonic Bounded Recurrence

A bounded recurrence is *monotonic* if it is of the form

$$\begin{aligned} f(\boldsymbol{\varepsilon}, y) &= g_\epsilon(y) \\ f(\mathbf{i}x, y) &= g_i(y, f(x, y) \restriction J(x, y)) \qquad (i = 0, 1) \end{aligned}$$

That is, the recurrence functions g_0, g_1 have no direct access to the recurrence argument x. We will adopt this variant, but with monotonicity compensated for by allowing simultaneous recurrence. I.e., a vector $\boldsymbol{f} = (f_1 \dots f_m)$ of functions is defined from m-ary function vectors $\boldsymbol{g}_\epsilon$, $\boldsymbol{g}_0$, and $\boldsymbol{g}_1$ by

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{\varepsilon}, y) &= \boldsymbol{g}_\epsilon(y) \\ \boldsymbol{f}(\mathbf{i}x, y) &= \boldsymbol{g}_i(y, \boldsymbol{f}(x, y) \restriction_m J(x, y)) \qquad (i = 0, 1) \end{aligned}$$

where $\langle z_1 \dots z_m \rangle \restriction_m b =_{\mathrm{df}} \langle z_1 \restriction b, \dots z_m \restriction b \rangle$

Proposition 1. [19] *Each instance of bounded recurrence can be derived using simultaneous monotonic bounded recurrence.*

In order to incorporate bounded recurrence into a definition of higher type functionals, Cook and Urquhart [7, 8] rephrased bounded recurrence as a functional operator, with reduction rules, to be adjoined to the simply typed lambda calculus, resulting in a calculus PV^ω. We use here a slight variant of their calculus. We introduce, for each m an identifier $\bar{\mathsf{R}}_m$, of type $(o \to o)^m \to (o^2 \to o)^{2m+1} \to (o^2 \to o)^m$, for m-ary simultaneous monotonic bounded recurrence. The reductions conveying the intended meaning are:

$$\begin{aligned} \bar{\mathsf{R}}_m(\boldsymbol{G}_\epsilon)(\boldsymbol{G}_0)(\boldsymbol{G}_1)J\boldsymbol{\varepsilon}Y &\Rightarrow_B (\boldsymbol{G}_\epsilon)Y \\ \bar{\mathsf{R}}_m(\boldsymbol{G}_\epsilon)(\boldsymbol{G}_0)(\boldsymbol{G}_1)J(\mathbf{i}X)Y &\Rightarrow_B ((\boldsymbol{G}_i)HY) \restriction_m (JXY) \qquad (i = 0,1) \\ &\qquad \text{where } H =_{\mathrm{df}} \bar{\mathsf{R}}\boldsymbol{G}_\epsilon\boldsymbol{G}_1\boldsymbol{G}_2JXY \end{aligned}$$

Our system $\boldsymbol{\lambda}_1\bar{\mathsf{R}}(\mathbb{W})$ is an extension of $\boldsymbol{\lambda}_1(C)$ with the constants $\bar{\mathsf{R}}_m$ $(m \geqslant 1)$ and $\restriction$. The additional reductions are the usual reductions for the predecessor and discriminator functions; the reductions above for $\bar{\mathsf{R}}_m$; and $x \restriction \boldsymbol{\varepsilon} \Rightarrow \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \restriction x \Rightarrow \boldsymbol{\varepsilon}$, $\mathbf{i}x \restriction \mathbf{j}w \Rightarrow \mathbf{i}(x \restriction w)$.

From Proposition 1 we obtain:

Proposition 2. *A functional over $\mathbb{W}$ is definable in PV^ω iff it is definable in $\boldsymbol{\lambda}_1\bar{\mathsf{R}}(\mathbb{W})$.*

2.4 Equational Programs

To delineate program *feasibility* in higher type we need a suitable broader notion of *computability* in higher type. In past works on feasibility in base type we considered a programming paradigm which is complete for computability in base type, namely equational computation model, in the style of Herbrand-Gödel, familiar from the extensive literature on algebraic semantics of programs. This rudimentary model is particularly suited for integration into logic, since its syntax is contained in the syntax of (equational) logic.

Thus, we consider equational programs for base type, augmented by Gödel's primitive recursion in all finite types, as follows. We refer to first-order types as defined above. Our primitive terms are, first, $\boldsymbol{\varepsilon} : o$, $\mathbf{0}, \mathbf{1}, \mathbf{p} : o \to o$, and $\mathbf{d} : o^3 \to o$. In addition, we include the combinators $\mathsf{K}_{\sigma\tau}$ and $\mathsf{S}_{\rho\sigma\tau}$ for all types. Terms are generated from these and typed variables using type-correct application, pairing, and projections. We do not include λ-abstraction, which is coded using the combinators. A program is a list of equations in base type; to convey an equation in other types we use projections and additional variables. For instance, if $\tau = (o \to o) \to o \times o$, then $E \approx E'$ for terms $E, E' : \tau$ is conveyed by the two equations $\pi_i(Ex^{o\to o}) \approx \pi_i(E'x^{o\to o})$, $i = 0, 1$.

For applications of our results it would be useful to refer to a broader notion of equational programming in higher type, not only in order to broaden the class of functionals considers, but even more importantly so as to extend the class of e*programming paradigms* considered. However, such extensions are orthogonal to our main concern here.

Given a program P, we write V_P for the vocabulary consisting of the constructors and the program-variables in P. We write $P \vdash^{=} E$ if E is a V_P-equation derivable from P in equational logic. That is,[6]

1. $P \vdash^{=} E$ for every $E \in P$;
2. $P \vdash^{=} t \approx t$ for every V_P-term t;
3. If $P \vdash^{=} E[u]$ then $P \vdash^{=} E[\mathbf{t}]$ for every V_P-term $\mathbf{t}$ and variable u;
4. If $P \vdash^{=} E[\mathbf{t}]$ and $P \vdash^{=} \mathbf{t} \approx \mathbf{t}'$, then $P \vdash^{=} E[\mathbf{t}']$.

Naturally, the reduction rules for the combinators $\mathsf{K}_{\sigma\tau}$ and $\mathsf{S}_{\rho\sigma\tau}$ are included as equations in a program, as needed to represent typed λ-abstraction.

3 Provable Programs of Second-Order Logic

3.1 Second Order Logic Augmented with Functional Quantifiers

We refer to a formalism for second order logic, with quantification on (usual, second order) relational variables, and functional-variables in all finite type. The formalism is second order, and not higher order, because we include no comprehension schemas for functions. Equality at base type, although second-odred definable, is included as a primitive logical constant.[7] The basic identifiers are thus the constants of the vocabulary in hand (here $V_0 = \{\boldsymbol{\varepsilon}, \mathbf{0}, \mathbf{1}\}$), the variables, and the program-functions. All these come with their types. We use throughout the usual Gentzen-Prawitz nqturql deduction system. We are keenly interested in restrictions of this formalism, $\mathbf{L}_2$, where set existence, i.e. the comprehension principle (which is conveyed in the natural deduction system by the relational $\forall$-elimination rule), is restricted to formulas in a certain syntactic classes $\mathcal{C}$. We denote the corresponding sub-formalism by $\mathbf{L}_2[\mathcal{C}]$.

3.2 Second Order Delineation of Data

It is well-known that inductively generated algebras are second-order definable. For instance, the natural numbers are second-order definable in the sense that in every structure the elements satisfying the following predicate N are precisely the denotations of the numerals $\mathbf{0}, \mathbf{s}(\mathbf{0})$....

$$\begin{aligned} N[x] \quad &\equiv_{\mathrm{df}} \quad \forall Q \ (\ Cl_N[Q] \ \rightarrow \ Q(x) \ ) \\ \text{where} \qquad Cl_N[Q] \quad &\equiv_{\mathrm{df}} \quad Q(\mathbf{0}) \wedge \forall u (Q(u) \rightarrow Q(\mathbf{s}(u))) \end{aligned}$$

Similarly, in every structure for a vocabulary containing V_0 the elements satisfying the following formulas $W[x]$ are precisely the denotations of the base

[6] Note that symmetry and transitivity of equality are derived from these rules.

[7] This allows to infer $x \approx x' \rightarrow \varphi[x] \rightarrow \varphi[x']$ for arbitrary formulas φ, even when comprehension is very weak.

terms:

$$W[x] \quad \equiv_{\mathrm{df}} \quad \forall Q \ (\ Cl_W[Q] \ \rightarrow \ Q(x) \ )$$

$$\text{where} \qquad Cl_W[Q] \quad \equiv_{\mathrm{df}} \quad Q(\boldsymbol{\varepsilon}) \wedge \forall u(Q(u) \rightarrow Q(\mathbf{0}(u))) \wedge \forall u(Q(u) \rightarrow Q(\mathbf{1}(u)))$$

In general, if A is an (inductively generated) free term algebra, we write $A[x]$ for $\forall Q \ Cl_A[Q] \rightarrow Q(x)$, where $Cl_A[Q]$ states that Q is closed under the constructors of A.

3.3 Second Order Definition of Functionality

For each type τ we define a formula Tot_τ, with one free variable of type τ. The definition is by discourse-level recurrence on τ, as follows.

$$Tot_o[x] \equiv W[x]$$
$$Tot_{\tau\rightarrow\sigma}(x) \equiv \forall y^\tau .\ Tot_\sigma[y] \rightarrow Tot_\tau[x(y)]$$

We say that a program P, for a program-function $\mathbf{f}$ of type τ, is *purely-provable* in an appropriately expressive formalism $\mathbf{L}$ if $\bar{P} \vdash_{\mathbf{L}} Tot_\tau(\mathbf{f})$, where $\bar{P}$ is the universal closure of the conjunction of the equations in P. For examples of provable functions, see [22]. For a second-order example, consider the iteration functional $J : (\mathbb{W} \rightarrow \mathbb{W}) \rightarrow \mathbb{W} \rightarrow \mathbb{W}$, given by the program $J(f)(\boldsymbol{\varepsilon}) \approx \boldsymbol{\varepsilon}$, $J(f)(\mathbf{i}x) \approx f(J(f)(x))$ $(i = 0, 1)$. Below is a derivation showing that J is purely provable. For readability we use an inference bar labeled with *parallel* to point out (differently displayed) identical formulas, and a double-bar for compound inferences with trivial details omitted.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(2)}{Tot_o[x]}}{W[x]} \ \|}{Cl_W[\lambda z.Tot_{o\rightarrow o}[Jfz]] \rightarrow Tot_{o\rightarrow o}[Jfx]} \quad \overset{\mathcal{D}}{Cl_W[\lambda z.Tot_{o\rightarrow o}[Jfz]]}}{Tot_{o\rightarrow o}[Jfx]}}{\forall x.Tot_o[x] \rightarrow Tot_{o\rightarrow o}[Jfx]} \ (2) \ \|}{Tot_{o\rightarrow o}[Jf]}}{\forall f.Tot_{o\rightarrow o}[f] \rightarrow Tot_{o\rightarrow o}[Jf]} \ (1) \ \|}{Tot_{(o\rightarrow o)\rightarrow o\rightarrow o}[J]}$$

where

$$\mathcal{D} \quad \equiv_{\mathrm{df}} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(1)}{Tot_{o\rightarrow o}[f]}}{Tot_{o\rightarrow o}[Jfz] \rightarrow Tot_{o\rightarrow o}[f(Jfz)]} \quad \overset{(3)}{Tot_{o\rightarrow o}[Jfz]}}{Tot_{o\rightarrow o}[f(Jfz)]}}{Tot_{o\rightarrow o}[f(J(f)(\mathbf{i}z))]}}{\forall z.Tot_{o\rightarrow o}[Jfz] \rightarrow Tot_{o\rightarrow o}[Jf(\mathbf{i}z)]} \ (3)}{W[\boldsymbol{\varepsilon}]}}{W[Jf\boldsymbol{\varepsilon}]} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overset{(1)}{Tot_{o\rightarrow o}[f]}}{Tot_{o\rightarrow o}[Jfz] \rightarrow Tot_{o\rightarrow o}[f(Jfz)]} \quad \overset{(3)}{Tot_{o\rightarrow o}[Jfz]}}{Tot_{o\rightarrow o}[f(Jfz)]}}{Tot_{o\rightarrow o}[f(J(f)(\mathbf{i}z))]}}{\forall z.Tot_{o\rightarrow o}[Jfz] \rightarrow Tot_{o\rightarrow o}[Jf(\mathbf{i}z)]} \ (3)}{Cl_W[\lambda z.Tot_{o\rightarrow o}[Jfz]}$$

Note that relational $\forall$-elimination is used here for the formally complex formula $Tot_{o\to o}[Jfz]$.

From Girard's [11], it is clear that the functions over $\mathbb{N}$ that are purely-provable in $\mathbf{L}_2$ are precisely the provably recursive functions of second-order arithmetic. By restricting comprehension we obtain successively smaller classes of functions. Since the provable functions of second order logic form such a vast class, one might expect that $\mathbf{L}_2$ is a poor starting point for delineating radically smaller complexity classes, such as FP, and that a first order formalisms would better suit the purpose. In fact, the opposite is the case.

3.4 Function Provability over Rudimentary Data-Axioms

Some caution is appropriate when comprehension is restricted beyond a certain point. The proof of Girard's result is based on an interpretation of second-order arithmetic in second-order logic, which requires comprehension for non-first-order formulas. Indeed, if comprehension is restricted to first-order formulas, then even subtraction for unary numerals is not provable [20]. This can be explained by the fact that data objects are used in computing in two orthogonal ways: as structured storage of bits of information, and as template that drive iterative constructs. The first aspect is exemplified by data-storage devices, whose memory architecture may in fact be non-sequential (e.g. hyper-cubes). Essential to this role is the ability to recognize each digit of the data visited. In contrast, the use of data as templates for iteration and recursion is umbilically tied to the inductive construction of data, on which the the second order definition of data is based. Data detection can be recovered, but at a cost, both logical and computational, that is no longer available in weak formalisms.

We define *Rudimentary Theory* for $\mathbb{W}$, $\mathbf{RT}(\mathbb{W})$, to have as vocabulary the constructors of $\mathbb{W}$ and a unary predicate identifier W_0, intended to range over $\mathbb{W}$.[8] The axioms are:

1. Closure properties of W_0, which we express as natural deduction rules:

$$\mathsf{W}_0(\varepsilon) \qquad \frac{\mathsf{W}_o(\mathbf{t})}{\mathsf{W}_0(\mathbf{it})} \qquad \frac{\mathsf{W}_o(\mathbf{it})}{\mathsf{W}_0(\mathbf{t})} \qquad (i = 0, 1,\ \mathbf{t}\ \text{ a term})$$

2. Determinateness of W_0: $\forall x\ (\ \mathsf{W}_0(x) \to (x \approx \varepsilon\ \vee\ x \approx \mathbf{0p}x\ \vee\ x \approx \mathbf{1p}x)\ )$ which we express by the natural deduction rule

$$\frac{\mathsf{W}_0(\mathbf{t}) \quad \varphi[\varepsilon] \quad \varphi[\mathbf{0}u] \quad \varphi[\mathbf{1}u]}{\varphi[\mathbf{t}]}$$

(u a free variable not free in open assumptions.)

[8] We use the subscript to disambiguate this primitive identifier from the defined second order predicate W.

We will say from now that a program P for a type τ functional f is *provable* in $\mathbf{L} = \mathbf{L}_2(\mathbb{W})$ if

$$\bar{P}, \mathbf{RT}(\mathbb{W}) \vdash_{\mathbf{L}} Tot_\tau(\mathbf{f}).$$

In that case we also say that the functional computed by P is *provable* in $\mathbf{L}$. The following is fairly straightforward:

Proposition 3. *Let $\mathcal{C}$ be a class of first order formulas closed under substitution of $\mathcal{C}$-definable formulas for relational variables.*[9] *Let $\mathbf{L} = \mathbf{L}_2[\mathcal{C}](\mathbb{W})$. Then the provable functionals of $\mathbf{L}$ are closed under composition and application.*

Let + stand for the class of positive first-order formulas, that is the formulas where no relational constant occurs in the negative scope of an implication (or negation). (Note that we do not consider equality as a relational identifier.) The main result concerning provability of first-order functions are:

Theorem 1. ([18, 20, 21]) *The provable functions of $\mathbf{L}_2[+](\mathbb{W})$ are precisely the functions computable in polynomial time.*

The main result of this paper is:

Theorem 2. *For every type τ and type-τ functional f over $\mathbb{W}$ the following are equivalent. (1) The functional f is in BFF. (2) The functional f is provable in $\mathbf{L}_2[+](\mathbb{W})$. (3) The functional f is λ-definable in the polymorphic lambda calculus $\boldsymbol{\lambda}_2[+](\mathbb{W})$ of [23].*

The Theorem will follow from the three implications proved below in Propositions 4, 5 and §6 (all truncated here due to space restrictions.)

4 Provability of the BFF Functionals

We start by proving that every functional in BFF is provable in $\mathbf{L}_2[+](\mathbb{W})$. Since our notion of provability refers to equational programs, we cast BFF as an equational calculus, using the combinators $\mathsf{K}_{\sigma\tau}$ and $\mathsf{S}_{\rho\sigma\tau}$, as above. The reductions for the combinators and the constants are then phrased as equations.[10] The recurrence operator is now a functional identifier, with the recurrence reductions given as part of the equational program.

Proposition 4. *If $(P, \mathbf{f})$ is a program corresponding to a term of $\boldsymbol{\lambda}_1\bar{\mathsf{R}}(\mathbb{W})$, then P is provable in $\mathbf{L}_2[+](\mathbb{W})$.*

The proof is in the full paper.

[9] Examples are the class of all first-order formulas, and the class of positive formulas, i.e. where no relational variable occurs in a negative position.

[10] These equations can be formulated as equations in base type by supplying typed variables as arguments.

5 From Set Abstraction to Type Abstraction

Let $\mathbf{2\lambda}$ be the Girard-Reynolds polymorphically typed λ-calculus [11, 26]. We posit a base type o, in addition to the types denoted by type-variables. Let $\mathbf{2\lambda}_o^p(\mathbf{W})$ be the extension of $\mathbf{2\lambda}$ with the constructor, destructor, and discriminator functions over $\mathbb{W}$, with their usual types over o. However, type application is restricted to type arguments without $\rightarrow$ or $\forall$ (i.e. for types generated from o and type variables using only $\times$). That is, type quantifiers in $\mathbf{2\lambda}_o^p(\mathbf{W})$ range over multiplicative types only. (See [23] for detail.)

Let $w \in \mathbf{W}$, and let $\bar{w}$ be the Church-Böhm-Berarducci abstraction term for w. In analogy to the Fortune-O'Donnell numerals [9, 25, 10], we have the *polymorphic form* of $\bar{w}$,

$$\bar{w}^{FO} =_{\mathrm{df}} \Lambda t.\, \lambda v_0^{t\rightarrow t} v_1^{t\rightarrow t} v_\epsilon^t.\, w[v_0, v_1, v_\epsilon],$$

which is of type $\omega \equiv \forall t.\, t \rightarrow (t \rightarrow t)^2 \rightarrow t$.

We say that an expression $E : \omega \rightarrow \omega$ *represents* a function $f : \mathbb{W} \rightarrow \mathbb{W}$ if $E\bar{w}^{FO} \equiv \overline{f(w)}^{FO}$ for all $w \in \mathbb{W}$. In [23] we showed that a function over $\mathbb{W}$ is represented in $\mathbf{2\lambda}_o^p(\mathbf{W})$ iff it is poly-time.

We now extend the definition of representability to higher type. The definition refers to the unrestricted calculus $\mathbf{2\lambda}$. For a type $\tau = \tau[o]$ let $\tau[\omega]$ be the polymorphic type obtained by replacing each occurrence of o with ω. We define the notion of representation for functionals of type τ by recurrence on τ. A functional of type τ will be represented by an expression of type $\tau[\omega]$. The recurrence base is the Fortune-O'Donnell representation: an expression M of type ω represents $w \in \mathbb{W}$ if M is equal under conversions to $\bar{w}$.

If τ is $\tau_1 \ldots \tau_r \rightarrow o$, then a term M of type $\tau[\omega]$ *represents* a functional f of type τover $\mathbb{W}$, if for all terms $A_1 : \tau_1[\omega], \ldots, A_r : \tau_r[\omega]$, if A_i represents $g_i : \tau_i$, then $MA_1 \cdots A_r$ represents $fg_1 \cdots g_r$.

Note that this definition disregards the values of functionals for arguments that are not representable. However, when we consider a sub-formalism of $\boldsymbol{\lambda}_2(\mathbb{W})$ such as $\boldsymbol{\lambda}_2[+](\mathbb{W})$, the definition still refers to the functionals definable in the full formalism, namely a very broad collection.

Proposition 5. *If a functional over $\mathbb{W}$ is provable in $\mathbf{L}_2[+](\mathbb{W})$, then it is representable in $\mathbf{2\lambda}_o^p(\mathbf{W})$.*

The proof uses a Curry-Howard style homomorphism κ, combining the first-order oblivion used [17] and the use of unit type to represent equality, in [22]. An outline of the definition is presented here in Tables 1 and 2.

Theorem 3. [Representation] *Let $(P, \mathbf{f})$ be a program computing a function f over $\mathbb{W}$. If $\mathcal{D}$ is a deduction of $\mathbf{L}_2[+](\mathbb{W})$ deriving $Tot_\tau(\mathbf{f})$ from P, then $\kappa\mathcal{D}$ represents in $\boldsymbol{\lambda}_2[+](\mathbb{W})$ the functional of τ computed by $(P, \mathbf{f})$.*

Table 1. The homomorphism κ from $\mathbf{L}_2[+](\mathbb{W})$ to $\boldsymbol{\lambda}_2[+](\mathbb{W})$: equality and data rules

$\mathcal{D}$	$\kappa\mathcal{D}$
$\mathbf{t} \approx \mathbf{t}$	$\bullet$
$\dfrac{\overset{\mathcal{D}_0}{\mathbf{t} \approx \mathbf{t}'} \quad \overset{\mathcal{D}_1}{\varphi[\mathbf{t}]}}{\varphi[\mathbf{t}']}$	$\kappa\mathcal{D}_1$
$\mathsf{W}_0(\varepsilon)$	ε
$\dfrac{\overset{\mathcal{D}_0}{\mathsf{W}_0(\mathbf{t})}}{\mathsf{W}_0(\mathbf{it})}\ (i = 0, 1)$	$\mathsf{i}\kappa\mathcal{D}_0$
$\dfrac{\overset{\mathcal{D}_0}{\mathsf{W}_0(\mathbf{it})}}{\mathsf{W}_0(\mathbf{t})}\ (i = 0, 1)$	$\mathsf{p}\kappa\mathcal{D}_0$
$\dfrac{\overset{\mathcal{T}}{\mathsf{W}_0(\mathbf{t})} \quad \overset{\mathcal{D}_\epsilon}{\varphi[\varepsilon]} \quad \overset{\mathcal{D}_0}{\varphi[\mathbf{0}u]} \quad \overset{\mathcal{D}_1}{\varphi[\mathbf{1}u]}}{\varphi[\mathbf{t}]}$	$\mathsf{d}(\kappa\mathcal{T})(\kappa\mathcal{D}_\epsilon)(\kappa\mathcal{D}_0)(\kappa\mathcal{D}_1)$

6 From Polymorphic Representability to BFF

We tackle the remaining implication of Theorem 2, and exhibit a semantic-preserving transformation ξ of terms M of $\boldsymbol{\lambda}_2[+](\mathbb{W})$ to terms of $\boldsymbol{\lambda}_1\bar{\mathsf{R}}(\mathbb{W})$. This is the trickiest of the three implications we prove to establish Theorem 2, attesting to the ad hoc nature of boundedness conditions. Details are in the full paper.

References

[1] Samuel Buss. *Bounded Arithmetic*. Bibliopolis, Naples, 1986. 367
[2] Samuel Buss. The polynomial hierarchy and intuitionistic bounded arithmetic. In *Structure in Complexity*, LNCS 233, pages 77–103, Berlin, 1986. Springer-Verlag. 367

[3] Peter Clote. A note on the relation between polynomial time functionals and constable's class k. In Hans Kleine-Büning, editor, *Computer Science Logic*, LNCS 1092, pages 145–160, Berlin, 1996. Springer-Verlag. 367

[4] A. Cobham. The intrinsic computational difficulty of functions. In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962. 367, 370

[5] Robert Constable. Type 2 computational complexity. In *Fifth Annual ACM Symposium on Theory of Computing*, pages 108–121, New York, 1973. ACM. 367

[6] Stephen Cook. Computability and complexity of higher type functions. In Y. Moschovakis, editor, *Logic from Computer Science*, pages 51–72. Springer-Verlag, New York, 1991. 368

[7] Stephen A. Cook and Alasdair Urquhart. Functional interpretations of feasible constructive arithemtic (extended abstract). In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 107–112, 1989. 368, 372

[8] Stephen A. Cook and Alasdair Urquhart. Functional interpretations of feasible constructive arithemtic. *Annals of Pure and Applied Logic*, 63:103–200, 1993. 368, 372

[9] Steven Fortune. *Topics in computational complexity.* Phd dissertation, Cornell University, 1979. 377

[10] Steven Fortune, Daniel Leivant, and Michael O'Donnell. The expressiveness of simple and second-order type structures. *Journal of the ACM*, 30(1):151–185, January 1983. 377

[11] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'anal yse, et son application a l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 63–92, Amsterdam, 1971. North-Holland. 375, 377

[12] Kurt Gödel. Über eine bisher noch nicht benutzte erweiterung des finiten standpunktes. *Dialectica*, 12:280–287, 1958. 368

[13] A. Grzegorczyk. Some classes of recursive functions. In *Rozprawy Mate. IV*. Warsaw, 1953. 370

[14] R. Irwin, B. M. Kapron, and J. Royer. On characterizations of the basic feasible functionals part i. *Journal of Functional Programming*, 11:117–153, 2001. 368

[15] B. M. Kapron and S. A. Cook. A new characerization of type-2 feasibility. *SIAM Journal of Computing*, 25:117–132, 1996. 367, 368

[16] Bruce Kapron and Stephen Cook. Characterizations of the basic feasible functionals of finite type. In S. Buss and P. Scott, editors, *Feasible Mathematics*, pages 71–95. Birkhauser-Boston, 1990. 368

[17] Daniel Leivant. Contracting proofs to programs. In P. Odifreddi, editor, *Logic and Computer Science*, pages 279–327. Academic Press, London, 1990. 368, 377

[18] Daniel Leivant. A foundational delineation of poly-time. *Information and Computation*, 110:391–420, 1994. (Special issue of selected papers from LICS'91, edited by G. Kahn). Preminary report: A foundational delineation of computational feasibility, in Proceedings of the Sixth IEEE Conference on Logic in Computer Science, IEEE Computer Society Press, 1991. 376

[19] Daniel Leivant. Ramified recurrence and computational complexity I: Word recurrence and poly-time. In Peter Clote and Jeffrey Remmel, editors, *Feasible Mathematics II*, Perspectives in Computer Science, pages 320–343. Birkhauser-Boston, New York, 1994. 371

[20] Daniel Leivant. Termination proofs and complexity certification. In N. Kobayashi and B. Pierce, editors, *Theoretical aspects of computer software*, volume 2215 of *LNCS*, pages 183–200, Berlin, 2001. Springer-Verlag. 375, 376

[21] Daniel Leivant. Calibrating computational feasibility by abstraction rank. In Gordon Plotkin, editor, *Seventeenth IEEE Annual Symposium on Logic in Computer Science*. IEEE Computer society Press, 2002. 376

[22] Daniel Leivant. Intrinsic reasoning about functional programs I: first order theories. *Annals of Pure and Applied Logic*, 114:117–153, 2002. 368, 374, 377

[23] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Fundamenta Informaticae*, 19:167–184, 1993. 369, 376, 377

[24] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. *JCSS*, 12:147–178, 1976. 367

[25] Michael O'Donnell. A programming language theorem which is independent of Peano Arithmetic. In *Eleventh Annual ACM Symposium on Theory of Computing*. ACM, 1979. 377

[26] John Reynolds. Towards a theory of type structures. In J. Loeckx, editor, *Conference on Porgrammin*, pages 408–425, Berlin, 1974. 377

[27] Anil Seth. Some desirable conditions for feasible functionals of type 2. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 320–331, Washington, DC, 1993. IEEE Computer Society Press. 368

Table 2. The homomorphism κ from $\mathbf{L}_2[+](\mathbb{W})$ to $\boldsymbol{\lambda}_2[+](\mathbb{W})$: logical rules

$\mathcal{D}$	$\kappa\mathcal{D}$
ψ^ℓ (labeled assumption)	$x_\ell^{\kappa\psi}$ (ℓ-th variable of type $\kappa\psi$)
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \varphi_0\end{matrix} \quad \begin{matrix}\mathcal{D}_1 \\ \varphi_1\end{matrix}}{\varphi_0 \wedge \varphi_1}$	$\langle \kappa\mathcal{D}_0, \kappa\mathcal{D}_1 \rangle$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \varphi_0 \wedge \varphi_1\end{matrix}}{\varphi_i}$	$\pi_i \kappa\mathcal{D}_0$
$\dfrac{\begin{matrix}(\ell) \\ \psi \\ \mathcal{D}_0 \\ \varphi\end{matrix}}{\varphi \to \psi}\ (\ell)$	$\lambda x_\ell^{\kappa\psi}.\, \kappa\mathcal{D}_0$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \varphi \to \psi\end{matrix} \quad \begin{matrix}\mathcal{D}_0 \\ \varphi\end{matrix}}{\psi}$	$(\kappa\mathcal{D}_0)(\kappa\mathcal{D}_1)$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \varphi[z]\end{matrix}}{\forall x\, \varphi[x]}$	$\kappa\mathcal{D}_0$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \forall x\, \varphi[x]\end{matrix}}{\varphi[\mathbf{t}]}$	$\kappa\mathcal{D}_0$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \varphi\end{matrix}}{\forall S\, \varphi[S]}$	$\Lambda S.\, \kappa\mathcal{D}_0$
$\dfrac{\begin{matrix}\mathcal{D}_0 \\ \forall S\, \varphi[S]\end{matrix}}{\varphi[\lambda z.\psi]}$	$(\kappa\mathcal{D}_0)(\kappa\psi)$

On Generalizations of Semi-terms of Particularly Simple Form*

Matthias Baaz and Georg Moser

Vienna University of Technology, Institut für Algebra und Computermathematik
E118.2, Wiedner Hauptstrasse 8–10, A-1040 Vienna
{baaz,moser}@logic.at

Abstract. We show that Gentzen's sequent calculus admits generalization of semi-terms of particularly simple form. This theorem extends one of the main results in [BS95] to languages $\mathcal{L}$ with functions of arbitrary arity and the central result in [KP88] to semi-terms.

Keywords: Structure of Proofs, Complexity of Programs, Proof Theory

1 Introduction

It is well-known that cut-free proofs in Gentzen's sequent calculus admit a much simpler structure than arbitrary proofs. E.g. recall that cut-free proofs have the *subformula property*: Any formula occurring in the proof is a subformula of the endformula. Furthermore, much is known about the term-structure of cut-free proofs. We can transform any cut-free proof Π of A into a most-general term-minimal cut-free proof Π' so that the maximal depth of terms t in Π' is elementarily bounded in the length[1] of the given proof Π and the logical complexity of A, cf. [KP88]. (Note that the logical structure of Π and Π' coincides.)

In this paper we study cut-free proofs in the context of *generalizations of proofs*, i.e., we are interested in the question whether we can generalize a given proof to a similar proof of a more general statement. Here one of the first questions is:

> *Is it possible to transform a given proof of $A(t)$ into a proof of $A(t')$, where t' is the result of replacing sufficiently deep subterms of t by corresponding variables*?

This form of generalization is usually called generalization of (particularly) simple form. Some calculi admit this type of generalization trivially without changing the (logical) structure of derivations. Take for example first-order resolution calculi: The generalizations are provided by lifting lemmas (cf. [CL73], Lemma 5.1). We conclude from the results stated in the first paragraph that cut-free proofs in Gentzen's LK admit generalization of simple form.

* The work on this paper was partly sponsored by FWF grant P15477-MAT.

[1] The length of a proof refers to the number of steps in the proof.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 382–397, 2002.
© Springer-Verlag Berlin Heidelberg 2002

To make this precise, we have to fix what is understood by "logical structure". Usually the logical structure of a sequent calculus proof is described as its *proof-skeleton*, i.e., as a rooted tree whose nodes are labeled by inference rules. We write $\tau(t)$ to denote the maximal depth of the term t.

For any sequent $S(a)$, and any proof-skeleton, there exists an $M \in \mathbb{N}$, such that for any term t it holds: If there exists a cut-free proof Π of $S(t)$ (in Gentzen's LK) with the fixed skeleton, then there exists a most-general term r such that (i) the transformed proof Π' proves $S(r)$, (ii) the proof-skeletons of Π and Π' coincide, (iii) $r\sigma = t$, for some substitution σ, and (iv) $\tau(r) \leq M$. We say that cut-free sequent calculus proofs admit *generalizations of particularly simple form with bound* M.

However, proof-skeletons are a too restrictive measure of the logical structure of proofs. We consider the following question:

> *Does Gentzen's LK admit generalization of simple form for terms containing bound variables?*

The answer to this question is negative, if we demand that the transformed proof has the same skeleton as the original one, cf. Section 2. However, if we admit controlled variations in the skeleton, then we can answer the question positively. We allow that single quantifier introductions are replaced by introductions of *blocks* of quantifiers. These changes necessarily trigger variations in the logical form of the endformula A.

Let an *extension* A' of the formula A be obtained by replacing strong quantifier occurrences $\mathsf{Q}x$ in A by $\mathsf{Q}x, \overline{z}$ for a (suitably defined) string of bound variables $\overline{z}$. (Note that A' is logical stronger than A.) Now, we can show the following:

For any sequent $S(a)$, and any skeleton, there exists an $M \in \mathbb{N}$, such that for any term or semi-term t it holds: If there exists a cut-free proof Π of $S(t)$ (in Gentzen's LK) with the fixed skeleton, then there exists a most-general term or semi-term r and a cut-free proof Π' such that

(i) the transformed proof Π' proves $S'(r)$,
(ii) the proof-skeletons of Π and Π' almost coincide: Single quantifier introductions are replaced by introductions of blocks of quantifiers,
(iii) $r\sigma = t$, and
(iv) $\tau(r) \leq M$.

Similar to above the bound M is computed by an elementary function depending only on the length of the given proof and the number of symbols in $A(a)$. Although this result is presented with respect to Gentzen's LK it is by no means necessary to stick to Gentzen's original formulation. In particular the theorem is true for *any* analytic sequent calculus that admits the usual quantifier rules.

We believe that this result is not only of interest in the area of generalization of proofs, but also of general interest, as we gain an extended insight into the

structure of cut-free proofs. In [Pud98] the correspondence between the structure of proofs and the complexity of programs is emphasized. In a similar way our results can be applied to study the complexity of programs via the study of (the structure of) proofs.

2 Preliminaries

Recall that *terms* are constructed from constants, free variables, and function symbols; while *semi-terms* are like terms but may as well contain bound variables. We employ an arbitrary (but equivalent) variant of Gentzen's sequent calculus [Gen34], denoted as **LK**. The *length* (denoted as $|\Pi|$) of a proof Π is the number of sequents in Π. The *size* (denoted as $\mathrm{size}(\Pi)$) of a proof Π is the number of symbols in Π.

We employ *proof-matrices* as partial proof-descriptions. Assume A can be written as $A(t_1, \ldots, t_n)$ such that all maximal occurring terms and semi-terms are indicated. Then A is called *term-free* if the t_i are distinct (free) variables.

Definition 1. *A* (proof-)matrix *is a rooted tree whose vertices are labeled by term-free sequent formulas. The leaves are marked by atomic sequents, only. The edges are marked by inference rules of* **LK**. *For each sequent the principal and auxiliary formulas are marked.*

It is important to note that the number of distinct proof-matrices for a given length k cannot be uniformly bounded, contrary to the fact that only finitely many skeletons of length bounded by k can exists.[2] This is due to the fact that arbitrary complex sequents can be attached to the nodes. However, if we restrict our attention to cut-free matrices together with a given endsequent, the subformula property enables us to consider only finitely many matrices.

We restate an example from [BW01]. This example will show that not even cut-free **LK** admits generalization of semi-terms of simple form, if the logical structure of the proof or the endsequent is kept fixed.

Table 1. Representation of even numbers

$$
\begin{array}{c}
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(s_3,s_4) \to P(s_3,s_4)}{\forall\alpha P(r_1(\alpha),r_2(\alpha)) \to P(s_3,s_4)} \quad \dfrac{P(s_5,s_6) \to P(s_5,s_6)}{\forall\alpha P(r_1(\alpha),r_2(\alpha)) \to P(s_5,s_6)}}{\forall\alpha P(r_1(\alpha),r_2(\alpha)), \forall\alpha P(r_1(\alpha),r_2(\alpha)) \to P(s_3,s_4) \wedge P(s_5,s_6)}}{\forall\alpha P(r_1(\alpha),r_2(\alpha)) \to P(s_3,s_4) \wedge P(s_5,s_6)}}{\forall\alpha P(r_1(\alpha),r_2(\alpha)) \to \exists\beta(P(s_3,r_4(\beta)) \wedge P(r_5(\beta),s_6))}}{\to \forall\alpha P(r_1(\alpha),r_2(\alpha)) \supset \exists\beta(P(s_3,r_4(\beta)) \wedge P(r_5(\beta),s_6))}
\end{array}
$$

[2] The length of a matrix Σ is defined as the number of (term-free) sequents in Σ.

Let s denote the successor function. We consider the matrix Σ given in Table 1 together with the endformula $A(f,a)$ $\forall x P(x, f(x)) \supset \exists z P(0,z) \wedge P(z,a)$, where f denotes an unary function variable and a is free variable. (To simplify the presentation of Σ, certain (mandatory) unification-steps have already be applied. Furthermore we write $r(\alpha)$ to indicate a variable r that can only be instantiated with a semi-term containing the bound variable α.)

If the basic language $\mathcal{L}$ contains at most unary function symbols, then the dependencies between different formula abstractions can be represented by a system of linear Diophantine equations. With respect to our example the obtained system of linear Diophantine equation reduce (by transitive closure and extensionality) to the equations: $f = \alpha$ and $a = f + \alpha$. Thus, the endformula becomes derivable by a proof with matrix Σ iff $f(x)$ becomes $s^n(x)$ and a is instantiated by $s^{2n}(0)$. We say that Σ *represents* the set of even numbers.

Now we show that **LK** doesn't admit generalization of semi-terms of simple form (for some bound M), if the proof-matrix of the initial proof is to be kept fixed. Assume to the contrary that **LK** admits generalization of semi-terms of simple form (for some bound M). Assume further a proof Π (with matrix Σ) of an instance $A(t,t')$ such that the depth of the (semi-)terms t, t' is $\geq M$. By assumption there exists a semi-term s and a term s', such that $A(s,s')$ is provable with matrix Σ. In particular there exists a $h < M$, such that $s' = s^h(b)$, where b is fresh free variable. This contradicts the fact that Σ represents the set of even numbers.

Therefore, if we are interested in generalization of semi-terms of particularly simple form, then we have to alter the logical structure. Note that the example shows that the central result of [KP88] is not (directly) applicable if we admit generalizations of semi-terms.

Consider a formula A, and let W be the set consisting of the variables in A that are bound by strong quantifiers[3] together with the constants occurring in A; let V be a subset of W. We frequently take the liberty to abbreviate a tuple of terms $t_1, \ldots, t_n$ by writing $\overline{t}$. The following definitions are parameterized wrt. V.

Definition 2. *Assume A can be written as $A(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ denote all maximal terms and semi-terms in A with the proviso that the included semi-terms contain only bound variables from V. Then $A(a_1, \ldots, a_n)$ denotes an* abstraction (wrt. V) *of $A(t_1, \ldots, t_n)$. (The variables $a_1, \ldots, a_n$ denote free variables.)*

Definition 3. *A* binding assignment (wrt. V) *δ is a function from the set of variables $\mathcal{V}$ into the power-set of V, i.e. $\delta\colon \mathcal{V} \to 2^V$.*

We extend the assignment δ to (semi-)terms: If t is a constant, then $\delta(t) = \{c\}$ for an arbitrary constant $c \in W$. Now let t be a (semi-)term, then $\delta(t) = \bigcup_{x \in \mathrm{var}(t)} \delta(x)$.

[3] Let $\mathsf{Q}x\ B(x)$, $\mathsf{Q} \in \{\forall, \exists\}$ be a subformula of A. If $\mathsf{Q}x\ B(x)$ occurs in the scope of an even (uneven) number of negation signs in A, then the occurrence of Q is called *strong* (*weak*) if $\mathsf{Q} \equiv \forall$ and *weak* (*strong*), otherwise.

Let $A(t_1, \ldots, t_n)$ and its abstraction $A(a_1, \ldots, a_n)$ be defined as above. Let δ denote a binding assignment.

Definition 4. *Let $A(s_1, \ldots, s_n)$ be an instance of the abstraction $A(a_1, \ldots, a_n)$. An* extension $A'(\overline{s})$ of $A(\overline{s})$ (wrt. V) *is obtained by replacing each occurrence of a strong quantifier $\mathsf{Q}y$ in $A(\overline{a})$ by $\mathsf{Q}y, \overline{z}$, if $y \in \delta(a_j)$ s.t. $\overline{z}$ is a subset of the bound variables in s_j.*

Let $\delta(a_j) = \{y_1, \ldots, y_k\}$. Assume $\overline{z}_1, \ldots, \overline{z}_k$ denote the chosen subsets of bound variables, respectively. Then we demand that the union of these subsets equals the set of bound variables in s_j.

Let Π a given cut-free proof. It simplifies the presentation if we fix the denotation of its end-sequent S. W.l.o.g. we assume S is closed and has the form

$$\rightarrow \exists \overline{x}_1 \forall \overline{y}_1 \cdots \exists \overline{x}_m \forall \overline{y}_m A(\overline{x}_1, \ldots, \overline{x}_m, \overline{y}_1, \ldots, \overline{y}_m)$$

where $\overline{x}_i, \overline{y}_j$ denote tuples of bound variables and A is quantifier-free. We can restrict our attention to the case where $\mathrm{card}(\overline{x}_i) = \mathrm{card}(\overline{y}_i) = 1$ for all i; this restriction does not imply a loss in generality, as the general case follows easily form the special one. We rewrite S

$$\rightarrow \exists x_1 \forall y_1 \cdots \exists x_m \forall y_m B(s_1(\overline{x}, \overline{y}), \ldots, s_k(\overline{x}, \overline{y}), t_1(\overline{y}), \ldots, t_p(\overline{y}), t_{p+1}, \ldots, t_{p+q})$$

such that B is quantifier-free and B does not contain any semi-terms not indicated above. The terms $t_1(\overline{y}), \ldots, t_p(\overline{y})$ do not contain other bound variables, than those indicated.

Let W be the set of variables bound by strong quantifiers and constants occurring in S; let V be a subset of W such that the variables occurring in V are exactly those that occur in the tuple $t_1, \ldots, t_p$. These variables will be called *distinguished* later on. If the set W contains constants, then V includes a constant c representing the constants occurring in W. The tuple of semi-terms $t_1, \ldots, t_p$ together with the term-tuple $t_{p+1}, \ldots, t_{p+q}$ are sometimes called *parameters*.

Using Definition 2 an abstraction $S(a_1, \ldots, a_p, a_{p+1}, \ldots, a_{p+q})$:

$$\rightarrow \exists x_1 \forall y_1 \cdots \exists x_m \forall y_m B(s_1(\overline{x}, \overline{y}), \ldots, s_k(\overline{x}, \overline{y}), a_1, \ldots, a_p, a_{p+1}, \ldots, a_{p+q})$$

of S is defined. (The a_i are sometimes called *abstraction* variables.) The end-sequent S naturally induces a specific binding assignment $\delta \colon \mathcal{V} \rightarrow 2^V$. Let V_i denote the distinguished variables in the parameter term t_i, $i = 1, \ldots, p$. Then set $\delta(a_i) = V_i$ for all i. Furthermore, if the tuple $t_{p+1}, \ldots, t_{p+q}$ is non-empty, then set $\delta(a_{p+i}) = \{c\}$, for all $i = 1, \ldots, q$. W.l.o.g. we assume that V can be written as $V \equiv \{y_{i_1}, \ldots, y_{i_r}, c\}$; $1 \leq i_1 < \cdots < i_r \leq m$.

Usually it is not necessary, to distinguish in our denotation between parameter-variables a_i that abstract semi-terms and variables a_{p+j} abstracting terms. If one the other hand a separation seems useful, we employ the binding assignment δ. Hence, we usually write $S(a_1, \ldots, a_n)$ as shorthand for the abstraction $S(a_1, \ldots, a_p, a_{p+1}, \ldots, a_{p+q})$.

We allow substitution to be applied to proofs. The set of free variables except the eigenvariables in Π is denoted as $\mathrm{var}(\Pi)$. Let Π be a proof, and σ be a substitution such that the domain of σ (denoted $\mathrm{dom}(\sigma)$) is a subset of $\mathrm{var}(\Pi)$. Then $\Pi\sigma$ denotes the proof obtained from Π by replacing every formula A in Π by $A\sigma$. (To make this definition independent of the choice of σ, we assume that $\Pi\sigma \equiv \Pi$, if $\mathrm{dom}(\sigma) \cap \mathrm{var}(\Pi) = \emptyset$.) Analogously the application of substitutions to proof-matrices is defined.

3 Preprocessing

Let Σ be the proof-matrix induced by the cut-free proof Π. Using the information coded in the endsequent, we will define an instantiation of the abstraction variables in Σ by (renaming of) semi-terms in the end-sequent. The obtained sequent-tree is called *instantiated* proof-matrix Σ'.

We start the construction of Σ' by setting Σ' equal to Σ and define instantiations of Σ' inductively: First assign $S(\overline{a})$ to the root of Σ'. If a node e in Σ' is not a leaf, then we assume inductively that terms or semi-terms have already been assigned to the variables in the sequent T labeling e. Consider a successor e' of e. Each side formula in T, defines term instances for the corresponding formula in the sequent T' that labels e'. Now we consider the principal formulas; we restrict our attention to the case where T follows from T' by a quantifier inference. The other cases are similar, but simpler.

(i) Assume that T follows by a *weak quantifier inference* from T'. Furthermore assume that the principal formula has the form $\exists x A(x)$ $(\forall x A(x))$ such that x occurs in a context of the form $s_i(\overline{x}, \overline{y})\rho$ $(i = 1, \ldots, k)$ where ρ is a variable renaming (ρ may rename free variables to bound variables). Let B be the auxiliary formula in T', then unify B with $A(\lambda)$; λ is a fresh abstraction variable. We set $\delta(\lambda) = \{c\}$.

(ii) Assume that T follows by a *strong quantifier inference* with principal formula $\forall y A(y)$, such that y occurs in the context $s_i(\overline{x}, \overline{y})\rho$, where ρ is a renaming. Let B be the auxiliary formula in T'. Unify B with $A(\lambda)$, where λ is some new abstraction variable λ. Set $\delta(\lambda) = \emptyset$.

(iii) Finally, assume T follows by a *strong quantifier inference* with principal formula $\forall y\ A(\lambda_1, \ldots, \lambda_m)$, where, $y \in \delta(\lambda_i)$ for all i. Let B be the auxiliary formula in T', then we unify B with $A(\mu_1, \ldots, \mu_m)$, where the μ_i are new abstraction variables. Moreover set $\delta(\mu_i) = \delta(\lambda_i) - \{y\}$.
Henceforth we refer to the positions of the variables λ_i (μ_i) in A (A') as *unsolved* positions. Furthermore we say the respective inference is *unsolved*.

This concludes the definition of the *instantiated proof-matrix* Σ'. □

Remark 1. In the given procedure a little bit of care is necessary, if we apply substitutions. Instantiations must not affect eigenvariables. This can be prevented by restricting substitutions to variables λ s.t. $\delta(\lambda) = \{c\}$.

Lemma 1. *Let Σ be a given cut-free proof-matrix with end-sequent $S(\overline{a})$. Assume there exists an instantiation $S(\overline{a})\rho$ which is provable with Σ (so that the binding function δ is respected). Then $S(\overline{a})\rho$ is provable with Σ' (so that the binding function δ is respected).*

4 Unification

Standard unification is not appropriate to find correct solutions for the unsolved positions in Σ'. In this section, we define *semi-term unification*, which will do the job nicely. Semi-term unification may be conceived as sorted unification with a specific (pseudo-linear) sort theory. The given unification procedure employs ideas from [Wei96].

We assume familiarity with the theory of standard unification, compare e.g. [BS01]. However, we will review some crucial notions. A *unification problem* U is either $\top$ or $\bot$ or a conjunction of equations $(s_1 = t_1 \wedge \cdots \wedge s_k = t_k)$.[4] A unification problem U is called *solved* if all s_i are pairwise distinct variables and $s_i \notin \mathrm{var}(t_j)$; for all i, j. If $U \equiv (x_1 = t_1 \wedge \cdots \wedge x_k = t_k)$ is in solved form, then $U^\star \equiv \sigma_1 \cdots \sigma_k$ ($\sigma_i \stackrel{\text{def}}{=} \{x_i \mapsto t_i\}$) is the unifier *induced* by U. A *weakening problem* is an unification problem of the form $x = t$ with $x \in \mathcal{V}; x \notin \mathrm{var}(t)$. Let σ, ρ be substitutions. If there exists a substitution ρ with $\tau \circ \rho = \sigma$, where $\circ$ denotes concatenation of substitutions, we say that τ is *more general* or an *extension* of σ.

Definition 5. *Let V be defined as above. Two terms s, t are* variants *if they can be transformed into each other by mappings of the form $\{\lambda_1 \mapsto \mu_1, \ldots, \lambda_n \mapsto \mu_n\}$, where $\delta(\lambda_i) = \emptyset$ and $\delta(\mu_i) = \{y\}$ and $y \in V$ for all i.*

Example 1. Assume $s \equiv h(a_1, \ldots, a_n)$ such that the a_i are fully indicated in s and $\delta(a_i) = \emptyset$ for all i. The term $t \equiv h(z_1, \ldots, z_n)$ is a variant of s if $\delta(z_i) = \{y\}$ for all i and $y \in V$.

Clearly the 'variant' relation is an equivalence relation.

Definition 6. *A* semi-term unification problem *is a triple $\Gamma \equiv \langle U, X, \delta\rangle$ where U denotes a standard unification problem, X is a partition of* $\mathrm{var}(U)$*; V is a set of of bound variables and $\delta\colon \mathcal{V} \to 2^V$ is a binding assignment.*

The problem $\langle U, X, \delta\rangle$ is solved by a substitution σ, called semi-term unifier*, if it is solved in the standard sense and σ in addition fulfills:* Let $C = \langle x_1, \ldots, x_n\rangle$ denotes a variables-class in X. Then $x_1\sigma, \ldots, x_n\sigma$ are variants and $\delta(x_i\sigma) \subseteq \delta(x_i)$ for all i.

The emphasized property of semi-term unification is sometimes called the *semi-term property*. We employ the usual rule-set for standard unification, extended by the rules *Partition* and *Weakening* as defined in Table 2 and Table 3.

[4] We often confuse the logical notation of a unification problem $(s_1 = t_1 \wedge \cdots \wedge s_k = t_k)$ and its multiset notation $\{s_1 = t_1, \ldots, s_k = t_k\}$.

Table 2. Partition

For both cases assume the picked equation is unmarked.
Pick an equation $x = s$, such that $s \equiv f(s_1, \ldots, s_m)$, f a function symbol and s is non-ground. $x = s \wedge U \longrightarrow x = s \wedge z = t \wedge x_{i_1} = s_{i_1} \wedge \cdots \wedge x_{i_n} = s_{i_n} \wedge U$ $X \oplus x \sim z \longrightarrow X \oplus x \sim z \oplus x_{i_1} \sim z_{i_1} \oplus \cdots \oplus x_{i_n} \sim z_{i_n}$ where $x \in \mathcal{V}$, $x \notin \mathrm{var}(s)$, the x_{i_j} $(1 \leq i_1 < \cdots < i_n \leq m)$ are fresh variables, and $t = f(s_1, \ldots, s_{i_1-1}, z_{i_1}, \ldots, z_{i_n}, s_{i_n+1}, \ldots, s_m)$ for fresh variables z_{i_j}. Mark the investigated equation.
Pick an equation $x = s$, such that for s, $\delta(s) \subseteq \delta(x)$ and $\delta(x) = \{y\} \subset V$. $x = s \wedge U \longrightarrow x = s \wedge z = t \wedge U$ $X \oplus x \sim z \longrightarrow X \oplus x \sim z$ where $x \in \mathcal{V}$ and s, t are variants. Mark the equation $x = s$.

As the partition X induces an (uniquely defined) equivalence relation $\sim$ it may be convenient to denote the partition X through the relation $\sim$. In the course of unification it may become necessary to extend the previous existing partition X; we write $X \oplus \langle x, y\rangle$ (or alternatively $X \oplus x \sim y$) to indicate the extension of X by the pair $\langle x, y\rangle$.[5] We set $\tau(\langle U, X, \delta\rangle) = \tau(U)$, where $\tau(U)$ denotes the maximal term-depth in U.

A related extension of standard unification, called *congruence unification* is presented in [BZ95]. Congruence unification can be conceived as standard unification plus the rule *Partition*, compare [BM01]. A congruence unification problem $\langle U, X\rangle$ is *solved* by an unifier σ, if σ is a standard unifier and σ fulfills the property: *If $x \sim y$, then $x\sigma, y\sigma$ are variants.* Congruence unification has similar properties as standard unification.

Theorem 1. *Let $\langle U, X\rangle$ be a congruence unification problem. Then there exists a finite set $\{\sigma_1, \ldots, \sigma_k\}$ of most general congruence unifiers of $\langle U, X\rangle$ iff $\langle U, X\rangle$ is solvable. Moreover for each i, $\tau(U\sigma_i) \leq \phi(\tau(U))$, where ϕ is an elementary function.*

Any unifier σ of a congruence unification problem can be represented in the form $(x_1 = t_1 \wedge \cdots \wedge x_k = t_k)$ s.t. all x_i are pairwise distinct variables and $x_i \notin \mathrm{var}(t_i)$ for all i. Moreover we can assume that σ meets the property *If $x \sim y$, then $x\sigma, y\sigma$ are variants.* An unification problem is in *congruence solved form* if these restriction are met. (Note that a congruence solved form is not necessarily a standard solved form.)

To deal properly with the binding function δ we change the usual definition of the unification rule *Application*, as follows.

$$x = t \wedge U \longrightarrow x = t \wedge U\{x \mapsto t\}$$

[5] If $\mathrm{var}(X) \cap \{x, y\} \neq \emptyset$, this extension will possibly change existing classes $C \in X$.

Table 3. Weakening

In all cases assume that the pair $\langle x, z\rangle \in X$ picked is either unmarked or the labels differs from $\langle \delta(x), \delta(z)\rangle$; furthermore assume $\delta(x) \neq \emptyset$.

$X \oplus x \sim z \longrightarrow X \oplus x \sim z$
Assume for the picked pair $x \sim z$, $\delta(x) = \{y\} \subset V$ and $\delta(z) = \emptyset$ holds. Mark the variable-pair $x \sim z$.

Assume f is a binary.
$U \longrightarrow x = f(x_1, x_2) \wedge z = f(z_1, z_2) \wedge U$
$X \oplus x \sim z \longrightarrow X \oplus x \sim z \oplus x_1 \sim z_1 \oplus x_2 \sim z_2$
Let V_1, V_2 (V_3, V_4) denote proper subsets of $\delta(x)$ ($\delta(z)$) such that $V_1 \cup V_2 = \delta(x)$ ($V_3 \cup V_4 = \delta(z)$). Set $\delta(x_1) = V_1$, $\delta(x_2) = V_2$ and $\delta(z_1) = V_3$, $\delta(z_2) = V_4$. Mark the variable-pair $\langle x_1, z_1\rangle$ ($\langle x_2, z_2\rangle$) with $\langle V_1, V_3\rangle$, ($\langle V_2, V_4\rangle$).

if $x \in \mathcal{V}$. In addition $x = t$ is marked as "unsolved" if $\delta(t) \not\subseteq \delta(x)$. Any marked equation is ignored in further unification steps, and a semi-term unification problem is only solved, if for all marked equations the corresponding constraints are fulfilled. (In particular an unsolved equation in the unification problem Γ cannot be used to define the (partial) solution $\Gamma^\star$ induced by Γ.)

The rule *Weakening*, see Table 3, is only applied if no other rule is applicable. In the definition of the rule we assume that the maximal arity of the function symbols in the basic language $\mathcal{L}$ is 2. It is easy to see how the definition is extended to the general case.[6]

Table 4. Deciding weakening problems

```
algorithm DecideECP(x_1 = t_1 ∧ ··· ∧ x_n = t_n, X, δ)
  begin
      U := {x_1 = t_1, ..., x_n = t_n}
      while U is not solved do
          Pick a variable pair ⟨x, z⟩ from X.
          Apply Weakening on with ⟨x, z⟩ respect to X and δ.
          Exhaustively apply unification steps to U, except Weakening.
          If U ≡ false then
              return false
          Remove all pairs x = s, s ground from U.
      end.
      return true
  end.
```

[6] Notice that in the intended application of semi-term unification, we can restrict our attention to unification problems, such that for each class $C \in X$ for at least one of the variables x, $\delta(x) \neq \emptyset$ holds.

Table 4 presents a non-deterministic algorithm which decides whether a conjunction of semi-term weakening problem has a solution. A solution σ is *minimal* if for any other solution λ of $x = t$, $\text{size}(t\sigma) \leq \text{size}(t\lambda)$. Let t be a term, assume the existence of two sub-terms t_1, t_2 of depth k, such that t_1 occur above t_2 (in the tree representation of t). If k is greater than 1, t_1, t_2 non-ground and $\delta(t_1) = \delta(t_2)$, then t is called *cyclic*. A solution of a weakening problem $x = t$ is *cyclic* if $t\sigma$ is.

We can transform `DecideECP` such that all possible weakening steps are enumerated; we obtain a finite representation of all minimal unifiers of semi-term weakening problems. The following lemma established a term bound on the solutions to semi-term weakening problems, obtained through `DecideECP`.

Lemma 2. *Let $\Gamma = \langle U, X, \delta \rangle$ be an semi-term unification problem, so that U is in (congruence) solved form. Let $\{\sigma_1, \ldots, \sigma_k\}$ denote the finite set of minimal semi-term unifiers of the unification problem $\langle U, X, \delta \rangle$. Then for each i, there exists a elementary function φ, such that $\tau(\Gamma\sigma_i) \leq \varphi(\tau(U), \text{card}(X), \text{card}(V))$.*

Combining Theorem 1 and Lemma 2 we conclude that semi-term unification for arbitrary term tuples remains decidable.

Theorem 2. *Let $U \equiv (s_1 = t_1 \wedge \cdots \wedge s_n = t_n)$ and $\Gamma = \langle U, X, \delta \rangle$ be a semi-term unification problem. Then there exists a finite set $\{\sigma_1, \ldots, \sigma_k\}$ of minimal semi-term unifiers of Γ iff Γ is solvable. Moreover for each i, there exists a elementary function ψ, such that $\tau(\Gamma\sigma_i) \leq \psi(\tau(U), \text{card}(X), \text{card}(V))$.*

Proof. First, we apply (altered) standard unification plus *Partition* rules to Γ. The obtained unification problem $\Gamma' \equiv \langle U', X', \delta \rangle$ is in congruence solved form. Due to Theorem 1 there exists an elementary function ϕ, such that

$$\tau(\Gamma') \leq \phi(\tau(U))$$

Second, we apply the procedure `DecideECP` to Γ'. The obtained unification problem $\Gamma'' \equiv \langle U'', X'', \delta' \rangle$ induces a minimal semi-term unifier. By Lemma 2 there exists an elementary function φ, such that

$$\tau(\Gamma'') \leq \varphi(\tau(U'), \text{card}(X'), \text{card}(V))$$

By definition $\tau(U') \leq \phi(\tau(U))$. (Note that $\tau(\Gamma') = \tau(U')$.) In the transformation of U into congruence solved form new equivalence classes are added, hence $\text{card}(X') \geq \text{card}(X)$. However, there exist only finitely many terms (up-to renaming) with fixed term-depth. Clearly there exists an (elementary) function $\phi'(d)$ that bounds the maximal number of terms in $\mathcal{L}$ with depth d. (Apart from d, ϕ' depends on the underlying signature $\mathcal{L}$.) From ϕ' we easily obtain a function ϕ'', that bounds the number of variables in Γ'; ϕ'' elementary. Per definition X' is a partition of variables in U', hence we have found an (elementary) bound of $\text{card}(X')$ depending only on $\tau(U')$ (and $\mathcal{L}$). In summary we obtain, $\tau(\Gamma'') \leq \varphi(\phi(\tau(U)), \phi''(\phi(\tau(U))), \text{card}(V))$. $\square$

5 The Final Touch

In this section we define a specific semi-term unification problem Γ. The finite set of minimal solutions of Γ is employed to define suitable instantiations of the unsolved positions in Σ'.

Let Σ' denote the instantiated proof-matrix; let δ denote the fixed binding function. We set $\Gamma = \langle U, X, \delta \rangle$. The set of equations U is defined by induction on the number of initial sequents in Σ'. For each initial sequent in Σ'

$$A(s_1, \ldots, s_n) \to A(t_1, \ldots, t_n)$$

we add the equations $s_i = t_i$ $(i = 1, \ldots, n)$ to the previously defined unification problem U.

To solve the yet uninstantiated unsolved positions in Σ', we introduce, by induction on the number of unsolved inference Q, equivalences between variables in U. Assume Q is of the following form.

$$\frac{\Gamma \to \Delta, A(\mu_1, \ldots, \mu_m)}{\Gamma \to \Delta, \forall y\ A(\lambda_1, \ldots, \lambda_m)} \tag{1}$$

such that $y \in \delta(\lambda_i)$ for all i. We add m equivalences to the previously defined partition X

$$\lambda_1 \sim \mu_1, \ldots, \lambda_m \sim \mu_m$$

This completes the definition of Γ. □

As the sequent S is provable (by the proof Π) the unification problem Γ is solvable. By Theorem 2 there exists a finite set of minimal solutions of Γ $\sigma_1, \sigma_2, \ldots, \sigma_k$. Let σ be an arbitrary minimal solution. We apply this solution to Σ'. The following lemma is an easy consequence of Theorem 2.

Lemma 3. *Assume σ is a minimal solution of Γ. Then σ uniquely defines an instance $S(t_1, \ldots, t_n)$ of the abstraction $S(\overline{a})$. This instance in turn uniquely defines an extension $S'(t_1, \ldots, t_n)$ such that $\tau(t_i) \leq \phi(|\Pi|, \mathrm{size}(S(\overline{a})))$, where ϕ is elementary.*

Remark 2. Notice that it is sufficient to consider minimal solutions. Any non-minimal solution of Γ will either be an instantiation of one of the solutions $\sigma_1, \ldots, \sigma_k$ or contain a cycle. However, with respect to cyclic solutions it is easy to see that any (non-minimal) cyclic solutions can be shortened by removing the cycle. Hence, we can always suppose that a given solution is cycle-free.

It remains to transform the sequent-tree $\Sigma'\sigma$ into a proof in the **LK**. Due to the definition of Γ if suffices to extend $\Sigma'\sigma$ at 'unsolved' quantifier introduction rules by additionally strong quantifier introduction rules to transform $\Sigma'\sigma$ into an **LK**-proof.

We extend $\Sigma'\sigma$—by induction on the number of unsolved inferences Q in Σ'—by additional quantifier inferences. Consider Q of the following form.

$$\frac{\Gamma\sigma \to \Delta\sigma, A(\mu_1, \ldots, \mu_m)\sigma}{\Gamma\sigma \to \Delta\sigma, \forall y\ A(\lambda_1, \ldots, \lambda_m)\sigma} \tag{2}$$

where $y \in \delta(\lambda_i)$ for all $i = 1, \ldots, m$. If we extend the 'variant' equivalence relation to formulas, we see that $A(\lambda_1, \ldots, \lambda_m)\sigma$ is a variant of $A(\mu_1, \ldots, \mu_m)\sigma$, i.e., there exists a renaming $\{a_1 \mapsto z_1, \ldots, a_n \mapsto z_n\}$ transforming the auxiliary formula into the principal formula of the inference, such that $\delta(a_i) = \emptyset$ and $\delta(z_i) = \{y\} \subset V$ for all i. Employing this substitution, we transform Q into a valid inference by replacing it with a sequence of n quantifier inference of the form

$$\frac{\Gamma\sigma \to \Delta\sigma, A(\mu_1, \ldots, \mu_m)\sigma}{\Gamma\sigma \to \Delta\sigma, \forall y \forall z_i\ A(\lambda_1, \ldots, \lambda_m)\sigma} \tag{3}$$

where $i = 1, \ldots, n$.

Lemma 4. *Let Π and S be defined as above. Then Π can be transformed into an* **LK***-proof Π' of an extension $S'(t_1, \ldots, t_n)$ of an instance of the abstraction of S.*

Proof. Let $\Sigma'\sigma$ be defined as above. The sequent-tree $\Sigma'\sigma$ is extended by additional quantifier inferences as described above, the obtained sequent-tree is called Ω.

It remains to verify that all the eigenvariable conditions are satisfied in Ω. Assume to the contrary the existence of a strong quantifier inference Q in Ω such that the eigenvariable a occurs in the lower sequent.

$$\frac{\Gamma \to \Delta, A(a)}{\Gamma \to \Delta, \forall z\ A(z)} \tag{4}$$

First recall the definition of the binding function δ: The given endsequent S induced a unique assignment of subsets of V to the abstraction variables in $S(\overline{a})$. During the generalization procedure the binding assignment δ is frequently extended, but it is impossible that previous values are changed.

W.l.o.g. we assume the existence of a sequent-formula $B(a)$ in Δ. Due to the construction the sequent-tree Ω is more general than the proof Π; i.e., there exists a substitution ρ instantiating (abstraction) variables by (semi-)terms in Π. In particular a must not occur in $B(a)\rho$, as otherwise Π would violate the eigenvariable condition itself. This implies that $\delta(a)$ either is the singleton $\{c\}$ or a subset of the distinguished variables.

Consider the occurrence of the eigenvariable a in $A(a)$. We distinguish two cases. First assume that Q is one of the newly introduced quantifier inferences. Then, by definition of the 'variant' relation, we have $\delta(a) = \emptyset$. Otherwise, we can assume that the inference Q was subject to the second case in the definition of the instantiated proof-matrix Σ'. Again we conclude that $\delta(a) = \emptyset$. In both cases we derive a contradiction. □

In summary we have shown the following theorem.

Theorem 3. *Any cut-free proof Π of $S(t_1, \ldots, t_n)$, where the t_i are either terms or semi-terms can be transformed into a proof Π' of $S'(r_1, \ldots, r_n)$ such that*

(i) there exists a substitution σ and $t_i = r_i\sigma$ for all $i = 1, \ldots, n$,

(ii) the proof-matrices of Π and Π' almost coincide: Single quantifier introductions in Π are replaced by sequences of quantifier introductions in Π',
(iii) $\tau(s_i) \leq \phi(|\Pi|, \mathrm{size}(S(\overline{a})))$, where the function ϕ is an elementary function.

Hence, the cut-free fragment of **LK** admits generalization of semi-terms of simple form, iff the logical form of the endsequent is altered. (The 'only if' is a consequence of the example in section 2.)

Remark 3. Notice that we have additionally proven that the terms and semi-terms in Π' are elementary bounded in the steps of Π and the size of S.

6 Parikh's Theorem

In this section we prove that the 'full' **LK** admits generalization of semi-terms of particularly simple form. To show this result, it suffices to demonstrate (i) how an arbitrary proof of $S(t_1, \ldots, t_n)$ can be transformed into a cut-free proof and that (ii) the length of the new proof is bounded in the length of the initial proof and the form of the abstraction $S(\overline{a})$.

Assume Π to be a proof of A, $|\Pi| = \mathrm{k}$. A result by Parikh [Par73] shows that the logical complexity of the formulas in Π can be bounded by an elementary function depending only on k and the logical complexity of S (denoted as $\mathrm{ld}(S)$). The idea of the proof is to use unification to eliminate redundant sub-formulas. For a modern presentation, see e.g. [Mos01].

Now assume an (arbitrary) proof Π of $S(t_1, \ldots, t_n)$ is given, whose length is k. Employing Parikh's Theorem there exists a proof Π' of S, with the same number of steps as Π so that the maximal logical depth of the formulas in Π' is bounded by an elementary function depending only on k and $\mathrm{ld}(S)$. As Π is arbitrary, its initial sequents need not be atomic. The same holds for the transformed proof Π'. It is easy to see how it is possible to replace non-atomic initial sequents by (short) derivations admitting atomic initial sequents only. Furthermore, the increase in length is bounded by an elementary function in the length of Π and the logical complexity of S. It remains to eliminate the cuts in Π'' using standard cut-elimination procedures, see e.g. [Bus98]. By the above argument the cut-degree of Π'' is elementarily bounded in the length of Π and $\mathrm{ld}(S)$. This implies that the length of the cut-free proof is bounded by a primitive recursive function in k and $\mathrm{ld}(S)$.

Theorem 4. *Any proof Π of $S(t_1, \ldots, t_n)$, where the t_i are either terms or semi-terms can be transformed into a proof Π' of $S'(r_1, \ldots, r_n)$ such that*
(i) there exists a substitution σ and $t_i = r_i\sigma$ for all $i = 1, \ldots, n$,
(ii) $\tau(r_i) \leq \phi(|\Pi|, \mathrm{size}(S(\overline{a})))$, where the function ϕ is a primitive recursive function.

7 Consequences

For the following, we assume that the formalizations under consideration are consistent and prove the usual axioms of equality. For a proof system **T**, we

write $\mathbf{T} \vdash A$ to denote the derivability of A in $\mathbf{T}$. We consider the following property, sometimes called Kreisel's conjecture.

$$\exists k \forall n\ \mathbf{T} \vdash^k A(s^n(0)) \text{ iff } \mathbf{T} \vdash \forall x A(x) \tag{5}$$

Example 2. If a formalization $\mathbf{T}$ of (a fragment of) arithmetic admits generalization of terms of simple form and proves $\forall x \exists y (x{=}0 \vee \cdots \vee x{=}s^{n-1}(0) \vee x{=}s^n(y))$ for all $n \in \mathbb{N}$ then for every formula $A(a)$ (5) holds.[7] We prove the following, somewhat more general form of (5)

$$\exists k \forall n_1 \cdots \forall n_r\ \mathbf{T} \vdash^k A(s^{n_1}(0), \ldots, s^{n_r}(0)) \text{ iff } \mathbf{T} \vdash \forall x_1 \cdots \forall x_r A(x_1, \ldots, x_r)\ .$$

Assume $\mathbf{T}$ admits generalization of simple form with bound M. Choose "sufficiently large" terms $s^{n_1}(0), \ \ldots, s^{n_m}(0)$. s.t. all n_i as well as $|n_i - n_j|$ (for $i \neq j$) are strictly greater than M. By assumption there exists a terms $r_1, \ldots, r_m$, s.t. $r_i = s^h(0)$ or $r_i = s^h(a_i)$ $(h < M)$ for free variables a_i and there exists a substitution σ such that $r_i\sigma = s^{n_1}(0)$, for $i = 1, \ldots, m$. Furthermore $\mathbf{T} \vdash A(r_1, \ldots, r_m)$. Employing $\forall n\ \mathbf{T} \vdash \forall x \exists y (x{=}0 \vee \cdots \vee x{=}s^{n-1}(0) \vee x{=}s^n(y))$ we obtain $\mathbf{T} \vdash \forall x_1 \cdots \forall x_m A(x_1, \ldots, x_m)$. □

We now derive consequences of the fact that large semi-terms can be generalized.

Example 3. If a formalization $\mathbf{T}$ admits generalization of semi-terms of simple form (with bound M) then for every formula $A(a, a_1, \ldots, a_r)$

$$\begin{gathered}\exists k\ \mathbf{T} \vdash^k \exists x \forall y A(x, s^{N_1}(y), \ldots, s^{N_r}(y)) \text{ implies} \\ \exists h\ \mathbf{T} \vdash \exists x \forall \overline{z} A(x, s^h(z_1), \ldots, s^h(z_r))\end{gathered}$$

where the N_i $(i = 1, \ldots, m)$ are "sufficiently large" wrt. M.

Consider the tuple $s^{N_1}(y), \ldots, s^{N_r}(y)$, as above we assume that all $N_i > M$ and $|N_i - N_j| > M$ $(i \neq j)$ There exists a tuple $s^{m_1}(z_1), \ldots, s^{m_r}(z_r)$, s.t. $m_i < M$ for all i. And there exists a substitution σ such that $s^{m_i}(z_i)\sigma = s^{n_i}(0)$, for $i = 1, \ldots, r$. We have $\mathbf{T} \vdash \exists x \forall z_1 \cdots \forall z_r A(x, s^{m_1}(z_1), \ldots, s^{m_r}(z_r))$. Finally set $h = \max\{m_1, \ldots, m_r\}$. □

Another application shows: if the existence of a bound beyond which a statement holds can be shown by a short proof then this bound can be made explicit within the formal system.

Example 4. If a formal system $\mathbf{T}$ admits generalization of semi-terms of simple form (with bound M) and proves $\forall x \exists y (x{<}y)$ as well as $\forall x (s^n(0){<}x \supset \exists y (x{=}s^{n+1}(y)))$ for all $n \in \mathbb{N}$ then for every formula $A(a)$

$$\exists k\ \mathbf{T} \vdash^k \exists x \forall y (x{<}y \supset A(s^N(y))) \quad \text{implies} \quad \exists h\ \mathbf{T} \vdash \forall y (s^h(0){<}y \supset A(y))$$

[7] As an example for $\mathbf{T}$ that admits generalization of simple form, consider the system $\mathrm{L}\exists_1$; i.e., a weak fragment of arithmetic extended with the schema of the least number principle for Σ_1-formulas, see [BP93, Pud98].

where N is chosen "sufficiently large" wrt. M.

We conclude as before that $\mathbf{T} \vdash \exists x \forall y \forall z (x{<}y \supset A(s^{h+1}(z)))$ for some h. This is logically equivalent to $\mathbf{T} \vdash \exists x \forall y (\neg(x{<}y) \vee \forall z A(s^{h+1}(z)))$. Now we use $\forall x \exists y (x{<}y)$ and $\forall x (s^h(0){<}x \supset \exists y (x{=}s^{h+1}(y)))$ to conclude $\mathbf{T} \vdash \forall y (s^h(0){<}y \supset A(y))$. □

8 Conclusion

The results of this paper indicate, that the notion of skeleton (and consequently the notion of length as measured by the number of steps) should not be considered as absolute proof invariants independent of the generalization problem under consideration. On the contrary, there is an intrinsic relation between the classes of proofs to be generalized and the notions of abstract proof structures needed for the calculation of most general proofs.

References

[BM01] M. Baaz and G. Moser. Herbrand's Theorem and Term Induction. Submitted to the Annals of Pure and Applied Logic, 2001. 389

[BP93] M. Baaz and P. Pudlák. Kreisel's conjecture for $L\exists_1$. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, pages 29–59. Oxford University, 1993. With a postscript by G. Kreisel. 395

[BS95] M. Baaz and G. Salzer. Semi-unification and geralization of a particularyly simply form. In L. Pacholski and J. Tiuryn, editors, *Proc.* 8th *Workshop CSL'94*, volume LNCS 933, pages 106–120. Springer Verlag, 1995. 382

[BS01] F. Baader and W. Snyder. Unification theory. In A. Voronkov, editor, *Handbook of Automated Reasoning*, volume I, pages 445–532. 2001. 388

[Bus98] S. R. Buss. An Introduction to Proof Theory. In S. R. Buss, editor, *Handbook of Proof Theory*, pages 1–79. Elsevier Science, 1998. 394

[BW01] Baaz and P. Wojtilak. Generalizing Proofs in Monadic languages. With a postscript by G. Kreisel. Submitted to the Annals of Pure and Applied Logic, 2001. 384

[BZ95] M. Baaz and R. Zach. Generalizing theorems in real closed fields. *Ann. of Pure and Applied Logics*, 75:3–23, 1995. 389

[CL73] C.-L. Chang and R. C. T. Lee. *Symbolic Logic an Mechanical Theorem Proving*. Academic Press, New York, 1973. 382

[Gen34] G. Gentzen. Untersuchungen über das logische Schließen I–II. *Math. Zeitschrift*, 39:176–210, 405–431, 1934. 384

[KP88] J. Krajíček and P. Pudlák. The number of proof lines and the size of proofs in first-order logic. *Arch. Math. Logic*, 27:69–84, 1988. 382, 385

[Mos01] G. Moser. *Term Induction*. PhD thesis, Vienna University of Technology, June 2001. 394

[Par73] R. J. Parikh. Some results on the length of proofs. *Trans. Amer. Math. Soc.*, pages 29–36, 1973. 394

[Pud98] P. Pudlak. The Lengths of Proofs. In S. Buss, editor, *Handbook of Proof Theory*, pages 547–639. Elsevier, 1998. 384, 395

[Wei96] C. Weidenbach. Unification in Pseudo-Linear Sort Theories is Decidable. In *13th International Conference on Automated Deduction, CADE-13*, LNCS. Springer, 1996. 388

Local Problems, Planar Local Problems and Linear Time

Régis Barbanchon and Etienne Grandjean

GREYC, Université de Caen
14032 Caen Cedex, France
{regis.barbanchon,etienne.grandjean}@info.unicaen.fr

Abstract. This paper aims at being a step in the precise classification of the many NP-complete problems which belong to NLIN (nondeterministic linear time complexity on random-access machines), but are seemingly not NLIN-complete. We define the complexity class LIN-LOCAL – the class of problems linearly reducible to problems defined by Boolean local constraints – as well as its planar restriction LIN-PLAN-LOCAL. We show that both "local" classes are rather computationally robust and that SAT and PLAN-SAT are complete in classes LIN-LOCAL and LIN-PLAN-LOCAL, respectively. We prove that some unexpected problems that involve some seemingly global constraints are complete for those classes. E.g., VERTEX-COVER and many similar problems involving cardinality constraints are LIN-LOCAL-complete. Our most striking result is that PLAN-HAMILTON – the planar version of the Hamiltonian problem – is LIN-PLAN-LOCAL and even is LIN-PLAN-LOCAL-complete. Further, since our linear-time reductions also turn out to be parsimonious, they yield new DP-completeness results for UNIQUE-PLAN-HAMILTON and UNIQUE-PLAN-VERTEX-COVER.

1 Introduction and Discussion

Since the publication of the famous Cook-Levin's theorem, two fundamental and complementary questions arise about time complexity: 1) What are the connections between deterministic time and nondeterministic time? 2) What is the precise complexity of usual NP-complete problems? It seems that any progress in proving complexity lower bounds for concrete NP-complete problems is conditioned by progress in both questions.

An interesting result concerning Question 1 is the separation result $\mathrm{DTIME}(n) \neq \mathrm{NTIME}(n)$ by Paul et al. [22] for linear time on Turing machines (TMs). However, its significance is weakened by the lack of any similar result known for other general-purpose computation models such as Random Access Machines (RAMs) and by the widespread feeling that linear time complexity on deterministic TMs is too restrictive.

Concerning Question 2, the second author defined and investigated (in a series of papers [12, 13, 15]), the classes DLIN and NLIN of problems which are (deterministically, resp. nondeterministically) decided in linear time on a certain type of RAM. It was argued [13, 15] that DLIN is robust and captures

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 397–411, 2002.
© Springer-Verlag Berlin Heidelberg 2002

the notion of linear time as used in algorithmic design. At least as importantly, the class NLIN contains most of the natural NP-complete problems, including the 21 problems of [17], as asserted in [12, 13], and a few of them, e.g., RISA (Reduction of Incompletely Specified Automata [9, 11]), are also NLIN-complete under DTIME(n)-reductions. This implies:

$$\text{DLIN} \neq \text{NLIN iff RISA} \notin \text{DLIN, and}$$
$$\text{RISA} \notin \text{DTIME}(n), \text{since DTIME}(n) \subsetneq \text{NTIME}(n) \subseteq \text{NLIN}.$$

In contrast, as argued in [13], it is unlikely that SAT is NLIN-complete because it can be solved on a RAM by the following algorithm (so called NSUBLIN algorithm) that performs $O(n)$ deterministic steps *and only* $O(n/\log n)$ *nondeterministic steps*:

Input: A propositional formula F of m variables $p_0, \cdots, p_{m-1}$.
(N) Guess an assignment $I \in \{0,1\}^m$ for $p_0, \cdots, p_{m-1}$.
(D) Check that $I \models F$.

Note that $n = length(F) \geq \sum_{i<m} length(p_i) = \Omega(m \log m)$ yields complexity $m = O(n/\log n)$ for Phase (N), and that Phase (D) is performed in deterministic linear time. More generally, many classical NP-complete problems, including HAMILTONIAN-CYCLE, VERTEX-COVER, 3COL, etc., have similar NSUBLIN algorithms. Further, the planar versions of those problems, PLAN-SAT, PLAN-VERTEX-COVER, etc., seem still easier NP-complete problems since a divide-and-conquer strategy based on a planar separator theorem [20] can be applied to solve them in deterministic *sub-exponential* time $2^{O(n^{1/2})}$.

In an effort to investigate the conjecture DLIN $\neq$ NLIN (seemingly weaker than P $\neq$ NP), the present paper aims at being a step in the precise classification of the many problems which lie "somewhere below" NLIN. First, it is striking to observe that a number of problems are linearly equivalent to SAT (e.g., 3COL, 3DM, KERNEL), i.e., are linearly reducible to SAT and conversely, as observed by several authors [4, 6, 13, 24]. Second, logic plays a fundamental role: As Fagin proved that Existential Second-Order logic (ESO) on finite first-order structures exactly characterizes NP [7], Grandjean et al. [14] proved that, on *unary functional* first-order structures (i.e., finite structures over a signature that consists of relation and function symbols of arity ≤ 1), NLIN is exactly characterized by the logic ESO(1), that is the set of sentences of the form: $\exists \bar{f}\ \forall x\ \varphi$, where $\bar{f}$ is a list of relations and function symbols of arity ≤ 1 and φ is a quantifier-free formula. Since we conjecture that SAT and the NSUBLIN problems are not NLIN-complete, we look for a sub-logic of ESO(1) that can express them. A natural candidate is the set of sentences of the form:

$$\exists \bar{U}\ \forall x\ \varphi\,, \tag{1}$$

where $\bar{U}$ is a list of *unary relation* symbols (i.e., set symbols) and φ is a quantifier-free formula. Lautemann and Weinzinger [18] investigated such a logic they denoted Monadic-NLIN and proved that it expresses a number of natural NP-complete problems including SAT, 3COL and KERNEL, on some kind of ordered

functional structures. Formulas (1) are special Monadic-ESO formulas, and unfortunately, it is well-known that this logic can only express "local" properties. In contrast, some easily computable (DLIN) properties such as graph connectivity cannot be defined in Monadic-ESO even in the presence of a built-in linear order [5, 8, 23]. Lautemann and Weinzinger [18] proved similar non-expressibility results for their logically defined class Monadic-NLIN. So we feel that the set of problems definable by Formulas (1) cannot be regarded as a complexity class. We think that any robust sequential time complexity class has to be closed under DLIN-reductions, because, on the one hand the sorting problem belongs to DLIN as proved in [13], and on the other hand we are convinced that DLIN is the minimal robust class for sequential time. This justifies the following definition of the complexity class LIN-LOCAL: A *local problem* is a set of unary functional first-order structures satisfying a sentence of the form (1) that we call a *local sentence*. A decision problem is LIN-LOCAL if it is DLIN-reducible to some local problem. The main feature of the class LIN-LOCAL is its *minimal* way to use nondeterminism by restricting it to happen in parallel at the end of any linear algorithm with an amount of $O(1)$ bits used per element.

A discussion of the notion of locality is required at this point. Any condition $(S \models \exists \bar{U}\ \forall x\ \varphi)$ is checked locally by consulting, for each element a of the structure S, the "colors" of a and of its "neighbors", i.e., the truth values of the monadic predicates of $a, f_0(a), \cdots, f_{k-1}(a)$, where $f_0, \cdots, f_{k-1}$ are the unary functions of S.[1]

Note that the "locality" results from the interaction of the local sentence with the underlying digraph $G(S) = (V, E)$ associated to S, that is defined by $V = Domain(S)$ and $E = \{(x, y) \in V^2,\ \exists f_i\ f_i(x) = y\}$. This graph is out-degree-bounded. It is natural to try to strengthen locality by adding one or both of the following semantical conditions over $G(S)$:

(B) Degree-Boundedness: $G(S)$ is *degree-bounded*. This can be obtained by requiring that each f_i be *bijective*.
(P) Planarity: $G(S)$ is *planar*.[2]

Regarding condition (P), we investigate a new complexity class, denoted LIN-PLAN-LOCAL, which is the class of decision problems DLIN-reducible to some planar local problem, i.e., a local problem over structures S whose underlying digraphs $G(S)$ are planar.

Let us now describe the main contributions of this paper. First, we justify the robustness of our complexity classes LIN-LOCAL and LIN-PLAN-LOCAL. Both are not modified if condition (B) is required together with several syntactical restrictions on φ, such as the use of at most two functions and at most one ESO monadic predicate. That strengthens the significance of the following series of

[1] Note that looking only at the immediate neighbors of a is possible because, as far as LIN-LOCAL problems are concerned, we can always assume w.l.g. that no functional composition occurs in φ.

[2] We will confuse a planar graph with one of its possible embeddings. This is justified by the fact that such a planar embedding is DLIN-computable [21].

inclusions of "linear classes" all conjectured to be strict:

$$\text{DLIN} \subseteq \text{LIN-PLAN-LOCAL} \subseteq \text{LIN-LOCAL} \subseteq \text{NLIN}\,.$$

One possible argument is that it would be a breakthrough if any of the following known inclusions in Turing Machine deterministic time classes could be improved:

$$\begin{aligned} \text{LIN-PLAN-LOCAL} &\subseteq \text{DTIME}(2^{O(n^{1/2})})\,, \\ \text{LIN-LOCAL} &\subseteq \text{DTIME}(2^{O(n/\log n)})\,, \\ \text{NLIN} &\subseteq \text{DTIME}(2^{O(n)})\,. \end{aligned}$$

Our second series of contributions are the proofs that many (planar) NP-complete problems are LIN-LOCAL-complete (resp. LIN-PLAN-LOCAL-complete). It is easily proved that SAT (resp. PLAN-SAT) is LIN-LOCAL-complete (resp. LIN-PLAN-LOCAL-complete). In other words, LIN-LOCAL (resp. LIN-PLAN-LOCAL) is exactly the set of problems DLIN-reducible to SAT (resp. PLAN-SAT). As a consequence, the numerous problems linearly equivalent to SAT (3COL, 3DM, KERNEL, etc., see [4] for a survey) are also LIN-LOCAL-complete, and we can prove that many of their planar restrictions are similarly LIN-PLAN-LOCAL-complete. The most surprising contributions of this paper are about some usual problems mixing local conditions with seemingly *global* (i.e., non-local) conditions:

- *cardinality conditions in the non-planar case*: problems such as VERTEX-COVER; All the cardinality problems are LIN-LOCAL and most of the usual NP-complete cardinality problems (e.g., VERTEX-COVER, DOMINATING-SET, MAX-SAT) are also LIN-LOCAL-complete.
- *connectivity conditions in the planar case*: the typical example is HAMILTON.[3] All the many variants of the planar HAMILTON problem are LIN-PLAN-LOCAL-complete. In particular, they are LIN-LOCAL, and hence all of them have a $O(2^{O(n^{1/2})})$ deterministic algorithm based on [20].

By lack of space, some of these results are presented in technical reports [1, 2, 3]. The "locality" of PLAN-HAMILTON contrasts with the conjecture that the general HAMILTON problem *is not* LIN-LOCAL. In that direction, Lautemann and Weinzinger [18] proved that HAMILTON does not belong to their class Monadic-NLIN, which means this problem *is not local* even in the presence of some kind of linear order.

Finally, we observe that all our reductions are not only DLIN-computable but also *parsimonious*. More precisely they state a bijective DLIN-computable correspondence between the solutions of the involved problems. As a side effect, this yields some new results about the status of some planar problems in

[3] The generic term HAMILTON refers to any of the many variants of the HAMILTONIAN-GRAPH problem: The input graph may be directed or not, and degree-bounded or not. We may test the existence of either a Hamiltonian cycle or a Hamiltonian path. In the latter case, the ends of the path may be fixed or free.

DP.[4] More importantly, the fact that our linear reductions can be made parsimonious strengthens our feeling that the LIN-LOCAL-complete problems (SAT, VERTEX-COVER, KERNEL, etc.) are very closely related to each other,

The paper is organized as follows: In Sect. 2, we define the classes LIN-LOCAL and LIN-PLAN-LOCAL and show their robustness. The LIN-LOCALITY (resp. LIN-PLAN-LOCALITY) of SAT (resp. PLAN-SAT) is also proved in this section. Section 3 is devoted to the LIN-LOCALITY of cardinality problems and Sect. 4 shows the LIN-PLAN-LOCALITY of PLAN-HAMILTON.

2 LIN-LOCAL Problems and SAT

In this section, we define precisely local structures, local sentences and our complexity classes LIN-LOCAL and LIN-PLAN-LOCAL. Moreover, we prove that those classes are rather robust under several changes of their definitions and that SAT and PLAN-SAT are respectively complete for them.

Definition 1 (Unary Structures). *A unary structure $S = (\mathcal{U}, \sigma)$ is a first-order structure over a finite universe $\mathcal{U}$ and a signature $\sigma = (\mathcal{F}, \mathcal{L})$, where $\mathcal{L}$ is a list of unary relations $L_0, \cdots, L_{p-1}$ (the* labelling *predicates), and $\mathcal{F}$ is a list of unary functions $f_0, \cdots, f_{k-1}$ (the* neighborhood *functions).*[5]

Example 1. The set of undirected graphs (without isolated vertex) can be represented, e.g., by a set $\mathcal{S}_G$ of unary structures $(\mathcal{U}, \sigma_G)$ with $\sigma_G = (\mathcal{F}_G, \mathcal{L}_G)$, $\mathcal{F}_G = (next, edge)$ and $\mathcal{L}_G = \emptyset$, as in Fig. 1. A graph $G(V, E)$ corresponds to a universe $\mathcal{U}$ of $2|E|$ elements, where each vertex $x \in V$ of degree d is represented by d elements $x_1, \cdots, x_d \in \mathcal{U}$ linked in a circular list via the function *next*, and where each edge (x, y) is represented by a circular list of length two linking two elements x_i and y_j via the function *edge*.

Definition 2 (Underlying Graph). *The underlying digraph of a unary structure $S = (\mathcal{U}, \sigma)$ with $\sigma = (\mathcal{F}, \mathcal{L})$ is defined by $G(S) = (V, E)$, where $V = \mathcal{U}$ and $E = \{(x, y) \in V^2,\ \exists f_i \in \mathcal{F}\ f_i(x) = y\}$. We say that S is planar if G is planar, i.e., has a planar embedding.*

Definition 3 (Local Problem, Description). *A* local problem *Π over a set $\mathcal{S}$ of unary σ-structures is the subset of $\mathcal{S}$ defined by $S \in \Pi$ iff $S \models \exists \mathcal{C}\ \forall x\ \varphi$ where $\mathcal{C}$ is a list of unary relation symbols $C_0, \cdots, C_{q-1}$ (the* coloring *predicates), and φ is a quantifier-free one-variable $(\sigma, \mathcal{C})$-formula. The tuple $(\mathcal{S}, \sigma, \mathcal{C}, \varphi)$ is called the description of the local problem and is identified to problem Π.*

Definition 4 (Planar Local Problem). *A* planar local problem *$\Pi = (\mathcal{S}, \sigma, \mathcal{C}, \varphi)$ is a local problem over a set $\mathcal{S}$ of planar structures.*

[4] A problem is in DP if it is defined by the conjunction of two conditions, one in NP, and the other one in co-NP.

[5] As usual, we confuse each relation or function symbol and its interpretation. Also, for convenience, we shall often view monadic predicates as functions to $\{0, 1\}$.

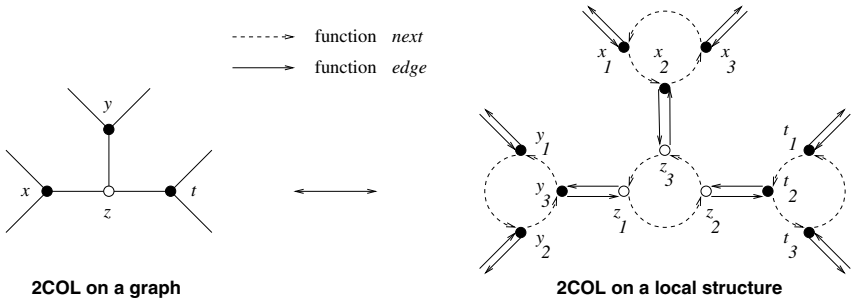


Fig. 1. 2COL on graphs and its translation on unary structures

Example 2. The problem $\Pi_{2col} = (\mathcal{S}_G, \sigma_G, \mathcal{C}_{2col}, \varphi_{2col})$, where $\mathcal{C}_{2col} = (Black)$ and φ_{2col} is $[Black(x) \Longrightarrow Black(next(x))] \wedge [Black(x) \Longrightarrow \neg Black(edge(x))]$, is the set of σ_G-structures associated to the graph problem 2COL. The problem $\Pi_{3col} = (\mathcal{S}_G, \sigma_G, \mathcal{C}_{3col}, \varphi_{3col})$ representing the graph problem 3COL can be similarly defined with $\mathcal{C}_{3col} = (Red, Green, Blue)$.

Definition 5 (Bijective Description, Minimal Description). *Let Π be a local problem of description $(\mathcal{S}, \mathcal{C}, \sigma, \varphi)$ with $\sigma = (\mathcal{F}, \mathcal{L})$. Π is a* bijective description *if $\mathcal{S}$ only uses bijective functions. Π is a* minimal description *if the equality is not used and no functional composition occurs in φ (i.e., φ is syntactically restricted to express conditions over the predicates f on x and its immediate neighborhood) and φ uses a minimal number of symbols: More precisely, at most one coloring predicate C_0, at most one labelling predicate L_0, and at most two neighborhood functions f_0 and f_1.*

Example 3. The local problem $(\mathcal{S}_G, \sigma_G, \mathcal{C}_{2col}, \varphi_{2col})$ is a minimal bijective description. The local problem $(\mathcal{S}_G, \sigma_G, \mathcal{C}_{3col}, \varphi_{3col})$ is a bijective but not minimal description, since it uses three coloring predicate symbols.

As previously argued, local problems cannot represent any consistent time complexity class if they are not closed under DLIN reductions. This justifies the following definition:

Definition 6 (LIN-LOCAL Class). *A decision problem Π is* LIN-LOCAL *if it is DLIN-reducible to a local problem Π'. Similarly, Π is* LIN-PLAN-LOCAL *if it is DLIN-reducible to a planar local problem Π'. For convenience, one says that any description of Π' is also a description of Π.*

It is easy to prove that (PLAN-)SAT is LIN-(PLAN)-LOCAL even with a bijective (but non-minimal) description [3]. It is trickier to prove the stronger theorem:

Theorem 1. – *SAT is LIN-LOCAL and has a minimal bijective description.*
– *PLAN-SAT is LIN-PLAN-LOCAL and has a minimal bijective description.*

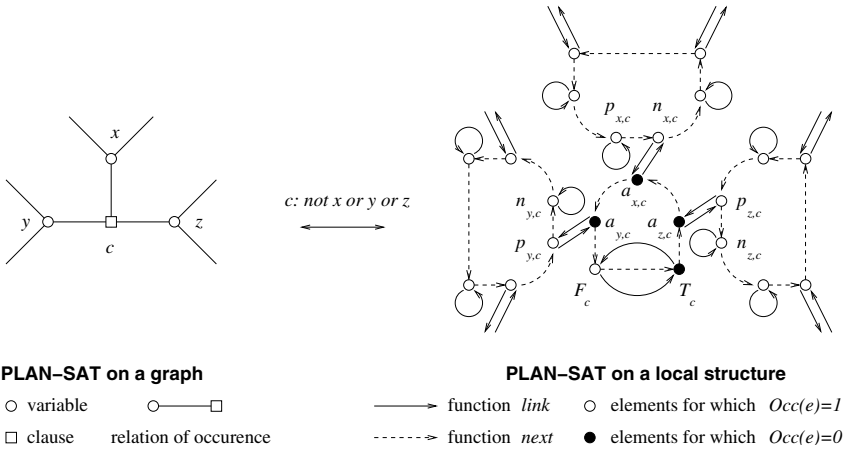


Fig. 2. Minimal bijective description for (PLAN-)SAT

Proof. We represent any (PLAN-)SAT instance by a (planar) $(\mathcal{F}, \mathcal{L})$-structure S where $\mathcal{F} = (next, link)$, $\mathcal{L} = (Occ)$, and $next$, $link$ are bijections.

- $\mathcal{U} = Domain(S)$ contains: for each clause c, the two elements T_c and F_c (the true and false constants), and for each occurrence of a variable v in a clause c , an element $a_{v,c}$ (the accumulators of the truth values in c) and two elements $p_{v,c}$ and $n_{v,c}$ (meant to represent v and $\neg v$ in c).
- The predicate Occ is mainly the label for occurrences: It maps all the $p_{v,c}$ and $n_{v,c}$ to 1 (true), and maps all the $a_{v,c}$ to 0 (false). A first trick is that it also maps all the F_c to 1 and all the T_c to 0.
- For each variable v, the function $next$ binds all the $p_{v,c}$ and $n_{v,c}$ in an alternating directed cycle. For each clause c, it also binds T_c, all the accumulators $a_{v,c}$ and F_c in this order in a directed cycle.
- The function $link$ mainly binds occurrences to accumulators: If a variable v occurs positively in a clause c then we define $link(p_{v,c}) = a_{v,c}$, $link(a_{v,c}) = p_{v,c}$, and the self-loop $link(n_{v,c}) = n_{v,c}$ (the symmetric case happens if v occurs negatively in c). The second trick is that for each clause c, we define the 2-cycle $link(T_c) = F_c$, $link(F_c) = T_c$.

The construction is clearly DLIN-computable and can be made planarity-preserving as shown in Fig. 2. The local formula uses *only one* color $True$ which holds the truth values of all the $p_{v,c}$ and $n_{v,c}$. For all the T_c (resp. F_c) it will be shown to be 1 (resp. 0), and for any accumulator $a_{v,c}$ it will hold the accumulated truth-values of the occurrences linked to all its successors via function $next$ up to F_c. The local sentence is $\exists\, True\ \forall x$:

$$\begin{aligned} &[Occ(x) \Longrightarrow (True(next(x)) \oplus True(x))] \\ \wedge\, &[\neg Occ(x) \Longrightarrow (True(x) \iff (True(next(x)) \vee True(link(x))))]\,. \end{aligned}$$

The first constraint coerces all the $n_{v,c}$ and $p_{v,c}$ of a variable v to have opposite values. Also, since $Occ(F_c) = 1$ and $next(F_c) = T_c$ for any clause c, it forces

that F_c and T_c have opposite values. The second constraint implies that, for each clause c, the value of the predicate $True$ is non-increasing along the arrows *next* from T_c to F_c (including T_c because $Occ(T_c) = 0$). Since $True(F_c) \neq True(T_c)$ because of the first constraint, this implies that $True(F_c) = 0$ and $True(T_c) = 1$. This also means that $True(T_c)$, which accumulates the truth-value of the final occurrence and the truth-value of F_c, indeed holds a copy of the truth-bit of the final accumulator. It follows that there is at least one $a_{v,c}$ such that $True(a_{v,c}) = 1$, i.e., such that the truth value of v (represented by $True(n_{v,c})$ and $True(p_{v,c})$) satisfies c. □

Theorem 2. – *SAT is LIN-LOCAL-complete.*
– *PLAN-SAT is LIN-PLAN-LOCAL-complete.*

Theorems 1 and 2 imply that:

Corollary 1. *Each LIN-LOCAL or PLAN-LIN-LOCAL problem has a minimal bijective description.*

The LIN-LOCAL-hardness of SAT is obtained by a straightforward unfolding of the universal quantifier $\forall x$ over the universe of the unary structure and is left to the reader. The proof of the LIN-PLAN-LOCAL-hardness of PLAN-SAT is technically more involved because the planarity of structures must be preserved, despite of the possible compositions occurring in φ. It needs the following lemma, whose proof is presented in [3].

Lemma 1. *Any local sentence $\exists \mathcal{C}\ \forall x\ \varphi$ is logically equivalent to another local sentence $\exists \mathcal{C}'\ \forall x\ \varphi'$ in CNF which is composition-free, i.e., such that no functional composition occurs in φ'.*

Proof (of Theorem 2, sketch). Assume now that φ verifies Lemma 1, the reduction to PLAN-SAT of a planar local problem $\Pi = (\mathcal{S}, \sigma, \mathcal{C}, \varphi)$ over a structure $S = (\mathcal{U}, \sigma)$ consists in building a planarity-preserving SAT-gadget of size $O(d(x))$ to simulate φ around each element $a \in \mathcal{U}$ of degree $d(a) = d^-(a) + d^+(a)$. All the gadgets are then connected following the embedding of $G(S)$. In [3] we present a uniform way to build such a gadget using Lichtenstein's planar crossover-box for PLAN-SAT [19, 16]. □

3 LIN-LOCAL Problems and Cardinality Problems

In this section, we show that augmenting the local constraints by constraints over the cardinalities of the unary relations does not change the class LIN-LOCAL in the general case. This does not seem to hold in the plane.

Definition 7 (Cardinality Problem). *Define $\#C$ to be the cardinality of a unary relation C. A cardinality constraint is a constraint of the form $(\#C_i \perp K)$ or of the form $(\#C_i \perp \#C_j)$ where C_i and C_j are unary relations symbols, K is a constant, and $\perp$ is a comparison relation among $=, \leq$. A cardinality*

problem is a problem characterized by both local constraints and cardinality constraints, i.e., by some sentence of the extended form $\exists\mathcal{C}\ (\forall x\ \varphi_1) \wedge \varphi_2$ *where* φ_1 *is a quantifier-free formula over* x, σ *and* $\mathcal{C}$, *and* φ_2 *is some Boolean combination of cardinality constraints.*

Example 4. A large number of natural NP-complete problems such as VERTEX-COVER, DOMINATING-SET, MAX-SAT, etc. [9] can be viewed as cardinality problems. E.g., the vertex-cover of a graph with less than K vertices can be converted into a cardinality problem $\Pi_{vc} = (\mathcal{S}_G, \sigma_{vc}, \mathcal{C}_{vc}, \varphi_{vc})$, where $\mathcal{C}_{vc} = (Cover, Count)$, and σ_{vc} is σ_G augmented by one monadic predicate *Repr* which identifies exactly one element per cycle of the function *next* (recall from Example 1 that such a cycle represents one vertex of the original graph). Clearly, Π_{vc} is defined by $\exists Cover, Count\ (\#Count \leq K) \wedge \forall x$:

$$\begin{aligned} & [Cover(x) \vee Cover(edge(x))] \\ \wedge\ & [Cover(x) \iff Cover(next(x))] \\ \wedge\ & [Count(x) \iff (Cover(x) \wedge Repr(x))]\,. \end{aligned}$$

We give a uniform argument that shows that each cardinality constraint is linearly SAT-expressible. The construction essentially uses a linear-sized SAT-adder that computes the correct cardinalities. As a consequence:

Theorem 3. *All the cardinality problems are LIN-LOCAL.*

Proof (sketch). Let C be a monadic predicate over a universe $\mathcal{U} = \{e_0, \cdots, e_{n-1}\}$. The main problem consists in building a SAT-gadget of size $O(n)$ which outputs a list of $\ell = O(\log n)$ Boolean variables holding the cardinality $\#C$ in binary.

W.l.g., assume that n is an exact power of 2, $n = 2^\ell$. Our adder uses a divide-and-conquer strategy on $\ell + 1$ levels (numbered from 0 to ℓ): Level 0 consists of a list of 2^ℓ 1-bits numbers $(X_0^0, \cdots, X_0^{2^\ell - 1})$, namely the bits $C(\mathcal{U})$ themselves. For any $1 < k \leq \ell$, level k consists of $2^{\ell-k}$ numbers $(X_k^0, \cdots, X_k^{2^{\ell-k}-1})$, such that $X_k^j = X_{k-1}^{2j} + X_{k-1}^{2j+1}$. Since the sum of two numbers of b bits fits in $b+1$ bits, each number at level k has $k+1$ bits. This way, the list of level ℓ consists of a single number X_ℓ^0 of $\ell+1$ bits holding $\#C$. Encoding all the binary additions with a carry-propagation scheme takes size and time $O(s(n))$ where $s(n)$ is the total number of bits over all levels, and $s(n) = \sum_{k=0}^{\ell}(k+1)2^{\ell-k} = O(n)$ as required. Finally, it is straightforward to build gadgets of size $O(\ell)$ to implement the arithmetic circuits for any comparison $\perp$ between any output cardinalities or constants. □

In [16], Hunt et al. show that #PLAN-VERTEX-COVER is #P-complete via a planarity-preserving and *weakly* parsimonious reduction from 1-EX-MONO-3SAT to VERTEX-COVER.[6] This problem is parsimoniously DLIN-equivalent to SAT, even in the plane. Since Hunt et al.'s reduction in [16] is also DLIN, this shows together with Theorem 3:

[6] Given a set of monotone 3-clauses (i.e., lists of 3 variables), 1-EX-MONO-3SAT is the problem of the existence of an assignment that satisfies *exactly* one variable in each 3-clause.

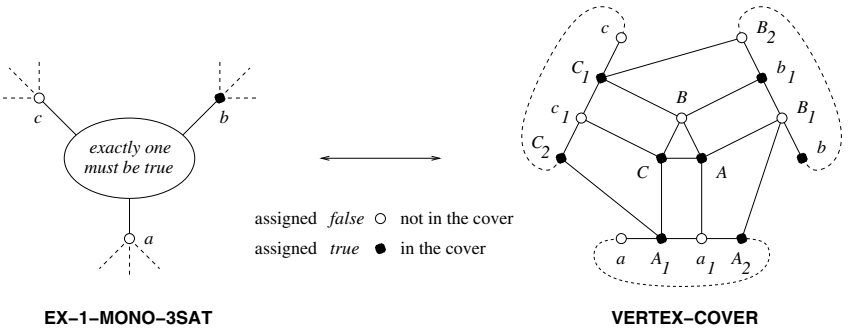


Fig. 3. The reduction from 1-EX-MONO-3SAT to VERTEX-COVER

Theorem 4. – *VERTEX-COVER is LIN-LOCAL-complete.*
– *PLAN-VERTEX-COVER is LIN-PLAN-LOCAL-hard.*

As noted above, Hunt et al.'s reduction is only *weakly* parsimonious. We improve it to make it parsimonious. This implies:

Theorem 5. *UNIQUE-PLAN-VERTEX-COVER is DP-complete under randomized polynomial reductions.*

Proof (sketch). Since it is known that UNIQUE-PLAN-1-EX-MONO-3SAT is DP-complete under randomized polynomial reductions, we only have to give a parsimonious polynomial reduction from PLAN-1-EX-MONO-3SAT to PLAN-VERTEX-COVER. Further, our reduction will be in DLIN. Let I be an input of PLAN-1-EX-MONO-3SAT with m 3-clauses (and hence $3m$ occurrences of variables). Our output-graph G for PLAN-VERTEX-COVER has $15m$ vertices and we ask for a cover K of cardinality $\leq 8m$. Each variable x in I of degree d (i.e., occurring d times) has an associated even cycle e_x in G of length $4d$ (i.e., 4 vertices by occurrence), and each 3-clause r in I has an associated triangle t_r in G. Occurrence-vertices are connected to 3-clause-vertices according to Fig. 3.

The truth-values of the variables a, b, c in I are witnessed by the membership to K of the corresponding vertices a, b, c in G. Simple arguments of cardinality developed in [3] imply the one-to-one correspondence between the configurations in I and G depicted in Fig. 3. □

4 LIN-PLAN-LOCAL Problems and PLAN-HAMILTON

In this section, we show that the many variants of the HAMILTON problem become LIN-PLAN-LOCAL when restricted to planar instances.

Theorem 6. *PLAN-HAMILTON is LIN-PLAN-LOCAL-complete.*

In [1], it was proved that all the cited variants of PLAN-HAMILTON are equivalent under parsimonious DLIN reductions. Thus, to show the LIN-LOCALITY of all these variants of PLAN-HAMILTON, we only have to find

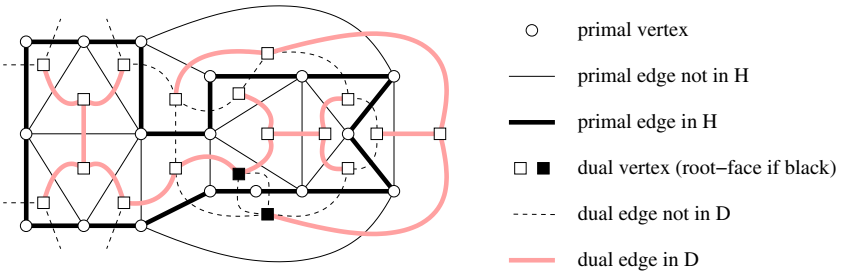


Fig. 4. A planar Hamiltonian cycle

a DLIN-reduction from, say, the planar undirected Hamiltonian cycle to PLAN-SAT. The converse DLIN reduction gives us the LIN-PLAN-LOCAL-hardness of PLAN-HAMILTON and is presented in [1] for space reasons. Since the latter reduction turns out to be parsimonious, it shows the DP-completeness of UNIQUE-PLAN-HAMILTON and answers a question stated as open in [16].

Corollary 2. *UNIQUE-PLAN-HAMILTON is DP-complete under randomized polynomial reductions.*

The rest of this paper is devoted to the proof of the LIN-PLAN-LOCALITY of PLAN-HAMILTON. Note that the problem of the (planar) Hamiltonian partition – i.e., the partition of the vertices of a graph into simple disjoint cycles – is easily DLIN-reducible to (PLAN-)SAT. However, in the general case, SAT does not seem to be able to detect whether there is only one cycle in such a partition. We show that it is indeed possible in the plane, using the following fact:

Fact 1 (Jordan Curve Theorem). *Any collection of k disjoint simple closed curves lying in a plane or a sphere split the surface into exactly $k+1$ maximal connected regions.*

Let $G(V, E)$ be a connected planar graph embedded in the plane, and let $G'(V', E')$ be its dual graph. Let H be a Hamiltonian partition in G. H is viewed as a set of edges. For any set of edges S, define $comp(S)$ to be the number of maximal connected components in S. Denote by H' the set of edges in G' that are dual to H. Define $D = E' \setminus H'$ (see Figs. 4 and 5). The following claim is an immediate consequence of Fact 1:

Claim 1. $comp(D) = comp(H) + 1$.

From now on, an arbitrary outer-face f_{out} is chosen for G. For any cycle C, denote $ext(C)$ (resp. $int(C)$) the exterior (resp. interior) region of C relative to f_{out}. We say that a cycle C_1 of H is a *max-cycle* if $C_2 \notin ext(C_1)$ for any cycle C_2 of H. Similarly, a cycle C_1 of H is a *min-cycle* if $C_2 \notin int(C_1)$ for any cycle C_2 of H. It is not difficult to see (though lengthy to prove) that:

Claim 2. *A connected component of D is acyclic (i.e.,is a tree) iff it lies:*

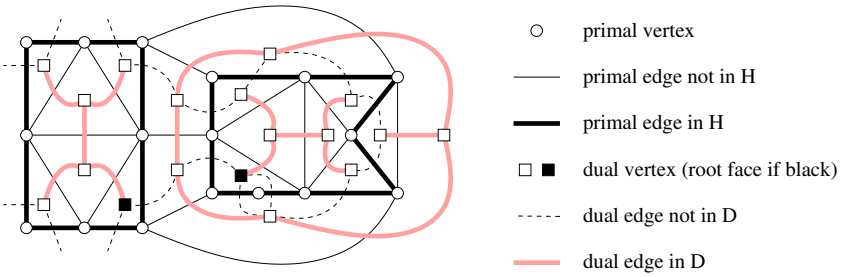


Fig. 5. A planar Hamiltonian partition into 2 disjoint cycles

- *either in the interior of a min-cycle,*
- *or in the exterior of a max-cycle, provided that H has no other max-cycle.*

The proof of Claim 2 is omitted for space reasons and is presented in [2]. (see Figs. 4 and 5 for an intuition). This gives us a first characterization of planar Hamiltonicity:

Claim 3. *$comp(H) = 1$ iff D is a forest of exactly two trees.*[7]

Proof. $(\Longrightarrow)$: If $comp(H) = 1$, then let C_1 be the unique cycle in H: by Claim 1, D has 2 components M_1 and M_2, lying in $int(C_1)$ and $ext(C_1)$ respectively. Since C_1 is both a min-cycle and the unique max-cycle in H, Lemma 2 applies twice, and M_1 and M_2 are both trees.

$(\Longleftarrow)$: In particular $comp(D) = 2$, and by Claim 1, $comp(H) = 1$. □

The idea of the reduction is to locally coerce H to be a Hamiltonian partition of disjoint cycles in the primal graph G while locally constraining D to be a forest of exactly two trees in the dual graph G'. While the former task is easy, the only way one can think of for the latter is to view the trees directed from the leaves to their roots and locally constrain each face but two (the two root faces) to select one adjacent face for its father in the same component of D. However, this leaves the possibility of generating non-tree parasitic components in D (called unicycles) that are trees whose roots are connected in a single circuit. Figure 5 shows such a unicycle. If unicycles occur in D, then H cannot be a Hamiltonian cycle, but these components are left undetected by the system of constraints described above because unicycles do not have any root. Hopefully, the following claim relaxes Claim 3 in a way that will allow us to forbid these unicycles.

Claim 4. *$comp(H) = 1$ iff two trees with adjacent roots exist in D.*

[7] This result is already known in connection with the 4-colors theorem, and the two trees of D are furthermore induced by their sets of vertices. Since this implies that G' is 4-colorable, this motivated a famous conjecture stating that any 3-connected cubic planar graph is Hamiltonian. The conjecture was proved false [25, 10].

Proof. ($\Longrightarrow$) : Since $comp(H) = 1$, Claim 3 applies and D is a forest of exactly two trees M_1 and M_2 lying in $int(C_1)$ and $ext(C_1)$ resp., where C_1 is the unique cycle of H. We only have to exhibit two adjacent roots r_1 and r_2: Choose an arbitrary edge e of C_1, and let $e' = (f, g)$ be its dual edge, such that f lies in $int(C_1)$ and g lies in $ext(C_1)$. $f \in M_1$ and $g \in M_2$, so we can choose $r_1 = f$ and $r_2 = g$.

($\Longleftarrow$) : Let M_1 and M_2 be two trees of D of adjacent roots r_1 and r_2. By Lemma 2, each one must lie either in the interior of a min-cycle or in the exterior of the unique max-cycle. Suppose $comp(H) > 1$, then there are two cases:

- $M_1 \in int(C_i)$ and $M_2 \in int(C_j)$ where C_i and C_j are disjoint min-cycles in H. Since G is planar and C_i and C_j are disjoint, any path from r_1 to r_2 in G' must contain an intermediate vertex lying in $ext(C_i) \cap ext(C_j)$. Hence, r_1 and r_2 are not adjacent, a contradiction.
- $M_1 \in int(C_i)$ and $M_2 \in ext(C_j)$ where C_i is a min-cycle and C_j is the unique max-cycle in H. Since the max-cycle is unique, $C_i \in int(C_j)$, and since $comp(H) > 1$, we conclude that $C_i \neq C_j$. Since G is a planar connected graph and C_i and C_j are disjoint, any path from r_1 to r_2 in G' contains a third vertex lying in $ext(C_i) \cap int(C_j)$. Hence, r_1 and r_2 are not adjacent, a contradiction. □

In [2], we show that we *do not need to guess* r_1 and r_2 because they can be chosen deterministically in linear time. Assume now r_1 and r_2 are fixed. We give the DLIN reduction from PLAN-HAMILTON to PLAN-SAT completing the proof of the LIN-PLAN-LOCALITY of PLAN-HAMILTON. For the sake of readability, we assume that the special clauses $1/N(\ell_1, \cdots, \ell_d)$ and $2/N(\ell_1, \cdots, \ell_d)$ – which are satisfied iff exactly one (resp. two) literal among the ℓ_i $(1 \leq i \leq d)$ are assigned true – are available (these special clauses are easy to implement parsimoniously in the plane using standard clauses, as shown in [2]). Here is the SAT-system satisfied iff G is Hamiltonian: (see also Fig. 4):

- *Set of variables:* Each edge $e \in E \cup E'$ has an associated Boolean variable $thick_e$, asserting that "$e \in H \cup D$". Each face $f \in V'$ of degree d has d associated Boolean variables $father_f^{e'}$, one for each edge $e' = (f, g) \in E'$, asserting that "$e' \in D$ and g is the father of f in D".
- *H is a Hamiltonian partition of G:* For each vertex $v \in V$ of degree d, with incident edges $e_1, \cdots, e_d$, generate the constraint $2/N(thick_{e_1}, \cdots, thick_{e_d})$.
- *D equals $E' - H'$:* For each edge $e \in E$ and its dual edge e', generate the clauses $(thick_e \vee thick_{e'})$ and $(\neg thick_e \vee \neg thick_{e'})$.
- *Each face distinct from r_1 and r_2 has exactly one father:* For each face $f \in V'$ of degree d, $f \notin \{r_1, r_2\}$, with incident edges $e'_1, \cdots, e'_d$, generate the constraint $1/N(father_f^{e'_1}, \cdots, father_f^{e'_d})$.
- *Both adjacent roots r_1 and r_2 have no father:* For each edge $e' \in E'$ incident to a root $r \in \{r_1, r_2\}$, generate the unit clause $(\neg father_r^{e'})$.
- *D is consistently directed:* For each edge $e' = (f, g) \in E'$, generate the clauses $(father_f^{e'} \Longrightarrow thick_{e'})$, $(father_g^{e'} \Longrightarrow thick_{e'})$, $(thick_{e'} \Longrightarrow father_f^{e'} \vee father_g^{e'})$, and $(\neg father_g^{e'} \vee \neg father_f^{e'})$.

The system is built in time $O(|G|+|G'|) = O(|G|)$, including the computation of the dual graph G', and its correctness is an immediate consequence of Claims 3 and 4. In [2], we show how to embed our SAT-system in the plane for each face of G.

5 Conclusion and Further Research

In relation to our class LIN-LOCAL, Lautemann and Weinzinger [18] previously defined Monadic-NLIN as the class of decision problems whose inputs are unary functional structures S and which are defined by some local sentence $\exists \mathcal{C}\ \forall x\ \varphi$ on any expanded structure $(S, Succ)$ where $Succ$ is a list of "compatible" successor functions. [18] proved that the class Monadic-NLIN is logically robust – since it is closed under some logical quantifier-free reduction (which is DLIN-computable) – and meaningful since it contains a number of complete problems via that logical reduction, including SAT, KERNEL, etc. However, we think that this class cannot be viewed as a complexity class because such a class should be closed under some computational device, which is is seemingly not the case for Monadic-NLIN. The main interest of our classes LIN-LOCAL and LIN-PLAN-LOCAL is their great wealth of complete problems – under DLIN reductions – some of which are surprising. Our most significant and most technical result states that the problem PLAN-HAMILTON is LIN-PLAN-LOCAL, which means that it is "essentially local". We conclude this paper by suggesting further research related to our work:

1. Give other logical or algebraic or computational definitions of the classes LIN-LOCAL and LIN-PLAN-LOCAL.
2. Prove that HAMILTON is not LIN-LOCAL; That would be a breakthrough since it implies both LIN-LOCAL $\subsetneq$ NLIN and HAMILTON $\notin$ DLIN which yields DLIN $\neq$ NLIN.
3. Give an intrinsic characterization of the class of problems DLIN-reducible to HAMILTON (it includes interesting NP-complete problems about trees, connectivity, etc.).

References

[1] R. Barbanchon. Planar Hamiltonian problems and linear parsimonious reductions. Tech. report, Les Cahiers du GREYC 1, 2001. (postscript available at http://www.info.unicaen.fr/algo/publications). 400, 406, 407

[2] R. Barbanchon. The problems Sat and Hamilton are equivalent under linear parsimonious reductions in the plane. Tech. report, Les Cahiers du GREYC 4, 2001. (postscript available at http://www.info.unicaen.fr/algo/publications). 400, 408, 409, 410

[3] R. Barbanchon and E. Grandjean. Local problems and linear time. Tech. report, Les Cahiers du GREYC 8, 2001. (postscript available at http://www.info.unicaen.fr/algo/publications). 400, 402, 404, 406

[4] Nadia Creignou. The class of problems that are linearly equivalent to Satisfiability or a uniform method for proving NP-completeness. *Theoretical Computer Science*, 145(1–2):111–145, 1995. 398, 400

[5] M. de Rougemont. Second order and inductive definability on finite structures. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 33:47–63, 1987. 399

[6] A. K. Dewdney. Linear time transformations between combinatorial problems. *Internat. J. Computer Math.*, 11:91–110, 1982. 398

[7] R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity and Computation*, 7:43–73, 1974. 398

[8] R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975. 399

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., 1979. 398, 405

[10] M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar hamiltonian circuit problem is NP-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976. 408

[11] E. Grandjean. A nontrivial lower bound for an NP problem on automata. *SIAM Journal on Computing*, 19:438–451, 1990. 398

[12] E. Grandjean. Linear time algorithms and NP-complete problems. *SIAM Journal on Computing*, 23(3):573–597, 1994. 397, 398

[13] E. Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16:183–236, 1996. 397, 398, 399

[14] E. Grandjean and F. Olive. Monadic logical definability of nondeterministic linear time. *Computational Complexity*, 7(1):54–97, 1998. 398

[15] E. Grandjean and T. Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. Tech. report, Les Cahiers du GREYC 10, 1999. To appear in SIAM Journal on Computing. 397

[16] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, and R. E. Stearns. The complexity of planar counting problems. *SIAM Journal on Computing*, 27(4):1142–1167, August 1998. 404, 405, 407

[17] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 398

[18] C. L. Lautemann and B. Weinzinger. Monadic-NLIN and quantifier-free reductions. In *CSL, 8th annual conference of the EACSL, Lect. Notes Comput. Sci.*, volume 1683 of *LNCS*, pages 322–337, 1999. 398, 399, 400, 410

[19] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982. 404

[20] R. J. Lipton and R. E. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615–627, 1980. 398, 400

[21] K. Mehlhorn and P. Mutzel. On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2):233–242, 1996. 399

[22] W. J. Paul, N. Pippenger, E. Szemeredi, and W. T. Trotter. On determinism versus non-determinism and related problems. In *24th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE Computer Society Press, 1982. 397

[23] T. Schwentick. On winning Ehrenfeucht games and Monadic NP. *Annals of Pure and Applied Logic*, 79(1):61–92, 1996. 399

[24] R. E. Stearns and H. B. Hunt III. Power indices and easier hard problems. *Mathematical Systems Theory*, 23(4):209–225, 1990. 398

[25] W. T. Tutte. On Hamilton circuits. *J. London Math. Soc.*, 21:98–101, 1946. 408

Equivalence and Isomorphism for Boolean Constraint Satisfaction*

Elmar Böhler[1], Edith Hemaspaandra[2], Steffen Reith[3], and Heribert Vollmer[4]

[1] Theoretische Informatik, Universität Würzburg
Am Hubland, D-97074 Würzburg, Germany
boehler@informatik.uni-wuerzburg.de

[2] Department of Computer Science, Rochester Institute of Technology
Rochester, NY 14623, U.S.A.
eh@cs.rit.edu‡

[3] LengfelderStr. 35b, D-97078 Würzburg, Germany
streit@streit.cc§

[4] Theoretische Informatik, Universität Hannover
Appelstr. 4, D-30167 Hannover, Germany
vollmer@informatik.uni-hannover.de¶

Abstract. A Boolean constraint satisfaction instance is a set of constraint applications where the allowed constraints are drawn from a fixed set $\mathcal{C}$ of Boolean functions. We consider the problem of determining whether two given constraint satisfaction instances are equivalent and prove a *dichotomy theorem* by showing that for all finite sets $\mathcal{C}$ of constraints, this problem is either polynomial-time solvable or coNP-complete, and we give a simple criterion to determine which case holds. A more general problem addressed in this paper is the isomorphism problem, the problem of determining whether there exists a renaming of the variables that makes two given constraint satisfaction instances equivalent in the above sense. We prove that this problem is coNP-hard if the corresponding equivalence problem is coNP-hard, and polynomial-time many-one reducible to the graph isomorphism problem in all other cases.

1 Introduction

A Boolean constraint satisfaction instance is a set of constraint applications where the allowed constraints are drawn from a fixed set $\mathcal{C}$ of Boolean functions. Let CSP($\mathcal{C}$) denote the problem of deciding whether a given set of constraint applications of $\mathcal{C}$ is satisfiable. Clearly, there are an infinite number of CSP($\mathcal{C}$) problems, and all these problems are in NP.

* Supported in part by grant NSF-INT-9815095/DAAD-315-PPP-gü-ab.

‡ Supported in part by an RIT FEAD grant. Work done in part while visiting Julius-Maximilians-Universität Würzburg.

§ Work done in part while employed at Julius-Maximilians-Universität Würzburg.

¶ Work done in part while employed at Julius-Maximilians-Universität Würzburg.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 412–426, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Ladner [Lad75] showed that, under the assumption that P $\neq$ NP, there are infinitely many polynomial-time many-one degrees in NP. One might therefore suspect that some of the CSP($\mathcal{C}$) problems are neither NP-complete, nor in P. However, in 1978, Schaefer proved the following remarkable *dichotomy theorem*: CSP($\mathcal{C}$) is either in P or NP-complete. He also completely characterized for which sets of constraints the problem is in P and for which it is NP-complete.

In recent years, there has been renewed interest in Schaefer's result and in constraint satisfaction problems. Creignou examined in [Cre95] how difficult it is to find assignments to constraint satisfaction problems that do not necessarily satisfy all clauses but that satisfy as many clauses as possible. Together with Hermann she studied the difficulty of determining the number of satisfying assignments of a given constraint satisfaction problem [CH96]. In [CH97], Creignou and Hérbrard discussed algorithms that generate all satisfying assignments. Kirousis and Kolaitis researched the complexity of finding minimal satisfying assignments for constraint satisfaction problems in [KK01], and Khanna, Sudan, Trevisan and Williamson examined the approximability of these problems [KSTW01]. Reith and Vollmer [RV00] looked at lexicographical minimal and maximal satisfying assignments of constraint satisfaction problems and of formulas that are built from Boolean functions out of algebraically closed classes in the sense of Post [Pos41]. In [RW00], Reith and Wagner examined various problems related to constraint satisfaction and Post's classes, such as the circuit value, counting, and threshold problems for restricted classes of Boolean circuits. The Ph.D. thesis of Reith [Rei01] contains a wealth of results about problems dealing with restricted Boolean circuits, formulas, and constraint satisfaction. Consult the excellent monograph [CKS00] for an almost completely up-to-date overview of further results and dichotomy theorems for constraint satisfaction problems.

Constraint satisfaction problems are used as a programming or query language in fields such as artificial intelligence and database theory, and the above complexity results shed light on the difficulty of design of systems in those areas. A problem of immense importance from a practical perspective is that of determining whether two sets of constraint applications express the same state of affairs (that is, are equivalent), for example, in the applications, if two programs or queries are equivalent, or if a program matches a given specification. Surprisingly, this problem has not yet been looked at from a complexity point of view. We investigate this problem in Section 3, and we obtain a complete classification of the complexity of determining if two given constraint satisfaction instances are equivalent (Theorem 6). For any finite set $\mathcal{C}$ of constraints, we show that the considered problem is either (1) solvable in polynomial time, or (2) complete for coNP. As in Schaefer's result, our proof is constructive in the sense that it allows us to easily determine, given $\mathcal{C}$, if (1) or (2) holds.

Besides the immediate practical relevance of the equivalence problem, we also see our results from Section 3 as contributions to the study of two other decision problems: First, the equivalence problem is a "sub-problem" of the minimization problem, i.e., the problem to find out, given a set of constraints,

if it can equivalently be expressed with a fewer number of constraints. Secondly, equivalence relates to the isomorphism problem, which has been studied from a theoretical perspective for various mathematical structures. Most prominently, the question if two given (directed or undirected) graphs are isomorphic is one of the few problems in NP neither known to be in P nor known to be NP-complete (see, e.g.,[KST93]). The most recent news about graph isomorphism is a number of hardness results (e.g., for NL, PL, and DET) given in [Tor00].

In Section 4, we address the isomorphism problem for CSPs. Related to our study are the papers [AT00, BRS98] presenting a number of results concerning isomorphism of propositional formulas. We show that the isomorphism problem for constraint applications is coNP-hard if the corresponding equivalence problem is coNP-hard, and polynomial-time many-one reducible to the just-mentioned graph isomorphism problem in all other cases (Theorem 17). We also show that for a number of these cases, the isomorphism problem is in fact polynomial-time many-one equivalent to graph isomorphism (Theorems 24 and 25). The proof of Theorem 17 can also be used to prove a general, non-trivial $\mathrm{P}_{||}^{\mathrm{NP}}$ (parallel access to NP) upper bound for the isomorphism problems for constraint satisfaction (Corollary 23).

2 Boolean Constraint Satisfaction Problems

We start by formally introducing constraint satisfaction problems. The definitions necessary for the equivalence and isomorphism problems will be given in the upcoming sections.

Definition 1. 1. A *constraint* C (of arity k) is a Boolean function from $\{0,1\}^k$ to $\{0,1\}$.
2. If C is a constraint of arity k, and $x_1, x_2, \ldots, x_k$ are (not necessarily distinct) variables, then $C(x_1, x_2, \ldots, x_k)$ is a *constraint application of* C.
3. If C is a constraint of arity k, and for $1 \leq i \leq k$, x_i is a variable or a constant (0 or 1), then $C(x_1, x_2, \ldots, x_k)$ is a constraint application of C *with constants*.

The decision problems examined by Schaefer are the following.

Definition 2. Let $\mathcal{C}$ be a finite set of constraints.

1. $\mathrm{CSP}(\mathcal{C})$ is the problem of, given a set S of constraint applications of $\mathcal{C}$, to decide whether S is satisfiable, i.e., whether there exists an assignment to the variables of S that satisfies every constraint application in S.
2. $\mathrm{CSP}_c(\mathcal{C})$ is the problem of, given a set S of constraint applications of $\mathcal{C}$ with constants, to decide whether S is satisfiable.

The complexity of CSP problems depends on those properties of constraints that we define next.

Definition 3. Let C be a constraint.

- C is *0-valid* if $C(\mathbf{0}) = 1$.
- C is *1-valid* if $C(\mathbf{1}) = 1$.
- C is *Horn* (a.k.a. weakly negative) if C is equivalent to a CNF formula where each clause has at most one positive variable.
- C is *anti-Horn* (a.k.a. weakly positive) if C is equivalent to a CNF formula where each clause has at most one negative variable.
- C is *bijunctive* if C is equivalent to a 2CNF formula.
- C is *affine* if C is equivalent to an XOR-CNF formula.
- C is *complementive* (a.k.a. C-closed) if for every $s \in \{0,1\}^k$, $C(s) = C(\overline{s})$, where k is the arity of C and $\overline{s} \in \{0,1\}^k =_{\text{def}} \mathbf{1} - s$, i.e., $\overline{s}$ is obtained by flipping every bit of s.

Let $\mathcal{C}$ be a finite set of constraints. We say $\mathcal{C}$ is 0-valid, 1-valid, Horn, anti-Horn, bijunctive, affine, or complementive if *every* constraint $C \in \mathcal{C}$ is 0-valid, 1-valid, Horn, anti-Horn, bijunctive, affine, or complementive, respectively. Finally, we say that $\mathcal{C}$ is *Schaefer* if $\mathcal{C}$ is Horn or anti-Horn or affine or bijunctive.

Schaefer's theorem can now be stated as follows.

Theorem 4 (Schaefer [Sch78]). *Let $\mathcal{C}$ be a finite set of constraints.*

1. *If $\mathcal{C}$ is 0-valid, 1-valid, or Schaefer, then* $\mathrm{CSP}(\mathcal{C})$ *is in* P*; otherwise,* $\mathrm{CSP}(\mathcal{C})$ *is* NP*-complete.*
2. *If $\mathcal{C}$ is Schaefer, then* $\mathrm{CSP}_c(\mathcal{C})$ *is in* P*; otherwise,* $\mathrm{CSP}_c(\mathcal{C})$ *is* NP*-complete.*

In this paper, we will study two other decision problems for constraint satisfaction problems. In the next section, we will look at the question of whether two given CSPs are equivalent. In Section 4, we address the isomorphism problem for CSPs. In both cases, we will prove dichotomy theorems.

3 The Equivalence Problem for Constraint Satisfaction

The decision problems studied in this section are the following.

Definition 5. Let $\mathcal{C}$ be a finite set of constraints.

1. $\mathrm{EQUIV}(\mathcal{C})$ is the problem of, given two sets S and U of constraint applications of $\mathcal{C}$, to decide whether S and U are equivalent, i.e., whether for every assignment to the variables, S is satisfied if and only if U is satisfied.
2. $\mathrm{EQUIV}_c(\mathcal{C})$ is the problem of, given two sets S and U of constraint applications of $\mathcal{C}$ with constants, to decide whether S and U are equivalent.

It is immediate that all these equivalence problems are in coNP. Note that in some sense, equivalence is at least as hard as non-satisfiability, since S is not satisfiable if and only if S is equivalent to 0. Thus, we obtain immediately that if $\mathcal{C}$ is not Schaefer, then $\mathrm{EQUIV}_c(\mathcal{C})$ is coNP-complete.

On the other hand, equivalence can be harder than satisfiability. For example, equivalence between Boolean formulas with $\wedge$ and $\vee$ (i.e., without negation) is coNP-complete [EG95] while non-satisfiability for these formulas is clearly in P. In this section, we will prove the following dichotomy theorem.

Theorem 6. *Let $\mathcal{C}$ be a finite set of constraints. If $\mathcal{C}$ is Schaefer, then* $\mathrm{EQUIV}(\mathcal{C})$ *and* $\mathrm{EQUIV}_c(\mathcal{C})$ *are in* P*; otherwise,* $\mathrm{EQUIV}(\mathcal{C})$ *and* $\mathrm{EQUIV}_c(\mathcal{C})$ *are* coNP*-complete.*

The cases of constraints with polynomial-time equivalence problems are easy to identify, using the following lemma, which states that $\mathrm{EQUIV}_c(\mathcal{C})$ is polynomial-time conjunctive truth-table reducible to $\overline{\mathrm{CSP}_c(\mathcal{C})}$. Conjunctive truth-table reducibility is a reducibility notion that is less strict than many-one reducibility, but stricter than Turing reducibility. A set A is polynomial-time conjunctive truth-table reducible to set B if there is a polynomial-time computable function f that computes on input x a list of strings (queries) $q_1, \ldots, q_\ell$ such that $x \in A$ if and only if for all $1 \leq i \leq \ell$, $q_i \in B$. Note that if B is in P, then so is A.

Lemma 7. $\mathrm{EQUIV}_c(\mathcal{C})$ *is polynomial-time conjunctive truth-table reducible to* $\overline{\mathrm{CSP}_c(\mathcal{C})}$.

Proof. Let S and U be two sets of constraint applications of $\mathcal{C}$ with constants. Note that S and U are equivalent if and only if $U \to A$ for every $A \in S$, and $S \to B$ for every $B \in U$. Here and in the rest of the paper, when we write a set of constraint applications S in a Boolean formula, we take this to be a shorthand for $\bigwedge_{\widehat{A} \in S} \widehat{A}$.

Given a constraint application $\widehat{A}$ with constants and a set $\widehat{S}$ of constraint applications of $\mathcal{C}$ with constants, it is easy to check whether $\widehat{S} \to \widehat{A}$ with at most 2^k conjunctive truth-table queries to $\mathrm{CSP}_c(\mathcal{C})$, where k is the maximum arity of $\mathcal{C}$: For every assignment to the variables in $\widehat{A}$ that does not satisfy $\widehat{A}$, substitute this partial truth assignment in $\widehat{S}$. $\widehat{S} \to \widehat{A}$ if and only if all of these substitutions result in sets of constraint applications of $\mathcal{C}$ with constants that are not satisfiable. □

If $\mathcal{C}$ is Schaefer, then $\mathrm{CSP}_c(\mathcal{C})$ is in P by Schaefer's theorem and we immediately obtain the following corollary.

Corollary 8. *Let $\mathcal{C}$ be a finite set of constraints. If $\mathcal{C}$ is Schaefer, then* $\mathrm{EQUIV}_c(\mathcal{C})$ *is in* P.

Having identified the easy equivalence cases, to prove our dichotomy theorem (Theorem 6), it remains to show that if $\mathcal{C}$ is not Schaefer, then $\mathrm{EQUIV}(\mathcal{C})$ is coNP-hard.

First of all, note that this would be easy to prove if we had constants in the language, since for all sets S of constraint applications of $\mathcal{C}$, S is not satisfiable if and only if S is equivalent to 0. Still, we can use this simple observation in the case where $\mathcal{C}$ is not 0-valid and not 1-valid.

Claim 9 *If $\mathcal{C}$ is not Schaefer, not 0-valid, and not 1-valid, then* $\mathrm{EQUIV}(\mathcal{C})$ *is* coNP*-hard.*

Proof. We will reduce $\overline{\mathrm{CSP}(\mathcal{C})}$ to $\mathrm{EQUIV}(\mathcal{C})$. Let S be a set of constraint applications of $\mathcal{C}$. As noted above, S is not satisfiable if and only if S is equivalent to 0. Let $C_0 \in \mathcal{C}$ be a constraint that is not 0-valid, and let $C_1 \in \mathcal{C}$ be a constraint that is not 1-valid. Note that, for any variable x, $\{C_0(x,\ldots,x), C_1(x,\ldots,x)\}$ is equivalent to 0, and thus $S \in \overline{\mathrm{CSP}(\mathcal{C})}$ if and only S is equivalent to $\{C_0(y,\ldots,y), C_1(y,\ldots,y)\}$. □

If $\mathcal{C}$ is not Schaefer, but is 0-valid or 1-valid, then every set of constraint applications of $\mathcal{C}$ is trivially satisfiable (by $\mathbf{0}$ or $\mathbf{1}$). In these cases, a reduction from $\overline{\mathrm{CSP}(\mathcal{C})}$ will not help, since $\overline{\mathrm{CSP}(\mathcal{C})}$ is in P. However, we will show that in these cases the problem of determining whether there exists a non-trivial satisfying assignment is NP-complete and we will use the complements of these satisfiability problems to reduce from.

Creignou and Hébrard prove the following result, concerning the existence of non-trivial satisfying assignments ([CH97, Proposition 4.7], their notation for our $\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}$ is SAT^*).

Proposition 10 ([CH97]). *If $\mathcal{C}$ is not Schaefer, then $\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ is* NP-*complete, where $\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ is the problem of, given a set S of constraint applications of $\mathcal{C}$, to decide whether there is a satisfying assignment for S other than $\mathbf{0}$ and $\mathbf{1}$.*

$\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ corresponds to the notion of "having a non-trivial satisfying assignment" in the case that $\mathcal{C}$ is 0-valid and 1-valid. We will reduce $\overline{\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})}$ to $\mathrm{EQUIV}(\mathcal{C})$ in this case in the proof of Claim 14 to follow.

For the cases that $\mathcal{C}$ is not 1-valid or not 0-valid, we obtain the following analogues of Proposition 10 from Creignou and Hébrard's proof of Proposition 10.

Proposition 11 (implicit in [CH97]).

1. *If $\mathcal{C}$ is not Schaefer and not 0-valid then $\mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$ is* NP-*complete, where $\mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$ is the problem of, given a set S of constraint applications of $\mathcal{C}$, to decide whether there is a satisfying assignment for S other than $\mathbf{1}$.*
2. *If $\mathcal{C}$ is not Schaefer and not 1-valid then $\mathrm{CSP}_{\neq\mathbf{0}}(\mathcal{C})$ is* NP-*complete, where $\mathrm{CSP}_{\neq\mathbf{0}}(\mathcal{C})$ is the problem of, given a set S of constraint applications of $\mathcal{C}$, to decide whether there is a satisfying assignment for S other than $\mathbf{0}$.*

Proof. Careful inspection of Creignou and Hébrard's proof of Proposition 10 shows that following holds if $\mathcal{C}$ is not Schaefer:

1. If $\mathcal{C}$ is not 0-valid and not 1-valid, then $\mathcal{L} = \{S \mid S \in \mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ and not $S(\mathbf{0})$ and not $S(\mathbf{1})\}$ is NP-complete (this is case 1 of Creignou and Hébrard's proof).
2. If $\mathcal{C}$ is 0-valid and not 1-valid, then $\mathcal{L}_0 = \{S \mid S \in \mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ and not $S(\mathbf{1})\}$ is NP-complete (this is case 2b of Creignou and Hébrard's proof).
3. If $\mathcal{C}$ is 1-valid and not 0-valid, then $\mathcal{L}_1 = \{S \mid S \in \mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ and not $S(\mathbf{0})\}$ is NP-complete (this is case 3b of Creignou and Hébrard's proof).

This almost immediately implies Proposition 11. Let $\mathcal{C}$ be not Schaefer and not 0-valid. If $\mathcal{C}$ is not 1-valid, then $\mathcal{L}$ trivially many-one reduces to $\mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$, since, for S a set of constraint applications of $\mathcal{C}$, $S \in \mathcal{L}$ if and only if not $S(\mathbf{0})$, not $S(\mathbf{1})$, and $S \in \mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$. Similarly, if $\mathcal{C}$ is 1-valid, then $\mathcal{L}_1$ trivially many-one reduces to $\mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$. This proves part (1) of Proposition 11. Part (2) follows by symmetry. □

Claim 12 *Let $\mathcal{C}$ be a finite set of constraints.*

1. *If $\mathcal{C}$ is 1-valid, not Schaefer, and not 0-valid, then* $\mathrm{EQUIV}(\mathcal{C})$ *is* coNP*-hard.*
2. *If $\mathcal{C}$ is 0-valid, not Schaefer and not 1-valid, then* $\mathrm{EQUIV}(\mathcal{C})$ *is* coNP*-hard.*

Proof. We will prove the first case; the proof of the second case is similar. We will reduce $\overline{\mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})}$ to $\mathrm{EQUIV}(\mathcal{C})$ as follows. Let S be a set of constraint applications of $\mathcal{C}$ and let $x_1, \ldots, x_n$ be the variables occurring in S. Note that $\mathbf{1}$ satisfies S, since every constraint in S is 1-valid. Therefore, $S \notin \mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\bigwedge_{i=1}^{n} x_i$. Let $C \in \mathcal{C}$ be not 0-valid. Since C is 1-valid, x_i is equivalent to $C(x_i, \ldots, x_i)$. It follows that $S \notin \mathrm{CSP}_{\neq\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\{C(x_i, \ldots, x_i) \mid 1 \leq i \leq n\}$. □

The final case is where $\mathcal{C}$ is both 0-valid and 1-valid. We need the following key lemma from Creignou and Hébrard which is used in their proof of Proposition 10.

Lemma 13 ([CH97], Lemma 4.9(1)). *Let $\mathcal{C}$ be a finite set of constraints that is not Horn, not anti-Horn, not affine, and 0-valid. Then either*

1. *There exists a set V_0 of constraint applications of $\mathcal{C}$ with variables x and y and constant 0 such that V_0 is equivalent to $x \to y$, or*
2. *There exists a set V_0 of constraint applications of $\mathcal{C}$ with variables x, y, z and constant 0 such that V_0 is equivalent to $(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge z) \vee (\overline{x} \wedge y \wedge z)$.*

Claim 14 *Let $\mathcal{C}$ be a finite set of constraints. If $\mathcal{C}$ is not Schaefer but both 0-valid and 1-valid, then* $\mathrm{EQUIV}(\mathcal{C})$ *is* coNP*-hard.*

Proof. We will reduce $\overline{\mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})}$ to $\mathrm{EQUIV}(\mathcal{C})$. Let S be a set of constraint applications of $\mathcal{C}$ and let $x_1, \ldots x_n$ be the variables occurring in S. Note that $\mathbf{0}$ and $\mathbf{1}$ satisfy S, since every constraint in S is 0-valid and 1-valid. Therefore, $S \notin \mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$.

First, suppose there is a constraint $C \in \mathcal{C}$ that is non-complementive. (This case is similar to Creignou and Hébrard's case 2a.) Let k be the arity of C and let $s \in \{0,1\}^k$ be an assignment such that $C(s) = 1$ and $C(\overline{s}) = 0$. Let $A(x, y)$ be the constraint application $C(a_1, \ldots, a_k)$, where $a_i = y$ if $s_i = 1$ and $a_i = x$ if $s_i = 0$. Then $A(0,0) = A(1,1) = 1$, since A is 0-valid and 1-valid; $A(0,1) = 1$, since $C(s) = 1$; and $A(1,0) = 0$, since $C(\overline{s}) = 0$. Thus, $A(x, y)$ is equivalent to $x \to y$. Since $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$ is equivalent to $\bigwedge_{1 \leq i,j \leq n}(x_i \to x_j)$, it follows that $S \notin \mathrm{CSP}_{\neq\mathbf{0},\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\bigwedge_{1 \leq i,j \leq n} A(x_i, x_j)$.

It remains to consider the case where every constraint in $\mathcal{C}$ is complementive. Let V_0 be the set of constraint applications of $\mathcal{C}$ with constant 0 from Lemma 13.

Let V_f be the set of constraint applications of $\mathcal{C}$ that results when we replace each occurrence of 0 in V_0 by f, where f is a new variable. There are two cases to consider, depending on the form of V_0.

Case 1: $V_0(x, y)$ is equivalent to $(x \to y)$. In this case, consider $V_f(f, x, y)$. Since $V_f(0, x, y)$ is equivalent to $x \to y$, and every constraint in S is complementive, it follows that $V_f(f, x, y)$ is equivalent to $(\overline{f} \wedge (x \to y)) \vee (f \wedge (y \to x))$. Thus, $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$ is equivalent to $\bigwedge_{1 \leq i,j \leq n} V_f(f, x_i, x_j)$, and it follows that $S \notin \mathrm{CSP}_{\neq \mathbf{0},\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\bigwedge_{1 \leq i,j \leq n} V_f(f, x_i, x_j)$.

Case 2: $V_0(x, y, z)$ is equivalent to $(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge z) \vee (\overline{x} \wedge y \wedge z)$. Since all constraints in V_0 are complementive, $V_f(f, x, y, z)$ behaves as follows: $V_f(0, 0, 0, 0) = V_f(0, 1, 0, 1) = V_f(0, 0, 1, 1) = V_f(1, 1, 1, 1) = V_f(1, 0, 1, 0) = V_f(1, 1, 0, 0) = 1$, and V_f is 0 for all other assignments. Note that $V_f(f, f, x_i, x_j)$ is equivalent to $(x_i \leftrightarrow x_j)$, and thus that $\bigwedge_{i=1}^{n} x_i \vee \bigwedge_{i=1}^{n} \overline{x_i}$ is equivalent to $\bigwedge_{1 \leq i,j \leq n} V_f(f, f, x_i, x_j)$. It follows that $S \notin \mathrm{CSP}_{\neq \mathbf{0},\mathbf{1}}(\mathcal{C})$ if and only if S is equivalent to $\bigwedge_{1 \leq i,j \leq n} V_f(f, f, x_i, x_j)$. □

Theorem 6 follows immediately from Corollary 8 and Claims 9, 12, and 14.

4 The Isomorphism Problem for Constraint Satisfaction

In this section, we study a more general problem: The question of whether a set of constraint applications can be made equivalent to a second set of constraint applications using a suitable renaming of its variables. We need some definitions.

Definition 15. 1. Let S be a set of constraint applications with constants over variables X and let π be a permutation of X. By $\pi(S)$ we denote the set of constraint applications that results when we replace simultaneously all variables x_i of S by $\pi(x_i)$.

2. Let S be a set of constraint applications over variables X. The number of satisfying assignments of S is $\#_1(S) =_{\mathrm{def}} ||\{\, I \mid I$ is an assignment to all variables in X that satisfies every constraint application in $S\,\}||$.

The isomorphism problem now is formally defined as follows.

Definition 16. 1. $\mathrm{ISO}(\mathcal{C})$ is the problem of, given two sets S and U of constraint applications of $\mathcal{C}$ over variables X, to decide whether S and U are isomorphic, i.e., whether there exists a permutation π of X such that $\pi(S)$ is equivalent to U.

2. $\mathrm{ISO}_c(\mathcal{C})$ is the problem of, given two sets S and U of constraint applications of $\mathcal{C}$ with constants over variables X, to decide whether S and U are isomorphic.

We remark that for S and U to be isomorphic, we require that formally they are defined over the same set of variables. Of course, this does not mean that all these variables actually have to occur textually in both formulas.

As in the case for equivalence, isomorphism is in some sense as least as hard as non-satisfiability, since S is not satisfiable if and only if S is isomorphic to 0. Thus, we immediately obtain that if $\mathcal{C}$ is not Schaefer, then $\mathrm{ISO}_c(\mathcal{C})$ is coNP-hard. Unlike the equivalence case however, we do *not* have a trivial coNP upper bound for isomorphism problems. In fact, there is some evidence [AT00] that the isomorphism problem for Boolean formulas is not in coNP. Note that determining whether two formulas or two sets of constraint applications are isomorphic is trivially in Σ_2^p. However, the isomorphism problem for formulas is not Σ_2^p-complete unless the polynomial hierarchy collapses [AT00]. In the sequel (Corollary 23) we will prove a stronger result for the isomorphism problem for Boolean constraints: We will prove a $\mathrm{P}_{||}^{\mathrm{NP}}$ upper bound for these problems, where $\mathrm{P}_{||}^{\mathrm{NP}}$ is the class of problems that can be solved in polynomial time with parallel access to an NP oracle. This class has many different characterizations, see, for example, Hemaspaandra [Hem89], Papadimitriou and Zachos [PZ83], Wagner [Wag90].

For equivalence, we obtained a polynomial-time upper bound for sets of constraints that are Schaefer. In contrast, we will show in the sequel that, for example, isomorphism for positive 2CNF formulas (i.e., isomorphism between two sets of constraint applications of $\{(0,1),(1,0),(1,1)\}$) is polynomial-time many-one equivalent to the graph isomorphism problem (GI).

The main result of this section is the following theorem.

Theorem 17. *Let $\mathcal{C}$ be a finite set of constraints. If $\mathcal{C}$ is Schaefer, then* $\mathrm{ISO}(\mathcal{C})$ *and* $\mathrm{ISO}_c(\mathcal{C})$ *are polynomial-time many-one reducible to* GI*, otherwise,* $\mathrm{ISO}(\mathcal{C})$ *and* $\mathrm{ISO}_c(\mathcal{C})$ *are* coNP*-hard.*

Note that if $\mathcal{C}$ is Schaefer, then the isomorphism problems $\mathrm{ISO}(\mathcal{C})$ and $\mathrm{ISO}_c(\mathcal{C})$ cannot be coNP-hard, unless NP = coNP. (This follows from Theorem 17 and the fact that GI is in NP.) Under the assumption that NP $\neq$ coNP, Theorem 17 thus distinguishes a hard case (coNP-hard) and an easier case (reducible to GI). In this sense, Theorem 17 is again a dichotomy theorem.

We will first prove the lower bound part of Theorem 17. In our proof, we will use the following property.

Lemma 18. *Let S and U be sets of constraint applications of $\mathcal{C}$ with constants. If S is isomorphic to U then $\#_1(U) = \#_1(S)$.*

Proof. First note that every permutation of the variables of S induces a permutation of the rows of the truth-table of S. Now let π be a permutation such that $\pi(S) \equiv U$. Then $\#_1(S) = \#_1(\pi(S))$ and $\#_1(\pi(S)) = \#_1(U)$. □

Claim 19 *If $\mathcal{C}$ is not Schaefer, then* $\mathrm{ISO}(\mathcal{C})$ *is* coNP*-hard.*

Proof. We first note that a claim analogous to Claim 9 also holds for isomorphism, i.e., if $\mathcal{C}$ is not Schaefer, not 0-valid, and not 1-valid, then $\mathrm{ISO}(\mathcal{C})$ is coNP-hard. For the proof, we use the same reduction as in the proof of Claim 9, i.e., we claim that $S \in \overline{\mathrm{CSP}(\mathcal{C})}$ if and only if S

is isomorphic to $\{C_0(y,\ldots,y), C_1(y,\ldots,y)\}$. This follows immediately, since $\{C_0(y,\ldots,y), C_1(y,\ldots,y)\}$ is equivalent to 0, and S is not satisfiable iff S is isomorphic to 0.

Next, we claim, analogously to Claim 12, that

(1) if $\mathcal{C}$ is 1-valid, not Schaefer, and not 0-valid, then $\mathrm{ISO}(\mathcal{C})$ is coNP-hard; and
(2) If $\mathcal{C}$ is 0-valid, not Schaefer, and not 1-valid, then $\mathrm{ISO}(\mathcal{C})$ is coNP-hard.

For the first case, we use the same reduction as in the proof of Claim 12. Note that if S is equivalent to $\{C(x_i,\ldots,x_i) \mid 1 \le i \le n\}$, then $(S, \{C(x_i,\ldots,x_i) \mid 1 \le i \le n\}) \in \mathrm{ISO}(\mathcal{C})$ via the identity permutation. For the converse, note that $S \notin \overline{\mathrm{CSP}_{\neq \mathbf{1}}(\mathcal{C})}$ iff $\#_1(S) \ge 2$ and that $\#_1(\{C(x_i,\ldots,x_i) \mid 1 \le i \le n\}) = 1$. The result now follows by Lemma 18. The proof of the second case is similar.

The remaining case is that of a set $\mathcal{C}$ that is not Schaefer, but both 0-valid and 1-valid. We use the same reduction as in the proof of Claim 14. Clearly, if $(S,U) \in \mathrm{EQUIV}(\mathcal{C})$ then also $(S,U) \in \mathrm{ISO}(\mathcal{C})$ via the identity permutation. To show the other direction, note that $S \notin \overline{\mathrm{CSP}_{\neq \mathbf{0},\mathbf{1}}(\mathcal{C})}$ iff $\#_1(S) \ge 3$, and that $\#_1(\bigwedge_{i=1}^n x_i \vee \bigwedge_{i=1}^n \overline{x_i}) = 2$. The result now follows by Lemma 18. This completes the proof of Claim 19. □

To complete the proof of Theorem 17, it remains to show that if $\mathcal{C}$ is Schaefer, then $\mathrm{ISO}(\mathcal{C})$ and $\mathrm{ISO}_c(\mathcal{C})$ are polynomial-time many-one reducible to GI. We will reduce $\mathrm{ISO}_c(\mathcal{C})$ to graph isomorphism for vertex-colored graphs, a GI variation that is polynomial-time many-one equivalent to GI.

Definition 20. VCGI *is the problem of, given two vertex-colored graphs* $G_i = (V_i, E_i, \chi_i)$, $i \in \{1,2\}$, $\chi_i : V \to \mathbf{N}$, *to determine whether there exists an isomorphism between* G_1 *and* G_2 *that preserves colors, i.e., whether there exists a bijection* $\pi\colon V_1 \to V_2$ *such that for all* $v, w \in V_1$, $\{v,w\} \in E_1$ *iff* $\{\pi(v), \pi(w)\} \in E_2$ *and* $\chi(v) = \chi(\pi(v))$.

Proposition 21 ([Fon76, BC79]). VCGI *is polynomial-time many-one equivalent to* GI.

By Proposition 21, to complete the proof of Theorem 17, it suffices to show the following.

Claim 22 *If* $\mathcal{C}$ *is Schaefer, then* $\mathrm{ISO}_c(\mathcal{C}) \le_m^p$ VCGI.

Proof. Suppose $\mathcal{C}$ is Schaefer, and let S and U be sets of constraint applications of $\mathcal{C}$ with constants over variables X. We will first bring S and U into normal form. Let $\widehat{S}$ be the set of all constraint applications A of $\mathcal{C}$ with constants such that all of A's variables occur in X and such that $S \to A$. Similarly, let $\widehat{U}$ be the set of all constraint applications B of $\mathcal{C}$ with constants such that all of B's variables occur in X and such that $U \to B$. It is clear that $S \equiv \widehat{S}$, since $S \subseteq \widehat{S}$ and $S \to \widehat{S}$. Likewise, $U \equiv \widehat{U}$. Note that $\widehat{S}$ and $\widehat{U}$ are polynomial-time computable (in $|(S,U)|$), since

1. there exist at most $||\mathcal{C}||(||X||+2)^k$ constraint applications A of $\mathcal{C}$ with constants such that all of A's variables occur in X, where k is the maximum arity of constraints in $\mathcal{C}$, and
2. since $\mathcal{C}$ is Schaefer, determining whether $S \to A$ and whether $U \to B$ takes polynomial time, using the same argument as in the proof of Lemma 7.

Note that we have indeed brought S and U into normal form, since $S \equiv U$ iff $\widehat{S} = \widehat{U}$. In addition, for any permutation π of X, if $\pi(S) \equiv U$, then $\pi(\widehat{S}) = \widehat{U}$. We remark that this approach of first bringing the sets of constraint applications into normal form is also followed in the proof of the coIP$[1]^{\mathrm{NP}}$ upper bound for the isomorphism problem for Boolean formulas [AT00].

It remains to show that we can in polynomial time encode $\widehat{S}$ and $\widehat{U}$ as vertex-colored graphs $G(\widehat{S})$ and $G(\widehat{U})$ such that there exists a permutation π of X with $\pi(\widehat{S}) = \widehat{U}$ if and only if $(G(\widehat{S}), G(\widehat{U})) \in$ VCGI.

Let $\mathcal{C} = \{C_1, \ldots, C_m\}$. We consider the set $P = \{C_{i_1}(x_{11}, x_{12}, \ldots, x_{1k_1}), C_{i_2}(x_{21}, x_{22}, \ldots, x_{2k_2}), \ldots, C_{i_\ell}(x_{\ell 1}, x_{\ell 2}, \ldots, x_{\ell k_\ell})\}$ of constraint applications of $\mathcal{C}$ with constants over variables X such that $i_1 \leq i_2 \leq i_3 \leq \cdots \leq i_\ell$. Define $G(P) = (V, E, \chi)$ as the following vertex-colored graph:

- $V = \{0,1\} \cup \{\, x \mid x \in X \,\} \cup \{\, a_{ij} \mid 1 \leq i \leq \ell, 1 \leq j \leq k_i \,\} \cup \{\, A_i \mid 1 \leq i \leq \ell \,\}$. That is, the set of vertices corresponds to the Boolean constants, the variables in X, the arguments of the constraint applications in P, and the constraint applications in P.
- $E = \{\, \{x, a_{ij}\} \mid x = x_{ij} \,\} \cup \{\, \{a_{ij}, A_i\} \mid 1 \leq i \leq \ell, 1 \leq j \leq k_i \,\}$.
- The vertex coloring χ will distinguish the different categories. Of course, we want to allow any permutation of the variables, so we will give all elements of X the same color. In addition, we also need to allow a permutation of constraint applications of the same constraint.
  - $\chi(0) = 0$, $\chi(1) = 1$,
  - $\chi(x) = 2$ for all $x \in X$,
  - $\chi(A_r) = 2 + j$ if $i_r = j$, and
  - $\chi(a_{ij}) = 2 + m + j$. (This will ensure that we do not permute the order of the arguments.)

If there is a permutation π of X such that $\pi(\widehat{S}) = \widehat{U}$, it is easy to see that $(G(\widehat{S}), G(\widehat{U})) \in$ VCGI. On the other hand, if $(G(\widehat{S}), G(\widehat{U})) \in$ VCGI via a permutation π of the vertices of $G(\widehat{S})$, then note that vertices corresponding to constraint applications can only be permuted together with those vertices corresponding to the arguments of that constraint application. In addition, because of the coloring, the order of arguments is preserved. Thus, if $\pi(A_i) = A_j$ then necessarily $\pi(a_{ir}) = a_{jr}$ for all $1 \leq r \leq k_i$, and because coloring is preserved, A_i in $G(\widehat{S})$ and A_j in $G(\widehat{U})$ correspond to applications of the same constraint. This part of the permutation corresponds to a permutation of the constraint applications in the set $\widehat{S}$. The remaining part of the permutation of $G(\widehat{S})$ is one that solely permutes vertices corresponding to variables in $\widehat{S}$, so $\pi(\widehat{S}) = \widehat{U}$. $\square$

Note that the construction used in proof of Claim 22 can be used to provide a general upper bound on $\mathrm{ISO}_c(\mathcal{C})$: Given sets S and U of constraint applications of $\mathcal{C}$ with constants, first bring S and U into the normal form ($\widehat{S}$ and $\widehat{U}$) described in the proof of Claim 22 (this can be done in polynomial time with parallel access to an NP oracle), and then determine if there exists a permutation π such that $\pi(\widehat{S}) = \widehat{U}$ (this takes one query to an NP oracle). The whole algorithm takes polynomial time with two rounds of parallel queries to NP, which is equal to $\mathrm{P}_{||}^{\mathrm{NP}}$ (Buss and Hay [BH91]). Thus, we have the following upper bound on the isomorphism problem for constraint satisfaction.

Corollary 23. *Let $\mathcal{C}$ be a finite set of constraints.* $\mathrm{ISO}(\mathcal{C})$ *and* $\mathrm{ISO}_c(\mathcal{C})$ *are in* $\mathrm{P}_{||}^{\mathrm{NP}}$.

Finally, we show that for some simple instances of Horn, bijunctive, and affine constraints, the isomorphism problem is in fact polynomial-time many-one *equivalent* to the graph isomorphism problem.

Theorem 24. GI *is polynomial-time many-one equivalent to* $\mathrm{ISO}(\{\{(0,1),(1,0),(1,1)\}\})$ *and to* $\mathrm{ISO}_c(\{\{(0,1),(1,0),(1,1)\}\})$.

Proof. It suffices to show that GI $\leq_m^p$ $\mathrm{ISO}(\{\{(0,1),(1,0),(1,1)\}\})$, since by Theorem 22, $\mathrm{ISO}_c(\{\{(0,1),(1,0),(1,1)\}\}) \leq_m^p$ GI.

Let $G = (V, E)$ be a graph and let $V = \{1, 2, \ldots, n\}$. We encode G in the obvious way as a set of constraint applications $S(G) = \{x_i \vee x_j \mid \{i, j\} \in E\}$. It is immediate that if G and H are two graphs with vertex set $\{1, 2, \ldots, n\}$, then G is isomorphic to H if and only if $S(G)$ is isomorphic to $S(H)$. □

Note that the constraint $\{(0,1),(1,0),(1,1)\}$ is the binary constraint $x \vee y$, denoted by OR_0 in [CKS00]. Theorem 24 can alternatively be formulated as: GI is polynomial-time many-one equivalent to the isomorphism problem for positive 2CNF formulas (with or without constants). Also, from [Tor00], we conclude that this simple isomorphism problem thus is hard for NL, PL, and DET.

Theorem 25. GI *is polynomial-time many-one equivalent to* $\mathrm{ISO}(\{\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}\})$ *and to* $\mathrm{ISO}_c(\{\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}\})$.

Proof. We show that GI $\leq_m^p$ $\mathrm{ISO}(\{\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}\})$; this suffices, since by Theorem 22, $\mathrm{ISO}_c(\{\{(1,0,0),(0,1,0),(0,0,1),(1,1,1)\}\}) \leq_m^p$ GI.

Let $G = (V, E)$ be a graph, let $V = \{1, 2, \ldots, n\}$, and enumerate the edges as $E = \{e_1, e_2, \ldots, e_m\}$. We encode G as a set of XOR_3 constraint applications in which propositional variable x_i will correspond to vertex i and propositional variable y_i will correspond to edge e_i. We encode G as $S(G) = S_1(G) \cup S_2(G) \cup S_3(G)$ where

- $S_1(G) = \{x_i \oplus x_j \oplus y_k \mid e_k = \{i, j\}\}$ ($S_1(G)$ encodes the graph),
- $S_2(G) = \{x_i \oplus z_i \oplus z_i' \mid i \in V\}$ ($S_2(G)$ will be used to distinguish x variables from y variables), and

- $S_3(G) = \{y_i \oplus y_j \oplus y_k \mid e_i, e_j, \text{and } e_k \text{ form a triangle in } G\}$. Note that for every $A \in S_3(G)$, $S_1(G) \to A$. We add these constraint applications to $S(G)$ to ensure that $S(G)$ is a maximum set of XOR_3 formulas.

We will show later that if G and H are two graphs with vertex set $\{1, 2, \ldots, n\}$ without isolated vertices, then G is isomorphic to H if and only if $S(G)$ is isomorphic to $S(H)$.

The proof of the theorem relies on the following claim, which shows that $S(G)$ is a maximum set of XOR_3 formulas. The proof of the claim can be found in the full version of the paper.

Claim 26 *Let $G = (V, E)$ be a graph such that $V = \{1, 2, \ldots, n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. Then for every triple of distinct propositional variables a, b, c in $S(G)$, the following holds: If $S(G) \to a \oplus b \oplus c$, then $a \oplus b \oplus c \in S(G)$. Note: we view $a \oplus b \oplus c$ as a* function*, and thus, for example, $a \oplus b \oplus c = c \oplus a \oplus b$.*

How can Claim 26 help us in the proof of Theorem 25? Note that if S and T are maximum sets of $\mathcal{C}$ constraint applications, then $S \equiv T$ if and only if $S = T$. (Here equality should be seen as equality between sets of functions.) So S is isomorphic to T if and only if there exists a permutation ρ of the variables of S such that $\rho(S) = T$.

We will now prove Theorem 25. Let G and H be two graphs. Remove the isolated vertices from G and H. If G and H thus modified do not have the same number of vertices or they do not have the same number of edges, then G and H are clearly not isomorphic. If G and H have the same number of vertices and the same number of edges, then rename the vertices in such a way that the vertex set of both graphs is $V = \{1, 2, \ldots, n\}$. Let $\{e_1, \ldots, e_m\}$ be an enumeration of the edges in G and let $\{e'_1, \ldots, e'_m\}$ be an enumeration of the edges in H.

We will show that G is isomorphic to H if and only if $S(G)$ is isomorphic to $S(H)$.

The left-to-right direction is trivial, since an isomorphism between the graphs induces an isomorphism between sets of constraint applications as follows. If $\pi : V \to V$ is an isomorphism from G to H, then we can define an isomorphism ρ from $S(G)$ to $S(H)$ as follows:

- $\rho(x_i) = x_{\pi(i)}$, $\rho(z_i) = z_{\pi(i)}$, $\rho(z'_i) = z'_{\pi(i)}$, for $i \in V$.
- For $e_k = \{i, j\}$, $\rho(y_k) = y_\ell$ where $e'_\ell = \{\pi(i), \pi(j)\}$.

For the converse, suppose that ρ is an isomorphism from $S(G)$ to $S(H)$. By the observation above, $\rho(S(G)) = S(H)$. Now look at the properties of the different classes of variables.

1. Elements from X are exactly those variables that occur at least twice and that also occur in an element of $S(G)$ together with two variables that occur exactly once. So, ρ will map X onto X.
2. Elements of Z are those variables that occur exactly once and that occur together with an element from X and another element that occurs exactly once. So ρ will map Z to Z.

3. Everything else is an element of Y. So, ρ will map Y onto Y.

For $i \in V$, define $\pi(i) = j$ iff $\rho(x_i) = x_j$. π is 1-1 onto by observation (1) above. It remains to show that $\{i, j\} \in E$ iff $\{\pi(i), \pi(j)\} \in E'$. Let $e_k = \{i, j\}$. Then $x_i \oplus x_j \oplus e_k \in S(G)$. Thus, $\rho(x_i) \oplus \rho(x_j) \oplus \rho(y_k) \in S(H)$. That is, $x_{\pi(i)} \oplus x_{\pi(j)} \oplus \rho(y_k) \in S(H)$. But that implies that $\rho(y_k) = y_\ell$ where $e'_\ell = \{\pi(i), \pi(j)\}$. This implies that $\{\pi(i), \pi(j)\} \in E'$. For the converse, suppose that $\{\pi(i), \pi(j)\} \in E'$. Then $x_{\pi(i)} \oplus x_{\pi(j)} \oplus y_\ell \in S(H)$ for $e_\ell = \{\pi(i), \pi(j)\}$. It follows that $x_i \oplus x_j \oplus \rho^{-1}(y_\ell) \in S(G)$. By the form of $S(G)$, it follows that $\{i, j\} \in E$. □

Note that the constraint $\{(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 1)\})$ is the constraint $x \oplus y \oplus z$, denoted by XOR_3 in [CKS00]. The proof of Theorem 25 also shows that $\mathrm{ISO}(\mathrm{XNOR}_3)$ and $\mathrm{ISO}_c(\mathrm{XNOR}_3)$ are many-one equivalent to GI, where XNOR_3 denotes the constraint $\{(0, 1, 1), (1, 0, 1), (1, 1, 0), (0, 0, 0)\}$. In addition, Theorem 25 holds if we replace the 3 by any fixed $k \geq 3$.

From Theorems 24 and 25, we conclude that, if we could show that isomorphism for bijunctive, anti-Horn (and, by symmetry, Horn), or affine CSPs is in P, then the graph isomorphism problem is in P, settling a long-standing open question in a very surprising way.

Acknowledgements: We would like to thank Lane Hemaspaandra for helpful conversations and suggestions, and the anonymous referees for helpful comments.

References

[AT00] M. Agrawal and T. Thierauf. The formula isomorphism problem. *SIAM Journal on Computing*, 30(3):990–1009, 2000. 414, 420, 422

[BC79] K. S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-01, University of Waterloo, 1979. 421

[BH91] S. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 90(2):86–102, 1991. 423

[BRS98] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on Boolean functions. *Theory of Computing Systems*, 31:679–693, 1998. 414

[CH96] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125:1–12, 1996. 413

[CH97] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 31(6):499–511, 1997. 413, 417, 418

[CKS00] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Applied Mathematics. SIAM, 2000. 413, 423, 425

[Cre95] N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51:511–522, 1995. 413

[EG95] T. Eiter and G. Gottlob. Identifying the minimal transversals of a hypergraph and related problems. *SIAM Journal on Computing*, 24(6):1278–1304, 1995. 415

[Fon76] M. Fontet. Automorphismes de graphes et planarité. In *Asterisque*, pages 73–90. 1976. 421

[Hem89] L. Hemachandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989. 420

[KK01] L. M. Kirousis and P. G. Kolaitis. The complexity of minimal satisfiability problems. In *Proceedings* 18th *Symposium on Theoretical Aspects of Computer Science*, volume 2010, pages 407–418. Springer Verlag, 2001. 413

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: its Structureal Complexity.* Progress in Theoretical Computer Science. Birkhäuser, 1993. 414

[KSTW01] S. Khanna, M. Sudan, L. Trevisan, and D. Williamson. The approximability of constraint satisfaction problems. *SIAM Journal on Computing*, 30(6):1863 – 1920, 2001. 413

[Lad75] R. Ladner. On the structure of polynomial-time reducibility. *Journal of the ACM*, 22:155–171, 1975. 413

[Pos41] E. L. Post. Two-valued iterative systems of mathematical logic. In *Annals of Math. Studies*, volume 5. Princeton University Press, 1941. 413

[PZ83] C. Papadimitriou and S. Zachos. Two remarks on the power of counting. In *Proceedings 6th GI Conference on Theoretical Computer Science*, volume 145 of *Lecture Notes in Computer Science*, pages 269 – 276. Springer Verlag, 1983. 420

[Rei01] S. Reith. *Generalized Satisfiability Problems.* PhD thesis, University of Würzburg, 2001. 413

[RV00] S. Reith and H. Vollmer. Optimal satisfiability for propositional calculi and constraint satisfaction problems. In *Proceedings* 25th *International Symposium on Mathematical Foundations of Computer Science*, volume 1893 of *Lecture Notes in Computer Science*, pages 640–649. Springer Verlag, 2000. 413

[RW00] S. Reith and K. W. Wagner. The complexity of problems defined by Boolean circuits. Technical Report 255, Institut für Informatik, Universität Würzburg, 2000. To appear in *Proceedings International Conference Mathematical Foundation of Informatics*, Hanoi, October 25–28, 1999. 413

[Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proccedings* 10th *Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978. 415

[Tor00] J. Torán. On the hardness of graph isomorphism. In *Proceedings* 41st *Foundations of Computer Science*, pages 180–186, 2000. 414, 423

[Wag90] K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990. 420

Travelling on Designs
Ludics Dynamics

Claudia Faggian

DPMMS – University of Cambridge, United Kingdom
c.faggian@dpmms.cam.ac.uk

Abstract. Proofs in Ludics are represented by designs. Designs (desseins) can be seen as an intermediate syntax between sequent calculus and proof nets, carrying advantages from both approaches, especially w.r.t. cut-elimination. To study interaction between designs and develop a geometrical intuition, we introduce an abstract machine which presents normalization as a token travelling along a net of designs. This allows a concrete approach, from which to carry on the study of issues such as: (i) which part of a design can be recognized interactively; (ii) how to reconstruct a design from the traces of its interactions in different tests.

Ludics is a new theory recently introduced by Girard in [6]. The program is to overcome the distinction between syntax and semantics: proofs are interpreted via proofs, and all properties are expressed and tested internally. Internally means interactively: the objects themselves test each other. The fundamental artifacts of Ludics are *designs,* which are both (i) an abstraction of formal proofs and (ii) a concretion of their semantical interpretation.

Designs have remarkable properties also as a syntax. They may be seen as an intermediate syntax between sequent calculus and proof-nets. Such a syntax carries advantages from both approaches, in particular w.r.t. cut-elimination. Designs: (i) Offer a concise syntax. (ii) Integrate a good treatment of the additives in a syntax that is still light to manipulate (this is a strong point of Ludics with respect to proof-nets and geometry of interaction). (iii) Are close to implementation, in that they make explicit the "addresses" and use tools typical of implementations, such as a dynamical approach to the context.

To have a concrete approach to designs and develop a geometrical intuition, we introduce an abstract machine, called *Loci Abstract Machine (LAM)*, which allows us to present normalization by a token travelling along a net of designs.

The LAM is the starting point from which we developed several tools for the operational study of designs. The path drawn by the token is a sequence of actions that represents the trace of the interaction between the designs. Conversely, we provide tools for reconstructing the agents from the traces of their interactions. A key operation we use exactly corresponds to a well-know operation of Games Semantics, the computation of the *view* ([7], [8]).

Note 1. By *design* we always intend the tree structure that in [6] is called *dessein.* If we refer to its sequent calculus presentation (i.e. *dessin*) we make it explicit.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 427–441, 2002.
© Springer-Verlag Berlin Heidelberg 2002

1 Ludics in a Nutshell

The program of Ludics is to overcome the distinction between *syntax* (the formalism) and *semantics* (its interpretation): proofs are interpreted via proofs. Syntax and semantics meet in the notion of *design*. Designs are both an abstraction of a formal proof, and a concretion of its semantic interpretation. This has been achieved working from two directions.

1. *Making semantics concrete.* This leads to enlarging the universe of proofs, in order to have enough inhabitants to be able to distinguish between them *inside* the system. *Paraproofs* are introduced.

2. *Abstracting from syntax.* The syntax of designs captures the geometrical structure underlying a sequent calculus proof. There are two crucial notions used to obtain this: *focalization* and *locations*. Focalization, which is an essential tool of proof-search ([1]), allows the definition of *synthetic connectives*. Locations are a major novelty of Ludics: proofs do not manipulate formulas, but their *addresses*. These are sequences of natural numbers, which can be thought of as the address in the memory where the formula is stored.

Para-proofs. Ludics provides a setting in which to any proof of A we can oppose (via cut-elimination) a proof of $A^{\perp}$. To this aim, it generalizes the notion of proof (para-proof).

A proof should be thought of in the sense of "proof search" or "proof construction": we start from the conclusion, and guess a last rule, then the rule above. What if we cannot apply any rule? A new rule is introduced, called daimon: $\dfrac{}{\vdash \Gamma}\,\dagger$. It allows us to assume any conclusion, without providing a justification.

Slices. To understand designs, it is useful to have in mind the notion of slice. A &-rule can be seen as the super-imposition of two unary rules: $(a\&b, a)$ and $(a\&b, b)$. Given a derivation, if for any &-rule we select one of the premises, we obtain a derivation where all &-rules are unary. This is called a *slice*. For example, the derivation

$$\dfrac{\dfrac{\dfrac{\cdots}{\vdash a, c}\,a \qquad \dfrac{\cdots}{\vdash b, c}\,b}{\vdash a\&b, c}\,(a\&b, \{a\}), (a\&b, \{b\})}{\vdash (a\&b) \oplus d, c}\,((a\&b) \oplus d, \{a\&b\})$$

can be decomposed into two slices:

$$\dfrac{\dfrac{\dfrac{\cdots}{\vdash a, c}\,a}{\vdash a\&b, c}\,(a\&b, \{a\})}{\vdash (a\&b) \oplus d, c}\,((a\&b) \oplus d, \{a\&b\}) \qquad \text{and} \qquad \dfrac{\dfrac{\dfrac{\cdots}{\vdash b, c}\,b}{\vdash a\&b, c}\,(a\&b, \{b\})}{\vdash (a\&b) \oplus d, c}\,((a\&b) \oplus d, \{a\&b\})$$

The &-rule is a set (the super-imposition) of two unary rules on the same formula. It is important to observe that *normalization is always carried out in a single slice:* selecting one of the premises of a &-rule is exactly what happens during normalization.

Synthetic Connectives. The calculus underlying ludics is 2nd order multiplicative-additive Linear Logic ($\mathbf{M}ALL^2$). Multiplicative and additive connectives of LL separate into two families: positives $(\otimes, \oplus, 1, 0)$ and negatives $(\parr, \&, \bot, \top)$. A cluster of operations of the same polarity can be decomposed in a single step, and can be written as a single connective, which is called a *synthetic connective*. A formula is positive (negative) if its outer-most connective is positive (negative).

In the formula $f = ((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$ we have a positive ternary connective $(-\oplus-)\otimes-$. The immediate subformulas of f are $p_1 \parr p_2, q^\bot, r^\bot$ (negative). To introduce this ternary connective there are two possible rules, obtained combining a Tensor-rule with one of the two possible Plus-rules:

$$\frac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{p^\bot, r^\bot\}) \qquad \text{or} \qquad \frac{\vdash q^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{q^\bot, r^\bot\})$$

Observe that *each rule* is labelled by a pair: (i) the focus and (ii) the subformulas which appear in the premises.

The dual formula $(p\&q)\parr r$ has a negative connective whose rule combines *the* Par-rule with *the* With-rule:

$$\frac{\vdash p, r, \Lambda \quad \vdash q, r, \Lambda}{\vdash (p\&q)\parr r, \Lambda}\ \{(f^\bot, \{p, q\}), (f^\bot, \{p, r\})\}$$

The rule is labelled by a set of pair: a pair (focus, set of subformulas) for each premise. This makes sense if we understand that each negative premise corresponds to an additive slice. Actually, we rather use the label $(f^\bot, \{\{p, r\}, \{q, r\}\})$ which is short for the one above.

To each positive rule corresponds a premise of the negative rule. During cut-elimination, the positive rule will select a negative premise. That is to say, the positive rule will select one slice. For example, the redex:

$$\frac{\dfrac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta}{\vdash (p^\bot \oplus q^\bot) \otimes r^\bot, \Gamma, \Delta}\ (f, \{p^\bot, r^\bot\}) \quad \dfrac{\vdash p, r, \Lambda \quad \vdash q, r, \Lambda}{\vdash (p\&q)\parr r, \Lambda}\ \{(f^\bot, \{p, r\}), (f^\bot, \{q, r\})\}}{\vdash \Gamma, \Delta, \Lambda}$$

reduces to:

$$\frac{\vdash p^\bot, \Gamma \quad \vdash r^\bot, \Delta \quad \vdash p, r, \Lambda}{\vdash \Gamma, \Delta, \Lambda}$$

Note 2. We write $((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$ for $(\downarrow (\uparrow p_1 \parr \uparrow p_2) \oplus \downarrow q^\bot) \otimes \downarrow r^\bot$. A positive rule can only be applied on positive formulas. Therefore we cannot directly form $((p_1 \parr p_2) \oplus q^\bot) \otimes r^\bot$; we need to use an operator which changes the polarity, the *Shift:* $\downarrow$. If N is negative, $\downarrow N$ is positive. However, we are going to deal with $\downarrow$ implicitly.

Locations. Each formula to be decomposed receives an address. Let f of the previous example have address ξ, and p, q, r be respectively located in $\xi 1, \xi 2, \xi 3$. The positive rules in the previous example can be rewritten as

Positive rules

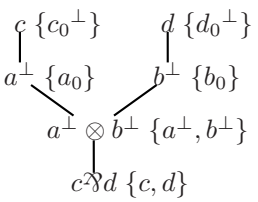
$$\frac{\xi 1 \vdash \Gamma \quad \xi 2 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}\ (\xi, \{1, 3\}) \qquad \frac{\xi 2 \vdash \Gamma \quad \xi 3 \vdash \Delta}{\vdash \xi, \Gamma, \Delta}\ (\xi, \{2, 3\})$$

Sequents of addresses are expressions of the form $\Xi \vdash \Lambda$ where: Ξ, Λ are finite sets of addresses, pairwise disjoint, and Ξ contains at most one address. Notice that negative formulas are written on the left-hand side. There is at most one negative formula.

Designs: Getting an Intuition. Designs capture the geometrical structure of sequent calculus derivations. To start from the sequent calculus is the simplest way to introduce designs. Consider the following derivation, where $a^\perp, b^\perp, c, d$ denote formulas which respectively decompose as $a_0, b_0, {c_0}^\perp, {d_0}^\perp$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash a_0, {c_0}^\perp}{\vdash a_0, c}\ (c, \{{c_0}^\perp\})}{\vdash a^\perp, c}\ \{(a, \{a_0\})\} \quad \dfrac{\dfrac{\vdash b_0, {d_0}^\perp}{\vdash b_0, d}\ (d, \{{d_0}^\perp\})}{\vdash b^\perp, d}\ \{(b^\perp, \{b_0\})\}}{\vdash c, d, a^\perp \otimes b^\perp}\ (\downarrow a^\perp \otimes b^\perp, \{\ a^\perp, b^\perp\})}{\vdash c \parr d, a^\perp \otimes b^\perp}\ \{(c \parr d, \{c, d\})\}$$

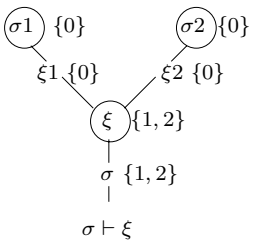
Let us forget everything in the sequent derivation, but the labels. The derivation above becomes the following tree of labels, which is in fact a (typed) design:



This formalism is more concise than the original sequent proof, but still carries all relevant information. To retrieve the sequent calculus counterpart is immediate. Rules and active formulae are explicitly given. Moreover we can *retrieve the context dynamically*. For example, when we apply the Tensor rule, we know that the context of $a^\perp \otimes b^\perp$ is c, d, because they are used above. After the decomposition of $a^\perp \otimes b^\perp$, we know that c is in the context of $a^\perp$ because it is used after $a^\perp$, and that d is in the context of $b^\perp$, because it appears after it.

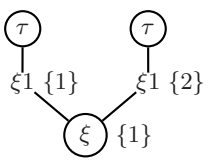
Since the sequent calculus is focalized, the proof construction follows the pattern: " (i) Decompose any negative formula; (ii)choose a positive focus, decompose it in its negative components, decompose the negatives; repeat (ii)." This is mirrored in the tree. In particular, polarities alternate, and a positive focus is always followed by its immediate sub-addresses.
Observe that the tree only branches on positive nodes. As a mnemonic aid, we represent the *positive nodes as vertices and the negative nodes as edges.*

To complete the process, let us now abstract from the type annotation (the formulas), writing only the addresses. In the example above, we locate $a^\perp \otimes b^\perp$ at the address ξ; for its subformulas a and b we choose the sub-addresses $\xi 1$ and $\xi 2$. Finally we locate a_0 in $\xi 10$ and b_0 in $\xi 20$. In the same way, we locate $c \parr d$ at the address σ and so on for its subformulas. Our design becomes:

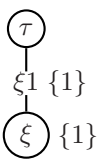


The pair (ξ, I) is called an *action*. As we have seen, ξ is an address (the address of a formula) and I a set of natural numbers, the relative addresses of the immediate subformulas we are using. ξ is called *focus* of the action. The daimon † is also an action.

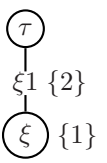
Where are the additives. The key to understand the &-rule in terms of design is to remember that the &-rule is a set (the super-imposition) of two actions on the (*same address*). Let us revisit our example of slices. Let us locate c in the address τ, $(a\&b) \oplus d$ in the address ξ, $(a\&b)$ in $\xi 1$, a in $\xi 11$, and b in $\xi 12$. The derivation of our previous example corresponds to the following design



whose two slices are



and



The actions $(\xi 1\{1\})$ and $(\xi 1\{2\})$ should be be thought of as unary &; the usual binary rule is recovered as the set of actions on the address $\xi 1$.

Design: Syntax. A *design* is given by *a base* and *a tree of actions* with some properties which we recall below.

A branch in the tree is called a *chronicle*. We think of the tree as oriented from the root upwards. If the action κ_1 is *before* κ_2, we write $\kappa_1 < \kappa_2$. We write $\kappa_1 <_1 \kappa_2$ if κ_2 immediately follows κ_1.

The base. A base is a sequent of addresses, which corresponds to the "initial" sequent of the derivation, the conclusion of the proof, the specification of the process. The base: (i) gives the addresses of the formulas we are going to decompose; (ii) induces a polarization of all the addresses (all the actions) in the design. According to its position, each address in the base has a *polarity*: positive (right hand side) or negative (l.h.s.). As in a synthetic connective the polarity of subformulas alternates at each layer, if ξ belongs to the base and is positive, ξi is negative, ξij is positive, and so on.

The tree of actions. A design $\mathfrak{D}$ of base $\Xi \vdash \Delta$ is: (i) a non empty tree of actions if the base is positive (there is only one first action), (ii) a (possibly empty)

forest of actions on the same initial focus if the base is negative (we can have a set of first actions on the same address).

Such a tree of actions satisfies the following conditions:
Root. The root (possibly roots in case of a negative base) focuses on an address of the base. If there is a negative address, that will be decomposed first.
Polarity. Polarities alternate.
Branching. The tree only branches on positive actions.
Focalization. The addresses used as focuses after a positive action (ξ, I) are immediate sub-addresses of ξ. Observe that $\dagger$ can only appear as a leaf, because it has no sub-addresses.
Sub-addresses. An address is either chosen in the base or has been created before (always $\xi < \xi i$). This simply corresponds to the *subformula property.*
Leaves. All maximal actions are positives.
Propagation (linearity). In all slices of $\mathfrak{D}$ each focus only appears once, where given a tree of action, a *slice* is a subtree such that the addresses $\xi i, i \in I$ after a positive action (ξ, I) are all distinct. This condition means that an addresses can be duplicated (reused) only in the context of a &.

Normalization. In Ludics there is no cut rule; a *cut* is a coincidence of addresses of opposite polarity in the base of two designs. A *cut-net* is a finite set $\mathfrak{R} = \{\mathfrak{D}_1, ...\mathfrak{D}_n\}$ of designs of respective bases $\Xi_i \vdash \Lambda_i$ The graph whose vertices are the $\Xi_i \vdash \Lambda_i$ and whose edges are the cuts is connected and acyclic. If we orient the edges from positive to negative, the design corresponding to the starting vertex is the *main design* of the cut-net. The uncut loci form a base, the *base of the cut-net.* A cut-net whose base is the empty sequent is said to be *closed.*

We call an address *closed* if it is a sub-address of a cut, *open* otherwise. The definition extends to actions.

The *normal form* of a cut net $\mathfrak{R}$ is indicated by $[\![\mathfrak{R}]\!]$. The normalization procedure on sequents of addresses mimics normalization in sequent calculus. In the next section, we will define normalization directly on the trees of actions.

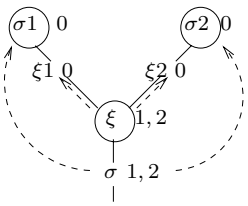
2 Slices as Proof-Nets

As a design, a slice is simply a tree of actions, where each address only appears once. Each action is uniquely determined by its focus. For this reason, when working with slices we often identify an action $\kappa = (\sigma, I)$ with its focus σ.

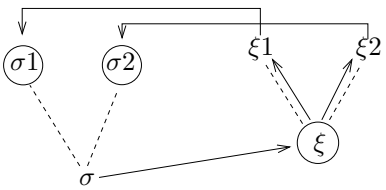
In a slice we are given *two orders,* corresponding to two kinds of information on the actions:

- the succession in time, recorded by the chronicles (the chronicles tree);
- the succession in space, corresponding to the relation of being sub-address (the prefix tree, which is analogous to a "sub-formula tree").

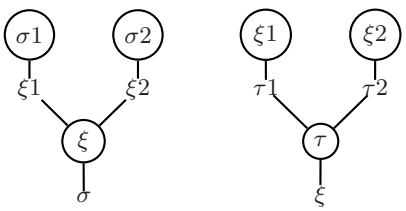
Let us again have a look at our previous example of design. We make explicit the relation of being a sub-address with a dashed arrow connecting σ to $\sigma 1$ and $\sigma 2$, and ξ to its sub-addresses, as follows:



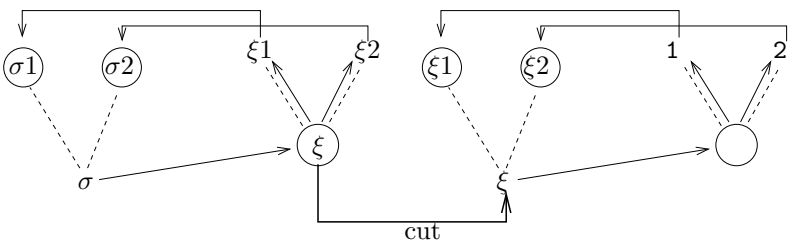
Consider a multiplicative proof-net, where the axioms are possibly "generalized axioms," that is hypothesis of the form $\vdash \Gamma$. Such a proof-net is a sub-formula tree with some extre information on the axiom links. If we emphasize the formula-tree rather than the chronicles-tree, we recognize something similar to a proof-net, added of some information on sequentialization. In particular this extre information allows us to establish the axioms links (generalized axioms, of the form $\xi \vdash \Gamma$) between the last-focused addresses, which are the leaves in the prefix tree. As we see below, in our example $\xi 1$ is connected to $\sigma 1$ and $\xi 2$ to $\sigma 2$.



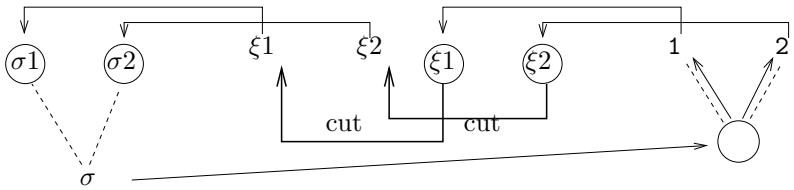
This suggests dealing with normalization as in proof-nets rather than as in sequent calculus. Essentially we mimic proof-nets normalization, as in the following example, where the cut-net



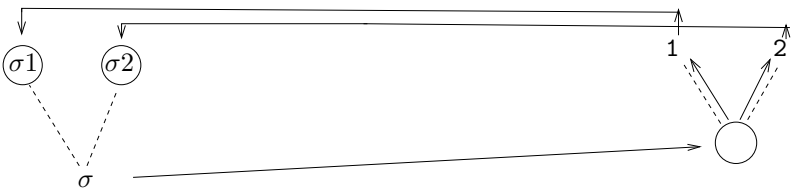
once written as



reduces as follows



and then to



In Ludics the situation is in general slightly more complex than in the above example, because the setting is not typed. Thus for example ξ could correspond on one side to the action $(\xi, \{1, 2, 3\})$ and on the other side to the action $(\xi, \{1, 2\})$, or just not appear at all. Observe however that what we actually do on proof-nets is to connect (or to identify) two nodes with the same label. This can be done on designs. This idea underlies both the normalization as "quotient of orders" described in [6] and the abstract machine we define in the next section.

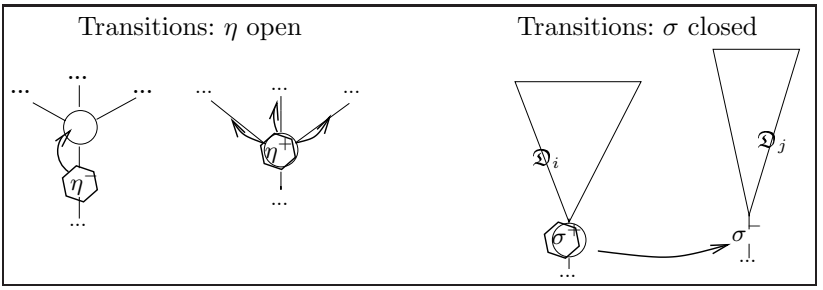
3 Loci Abstract Machine

Normalization of a cut-net $\mathfrak{R}$ can be presented by a token traveling along the net. This is implemented by a machine which we call Loci Abstract Machine (LAM). We first present a minimal version, which we indicate by $\mathbf{LAM_0}$, working on slices. Since in a slice there is no "additive duplication," normalization of slices is simpler than normalization of general designs. However, one could always work "by slices:" normalize slice by slice, and then put them together. In Section 5 we will generalize it.

The figure below presents the machine graphically. The key point is that when the same address σ appears in distinct designs, we can move from one design to the other, passing from σ^+ to σ^-. Observe that the token is always going *upwards*. While the token moves around, it draws a path on the cut-net. Each path will represent a chronicle of the normal form $[\![\mathfrak{R}]\!]$, as soon as we hide the closed actions (internal communication).

Initialization: The token enters the net on the root of the main design ($Main$).

Transitions: When the token is on an *open action,* κ it follows the chronicles order, moving *upwards* to the actions which immediately follow κ in the slice. When the token enters a *(positive) closed action*, it exits at the corresponding negative action (then changing of design).



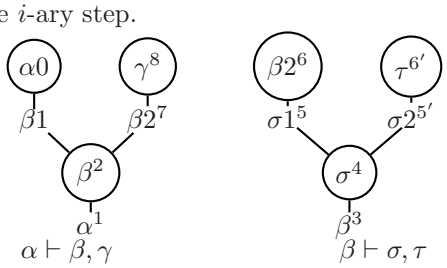
Below we give a formal definition of the machine. At the end of this section we will give an example of execution.

A *token* is given by a pair (s, κ). The action κ represents the current position of the token, while s is a list of actions, which records the path followed by the token. Each time the token enters an open action, that action is attached to the list. The transitions only depend on the position; the sequence of actions is only recorded to produce the normal form. We denote the empty sequence by ϵ.

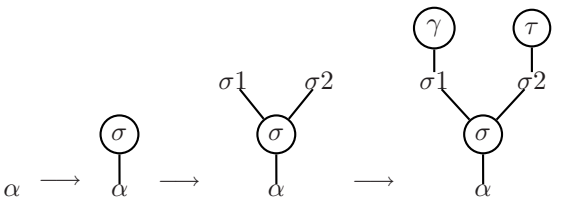
Let T be the set of all positions reached by the tokens.
Initialization. If $Main \neq \emptyset$ then $T := \{(\epsilon, \kappa)\}$, where κ is the root of the main design $(Main)$.
Transitions.
(i) Let η be an *open* action (recall that open means not cut).
If $(s, \eta) \in T$ then $T := T \cup (s\eta, \kappa)$ for all $\kappa >_1 \eta$.
(ii) Let σ be a *closed* action (the focus is sub-address of a cut).
If $(s, \sigma) \in T$ and $\sigma^- \in \mathfrak{R}$ then $T := T \cup (s, \kappa)$ for $\kappa >_1 \sigma^-$.
Result. $[\![\mathfrak{R}]\!] = \{c : c \sqsubseteq s^+ \text{ and } (s^+, \kappa) \in T\}$, where s^+ is a sequence whose last action is positive.

Comments. When we enter a closed action σ, it is necessarily positive. We proceed to the corresponding negative action (then changing of design). If σ^- exists we move to the (unique) action which follows σ^-. If not, there is no way to extend s, and we are finished with that token. Notice that in this case s terminates on a negative action. Each maximal positive path describes a maximal chronicle of the normal form.

Example of Execution. Consider the following cut-net, where the bases are respectively $\alpha \vdash \beta, \gamma$ and $\beta \vdash \sigma, \tau$. We decorate it with the path followed by the tokens: i indicate the i-ary step.



On σ the computation splits in two flows. There are two normalization paths, which are: $\alpha, \beta, \sigma, \sigma 1, \beta 2, \gamma$ and $\alpha, \beta, \sigma, \sigma 2, \tau$. As the token travels along, we only record the open actions and the normal form grows as follows:



From here it is immediate to recover the sequent calculus presentation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \sigma 10, \gamma}}{\sigma 1 \vdash \gamma} \quad \dfrac{\dfrac{}{\vdash \sigma 20, \tau}}{\sigma 2 \vdash \tau}}{\vdash \alpha 0, \sigma, \gamma, \tau}}{\alpha \vdash \sigma, \gamma, \tau}$$

Designs vs. sequent calculus normalization We could have presented the same cut-net with the syntax of sequent calculus.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \beta 10, \alpha 0}\,\alpha 0}{\beta 1 \vdash \alpha 0}\,\beta 1 \quad \dfrac{\dfrac{}{\vdash \beta 20, \gamma}\,\gamma}{\beta 2 \vdash \gamma}\,\beta 2}{\vdash \alpha 0, \beta, \gamma}\,\beta, \{1,2\}}{\alpha \vdash \beta, \gamma}\,\alpha \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \sigma 10, \beta 2}\,\beta 2}{\sigma 1 \vdash \beta 2}\,\sigma 1 \quad \dfrac{\dfrac{}{\vdash \sigma 20, \tau}\,\tau}{\sigma 2 \vdash \tau}\,\sigma 2}{\vdash \beta 1, \beta 2, \sigma, \tau}\,\sigma}{\beta \vdash \sigma, \tau}\,\beta$$

The reader is free to normalize on the sequent calculus, to check that the resulting normal form is actually the one associated to the result on designs.

4 Disputes and Chronicles Extraction

In the previous section we presented normalization by a token traveling around the cut-net. The token draws a path, which is a chronicle of the normal form, as soon as we ignore the closed actions. To calculate the normal form we only need to record the open actions. However, the *normal form is not necessarily the most interesting thing in normalization.* In Ludics, the most important case of cut-net is by far the closed one. If normalization converges, the normal form reserves no surprise: it is $\dfrac{}{\vdash}\,\dagger$. What is interesting is the interaction itself, that is the sequence of actions that have actually been visited (used) during the normalization.

We call *normalization path* the sequence of actions visited during the normalization of a cut-net. We indicate by $Paths(\mathfrak{R})$ the collection of all normalization paths on $\mathfrak{R}$. We call *dispute* the sequence of actions visited during the normalization of a *closed* net. If the net is $\{\mathfrak{D}, \mathfrak{E}\}$, we indicate the dispute by $[\mathfrak{D} \rightleftharpoons \mathfrak{E}]$.

Remark 1. It is immediate to modify the abstract machine given in the previous section into a machine that keeps track of all the visited actions.

Views. In a design, action with the same focus may appear several times, because of the use of n-ary negative rules (additives!). Each occurrence of an action κ is identified by the minimal chronicle $\mathfrak{c}\kappa$ in which it appears. We can see this as the position of that κ. As we shall see, for each action κ used in the normalization, the normalization path allows us to retrieve its position.

The key is to invert the process of constructing the path. This is in fact a well-known operation of HO-Nickau games [7], [8] the *view* operation. The notion of view is relative to a player, or to a parity in our setting. Let us recall some technical notions we need.

The space of addresses, and thus of actions, is split between two players: Even and Odd, according to the length of the address. A *base* has the same parity (even or odd) as the addresses on its positive side (all addresses on the right-hand side –positive– have the same parity, opposite to that of the address on the left-hand side). The empty base $\vdash$ is defined positive. A *design* is even or odd according to its base. An *action* is even or odd according to its focus. When an action (or a base, or a design) *has parity Even (Odd)* we also say that it *belongs to Even (Odd).*

The polarity (positive or negative) of each action in a design is *relative* to the parity (even, odd) of the design. In a design of base X, each X action is positive. We use the variable X, for X either Even or Odd, and $\overline{X}$ for the dual. To explicit if an action κ is Even or Odd, positive or negative we use the notation: $\kappa^E, \kappa^O, \kappa^+, \kappa^-$.

Any *cut-net* $\{D_i\}$ splits into two components: the collection of even designs $(\mathfrak{D}_i^E)$, and the collection of odd designs $(\mathfrak{D}_j^O)$. Hence we can write $\mathfrak{R}$ as $\{(\mathfrak{D}_i^E), (\mathfrak{D}_j^O)\}$. We extend the notation for disputes to this case, writing $[(\mathfrak{D}_i^E) \rightleftharpoons (\mathfrak{D}_j^O)]$.

Let us define the function *view* on $p \in Paths(\mathfrak{R})$. Observe that each action κ in p has a parity (Even/Odd). If κ belongs to X, it is X-positive and $\overline{X}$-negative. Given an action $(\xi, I) \in p$, we say that it is *initial* if ξ is not a sub-address of any other address in p (ξ belongs to the base of one of the designs in the cut-net).

Definition 1 (Views). *Let $p \in Paths(\mathfrak{R})$ and X be either Even or Odd. Its view $\ulcorner p \urcorner^X$ of p is defined as follows (positive and negative is relative to X).*

- $\ulcorner \epsilon \urcorner = \epsilon$;
- $\ulcorner s\kappa^+ \urcorner = \ulcorner s^- \urcorner \kappa$;
- $\ulcorner s\kappa^- \urcorner = \kappa$ *if* κ *is initial;*
- $\ulcorner s\kappa' t\kappa^- \urcorner = \ulcorner s \urcorner \kappa' \kappa$ *if* $\kappa = (\xi i, K)$ *and* $\kappa' = (\xi, I)$.

We denote Odd view *by* $\ulcorner q \urcorner^O$ *and the* Even view *by* $\ulcorner q \urcorner^E$. *It is convenient to adopt the following convention: by* $\ulcorner q\kappa^+ \urcorner$ *we mean the view of the player for which* κ *is positive. If* κ *belongs to* X, *then* $\ulcorner q\kappa^+ \urcorner = \ulcorner q\kappa \urcorner^X$ *and* $\ulcorner q\kappa^- \urcorner = \ulcorner q\kappa \urcorner^{\overline{X}}$

Notice that the notion of view applies to any $p = [(\mathfrak{D}_i^E) \rightleftharpoons (\mathfrak{D}_j^O)]$.

Chronicles Extraction. Let $\mathfrak{R}$ be a cut-net whose designs are all slices and $p \in Paths(\mathfrak{R})$. We have that:

Proposition 1. *Let $\mathfrak{R}$ be a cut-net of slices, $p \in Paths(\mathfrak{R})$ and $q\kappa \sqsubseteq p$. If κ appears positive in $\mathfrak{R}$, then the chronicle $\mathfrak{c}\kappa^+ \in \mathfrak{R}$ is given by $\ulcorner q\kappa^+ \urcorner$. If κ appears negative in $\mathfrak{R}$, then the chronicle $\mathfrak{c}\kappa^- \in \mathfrak{R}$ is $\ulcorner q\kappa^- \urcorner$.*

Notice that an open action κ will appear in $\mathfrak{R}$ either positive or negative, never both.

Proof. The proof is by induction on the length of $q\kappa$. Let κ be an *open* action. The action η visited just before κ by normalization is the action that precedes κ in the chronicle. Let $q = q'\eta$ and $\mathfrak{c}\kappa = \mathfrak{c}'\eta\kappa$. By induction, $\ulcorner q'\eta \urcorner = \mathfrak{c}'\eta$. If κ is positive, $\ulcorner q'\eta\kappa^+ \urcorner = \ulcorner q'\eta \urcorner\kappa$. If κ is negative, $\ulcorner q'\eta\kappa^- \urcorner = \ulcorner q \urcorner\eta^+\kappa$, because the focus of κ is sub-address of that of η.

Let κ be *closed*. The positive case is as before. $\mathfrak{c}\kappa^-$ is of the form $\mathfrak{c}(\xi, I)^+(\xi i, J)^-$, where $(\xi, I) < (\xi i, J)^-$ in p. Hence $q'(\xi, I) \sqsubseteq q$, $\ulcorner q\kappa^- \urcorner = \ulcorner q'(\xi, I) \urcorner(\xi i, J)$ and $\ulcorner q'(\xi, I) \urcorner = \mathfrak{c}(\xi, I)$.

Proposition 1 has immediate *consequences* which we develop in the next sections.

5 LAM$_+$: Generalized Version

The normalization procedure given in Section 3 is well defined since in the case of slices there is only one occurrence of any focus. At the same time, it is idealized in the sense that we assume that the machine is able to find the next action by itself, in particular when moving from σ^+ to σ^-. Moreover, it would not be feasible if we were not working by slices: in a general design, the same action may appear several times (additive duplications). However, the sequence of visited actions carries all information needed to retrieve the position of any of its action (Proposition 1). In particular, when we enter a positive action κ^+ we are able to retrieve the chronicle that identifies the negative action κ^- to which we have to move. Assume p is the sequence of actions we have visited so far, and we enter the positive action κ^+. We then move to the action κ^- identified by the chronicle $\mathfrak{d} = \ulcorner p\kappa^- \urcorner$.

We can therefore define the following general machine to normalize arbitrary designs. Let $Paths(\mathfrak{R})$ be the set of all paths described on $\mathfrak{R}$. We have that:

- $\epsilon \in Paths(\mathfrak{R})$
- Let η be open and of polarity $x \in \{+, -\}$.
  If $p\eta \in Paths(\mathfrak{R})$ and $\ulcorner p\eta^x \urcorner\kappa \in \mathfrak{R}$ then $p\eta\kappa \in Paths(\mathfrak{R})$.
- Let σ be a closed action.
  If $p\sigma \in Paths(\mathfrak{R})$ and $\ulcorner p\sigma^- \urcorner\kappa \in \mathfrak{R}$ then $p\sigma\kappa \in \mathfrak{R}$.

Let $Norm(\mathfrak{R}) = \{hide(p) : p \in Paths(\mathfrak{R})\}$, where $hide(p)$ is p where we have deleted (hidden) all closed actions. We have that $[\![\mathfrak{R}]\!] = \{s, s \sqsubseteq q^+, q^+ \in Norm(\mathfrak{R})\}$, where q^+ is a sequence whose last action is positive.

6 Calculating the Pull-Back

The normalization of a closed cut-net produces a unique maximal path, the dispute. If we are given a dispute, we can calculate the minimal cut-net that produces it. We indicate this operation by $Pull(p)$.

Let $p = [\mathfrak{D} \rightleftharpoons \mathfrak{E}]$. $Pull^E(p)$ is defined as $\{\ulcorner q \urcorner^E : q \sqsubseteq r^+ \sqsubseteq p, q \neq \epsilon\}$. $Pull^O(p)$ is defined symmetrically. $Pull(p) = \{Pull^E, Pull^O\}$.

It is immediate, and it is important to notice, that $Pull(p)$ only depends on p. Thus for any cut-net $\mathfrak{R}$, the normalization produces the dispute p iff $Pull(p) \subseteq \mathfrak{R}$. As a consequence

Proposition 2. *Given a cut-net $\mathfrak{R}$ whose normalization produces the dispute p, $Pull(p)$ gives the the minimal $\mathfrak{R}_0 \subseteq \mathfrak{R}$ which produces p.*

In [6] $\mathfrak{R}_0$ is called the *pull-back* of p along $\mathfrak{R}$.

It is easy to extend the definition above to any closed cut net $\mathfrak{R}$. In such a case $Pull^E(p)$ and $Pull^O(p)$ are a set of chronicle that we can split into a collection of designs.

7 Computing a Counter-Design

Let us present another way to use the same machine "the other way round:" given a slice and a path on it, we calculate a counterdesign realizing the path.

A *path* p on a slice $\mathfrak{S}$ is a sequence of actions such that for any $p' \sqsubseteq p$ the region of $\mathfrak{S}$ covered by p' contains the root and is a tree. Now suppose we freely draw such a path on a slice. Is there a counter-design which realizes that path? Can we produce it? If we know that the counter-design exists, we can calculate the pull-back. Otherwise, we can build the counter-design "by hand" as follows.

Procedure. Assume we have a slice $\mathfrak{S}$ and a path $p = \kappa_0, ...\kappa_n$ on it. Our aim is to build a counter-design $\mathfrak{T}$ such that $[\mathfrak{S} \rightleftharpoons \mathfrak{T}] = p$. (We focus our discussion on the case where $\mathfrak{S}$ has base $\vdash \xi$ or $\xi \vdash$; the case of base $\Xi \vdash \Lambda$ is similar, but we have a family $\mathfrak{T}_i$ of counter-designs).

The *base* of $\mathfrak{T}$ is determined. To *build* $\mathfrak{T}$, we progressively place the actions of p to form a tree. The polarity of the actions in $\mathfrak{T}$ is opposite to that in $\mathfrak{S}$, as is the polarity of the base. If κ_i is negative in $\mathfrak{T}$, there is no ambiguity on where to place it: either it is the root, or it is of the form ξi, and we place it just after ξ (which is positive). If κ_{i+1} is positive in $\mathfrak{T}$, we need to place it just after κ_i (which is negative in $\mathfrak{T}$). In fact once the normalization is on a positive action κ_i^+ in $\mathfrak{S}$, it moves to the negative action κ_i^- in $\mathfrak{T}$, and then κ_{i+1}.

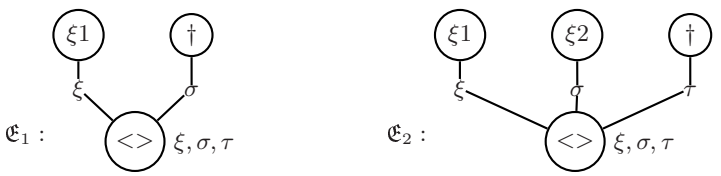
At any stage in $\mathfrak{T}$ there is at most one maximal branch terminating with a negative action. If κ_n, the last action of p, is *negative* in $\mathfrak{T}$, we complete $\mathfrak{T}$ with a daimon (†) after κ_n. By construction, the normalization applied to $\{\mathfrak{S}, \mathfrak{T}\}$ produces p. We need to check that the tree we build is actually a design. The only property that is not guaranteed by construction is that of *sub-address* on positive focus.

8 An Application: What Can Be Observed Interactively?

The program of Ludics is that of an interactive approach to logic. Ideally, we should be able to express and to test interactively the properties we ask to designs. Therefore what we know of a design is what we can see testing it against a counter-design. What part of a design can be visited during normalization? Normalization is always carried out in a single slice. Given a slice, can we build a counter-design which is able to completely explore it? Even if we only consider finite slices, the answer is no, as shown by the following example:

As we have sketched on the right hand side, such a design corresponds to a purely multiplicative structure. In fact we can easily type it, for example letting $F(\xi) = F(\xi 1) \otimes F(\xi 2)$, $F(<>) = F(\xi) \parr F(\sigma) \parr F(\tau)$, where by $F(*)$ we indicate the formula associated to the address $*$.

Let us build a counter-design to explore this slice. The path will start with $<>$, move to ξ, and then choose one of the branches, going either to $\xi 1$ or $\xi 2$. The two choices are symmetrical, so let us take $\xi 1$. At σ we are forced to stop, because there is no way to move to the other branch. The counter-design we have built is the following one ($\mathfrak{E}_1$).



The corresponding path is $<>, \xi, \xi 1, \sigma$, while the path we would like to have is $<>, \xi, \xi 1, \sigma, \xi 2, \tau$. $\mathfrak{E}_2$ (above) is the tree of actions that would realize this path. However, it is not a design, because it does not satisfy the sub-address condition ($\xi \not< \xi 2$).

An immediate consequence is that we *cannot interactively detect* the use of *weakening*, even in a slice. Consider again the example above, now assuming that the root is the action $(\xi, \{\xi, \sigma, \tau, \lambda\})$. The root creates an address, λ, which is never used. However, we cannot interactively detect that λ is weakened. Either we explore the left branch, or the right one. In the first case we see that σ is used. The other addresses, τ and λ, are possibly used after $\xi 2$. In the second case we see that τ is used, σ and λ being possibly used after $\xi 1$.

9 Related and Further Work

Interaction is central in Ludics, so it is important to have a theory telling what can be interactively recognized, and it is rather natural to take interaction traces as primitive and study designs from them.

In this paper we developed a concrete approach to designs, which gives us effective tools to address issues such as the following ones (see [4]).

(i) Study geometrical properties of the normalization paths, in the style of Geometry of Interaction.

(ii) Rebuild a slice out of a prefix tree of addresses.

(iii) Characterize the (parts of) designs that can be observed interactively: the designs that can be explored in a test (in a single run of normalization) represent the primitive units of observability.

(iv) Present designs as the collections of their disputes, which allows then establish a bridge with Games Semantics [5].

Related Work. Our normalization on designs (rather than on the sequent calculus) is analogous to the order quotient defined in [6], though it was developed independently. Our approach is more local, hence easier to use for actual computations. Actually, what the machine does is to calculate the balanced slice. On the other hand, Girard's theory provides a synthetic view, which better suits the development of general results.

The notion of design is very close to that of abstract Böhm tree introduced by Curien as a generalization of lambda terms and as a concrete syntax for games. The way we proceed closely relates our work to the abstract machines studied by Curien and Herbelin in [3]. Our generalized LAM is actually an instance of the View abstract machine, introduced by Coquand in [2].

References

[1] J.-M. Andreoli and R. Pareschi. Linear objects: logical processes with built-in inheritance. *New Generation Computing*, 9(3-4):445–473, 1991. 428

[2] T. Coquand. A semantics of evidence for classical arithmetic. *Journal of Symbolic Logic*, (60), 1995. 441

[3] P.-L. Curien and H. Herbelin. Computing with abstract bohm trees. In *Third Fuji International Conference on Functional and Logic Programming*, Kyoto, 1998. Word Scientific. 441

[4] C. Faggian. *On the Dynamics of Ludics. A Study of Interaction.* PhD thesis, Université Aix-Marseille II, 2002. 441

[5] C. Faggian and M. Hyland. Designs, disputes and strategies. In *CSL 2002 (this volume)*, LNCS. Springer, 2002. 441

[6] J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science*, 2001. 427, 434, 439, 441

[7] M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 2000. 427, 437

[8] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, LNCS. Springer, 1994. 427, 437

Designs, Disputes and Strategies

Claudia Faggian and Martin Hyland

DPMMS – University of Cambridge

Abstract. Ludics has been proposed by Girard as an abstract general approach to proof theory. We explain how its basic notions correspond to those of the "innocent strategy" appraoch to Games Semantics, and thus establish a clear connection between the two subjects.

1 Introduction

Interaction has become an important notion both in theoretical computer science and in proof theory. From the computational point of view, when running an application the result of computation (if there is any) is not necessarily the most interesting aspect. The dynamics, the process of computation itself may play the central role. Moreover, composition of programs is in general a rich two-directions process, which entails communication and exchanges between the components. A paradigm of *computation as interaction* underlies several models of computation. This paradigm is particularly significant today, since for reactive systems, the process of interaction rather than any final result is what is at issue.

Important progress in logic has also lead to interactive and dynamical models. Major examples are Geometry of Interaction and Games Semantics. The Geometry of Interaction [5], which arose from Linear Logic, interprets normalization (computation) as a flow of information circulating around a net. Games Semantics interprets computation as a dialog between two parties, the program (player) and the environment (opponent), each one following its own "strategy". Games Semantics (see [2] for a survey) has been both an important development in logic, and a successful approach to the semantics of programming language. The strength of these models is to capture the dynamical aspects of computation, so as to take into account both qualitative (correctness) and quantitative (efficiency) aspects of programming languages. Ludics, recently introduced by Girard in [6], is a further step in this development, the fundamental notion in the theory being that of interaction.

The basic objects of Ludics are *designs,* which are both (i) an abstraction of formal proofs and (ii) a concretion of their semantical interpretation. A design can be described as the skeleton of a sequent calculus derivation, where we do not manipulate formulas, but their locations (the addresses where the formulas are stored). A design can also be presented in a very natural way as the collection of its possible interactions. Our paper focuses on this presentation . An advantage of the approach we follow is to establish a bridge with the notions of Game Semantics, in particular with HON Games [7], [9]. In fact, we are going to make precise the following correspondences:

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 442–457, 2002.
© Springer-Verlag Berlin Heidelberg 2002

actions – moves
disputes – plays
chronicles – views
designs – innocent strategies

The crucial correspondence is

"view - chronicle - sequent calculus branch."

(In what follows one should keep in mind the concrete interpretation of a chronicle as a branch in a sequent calculus derivation, a design being the "skeleton" of a sequent calculus derivation.)

The correspondence view-chronicle is the key to translating between Ludics and Games Semantics settings. We expect to be able to transfer experiences and techniques between the two settings.

2 Ludics: Designs

2.1 The Universe of Proofs

The program of Ludics is to overcome the distinction between syntax (the formal system) on one side and semantics (its interpretation) on the other side. Rather then having two separate worlds, proofs are interpreted via proofs. To determine and test properties, a *proof* of A should be tested with *proofs* of $A^{\perp}$. Ludics provides a setting in which proofs of A interact with proofs of $A^{\perp}$; to this end, it generalizes the notion of proof.

A proof should be thought in the sense of "proof search" or "proof construction": we start from the conclusion, and guess a last rule, then the rule above. What if we cannot apply any rule? A new rule is introduced, called daimon:

$$\frac{}{\vdash \Gamma}\ \dagger$$

Such a rule allow us to assume any conclusion, without providing a justification.

The syntax of proofs is not the sequent calculus, but a more abstract formalism, close to Böhm trees, and called "design". The proofs do not manipulate formulas, but addresses. These are sequences of natural number, which can be thought of as the address in the memory where the formula is stored.

2.2 Designs

Let us first give an intuition of what is a design. This should be enough to follow the rest of the paper. At the end of the section we recall the formal definitions. We will not really enter in the details of the logical calculus associated to designs, which is a focalized version of second order multiplicative-additive Linear Logic ($\mathbf{M}ALL_2$).

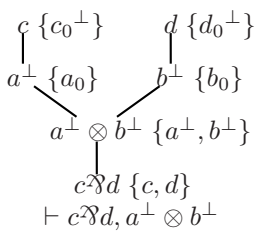
Designs capture the geometrical structure of sequent calculus derivations. The simplest way to introduce designs is to start from the sequent calculus. Let

us consider the following derivation, where the rules are labelled by the active formula and the subformulas which appear in the premises[1]: for example, $\oplus_L$ would be labelled as $(a \oplus b, \{a\})$.

$$\dfrac{\dfrac{\dfrac{\vdash a_0, {c_0}^\perp}{\vdash a_0, c}\,(c, \{{c_0}^\perp\})}{\vdash a^\perp, c}\,(a, \{a_0\}) \qquad \dfrac{\dfrac{\vdash b_0, {d_0}^\perp}{\vdash b_0, d}\,(d, \{{d_0}^\perp\})}{\vdash b^\perp, d}\,(b^\perp, \{b_0\})}{\dfrac{\vdash c, d, a^\perp \otimes b^\perp}{\vdash c \parr d, a^\perp \otimes b^\perp}\,(c \parr d, \{c, d\})}\,(a^\perp \otimes b^\perp, \{\, a^\perp, b^\perp\})$$

$a^\perp, b^\perp, c, d$ are formulas that respectively decompose into $a_0, b_0, {c_0}^\perp, {d_0}^\perp$.

Let us forget everything in the sequent derivation, but the labels. The derivation above becomes the following tree of labels, which is in fact a (typed) design:



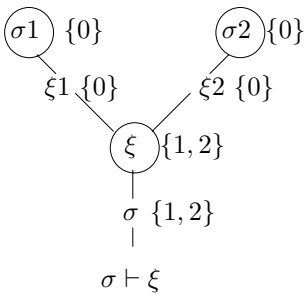
This formalism is more concise than the original sequent proof, but still carries all relevant information to retrieve its sequent calculus counter-part.

What makes this formalism possible is *focalization*. Multiplicative and additive connectives of Linear Logic (MALL) split into two families: positives $(\otimes, \oplus, 1, 0)$ and negatives $(\parr, \&, \perp, \top)$. A cluster of operations of the same polarity can be decomposed in a single step. Such a cluster can be written as a single connective, which is called a *synthetic connective*. For example the formula $(P^\perp \oplus Q^\perp) \otimes R^\perp$ has as immediate subformulas $P^\perp, Q^\perp, R^\perp$, to which we applied the connective $(- \oplus -) \otimes -$

As a consequence, in a derivation positive and negative synthetic connectives alternate at each step.

To complete the process, let us now abstract from the type annotation (the formulas), writing only the addresses. In the example above, we locate $a^\perp \otimes b^\perp$ at the address ξ; for its subformulas a and b we choose the sub-addresses $\xi 1$ and $\xi 2$. Finally we locate a_0 in $\xi 10$ and b_0 in $\xi 20$. In the same way, we locate $c \parr d$ at the address σ and so on for its subformulas. Our design becomes:

[1] In first approximation, we slightly simplify the labels.



where we have circled the address of positive formulas (we will give more detailed on the polarity –positive or negative– of the addresses in Section 2.3).

The pair (ξ, I) is called an *action*. ξ is an address (a list of natural numbers, intended as the address of the formula) and $I \in \mathcal{P}_f(\mathbb{N})$ is a finite set of natural numbers, the relative addresses of the immediate subformulas we are considering. ξ is called *focus* of the action.

$\dagger$ is also an action.

A *design* is given by:

a base, which is a sequent giving the conclusion of the proof (the specification of the process) and

a tree of actions with some properties that we recall in Section 2.3. A branch in the tree is called a *chronicle*. If κ_1 is before κ_2 we write $\kappa_1 < \kappa_2$.

Additives. The example we have used is simple, in that we have used a multiplicative proof, where each formula (each address) only appears once. What about the "additives"? Informally speaking, an &-rule can be seen as the superimposition of two unary rules: $(a\&b, a)$ and $(a\&b, b)$. Given a derivation, if for any &-rule we select one of the premises, we obtain a derivation (where all &-rules are unary). This is called a slice. For example, the derivation

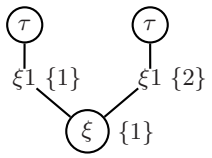
$$\dfrac{\dfrac{\dfrac{\cdots}{\vdash a, c}\ (a, ...) \quad \dfrac{\cdots}{\vdash b, c}\ (b, ...)}{\vdash a\&b, c}\ (a\&b, \{a\}), (a\&b, \{b\})}{\vdash (a\&b) \oplus d, c}\ ((a\&b) \oplus d, \{a\&b\})$$

can be decomposed into two slices:

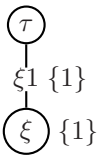
$$\dfrac{\dfrac{\dfrac{\cdots}{\vdash a, c}\ (a, ...)}{\vdash a\&b, c}\ (a\&b, \{a\})}{\vdash (a\&b) \oplus d, c}\ ((a\&b) \oplus d, \{a\&b\}) \quad \text{and} \quad \dfrac{\dfrac{\dfrac{\cdots}{\vdash b, c}\ (b, ...)}{\vdash a\&b, c}\ (a\&b, \{b\})}{\vdash (a\&b) \oplus d, c}\ ((a\&b) \oplus d, \{a\&b\})$$

Therefore, the &-rule is a set (the super-imposition) of two actions on the *same address*.

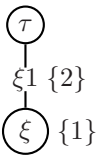
This is the key to understand the &-rule in terms of designs. Taking again the above examples, let us locate c in the address τ, $(a\&b) \oplus d$ in the address ξ, $(a\&b)$ in $\xi 1$, a in $\xi 11$, and b in $\xi 12$. The derivation of our previous example corresponds to the following design



whose two slices are



and



The actions ($\xi 1\{1\}$) and ($\xi 1\{2\}$) should be be thought of as unary &, while the usual binary rule is recovered as the set of actions on the address $\xi 1$.

2.3 Designs as Sets of Chronicles

A *design* is given by a base and a tree of actions with some properties that we are going to present. A *base* is a sequent of addresses which correspond to the "initial" sequent of the derivation, the conclusion of the proof. Focalization leads to consider only sequents of the form $\Xi \vdash \Lambda$,where Ξ has at most one element (and Λ is finite). The base (i) gives the addresses of the formulas we are going to decompose, (ii) establishes the polarity of the addresses, (iii) establishes a dependency relation between the addresses.

A sequent has a positive side (right-hand side) and a negative side (left-hand side). According to its position (r.h.s. or l.h.s.), each address in the base has a *polarity: positive or negative.* We have seen that in a synthetic connective the polarity of subformulas alternates at each layer, so if ξ is positive, ξi is negative, ξij is positive. According to its length, we say that an address is even or odd. This is called the *parity* of an address. Relative to the addresses ξ given in the base, sub-addresses of ξ with the same parity as ξ have the same polarity, sub-addresses of ξ with opposite parity have opposite polarity.

Designs are described in [6] as sets of chronicles. The definition is in two steps:

1. definition of *chronicle,* that is a *formal branch* in a focalized sequent calculus derivation,

2. definition of a *coherence condition* making a set of chronicles all belong to the same proof.

Definition 1 (Chronicle). *A chronicle* $\mathfrak{c}$ *of base* $\Xi \vdash \Lambda$ *is a sequence of actions* $\langle \kappa_0, \kappa_1, ...\kappa_n \rangle$ *such that:*

Alternation. *The polarity of* κ_j *is equal to that of the base for* j *even, opposite for* j *odd.*

Daimon. *For* $j < n$, κ_j *is not a daimon.*

Positive focuses. *The focus of a positive action* κ_p *either belongs to the basis or is an address* ξi *generated by* a *previous action:* $\kappa_q = (\xi, I), i \in I$ *and* $\kappa_q < \kappa_p$.

Negative focuses. *The focus of a negative action* κ_p *either belongs to the basis or is an address* ξi *generated by* the *previous action:* $\kappa_{p-1} = (\xi, I), i \in I$

Destruction of Focuses. *Focuses are pairwise distinct.*

Definition 2 (Coherence). *The chronicles* $\mathfrak{c}, \mathfrak{c}'$ *are* coherent *when*

Comparability. *Either one extends the other, or they first differ on negative actions, i.e. if* $\mathfrak{c}_1 = \mathfrak{c}\kappa_1 * \mathfrak{e}_1, \mathfrak{c}_2 = \mathfrak{c} * k_2 * \mathfrak{e}_2$ *with* $\kappa_1 \neq \kappa_2$ *then* κ_1, κ_2 *are negative.*

Propagation. *If* $\mathfrak{c}_1, \mathfrak{c}_2$ *first differ on* κ_1, κ_2 *with distinct focuses, then all ulterior focuses are distinct.*

Definition 3 (Design). *A design* $\mathfrak{D}$ *of base* $\Xi \vdash \Lambda$ *is a set of chronicles of base* $\Xi \vdash \Lambda$ *such that:*

Arborescence. $\mathfrak{D}$ *is closed under restriction.*

Coherence. *The chronicles of* $\mathfrak{D}$ *are pairwise coherent.*

Positivity. *If* $\mathfrak{c} \in \mathfrak{D}$ *has no extension in* $\mathfrak{D}$*, then its last action is positive.*

Totality. *Then* $\mathfrak{D}$ *is non empty.*

One is also interested in the empty design on a positive base, which is called *partial* and indicated by $\vdash \Omega$.

Notice that the above definition admits the empty chronicle, which is more natural in the setting Game Semantics, even though this is not the case in [6].

Cuts and Normalization. A set of designs to be cut together is called a cut-net. A cut between two designs is a coincidence of addresses of opposite polarity in the base of the two designs (one appears on the right-hand side, one on the left-hand side of two distinct bases).

By far, in Ludics the most important case of cut-net is the *closed* case: all addresses are cut. Given a base, its opposite is the base (or family of bases) which allow us to close the net. The opposite of $\vdash \xi$ is $\xi \vdash$; the opposite of $\xi \vdash \lambda_1, \ldots, \lambda_n$ is the family $\vdash \xi, \quad \lambda_1 \vdash, \quad \lambda_n \vdash$.

Given two design $\mathfrak{D}, \mathfrak{E}$ the normal form is indicated as $[\mathfrak{D}, \mathfrak{E}]$. This is a (possibly partial) design. Its base is given by the uncut addresses; if $\mathfrak{D}, \mathfrak{E}$ have opposite base, since all addresses are cut, $\mathfrak{D}, \mathfrak{E}$ has conclusion $\vdash$. The normalization process then builds the tree of actions (the proof) which justifies this conclusion, as the result of the interaction between the cut designs. We start we no data (the empty design): this is our initial *partial* result. If $\mathfrak{D}, \mathfrak{E}$ do "cooperate," eventually we have a rule (an action) to justify the conclusion. In this case we will obtain a design of the form $\dfrac{}{\vdash}\,\dagger$ (the †-rule is actually the only one able to justify the empty sequent). In this case, normalization is said to converge, and $\mathfrak{D}, \mathfrak{E}$ are said to be *orthogonal*. However, it could also be that the two disigns are just unable to communicate, and normalization does not deliver any result. In this case, we remain with the partial design: $\dfrac{}{\vdash}\,\Omega$.

More interesting than the normal form, is the process of calculation itself, that is the interaction between the designs. The sequence of actions produced by this interaction is called *dispute*.

A design can also be presented as the collection of its possible interactions. In the following we will first characterize the *sequences of actions that correspond*

to a dispute. We will then characterize the *set of disputes which correspond to* interactions of the same *design,* and verify that we have *all* of them. We therefore need: (i) a "coherence condition" to guarantee that a set of disputes is compatible, meaning that all the disputes are paths on the same design, and (ii) a "saturation condition" to guarantee we have all the possible paths.

3 Arenas, Players and Legal Positions

Let us revisit some basic notions of Game Semantics in order to express the setting of Ludics.

As we have seen, an action is a pair (ξ, I) where ξ is a sequence of natural numbers, called an address, and I is a finite set of natural numbers. Each action is a "move" in Games Semantics term. The associated "dependency tree" is the universal Arena $\mathcal{U}$.

Players: The universe of addresses (and therefore of actions) is split between two *players*: one owning the even-length addresses, the other owning the odd-length addresses. Since it is convenient to fix a point of view, we will call Proponent (P) the player who starts, and Opponent (O) the other. Notice that in Ludics there is a complete symmetry between the two players: they obey the same rules. Games Semantics is generally biased toward Proponent. We will come back to this in Section 5.

Arena: An *arena* is given by a set of *moves*, a *labelling function* telling which player owns each move, and an *enabling relation* establishing a dependency relation between moves.

In the setting of Ludics, the dependency is induced by the sub-address relation *and* by the base. We say that (ξ, I) justifies $(\xi i, J)$, if $i \in I$. Moreover, if the base is $\eta \vdash \xi$, one can access ξ only after having accessed η. In this sense, ξ depends upon η.

Definition 4 (Universal Arena). *The Universal Arena $\mathcal{U}$ on the base $\vdash <>$ is given by (the initial solution of)*

$$\mathcal{U} = \textstyle\sum_{J \in P}(1 + J \times \mathcal{U})$$

where $\sum$ is the parallel composition of partial orders, and $+$ is the serial composition.

The universal arena $\mathcal{U}$ can be relocated to any initial address ξ. The moves of $\xi(\mathcal{U})$ are those of $\mathcal{U}$ with the renaming $\xi(\sigma, I) = (\xi\sigma, I)$.

The Arena on the base $\vdash \xi_1, \ldots, \xi_n$ is given by $\sum_{1 \leq i \leq n} \{\xi_i\} \times \mathcal{U}$

The Arena on the base $\eta \vdash \xi_1, \ldots, \xi_n$ is given by $\{\eta\} \times \mathcal{U} \leftarrow \sum_{1 \leq i \leq n} \{\xi_i\} \times \mathcal{U}$ (We do not explain the familiar operator $\leftarrow$ further here)

We extend the universal arena with a formal action † called *daimon.* † can be played by any player. It does not justify and is not justified by any other action.

Given the arena on a certain base, we call *initial* any action whose address belongs to the base. Notice that on the base $\eta \vdash \xi_i$, actions on either η or ξ_i are

initial moves, but any action of address ξ_i depends upon η.

Definition 5 (Linear positions). *A sequence of actions s is a* linear position, *or* play *if it satisfies the following conditions:*

Alternation *Parity alternates*
Justification *Each move is either initial or is justified by an earlier move.*
Linearity *Any address appears at most once.*
Daimon *Daimon (†) can only appear as last move.*

We call *terminating* the plays whose last move is †. ϵ indicates the empty sequence.

Notation. To indicate the players, we will use the variable X, $X \in \{P, O\}$, and $\overline{X}$ for its dual (the other player).

We will also use the notion of polarity: positive and negative. A move is positive for a player if it belongs to that player, negative if it belongs to the other.

P-move ("move belonging to P") = P-positive ("move positive for P") = O-negative ("move negative for O").

Each position belongs to one of the players, according to the last move (or more precisely to who is to play). We call P-Position a position that expects an action by Opponent, typically, a position whose last move is P. An O-Position is a position where P is to play. Since Proponent is the player who starts, we have that ϵ and all even-length positions are O-position, all odd-length positions are P-positions.

A P-position is a positive position for P, and a negative position for O. We use the notation p^P, p^O, p^+, p^-.

Let as recall the key notion of view.

Definition 6 (Views). *Let q be a linear position and $X \in \{O, P\}$ a player. Its view $\ulcorner q \urcorner^X$ of q is inductively defined as follows. When there is no ambiguity on the player, we simply write $\ulcorner q \urcorner$ for $\ulcorner q \urcorner^X$. Below, positive and negative is relative to X.*

- $\ulcorner \epsilon \urcorner = \epsilon$;
- $\ulcorner s\kappa^+ \urcorner = \ulcorner s^- \urcorner \kappa^+$;
- $\ulcorner s\kappa^- \urcorner = \kappa^-$ *if κ is initial;*
- $\ulcorner s\kappa' t\kappa^- \urcorner = \ulcorner s^- \urcorner \kappa' \kappa$, *if $\kappa = (\xi i, J)$ and $\kappa' = (\xi, I)^+$.*

We denote Opponent view *by $\ulcorner q \urcorner^O$ and* Proponent view *by $\ulcorner q \urcorner^P$. Moreover, by $\ulcorner q\kappa^+ \urcorner$ we mean the view of the player for which κ is positive. If κ belongs to X, then $\ulcorner q\kappa^+ \urcorner = \ulcorner q\kappa \urcorner^X$ and $\ulcorner q\kappa^- \urcorner = \ulcorner q\kappa \urcorner^{\overline{X}}$.*

Definition 7 (Legal positions). *A linear position p is* legal *if it satisfies the following condition:*

Visibility *If* $t\kappa \sqsubseteq p$ *and* κ *is non initial, then the justifier of* κ *occurs in* $\ulcorner t\kappa^{+}\urcorner$.

According to our convention, this means that if κ *is a P-move, its justifier occurs in* $\ulcorner t\kappa\urcorner^{P}$*, and therefore in* $\ulcorner t\urcorner^{P}$*, if* κ *is an O-move, its justifier occurs in* $\ulcorner t\urcorner^{O}$.

3.1 Designs

Let us revisit the presentation of designs. We first recall normalization. The interaction among the designs of a cut-net leads us to access some of the actions of the two designs, in a sequence which in Ludics is called dispute. Normalization converges if eventually we reach a daimon (†). Daimon is in fact is a special symbol which indicates termination (one of the players gives up). Otherwise, normalization diverges, and the result is "partial". When normalization converges, $\mathfrak{D}$ is said orthogonal to $\mathfrak{E}$ ($\mathfrak{D}\perp\mathfrak{E}$).

Let $\mathfrak{D}$ be a design of base $\vdash<>$ and $\mathfrak{E}$ a counter-design of base $<>\vdash$. From now on we focus on this case to simplify presentation[2].

We define the plays according to these two designs, $\mathcal{P} = Plays(\mathfrak{D};\mathfrak{E})$, as:
$\epsilon \in \mathcal{P}$
$p \in \mathcal{P}$ is a P to play position and $\ulcorner p\urcorner^{P}\kappa \in \mathfrak{D}$ then $p\kappa \in \mathcal{P}$
$p \in \mathcal{P}$ is an O to play position and $\ulcorner p\urcorner^{O}\kappa \in \mathfrak{E}$ then $p\kappa \in \mathcal{P}$

Fact 1 *$\mathcal{P}$ is totally ordered by initial segment.*

We indicate by $[\mathfrak{D} \rightleftharpoons \mathfrak{E}]$ the (possibly infinite) sequence of actions which is the sup of $Plays(\mathfrak{D};\mathfrak{E})$. A sequence ending with a daimon is called a *dispute*. In such a case, $\mathfrak{D}$ is said to be orthogonal to $\mathfrak{E}$: $\mathfrak{D}\perp\mathfrak{E}$.

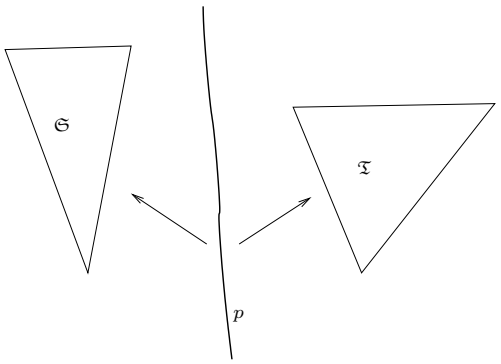
Fact 2 (Chronicles) *If* $p \in Plays(\mathfrak{D};\mathfrak{E})$*, then for any* $q \sqsubseteq r^{P} \sqsubseteq p$*,* $\ulcorner q\urcorner^{P}$ *is a chronicle of* $\mathfrak{D}$*. For any* $q \sqsubseteq r^{O} \sqsubseteq p$*,* $\ulcorner q\urcorner^{O} \in \mathfrak{E}$

Proposition 1 (Disputes as legal positions). *If* $p \in Plays(\mathfrak{D};\mathfrak{E})$ *then* p *is a legal position. Therefore in particular any dispute is a legal position on the universal arena.*

Conversely, we shall show that, given a legal position p, we can extract a design $\mathfrak{S}$ *and* a counter-design $\mathfrak{T}$ s.t. $[\mathfrak{S} \rightleftharpoons \mathfrak{T}] = p$. $\mathfrak{S}, \mathfrak{T}$ are minimal such designs.

Definition 8. *Let* p *be a finite legal position on the universal arena.*
$Des^{P}(p) = \{\ulcorner q\urcorner^{P} : q \sqsubseteq r^{P} \sqsubseteq p\}$.
$Des^{O}(p) = \{\ulcorner q\urcorner^{O} : q \sqsubseteq r^{O} \sqsubseteq p\}$.

[2] In the general case one deals with a family of designs.



Proposition 2. *Let p be a legal position on the universal arena.*

(i) $Des^P(p)$, $Des^O(p)$ *are designs on bases* $\vdash <>$, $<> \vdash$ *respectively.*
(ii) $[Des^P(p) \rightleftharpoons Des^O(p)] = p$.
(iii) If $p \in Plays(\mathfrak{D}; \mathfrak{E})$ *then* $Des^P(p) \subseteq \mathfrak{D}$ *and* $Des^O(p) \subseteq \mathfrak{E}$.

Proof. (i) Let us just check Coherence. Assume $\mathfrak{c}_1, \mathfrak{c}_2$ are incomparable, $\mathfrak{c}_1 \sqsupseteq \mathfrak{c}\kappa\kappa_1$ and $\mathfrak{c}_2 \sqsupseteq \mathfrak{c}\kappa\kappa_2$, where $\kappa_1 \neq \kappa_2$. If $\mathfrak{c}_1 \bigwedge \mathfrak{c}_2$ were not positive, then κ_1, κ_2 would be. Therefore $\mathfrak{c}\kappa\kappa_1 = \ulcorner s_1\kappa\kappa_1 \urcorner^P$, $\mathfrak{c}\kappa\kappa_2 = \ulcorner s_2\kappa\kappa_2 \urcorner^P$, and since linearity forces $s_1\kappa = s_2\kappa$, thus $\kappa_1 = \kappa_2$.

Examples. $Des^P(\epsilon) = \emptyset$, which corresponds to the partial design $\dfrac{}{\vdash <>}\ \Omega$

$Des^O(\epsilon) = \{\epsilon\}$, which corresponds to the derivation $\dfrac{}{<> \vdash}\ (<>, \emptyset)$

$Des^P(< \dagger >) = \emptyset$, which corresponds to the partial design $\dfrac{}{\vdash <>}\ \dagger$

$Des^O(< \dagger >) = \{\epsilon\}$, as before.

3.2 Designs as Set of Disputes

A design can be described by the set of its possible interaction (plays or disputes).

Given a design $\mathfrak{D}$, let us define

$$Plays(\mathfrak{D}) = \bigcup Plays(\mathfrak{D}; \mathfrak{E}), \text{ for } \mathfrak{E} \text{ design of opposite base.}$$

We have that $Plays(\mathfrak{D}) \cap Plays(\mathfrak{E}) = Plays(\mathfrak{D}; \mathfrak{E})$.

Fact 3 *If* $p \in \mathfrak{D}$ *then* $p \in Plays(\mathfrak{D})$

Fact 4 $Plays(\mathfrak{D}) = \{q$ *legal positions, s.t.*$\forall q \sqsubseteq r^+ \sqsubseteq p, \ulcorner q \urcorner \in \mathfrak{D}\}$

Proposition 3. $\mathfrak{D}$ *is recovered from* $Plays(\mathfrak{D})$ *by*

$$\mathfrak{D} = \{\ulcorner q \urcorner, q \sqsubseteq r, r \text{ is a positive position}, r \in Plays(\mathfrak{D})\}$$

Let *Disp* $\mathfrak{D}$ be the set $\{[\mathfrak{D} \rightleftharpoons \mathfrak{E}], \mathfrak{D} \perp \mathfrak{E}\}$ of terminating plays.

As any non-terminating positive play can be immediately terminated by the opponent with a daimon, any positive play belongs to *Disp* $\mathfrak{D}$. Therefore the set of disputes is enough to recover $\mathfrak{D}$.

Proposition 4. *$\mathfrak{D}$ is recovered from Disp $\mathfrak{D}$ by*

$$\mathfrak{D} = \{\ulcorner q \urcorner, q \sqsubseteq r^+ \in \mathit{Disp}\ (\mathfrak{D})\}$$

The set of possible interactions of a design can be characterized directly using the notions of Game Semantics.

4 Strategies

The universal arena gives us a game in the usual sense. There are two natural choices to give the game tree: either we consider the tree of all plays (all linear positions), or we consider only the tree of legal plays. We start by adopting this second choice, but we will come back to the first in Section 5.

There are a number of standard representations of the simplest notion of deterministic P-strategy for games. One can take any of the following.

(i) All finite plays "in accord" with the strategy.
(ii) All prefix of finite plays ending with P-moves.
(iii) All finite plays ending with P-moves.
(iv) All finite plays ending with P-moves plus all finite plays ending in O-moves to which P has no response.

These are equivalent and here is convenient to use the form (i).

Definition 9 (X-Strategy). *A P-strategy (O-strategy) S on the universal arena is a non-empty collection of plays (on that arena) which is*

1. *closed under prefix;*
2. *if $p, q \in S$ are* incomparable *then $p \bigwedge q$ is a positive position (a P-position for a P-Strategy, an O-position for an O-strategy);*
3. *if $p \in S$ is a positive position, then for all legal positions $p\kappa$, $p\kappa \in S$.*

We will call *pre-strategy* the way to present a strategy corresponding to the alternative (ii). A pre-strategy is therefore a non empty collection of plays which satisfies conditions (1) and (2) above, and such that all maximal plays are positive.

Definition 10 (Innocent Strategy). *An X-strategy S is innocent when:*
if p, q are negative positions, $\ulcorner p \urcorner^X = \ulcorner q \urcorner^X$, $p\kappa \in S$ and $q \in S$ then $q\kappa \in S$.

It is immediate by construction that

Fact 5 *If $\mathfrak{D}$ is a design of base $\vdash <>$ then $\mathit{Plays}(\mathfrak{D})$ is an innocent Player strategy (in the game given by the tree of legal plays). If $\mathfrak{E}$ is a design of base $<> \vdash$ then $\mathit{Plays}(\mathfrak{E})$ is an innocent opponent strategy.*

It is well known in Games Semantics that

(i) the collection of views of an innocent strategy generates the complete strategy, and
(ii) the collections of views of an innocent strategy S is contained in S.

Our main claim is that a design can be seen as the collection of views of an innocent strategy. From the views we can recover the strategy, from the strategy we can extract the views. Section 4.1 reviews these notions. Section 4.2 comes back to designs.

4.1 Innocent Strategies: Views and Plays

The views of an innocent strategy are enough to describe the strategy. When we do this, it is rather natural not to consider the views to which the player does not reply.

Definition 11 (Views(S)). *Let S be an X-strategy. We define*

$$Views(S) = \{\ulcorner q \urcorner^X, q \sqsubseteq p^+ \in S\}$$

We recall some properties of innocent strategies from this perspective.

Fact 6 (Closure under view) *If S is an innocent X-strategy then we have $Views(S) \subseteq S$.*

Fact 7 (Saturation) *Let T be any strategy and S an innocent strategy. If $Views(T) \subseteq S$ then $T \subseteq S$.*

Fact 8 (Determinism under view) *Let S be an innocent X-strategy. If $pab \in S, qac \in S, \ulcorner pa \urcorner = \ulcorner qa \urcorner$ then $b = c$.*

This in particular means that $Views(S)$ itself satisfies determinism (cf. (ii) in Definition 9).

Plays vs. Views. We say that a set of positions $\mathcal{V}$ is *stable under view* if $\ulcorner p \urcorner = p$ for all $p \in \mathcal{V}$.

Definition 12 (Plays($\mathcal{V}$)). *Let $\mathcal{V}$ be a pre-strategy stable under view. We define $Plays(\mathcal{V})$ as in 4:*
$Plays(\mathcal{V}) = \{q$ legal positions, $s.t. \forall q \sqsubseteq r^+ \sqsubseteq p, \ulcorner q \urcorner \in \mathcal{V}\}$

Proposition 5. *Let S be an innocent strategy.*
$Plays(Views(S)) = S$
$Views(S)$ is a pre-strategy, stable under view.

Proposition 6. *Let $\mathcal{V}$ be a pre-strategy stable under view.*
$Views(Plays(\mathcal{V})) = \mathcal{V}$
$Plays(\mathcal{V})$ is the smallest innocent strategy which contains $\mathcal{V}$.

Proof. Notice that $Plays(\mathcal{V})$ is deterministic because $\mathcal{V}$ is.

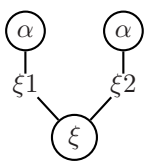
If S is an innocent strategy and $\mathcal{V} \subseteq S$ then from $Views(Plays(\mathcal{V})) = \mathcal{V}$ and Proposition 7 we deduce that $Plays(\mathcal{V}) \subseteq S$.

4.2 Designs as Innocent Strategies

Fact 9 *Let $\mathfrak{D}$ be a design. Then $\mathfrak{D}$ is a pre-strategy stable under view.*

Unfortunately, the converse is not necessarily true.

Consider for example the innocent strategy generated by the following two plays: $\{\langle \xi^+, \xi 1, \alpha\rangle, \langle \xi^+, \xi 2, \alpha\rangle\}$. To this we would associate the following tree of views:



Even though all plays are linear, we do not obtain a design, in that propagation is not satisfied (in the next section we explain better what does this mean).

A first solution is simply to translate the condition of propagation from chronicles to views. We will give a more natural solution in Section 5.

Definition 13 (Propagation). *A strategy S satisfies the propagation condition if:*

If $t\kappa, t'\kappa \in Views(S)$ and $t = \mathfrak{c} * (\xi, I) * \mathfrak{d}$, $t' = \mathfrak{c} * (\xi', I') * \mathfrak{d}'$ then $\xi = \xi'$.

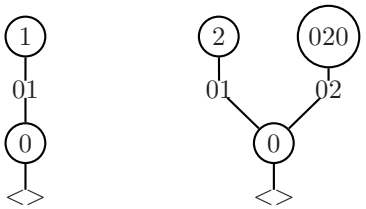
Fact 10 *Let $\mathcal{V}$ be a pre-strategy which is stable under view and which also satisfies* propagation*, then $\mathcal{V}$ is a design.*

Fact 11 *(i) Let $\mathfrak{D}$ be a design. $Plays(\mathfrak{D})$ is an innocent strategy, the smallest innocent strategy which contains $\mathfrak{D}$.*
(ii) Let S be an innocent strategy which satisfies propagation*. Then $ViewS$ is a design.*
(iii) $Plays(ViewsS) = S$ and $Views(Plays\mathfrak{D}) = \mathfrak{D}$

Innocence. Notice that a strategy which is not innocent does not correspond to any construct in Ludics.

Let us consider the strategy S on $\vdash <>$ given by the closure under prefix of
$\{p_1 = \langle(<>, \{0,1,2\}), (0, I_0), (01, I_{01}), (1, J)\rangle$ and
$p_2 = \langle(<>, \{0,1,2\}), ((0, I_0), (02, I_{02}), (020, I_{020}), (01, I_{01}), (2, K))\}\rangle$

S is an O-strategy. $Des^O(p_1)$ and $Des^O(p_2)$ respectively produce the trees:



The first two chronicles cannot co-exist in the same design.

5 Linearity

As we have seen, there is only one delicate point to establish a correspondence between designs and innocent strategies, namely that it is not enough to consider linear legal positions.

The objects described by an innocent strategy are linear for all computational purpose, but we would not reach a full completeness result for **MALL**. Typically, to the example in Section 4.2, we could associate the following proof.

$$\dfrac{\dfrac{\dfrac{0 \vdash A}{\vdash \downarrow \top, A}}{\downarrow A^{\perp} \vdash \downarrow \top} \quad \dfrac{\dfrac{0 \vdash B}{\vdash \downarrow \top, B}}{\downarrow B^{\perp} \vdash \downarrow \top}}{\vdash (\downarrow\uparrow A) \otimes (\downarrow\uparrow B), \downarrow \top}$$

The formula $\downarrow \top$ appears in the context of both component of the tensor. No play satisfying visibility can detect that α (the address of the formula $\downarrow \top$) is used twice, visiting both branches of the design.

The solution we gave earlier was to ask the condition of propagation, which is a way explicitly to demand the separation of the contexts on a Tensor rule. Games suggest a better solution in the use of a more liberal notion of play, as in [1].

Let us come back to Section 4 and consider the other possible choice for the game tree: using all linear positions (not only legal ones).

Given a design $\mathfrak{D}$ let us consider

$$Plays^*(\mathfrak{D}) = \{p \text{ linear plays such that for all } q \sqsubseteq r^+ \sqsubseteq p, \ulcorner q \urcorner \in \mathfrak{D}\}$$

Fact 12 *If $\mathfrak{D}$ is a design, then $Plays^*(\mathfrak{D})$ is an innocent strategy (in the game given by the tree of all linear plays).*

Remark 1. In general, there will be $p \in Plays*(\mathfrak{D})$ in which the *opponent* does not play innocently. A position in which the *player* does not play innocently never appears in $Plays^*(\mathfrak{D})$.

Proposition 7. *If S is an innocent strategy in the tree of linear plays, then $Views(S)$ is a design.*

Extracting Strategies from a Play. We have shown that to a play p we can associate both a design and a counter-design. To be able to extract both a strategy and a counter-strategy it is essential that p is linear. For example, to the play $\langle \alpha, \alpha 0, \alpha \rangle$ we can associate a design, but not a counter-design. In other words, this play belongs to an innocent strategy, but not to an innocent counter-strategy. Notice that the issue of lifting a play to a strategy (not a counter-strategy) was addressed by Danos Herbelin and Regnier in [3].

6 Further Work

This work suggests several directions to be explored. A natural continuation is to develop a presentation of Ludics based on disputes. Moreover, since we establish a bridge between Ludics and Game Semantics, we expect to be able to transfer experiences and techniques between the two settings. The use of plays rather than views (chronicles) could allow for a finer analysis. We have seen that designs correspond to innocent strategies. It is a natural question to ask what would be the analogue of *general strategies* in Ludics. Conversely, to what would lead the notion of *location* in Games?

In this paper we only consider the first concepts in Ludics. We intend further to consider the constructions of *behaviour* and *incarnation* from the perspective of Game Semantics. It seems natural to apply the framework of Abstract Games [8]. A category of behaviours is obtained using orthogonality and double gluing. We would like to clarify the relation between that structure and the "realizability" structure on behaviours given in Ludics. Furthermore it would be interesting to investigate the extent to which behaviours regarded as abstract games can be presented as concrete games. (First steps in this direction were given in [4]).

References

[1] S. Abramsky, K. Honda, and G. McCusker. A fully abstract game semantics for general references. In *Proceedings LICS'98*. IEEE Computer Society Press, 1998. 455

[2] S. Abramsky and G. McCusker. *Computational Logic*, chapter Game semantics. Springer-Verlag, 1999. 442

[3] V. Danos, H. Herbelin, and L. Regnier. Games semantics and abstract machines. In *Proceedings LICS'96*. IEEE Computer Society Press, 1996. 456

[4] C. Faggian. *On the Dynamics of Ludics. A Study of Interaction*. PhD thesis, Université Aix-Marseille II, 2002. 456

[5] J.-Y. Girard. Geometry of interaction i: Interpretation of system f. In Z. A. Ferro R.m Bonotto C., Valentini S., editor, *Logic Colloquium 88*, pages 221–260. North Holland, 1989. 442

[6] J.-Y. Girard. Locus solum. *Mathematical Structures in Computer Science*, 2001. 442, 446, 447

[7] M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 2000. 442

[8] M. Hyland and A. Schalk. Abstract Games for Linear Logic. *Electronic Notes in Theoretical Computer Science*, 29:1–24, 1999. 456

[9] H. Nickau. Hereditarily sequential functionals. In *Proceedings of the Symposium on Logical Foundations of Computer Science: Logic at St. Petersburg*, LNCS. Springer, 1994. 442

Classical Linear Logic of Implications

Masahito Hasegawa

Research Institute for Mathematical Sciences, Kyoto University
hassei@kurims.kyoto-u.ac.jp

Abstract. We give a simple term calculus for the multiplicative exponential fragment of Classical Linear Logic, by extending Barber and Plotkin's system for the intuitionistic case. The calculus has the non-linear and linear implications as the basic constructs, and this design choice allows a technically managable axiomatization without commuting conversions. Despite this simplicity, the calculus is shown to be sound and complete for category-theoretic models given by $*$-autonomous categories with linear exponential comonads.

1 Introduction

We propose a linear lambda calculus called *Dual Classical Linear Logic (DCLL)* for the multiplicative exponential fragment of Classical Linear Logic [10] (often called MELL in the literature). It can be regarded as an extension of the *Dual Intuitionistic Linear Logic (DILL)* of Barber and Plotkin [1, 2].

The main feature of DCLL is its *simplicity*: just three logical connectives (intuitionistic implication $\to$, linear implication $\multimap$ and the bottom type $\bot$) and six axioms for the equational theory on terms (proofs) which are just the familiar $\beta\eta$ axioms of the lambda calculus (each for $\to$ and $\multimap$) plus two axioms saying that the type $(\sigma \multimap \bot) \multimap \bot$ is canonically isomorphic to σ. In particular we can avoid axioms for *commuting conversions*, which have always been troublesome on term calculi for Linear Logic. Other logical connectives and their proof expressions of MELL are easily derived in DCLL; for instance the exponential ! is given by $!\sigma \equiv (\sigma \to \bot) \multimap \bot$. All the desired equalities between terms, including the commuting conversions, are provable from the simple axioms of DCLL.

Thus DCLL can be used as a compact linear syntax for reasoning about MELL, to compliment the drawbacks of conventional proof nets-based presentations which are often tiresome to formulate and deal with. For instance, it is much easier to describe and analyze the translations between type systems if we use term calculi like DCLL instead of graph-based systems. Also techniques of logical relations (e.g. [11, 23]) seem to work more smoothly on term-based systems. As future work, we plan to study the compilations of call-by-value programming languages into linearly typed intermediate languages [6, 13] using DCLL as a target calculus. In fact, our choice of the logical connectives has been motivated by this research direction – see the discussion in Sec. 6.

Despite its simplicity, it is shown that DCLL is sound and complete for categorical models of MELL given by $*$-autonomous categories with symmetric

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 458–472, 2002.
© Springer-Verlag Berlin Heidelberg 2002

monoidal comonads satisfying some coherence conditions (to be called linear exponential comonads). It turns out that our simple axioms are sufficient for giving such a categorical structure on the term model. Although this may not be of big surprise, there seem not many systems for Linear Logic supported by this sort of semantic completeness at the level of proofs, and we think that this completeness result gives a justification on our design of DCLL.

This paper is organized as follows. We introduce the system DCLL in Sec. 2, with some discussions on its alternative formulations. Sec. 3 gives a comparison of DCLL with its precursor DILL. Sec. 4 then states the completeness result of DCLL with respect to the categorical models of MELL. In Sec. 5 the extension with additives (hence a full propositional Classical Linear Logic) is discussed. We conclude the paper by giving some discussions on future work at Sec. 6. Appendix A gives a summary of DILL, while Appendix B is devoted to a variant of DCLL based on the $\lambda\mu$-calculus, called μDCLL. Appendix C describes an alternative axiomatization of DCLL (and MLL) with no base type.

Acknowledgements I am grateful to Hayo Thieletcke for drawing my attention to the $\{\rightarrow, \multimap\}$-fragment. I thank Martin Hofmann, Yoshihiko Kakutani and Valeria de Paiva for discussions and comments related to this work.

2 DCLL

2.1 The System DCLL

In this "dual-context"[1] formulation of the linear lambda calculus, a typing judgement takes the form $\Gamma\ ;\ \Delta \vdash M : \tau$ in which Γ represents an intuitionistic (or additive) context whereas Δ is a linear (multiplicative) context. While the variables in Γ can be used in the term M as many times as we like, those in Δ must be used exactly once. A typing judgement $x_1 : \sigma_1, \ldots, x_m : \sigma_m\ ;\ y_1 : \tau_1, \ldots, y_n : \tau_n \vdash M : \sigma$ can be considered as the proof of the sequent $!\sigma_1, \ldots, !\sigma_m, \tau_1, \ldots, \tau_n \vdash \sigma$, or the proposition $!\sigma_1 \otimes \ldots \otimes !\sigma_m \otimes \tau_1 \otimes \ldots \otimes \tau_n \multimap \sigma$.

As mentioned in the introduction, the system features both intuitionistic (non-linear) arrow type $\rightarrow$ and linear arrow type $\multimap$. We use $\boldsymbol{\lambda} x^\sigma.M$ and $M \mathbin{@} N$ for the non-linear lambda abstraction and application respectively, while $\lambda x^\sigma.M$ and $M\,N$ for the linear ones. For expressing the duality of Classical Linear Logic, there also is a special combinator C_σ which serves as the isomorphism from $(\sigma \multimap \bot) \multimap \bot$ to σ (which, however, can be eliminated when we have no base type – see the discussion at the end of this section).

Types and Terms

$$\begin{aligned} \sigma &::= b \mid \sigma \rightarrow \sigma \mid \sigma \multimap \sigma \mid \bot \\ M &::= x \mid \boldsymbol{\lambda} x^\sigma.M \mid M \mathbin{@} M \mid \lambda x^\sigma.M \mid M\,M \mid \mathsf{C}_\sigma \end{aligned}$$

where b ranges over a set of base types. We may omit the type subscripts for ease of presentation.

[1] As noted in [2] the word "dual" of DILL (and DCLL) comes from this dual-context typing, and has nothing to do with the duality of Classical Linear Logic.

Typing

$$\frac{}{\Gamma_1, x:\sigma, \Gamma_2 \ ; \ \emptyset \vdash x:\sigma}\ (\text{Int-Ax}) \qquad \frac{}{\Gamma \ ; \ x:\sigma \vdash x:\sigma}\ (\text{Lin-Ax})$$

$$\frac{\Gamma, x:\sigma_1 \ ; \ \Delta \vdash M:\sigma_2}{\Gamma \ ; \ \Delta \vdash \boldsymbol{\lambda} x^{\sigma_1}.M : \sigma_1 \to \sigma_2}\ (\to \text{I}) \quad \frac{\Gamma \ ; \ \Delta \vdash M:\sigma_1 \to \sigma_2 \quad \Gamma \ ; \ \emptyset \vdash N:\sigma_1}{\Gamma \ ; \ \Delta \vdash M \,@\, N : \sigma_2}\ (\to \text{E})$$

$$\frac{\Gamma \ ; \ \Delta, x:\sigma_1 \vdash M:\sigma_2}{\Gamma \ ; \ \Delta \vdash \lambda x^{\sigma_1}.M : \sigma_1 \multimap \sigma_2}\ (\multimap \text{I}) \quad \frac{\Gamma \ ; \ \Delta_1 \vdash M:\sigma_1 \multimap \sigma_2 \quad \Gamma \ ; \ \Delta_2 \vdash N:\sigma_1}{\Gamma \ ; \ \Delta_1 \sharp \Delta_2 \vdash M\,N : \sigma_2}\ (\multimap \text{E})$$

$$\frac{}{\Gamma \ ; \ \emptyset \vdash \mathsf{C}_\sigma : ((\sigma \multimap \bot) \multimap \bot) \multimap \sigma}\ (\mathsf{C})$$

where $\Delta_1 \sharp \Delta_2$ is a merge of Δ_1 and Δ_2 [2]. Thus, $\Delta_1 \sharp \Delta_2$ represents one of possible merges of Δ_1 and Δ_2 as finite lists. We assume that, when we introduce $\Delta_1 \sharp \Delta_2$, there is no variable occurring both in Δ_1 and in Δ_2. We write $\emptyset$ for the empty context. We note that any typing judgement has a unique derivation (hence a typing judgement can be identified with its derivation).

Axioms

$$\begin{array}{llcll}
(\beta_\to) & (\boldsymbol{\lambda} x.M) \,@\, N & = & M[N/x] & \\
(\eta_\to) & \boldsymbol{\lambda} x.M \,@\, x & = & M & (x \notin FV(M)) \\
(\beta_\multimap) & (\lambda x.M)\, N & = & M[N/x] & \\
(\eta_\multimap) & \lambda x.M\, x & = & M & \\
(\mathsf{C}_1) & L\,(\mathsf{C}_\sigma\, M) & = & M\, L & (L : \sigma \multimap \bot) \\
(\mathsf{C}_2) & \mathsf{C}_\sigma\,(\lambda k^{\sigma \multimap \bot}.k\, M) & = & M &
\end{array}$$

where $M[N/x]$ denotes the capture-free substitution. Note that there is no side condition $x \notin FV(M)$ for the axiom $(\eta_\multimap)$ (and similarly for (C_2)), as linearity prevents x from occuring in M. The equality judgement $\Gamma \ ; \ \Delta \vdash M = N : \sigma$ for $\Gamma \ ; \ \Delta \vdash M : \sigma$ and $\Gamma \ ; \ \Delta \vdash N : \sigma$ is defined as usual.
We note that the axiom (C_1) is equivalent to $\lambda k^{\sigma \multimap \bot}.k\,(\mathsf{C}_\sigma\, M) = M$; thus the last two axioms say that C_σ is the inverse of $\lambda x^\sigma.\lambda k^{\sigma \multimap \bot}.k\, x : \sigma \multimap (\sigma \multimap \bot) \multimap \bot$.

Lemma 1. *The "naturality" of* C *is provable in DCLL:*

$$L^{\sigma \multimap \tau}\,(\mathsf{C}_\sigma\, M^{(\sigma \multimap \bot) \multimap \bot}) \ = \ \mathsf{C}_\tau\,(\lambda k^{\tau \multimap \bot}.M\,(\lambda x^\sigma.k\,(L\,x))) \ : \ \tau$$

<u>Proof:</u>

$$L\,(\mathsf{C}\,M) \overset{\mathsf{C}_2}{=} \mathsf{C}\,(\lambda k.k\,(L\,(\mathsf{C}\,M))) \overset{\beta_\multimap}{=} \mathsf{C}\,(\lambda k.(\lambda x.k\,(L\,x))\,(\mathsf{C}\,M)) \overset{\mathsf{C}_1}{=} \mathsf{C}\,(\lambda k.M\,(\lambda x.k\,(L\,x))).$$

□

2.2 Alternative Formulations of DCLL

Formulation Based on the $\lambda\mu$-calculus. Instead of the combinator C for the double-negation elimination, we could use the syntax of the $\lambda\mu$-calculus [21] for expressing the duality, as done in [17] for the multiplicative fragment (MLL).

We do not take this approach here as our presentation using C seems sufficiently simple, while the $\lambda\mu$-calculus style formulation requires to introduce yet another typing context. For completeness, in Appendix B we present such a system (μDCLL) which is routinely seen to be equivalent to DCLL. A potential benefit of the $\lambda\mu$-calculus approach is that it may give a confluent and normalizing reduction system (which cannot be expected for DCLL); also it allows natural treatment of the connective $\parr$ (by introducing the binary μ-bindings). See also [8] for relevant results.

Axiomatization without C. In DCLL, the following equations are provable:

Lemma 2.

1. $\mathsf{C}_\perp = \lambda m^{(\perp \multimap \perp) \multimap \perp}.m\,(\lambda x^\perp.x)$
2. $\mathsf{C}_{\sigma\to\tau} = \lambda m^{((\sigma\to\tau)\multimap\perp)\multimap\perp}.\boldsymbol{\lambda} x^\sigma.\mathsf{C}_\tau\,(\lambda k^{\tau\multimap\perp}.m\,(\lambda f^{\sigma\to\tau}.k\,(f \mathbin{@} x)))$
3. $\mathsf{C}_{\sigma\multimap\tau} = \lambda m^{((\sigma\multimap\tau)\multimap\perp)\multimap\perp}.\lambda x^\sigma.\mathsf{C}_\tau\,(\lambda k^{\tau\multimap\perp}.m\,(\lambda f^{\sigma\multimap\tau}.k\,(f\,x)))$

<u>Proof:</u>

1. $\mathsf{C}_\perp\, m = (\lambda x^\perp.x)\,(\mathsf{C}_\perp\, m) = m\,(\lambda x^\perp.x)$.
2. $\mathsf{C}_{\sigma\to\tau}\, m \mathbin{@} x = \mathsf{C}_\tau\,(\lambda k.k\,(\mathsf{C}_{\sigma\to\tau}\, m \mathbin{@} x)) = \mathsf{C}_\tau\,(\lambda k.(\lambda f.k\,(f \mathbin{@} x))\,(\mathsf{C}_{\sigma\to\tau}\, m)) = \mathsf{C}_\tau\,(\lambda k.m\,(\lambda f.k\,(f \mathbin{@} x)))$.
3. $\mathsf{C}_{\sigma\multimap\tau}\, m\, x = \mathsf{C}_\tau\,(\lambda k.k\,(\mathsf{C}_{\sigma\multimap\tau}\, m\, x)) = \mathsf{C}_\tau\,(\lambda k.(\lambda f.k\,(f\,x))\,(\mathsf{C}_{\sigma\multimap\tau}\, m)) = \mathsf{C}_\tau\,(\lambda k.m\,(\lambda f.k\,(f\,x)))$. □

This implies that, if we do not have base types, all DCLL terms can be expressed as just (non-linear and linear) lambda terms, without using the combinator C. By induction we can show

Proposition 1. *For $\sigma = \sigma_1 \Rightarrow_1 \ldots \sigma_n \Rightarrow_n \perp$ (where $\Rightarrow_i$ is either $\to$ or $\multimap$)*

$$\mathsf{C}_\sigma\, M \star_1 N_1 \ldots \star_n N_n = M\,(\lambda f^\sigma.f \star_1 N_1 \ldots \star_n N_n) : \perp$$

is provable in DCLL, where $M : (\sigma \multimap \perp) \multimap \perp$, $N_i : \sigma_i$, and $\star_i$ is a non-linear application if $\Rightarrow_i$ is $\to$, or a linear application if $\Rightarrow_i$ is $\multimap$. □

If we *define* C's as lambda terms by the equations of Lem. 2 or Prop. 1, then the axiom (C_2) follow just from the $\beta\eta$-axioms for $\to$ and $\multimap$. Therefore it is possible to axiomatize DCLL with no base type as a quotient of the $\{\to, \multimap\}$-calculus on the single base type $\perp$ obtained by adding the axiom (C_1) for these defined C's. In fact all of them are derivable from the following single instance and the $\beta\eta$-axioms for $\to$ and $\multimap$:

$$L\,(\lambda x^\sigma.M\,(\lambda f^{\sigma\multimap\perp}.f\,x)) = M\,L$$

where $L : (\sigma \multimap \perp) \multimap \perp$ and $M : ((\sigma \multimap \perp) \multimap \perp) \multimap \perp$.[2] So it suffices to have the standard $\beta\eta$-axioms and this equation; Appendix C describes the resulting system (as well as its multiplicative fragment MLL).

[2] This in fact amounts to the infamous (in)equality known as "triple unit problem" (which asks if two canonical endomorphisms on $((A \multimap I) \multimap I) \multimap I$ are the same in a symmetric monoidal closed category, see [19, 16]) if one replaces $\perp$ by I.

3 DILL in DCLL

The primitive constructs of DILL (summarized in Appendix A) can be defined in DCLL as follows:

$$
\begin{array}{ll}
I & \equiv \bot \multimap \bot \\
\sigma_1 \otimes \sigma_2 & \equiv (\sigma_1 \multimap \sigma_2 \multimap \bot) \multimap \bot \\
!\,\sigma & \equiv (\sigma \to \bot) \multimap \bot \\
 & \\
* & \equiv \lambda x^{\bot}.x \\
\mathsf{let}\ *\ \mathsf{be}\ M^{I}\ \mathsf{in}\ N^{\tau} & \equiv \mathsf{C}_{\tau}\,(\lambda k^{\tau \multimap \bot}.M\,(k\,N)) \\
M^{\sigma_1} \otimes N^{\sigma_2} & \equiv \lambda k^{\sigma_1 \multimap \sigma_2 \multimap \bot}.k\,M\,N \\
\mathsf{let}\ x^{\sigma_1} \otimes y^{\sigma_2}\ \mathsf{be}\ M^{\sigma_1 \otimes \sigma_2}\ \mathsf{in}\ N^{\tau} & \equiv \mathsf{C}_{\tau}\,(\lambda k^{\tau \multimap \bot}.M\,(\lambda x^{\sigma_1}.\lambda y^{\sigma_2}.k\,N)) \\
!M^{\sigma} & \equiv \lambda h^{\sigma \to \bot}.h \mathbin{@} M \\
\mathsf{let}\ !x^{\sigma}\ \mathsf{be}\ M^{!\sigma}\ \mathsf{in}\ N^{\tau} & \equiv \mathsf{C}_{\tau}\,(\lambda k^{\tau \multimap \bot}.M\,(\boldsymbol{\lambda} x^{\sigma}.k\,N))
\end{array}
$$

(It is also possible to introduce connectives ? and ⅋ by $?\sigma \equiv (\sigma \multimap \bot) \to \bot$ and $\sigma_1 \parr \sigma_2 \equiv (\sigma_1 \multimap \bot) \multimap (\sigma_2 \multimap \bot) \multimap \bot$, though giving the term expressions associated to these connectives seems less obvious.)

Below we shall see that this encoding is sound, for both the typing and equational theory.

Lemma 3. *Derivation rules of typing judgements in DILL are admissible in DCLL.*

Proof: We shall spell out the cases of introduction and elimination rules for !

$$
\frac{\Gamma\ ;\ \emptyset \vdash M : \sigma}{\Gamma\ ;\ \emptyset \vdash !M : !\sigma}\ (!\,\mathrm{I}) \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : !\sigma \quad \Gamma, x : \sigma\ ;\ \Delta_2 \vdash N : \tau}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ !x^{\sigma}\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau}\ (!\,\mathrm{E})
$$

which are derivable in DCLL as follows.

$$
\dfrac{\dfrac{\dfrac{}{\Gamma\ ;\ h : \sigma \to \bot \vdash h : \sigma \to \bot}\ \text{Lin-Ax} \qquad \Gamma\ ;\ \emptyset \vdash M : \sigma}{\Gamma\ ;\ h : \sigma \to \bot \vdash h \mathbin{@} M : \bot}\ \to\mathrm{E}}{\Gamma\ ;\ \emptyset \vdash !M \equiv \lambda h^{\sigma \to \bot}.h \mathbin{@} M : (\sigma \to \bot) \multimap \bot \equiv !\sigma}\ \multimap\mathrm{I}
$$

$$
\dfrac{\dfrac{\Gamma\ ;\ \Delta_1 \vdash M : !\sigma \equiv (\sigma \to \bot) \multimap \bot \qquad \dfrac{\dfrac{\dfrac{}{\Gamma, x : \sigma\ ;\ k : \tau \multimap \bot \vdash k : \tau \multimap \bot}\ \text{Lin-Ax} \qquad \Gamma, x : \sigma\ ;\ \Delta_2 \vdash N : \tau}{\Gamma, x : \sigma\ ;\ \Delta_2, k : \tau \multimap \bot \vdash k\,N : \bot}\ \multimap\mathrm{E}}{\Gamma\ ;\ \Delta_2, k : \tau \multimap \bot \vdash \boldsymbol{\lambda} x^{\sigma}.k\,N : \sigma \to \bot}\ \to\mathrm{I}}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2, k : \tau \multimap \bot \vdash M\,(\boldsymbol{\lambda} x^{\sigma}.k\,N) : \bot}\ \multimap\mathrm{E}}{\dfrac{\Gamma\ ;\ \emptyset \vdash \mathsf{C}_{\tau} : ((\tau \multimap \bot) \multimap \bot) \multimap \tau \qquad \Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \lambda k^{\tau \multimap \bot}.M\,(\boldsymbol{\lambda} x^{\sigma}.k\,N) : (\tau \multimap \bot) \multimap \bot}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ !x^{\sigma}\ \mathsf{be}\ M\ \mathsf{in}\ N \equiv \mathsf{C}_{\tau}\,(\lambda k^{\tau \multimap \bot}.M\,(\boldsymbol{\lambda} x^{\sigma}.k\,N)) : \tau}\ \multimap\mathrm{E}}\ \multimap\mathrm{I}
$$

The cases of I and $\otimes$ are derived similarly. □

Theorem 1. *Equality axioms in DILL are admissible in DCLL.*

<u>Proof:</u> The β-axioms are easy:

$$\begin{aligned}
\mathsf{let}\ *\ \mathsf{be}\ *\ \mathsf{in}\ N &\equiv \mathsf{C}\,(\lambda k.(\lambda x.x)\,(k\,N)) \\
&= \mathsf{C}\,(\lambda k.k\,N) \\
&= N
\end{aligned}$$

$$\begin{aligned}
\mathsf{let}\ x \otimes y\ \mathsf{be}\ M_1 \otimes M_2\ \mathsf{in}\ N &\equiv \mathsf{C}\,(\lambda k.(\lambda h.h\,M_1\,M_2)\,(\lambda x.\lambda y.k\,N)) \\
&= \mathsf{C}\,(\lambda k.(\lambda x.\lambda y.k\,N)\,M_1\,M_2) \\
&= \mathsf{C}\,(\lambda k.k\,N[M_1/x, M_2/y]) \\
&= N[M_1/x, M_2/y]
\end{aligned}$$

$$\begin{aligned}
\mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N &\equiv \mathsf{C}\,(\lambda k.(\lambda h.h \mathbin{@} M)\,(\boldsymbol{\lambda} x.k\,N)) \\
&= \mathsf{C}\,(\lambda k.(\boldsymbol{\lambda} x.k\,N) \mathbin{@} M) \\
&= \mathsf{C}\,(\lambda k.k\,N[M/x]) \\
&= N[M/x]
\end{aligned}$$

The η-axioms are slightly more subtle.

$$\begin{aligned}
\mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ * &\equiv \mathsf{C}\,(\lambda k.M\,(k\,(\lambda x.x))) \\
&= \lambda y.(\lambda k.M\,(k\,(\lambda x.x)))\,(\lambda f.f\,y) && \text{(Prop.1)} \\
&= \lambda y.M\,((\lambda f.f\,y)\,(\lambda x.x)) \\
&= \lambda y.M\,((\lambda x.x)\,y) \\
&= \lambda y.M\,y \\
&= M
\end{aligned}$$

$$\begin{aligned}
\mathsf{let}\ x \otimes y\ \mathsf{be}\ M\ \mathsf{in}\ x \otimes y &\equiv \mathsf{C}\,(\lambda k.M\,(\lambda xy.k\,(\lambda n.n\,x\,y))) \\
&= \lambda u.(\lambda k.M\,(\lambda xy.k\,(\lambda n.n\,x\,y)))\,(\lambda f.f\,u) && \text{(Prop.1)} \\
&= \lambda u.M\,(\lambda xy.(\lambda f.f\,u)\,(\lambda n.n\,x\,y)) \\
&= \lambda u.M\,(\lambda xy.u\,x\,y) \\
&= \lambda u.M\,u \\
&= M
\end{aligned}$$

$$\begin{aligned}
\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x &\equiv \mathsf{C}\,(\lambda k.M\,(\boldsymbol{\lambda} x.k\,(\lambda h.h \mathbin{@} x))) \\
&= \lambda u.(\lambda k.M\,(\boldsymbol{\lambda} x.k\,(\lambda h.h \mathbin{@} x)))\,(\lambda f.f\,u) && \text{(Prop.1)} \\
&= \lambda u.M\,(\boldsymbol{\lambda} x.(\lambda f.f\,u)\,(\lambda h.h \mathbin{@} x)) \\
&= \lambda u.M\,(\boldsymbol{\lambda} x.(\lambda h.h \mathbin{@} x)\,u) \\
&= \lambda u.M\,(\boldsymbol{\lambda} x.u \mathbin{@} x) \\
&= \lambda u.M\,u \\
&= M
\end{aligned}$$

There remain (30 instances of) axioms for commuting conversions which, for instance, can be shown as

$$\begin{aligned}
L\,(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N) &\equiv L\,(\mathsf{C}\,(\lambda k.M\,(\boldsymbol{\lambda} x.k\,N))) \\
&= \mathsf{C}\,(\lambda h.(\lambda k.M\,(\boldsymbol{\lambda} x.k\,N))\,(\lambda y.h\,(L\,y))) && \text{(Lem. 1)} \\
&= \mathsf{C}\,(\lambda h.M\,(\boldsymbol{\lambda} x.(\lambda y.h\,(L\,y))\,N)) \\
&= \mathsf{C}\,(\lambda h.M\,(\boldsymbol{\lambda} x.h\,(L\,N))) \\
&\equiv \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ L\,N
\end{aligned}$$

$$\begin{aligned}
\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ \lambda y.N &\equiv \mathsf{C}\,(\lambda k.M\,(\boldsymbol{\lambda} x.k\,(\lambda y.N))) \\
&= \lambda y.\mathsf{C}\,(\lambda h.(\lambda k.M\,(\boldsymbol{\lambda} x.k\,(\lambda y.N)))\,(\lambda f.h\,(f\,y))) && \text{(Lem. 2)} \\
&= \lambda y.\mathsf{C}\,(\lambda h.M\,(\boldsymbol{\lambda} x.(\lambda f.h\,(f\,y))\,(\lambda y.N))) \\
&= \lambda y.\mathsf{C}\,(\lambda h.M\,(\boldsymbol{\lambda} x.h\,N)) \\
&\equiv \lambda y.(\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N)
\end{aligned}$$

We leave the other cases as exercises for the interested readers. □

4 Completeness for Categorical Models

An important implication of Thm. 1, together with the result in [2] (completeness via the term model construction), is that the term model of DCLL forms a model of DILL, i.e., a symmetric monoidal closed category equipped with a symmetric monoidal comonad satisfying certain coherence conditions (see e.g. [7]) which we shall call a "linear exponential comonad" (following [15]).[3]

Definition 1 (linear exponential comonad). *A symmetric monoidal comonad* $! = (!, \varepsilon, \delta, m_{A,B}, m_I)$ *on a symmetric monoidal category* $\mathcal{C}$ *is called a* linear exponential comonad *when the category of its coalgebras is a category of commutative comonoids – that is:*

- *for each free* $!$*-coalgebra* $(!A, \delta_A)$ *there are specified monoidal natural transformations* $e_A : !A \to I$ *and* $d_A : !A \to !A \otimes !A$ *which form a commutative comonoid* $(!A, e_A, d_A)$ *in* $\mathcal{C}$ *and also are coalgebra morphisms from* $(!A, \delta_A)$ *to* (I, m_I) *and* $(!A \otimes !A, m_{!A,!A} \circ (\delta_A \otimes \delta_A))$ *respectively, and*
- *any coalgebra morphism from* $(!A, \delta_A)$ *to* $(!B, \delta_B)$ *is also a comonoid morphism from* $(!A, e_A, d_A)$ *to* $(!B, e_B, d_B)$.

Moreover, the symmetric monoidal closed category given by the term model of DCLL is a $*$*-autonomous category* [3, 4] if we take $\perp$ as the dualizing object. Recall that a $*$-autonomous category can be characterized as a symmetric monoidal closed category with an object $\perp$ such that the canonical morphism from σ to $(\sigma \multimap \perp) \multimap \perp$ is an isomorphism — in the term model of DCLL, the inverse is given by the combinator C_σ.

On the other hand, all the axioms of DCLL are sound with respect to interpretations in such categorical models, where a typing judgement

$$x_1 : \sigma_1, \ldots, x_m : \sigma_m \ ; \ y_1 : \tau_1, \ldots, y_n : \tau_n \vdash M : \sigma$$

is inductively interpreted as a morphism $[\![x_1 : \sigma_1, \ldots \ ; \ y_1 : \tau_1, \ldots \vdash M : \sigma]\!]$ from $![\![\sigma_1]\!] \otimes \ldots \otimes ![\![\sigma_m]\!] \otimes [\![\tau_1]\!] \otimes \ldots \otimes [\![\tau_n]\!]$ to $[\![\sigma]\!]$ in the $*$-autonomous category with the linear exponential comonad $!$. Thus we have:

Theorem 2 (categorical completeness). *The equational theory of DCLL is sound and complete for categorical models given by* $*$*-autonomous categories with linear exponential comonads:* $\Gamma \ ; \ \Delta \vdash M = N : \sigma$ *is provable if and only if* $[\![\Gamma \ ; \ \Delta \vdash M : \sigma]\!] = [\![\Gamma \ ; \ \Delta \vdash N : \sigma]\!]$ *holds for every such models.* □

[3] In [2] a model of DILL is described as a symmetric monoidal adjunction between a cartesian closed category and a symmetric monoidal closed category (Benton's LNL model [5]). It is known that such an "adjunction model" gives rise to a linear exponential comonad on the symmetric monoidal closed category part. Conversely, a symmetric monoidal closed category with a linear exponential comonad has at least one symmetric monoidal adjunction from a cartesian closed category so that it induces the linear exponential comonad (such an adjunction is not unique in general, though). Therefore, for our purpose (the completeness result as stated here), it does not matter which class of structures we choose as models. However we must be careful when we talk about the morphisms between models, e.g. to use the term model of DILL (or DCLL) as a classifying category of such structures.

5 Additives

It is fairly routine to enrich DCLL with additives. We add the cartesian product & and its unit $\top$, and terms

$$\frac{}{\Gamma\ ;\ \Delta \vdash \langle\,\rangle : \top}\ (\top\,\mathrm{I}) \qquad \frac{\Gamma\ ;\ \Delta \vdash M : \sigma \quad \Gamma\ ;\ \Delta \vdash N : \tau}{\Gamma\ ;\ \Delta \vdash \langle M, N\rangle : \sigma \,\&\, \tau}\ (\&\,\mathrm{I})$$

$$\frac{\Gamma\ ;\ \Delta \vdash M : \sigma \,\&\, \tau}{\Gamma\ ;\ \Delta \vdash \mathsf{fst}_{\sigma,\tau}\, M : \sigma}\ (\&\,\mathrm{E}_L) \qquad \frac{\Gamma\ ;\ \Delta \vdash M : \sigma \,\&\, \tau}{\Gamma\ ;\ \Delta \vdash \mathsf{snd}_{\sigma,\tau}\, M : \tau}\ (\&\,\mathrm{E}_R)$$

and the standard axioms

$$\begin{array}{lll} M & = \langle\,\rangle & (M : \top) \\ \mathsf{fst}\,\langle M, N\rangle & = M & \\ \mathsf{snd}\,\langle M, N\rangle & = N & \\ \langle \mathsf{fst}\, M, \mathsf{snd}\, M\rangle & = M & \end{array}$$

Again we do not need any additional axiom for commuting conversions. Furthermore, it is possible to eliminate the C combinators for additives as we can prove (using Lem. 1 for the latter case)

Lemma 4.

1. $\mathsf{C}_\top = \lambda m^{(\top \multimap \bot) \multimap \bot}.\langle\,\rangle$
2. $\mathsf{C}_{\sigma \,\&\, \tau} =$
   $\lambda m^{((\sigma \,\&\, \tau) \multimap \bot) \multimap \bot}.$
   $\langle \mathsf{C}_\sigma\,(\lambda k^{\sigma \multimap \bot}.m\,(\lambda z^{\sigma \,\&\, \tau}.k\,(\mathsf{fst}_{\sigma,\tau}\, z))), \mathsf{C}_\tau\,(\lambda h^{\tau \multimap \bot}.m\,(\lambda z^{\sigma \,\&\, \tau}.h\,(\mathsf{snd}_{\sigma,\tau}\, z)))\rangle$

□

In particular, if we do not have base types, it is possible to axiomatize DCLL with additives as a quotient of a typed lambda calculus (with $\to$, $\multimap$, $\top$, &) on a single base type $\bot$, in the same way as described at the end of Sec. 2.

The coproduct $\oplus$ and its unit 0 are given by $\sigma_1 \oplus \sigma_2 \equiv ((\sigma_1 \multimap \bot) \,\&\, (\sigma_2 \multimap \bot)) \multimap \bot$ and $0 \equiv \top \multimap \bot$ as usual. The associated term constructs are

$$\frac{\Gamma\ ;\ \Delta \vdash M : \sigma}{\Gamma\ ;\ \Delta \vdash \mathsf{inl}_{\sigma,\tau}\, M \equiv \lambda k^{(\sigma \multimap \bot) \,\&\, (\tau \multimap \bot)}.\mathsf{fst}_{\sigma \multimap \bot, \tau \multimap \bot}\, k\, M : \sigma \oplus \tau}\ (\oplus\,\mathrm{I}_L)$$

$$\frac{\Gamma\ ;\ \Delta \vdash N : \tau}{\Gamma\ ;\ \Delta \vdash \mathsf{inr}_{\sigma,\tau}\, N \equiv \lambda k^{(\sigma \multimap \bot) \,\&\, (\tau \multimap \bot)}.\mathsf{snd}_{\sigma \multimap \bot, \tau \multimap \bot}\, k\, N : \sigma \oplus \tau}\ (\oplus\,\mathrm{I}_R)$$

$$\frac{\Gamma\ ;\ \Delta \vdash L : \sigma \oplus \tau \quad \Gamma\ ;\ \Delta, x : \sigma \vdash M : \theta \quad \Gamma\ ;\ \Delta, y : \tau \vdash N : \theta}{\begin{array}{l}\Gamma\ ;\ \Delta \vdash \mathsf{case}\ L\ \mathsf{of}\ \mathsf{inl}\, x^\sigma \mapsto M \parallel \mathsf{inr}\, y^\tau \mapsto N \equiv \\ \qquad\qquad \mathsf{C}_\theta\,(\lambda k^{\theta \multimap \bot}.L\,\langle \lambda x^\sigma.k\, M, \lambda y^\tau.k\, N\rangle) : \theta\end{array}}\ (\oplus\,\mathrm{E})$$

They do satisfy the standard axioms for coproducts as well as a number of commuting conversion axioms.

A category-theoretic model of DCLL extended with additives can be given as a $*$-autonomous category with a linear exponential comonad and finite products. The soundness and completeness results in the last section easily extend for this setting.

6 Discussions and Future Work

6.1 DCLL as a Typed Intermediate Language

The design of DCLL is heavily inspired from our experience (and still on-going project) on the study of compiling (mostly call-by-value typed) programming languages into linearly typed intermediate languages [13], which has been briefly mentioned in the introduction.

In [6] the $\{\to, \multimap\}$-fragment of DILL (with recursive types) is used as the target language of CPS transformations. In [13] we extend the idea of [6] to general monadic transformations into the $\{!, \multimap\}$-fragment of DILL, and have observed that the $\{\to, \multimap\}$-fragment is full in the $\{!, \multimap\}$-fragment[4] (hence both approaches essentially agree, as long as we talk about CPS transformations). In these studies the "linearly-used continuation monad" $((-) \to \theta) \multimap \theta$ plays the key role[5] $:\to$ for continuations, and $\multimap$ for the linearity of their passing. The choice of connectives of DCLL then comes to us naturally; $\to$ and $\multimap$ come first, and we regard the exponential ! as the special case of the linearly-used continuation monad by letting θ be $\bot$: $!\sigma \simeq (!\sigma \multimap \bot) \multimap \bot \simeq (\sigma \to \bot) \multimap \bot$.

It is also interesting to re-examine the previous work on applying Classical Linear Logic to programming languages with control features [9, 20] using DCLL; in particular Filinski's work [9] seems to share several ideas with the design of DCLL.

6.2 Is "!" better than "$\to$"?

A possible criticism on DCLL is on its indirect treatment of the exponentials, which have been regarded as the central feature of Linear Logic by many people (though there are some exceptions, e.g. [24, 22, 18][6]). We used to consider ! as a primitive and $\to$ as a derived connective as $\sigma \to \tau \equiv !\sigma \multimap \tau$, but not in the other way (i.e. $!\sigma \equiv (\sigma \to \bot) \multimap \bot$ as we do in DCLL).

However, even in the Intuitionistic Linear Logic, the full completeness of the $\{\to, \multimap\}$-fragment in the $\{!, \multimap\}$-fragment tells us that $\to$ is no less delicate than ! at the level of proofs (terms), while $\{\to, \multimap\}$ enjoys much simpler term structures and nice properties like confluence and strong normalization. And, in Classical Linear Logic, $\{\to, \multimap, \bot\}$ is literally isomorphic to $\{!, \multimap, \bot\}$ — then it is not unnatural to use the technically simpler presentation.

[4] This result is shown by mildly extending the proof of full completeness of Girard's translation from the simply typed lambda calculus into the $\{!, \multimap\}$-fragment of DILL [12].

[5] This is *not* a monad on the term model of DILL; it is a monad on a suitable subcategory of the category of !-coalgebras.

[6] In particular Plotkin's system [22] is the *second-order* $\{\to, \multimap\}$-calculus in which other connectives of DILL including ! are definable in the similar way as we do in DCLL, for example $!\sigma$ as $\forall X.(\sigma \to X) \multimap X$. In fact it suffices to add an axiom $L^{\sigma \multimap \tau}\,(M^{\forall X.(\sigma \multimap X) \multimap X}\,\sigma\,(\lambda x^{\sigma}.x)) = M\,\tau\,L$ (which just says $\sigma \simeq \forall X.(\sigma \multimap X) \multimap X$) to give the structure of models of DILL to the term model of this calculus – the story is completely analogous to the case of DCLL.

Moreover, as mentioned above, DCLL do have natural advantages in programming language theory. From such an application-oriented view, we think that the simplicity of DCLL is undeniably attractive. See also [18] for relevant discussions on the $\{\rightarrow, \multimap, \otimes, I, \&, \top\}$-fragment and its fibration-based models (which can be adopted for DCLL without problem).

6.3 Why not $\sigma^{\perp\perp} = \sigma$

Another possible source of criticism would be the way we deal with the duality, which again is the essential feature of Classical Linear Logic. Many systems for Classical Linear Logic, especially those of proof nets, identify the type $\sigma^{\perp\perp}$ $(= (\sigma \multimap \perp) \multimap \perp)$ with σ. On the other hand, in DCLL (and some other term-based systems like [8]) they are just isomorphic, and we explicitly have terms for the isomorphisms. The essential reason of this non-identification in DCLL is that we intend it to have $*$-autonomous categories with linear exponential comonads as models, rather than those with *strict* involution (i.e. $(-)^{\perp\perp}$ is the identity functor and the canonical isomorphism $\sigma \xrightarrow{\simeq} \sigma^{\perp\perp}$ is an identity arrow), as we think that having a strict involution is not a natural assumption on semantic models. (However, it might be the case that any $*$-autonomous category is equivalent to a $*$-autonomous category with strict involution, and if this is true, this design choice would be just a matter of taste.)

6.4 ILL vs. CLL

We believe that the relationship between Intuitionistic Linear Logic and Classical Linear Logic – at the level of proofs rather than that of provability – has not been sufficiently sorted out yet. Let us state the problems in terms of DCLL.

The first question concerns the converse of Thm. 1.

Conjecture 1 (conservativity, or completeness). The equational theory of DCLL is conservative over that of DILL. That is, $\Gamma \; ; \; \Delta \vdash M = N : \sigma$ is provable in DILL if and only if it is provable in DCLL (via the encoding given in Sec. 3 – the "only if" part follows from Thm. 1).

The second question is on the *fullness* of Intuitionistic Linear Logic in Classical Linear Logic.

Conjecture 2 (fullness). DILL is full in DCLL. That is, if $\Gamma \; ; \; \Delta \vdash N : \sigma$ is derivable in DCLL and all the types in Γ, Δ and σ stay in DILL, then there exists a DILL-term $\Gamma \; ; \; \Delta \vdash M : \sigma$ so that $\Gamma \; ; \; \Delta \vdash M = N : \sigma$ is provable in DCLL.

Note that the corresponding results for multiplicative fragments are already known: MILL is fully complete in MLL, see for instance [15]. We also know that MILL is fully complete in DILL [11] – but how about DILL and DCLL? In fact, one of our motivations to introduce DCLL has been to provide a manageable foundation for attacking this question. We expect that this will be positively solved by using the model construction techniques (categorical glueing / logical relations) in [23, 15].

6.5 Decidability of the Equational Theory

Another natural question on DCLL is

Conjecture 3 (decidability). The equational theory of DCLL is decidable.

We shall note that the equational theory of DILL is known to be decidable, see [1]. The same is true for MLL (in [17] the corresponding coherence problem for $*$-autonomous categories is solved). We hope that some rewriting techniques are effective for this purpose, especially using some $\lambda\mu$-calculus style variant of DCLL (e.g. μDCLL given in Appendix B). However, even though DCLL avoids to deal with commuting conversions explicitly, we still have to work up to certain equivalence classes of terms, e.g. as in [17] (for instance $\lambda x^{\perp}.\lambda f^{\perp\multimap\perp}.\lambda g^{\perp\multimap\perp}.f\,(g\,x) = \lambda x^{\perp}.\lambda f^{\perp\multimap\perp}.\lambda g^{\perp\multimap\perp}.g\,(f\,x)$ holds in DCLL, but there is no natural way to give an orientation on this equation).

References

[1] Barber, A. (1997) *Linear Type Theories, Semantics and Action Calculi.* PhD Thesis ECS-LFCS-97-371, University of Edinburgh. 458, 468

[2] Barber, A. and Plotkin, G. (1997) Dual intuitionistic linear logic. Submitted. An earlier version available as Technical Report ECS-LFCS-96-347, LFCS, University of Edinburgh. 458, 459, 460, 464

[3] Barr, M. (1979) $*$-*Autonomous Categories.* Springer Lecture Notes in Math. **752**. 464

[4] Barr, M. (1991) $*$-autonomous categories and linear logic. *Math. Struct. Comp. Sci.* **1**, 159–178. 464

[5] Benton, P. N. (1995) A mixed linear and non-linear logic: proofs, terms and models (extended abstract). In *Computer Science Logic (CSL'94)*, Springer Lecture Notes in Comput. Sci. **933**, pp. 121–135. 464

[6] Berdine, J., O'Hearn, P. W., Reddy, U. S. and Thielecke, H. (2001) Linearly used continuations. In *Proc. ACM SIGPLAN Workshop on Continuations (CW'01)*, Technical Report No. 545, Computer Science Department, Indiana University, pp. 47–54. 458, 466

[7] Bierman, G. M. (1995) What is a categorical model of intuitionistic linear logic? In *Proc. Typed Lambda Calculi and Applications (TLCA'95)*, Springer Lecture Notes in Comput. Sci. **902**, pp. 78–93. 464

[8] Bierman, G. M. (1999) A classical linear lambda-calculus. *Theoret. Comp. Sci.* **227**(1-2), 43–78. 461, 467

[9] Filinski, A. (1992) Linear continuations. In *Proc. Principles of Programming Languages (POPL'92)*, pp. 27–38. 466

[10] Girard, J.-Y. (1987) Linear logic. *Theoret. Comp. Sci.* **50**, 1–102. 458

[11] Hasegawa, M. (1999) Logical predicates for intuitionistic linear type theories. In *Proc. Typed Lambda Calculi and Applications (TLCA'99)*, Springer Lecture Notes in Comput. Sci. **1581**, pp. 198–213. 458, 467

[12] Hasegawa, M. (2000) Girard translation and logical predicates. *J. Funct. Programming* **10**(1), 77–89. 466

[13] Hasegawa, M. (2002) Linearly used effects: monadic and CPS transformations into the linear lambda calculus. In *Proc. Functional and Logic Programming (FLOPS2002)*, Springer Lecture Notes in Comput. Sci. 458, 466

[14] Hofmann, M., Pavlović, D. and Rosolini, P. (eds.) (1999) *Proc. 8th Conf. on Category Theory and Computer Science*. Electron. Notes Theor. Comput. Sci. **29**. 469

[15] Hyland, M. and Schalk, A. (200x) Glueing and orthogonality for models of linear logic. To appear in *Theoret. Comp. Sci.* 464, 467

[16] Kelly, G. M. and Mac Lane, S. (1971) Coherence in closed categories. *J. Pure Appl. Algebra* **1**(1):97–140. 461

[17] Koh, T. W. and Ong, C.-H. L. (1999) Explicit substitution internal languages for autonomous and $*$-autonomous categories. In [14]. 460, 468

[18] Maietti, M. E., de Paiva, V. and Ritter, E. (2000) Categorical models for intuitionistic and linear type theory. In *Foundations of Software Science and Computation Structure (FoSSaCS 2000)*, Springer Lecture Notes in Comput. Sci. **1784**, pp. 223–237. 466, 467

[19] Murawski, A. S. and Ong, C.-H. L. (1999) Exhausting strategies, Joker games and IMLL with units. In [14]. 461

[20] Nishizaki, S. (1993) Programs with continuations and linear logic. *Science of Computer Programming* **21**(2), 165–190. 466

[21] Parigot, M. (1992) $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proc. Logic Programming and Automated Reasoning*, Springer Lecture Notes in Comput. Sci. **624**, pp. 190–201. 460

[22] Plotkin, G. (1993) Type theory and recursion (extended abstract). In *Proc. Logic in Computer Science (LICS'93)*, pp. 374. 466

[23] Streicher, T. (1999) Denotational completeness revisited. In [14]. 458, 467

[24] Wadler, P. (1990) Linear types can change the world! In *Proc. Programming Concepts and Methods*, North-Holland, pp. 561–581. 466

A Dual Intuitionistic Linear Logic

Types and Terms

$$\begin{aligned}\sigma &::= b \mid I \mid \sigma \otimes \sigma \mid \sigma \multimap \sigma \mid !\sigma \\ M &::= x \mid * \mid \mathsf{let}\ * \ \mathsf{be}\ M\ \mathsf{in}\ M \mid M \otimes M \mid \mathsf{let}\ x^\sigma \otimes x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ M \mid \\ &\quad \lambda x^\sigma.M \mid MM \mid !M \mid \mathsf{let}\ !x^\sigma\ \mathsf{be}\ M\ \mathsf{in}\ M\end{aligned}$$

Typing

$$\frac{}{\Gamma_1, x:\sigma, \Gamma_2\ ;\ \emptyset \vdash x:\sigma}\,(\text{Int-Ax}) \qquad \frac{}{\Gamma\ ;\ x:\sigma \vdash x:\sigma}\,(\text{Lin-Ax})$$

$$\frac{}{\Gamma\ ;\ \emptyset \vdash * : I}\,(I\,\mathrm{I}) \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : I \quad \Gamma\ ;\ \Delta_2 \vdash N : \sigma}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ N : \sigma}\,(I\,\mathrm{E})$$

$$\frac{\Gamma\ ;\ \Delta_1 \vdash M : \sigma_1 \quad \Gamma\ ;\ \Delta_2 \vdash N : \sigma_2}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash M \otimes N : \sigma_1 \otimes \sigma_2}\,(\otimes\,\mathrm{I}) \qquad \frac{\begin{array}{c}\Gamma\ ;\ \Delta_1 \vdash M : \sigma_1 \otimes \sigma_2 \\ \Gamma\ ;\ \Delta_2, x:\sigma_1, y:\sigma_2 \vdash N : \tau\end{array}}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ x^{\sigma_1} \otimes y^{\sigma_2}\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau}\,(\otimes\,\mathrm{E})$$

$$\frac{\Gamma\ ;\ \Delta, x:\sigma_1 \vdash M : \sigma_2}{\Gamma\ ;\ \Delta \vdash \lambda x^{\sigma_1}.M : \sigma_1 \multimap \sigma_2}\,(\multimap\,\mathrm{I}) \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : \sigma_1 \multimap \sigma_2 \quad \Gamma\ ;\ \Delta_2 \vdash N : \sigma_1}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash M\,N : \sigma_2}\,(\multimap\,\mathrm{E})$$

$$\frac{\Gamma\ ;\ \emptyset \vdash M : \sigma}{\Gamma\ ;\ \emptyset \vdash !M : !\sigma}\,(!\,\mathrm{I}) \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : !\sigma \quad \Gamma, x:\sigma\ ;\ \Delta_2 \vdash N : \tau}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N : \tau}\,(!\,\mathrm{E})$$

Axioms

$$\begin{array}{rrr} \mathsf{let}\ *\ \mathsf{be}\ *\ \mathsf{in}\ M = M & & \mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ * = M \\ \mathsf{let}\ x \otimes y\ \mathsf{be}\ M \otimes N\ \mathsf{in}\ L = L[M/x, N/y] & & \mathsf{let}\ x \otimes y\ \mathsf{be}\ M\ \mathsf{in}\ x \otimes y = M \\ (\lambda x.M)\, N = M[N/x] & & \lambda x.M\, x = M \\ \mathsf{let}\ !x\ \mathsf{be}\ !M\ \mathsf{in}\ N = N[M/x] & & \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ !x = M \end{array}$$

$$\begin{array}{rl} C[\mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ N] & = \mathsf{let}\ *\ \mathsf{be}\ M\ \mathsf{in}\ C[N] \\ C[\mathsf{let}\ x \otimes y\ \mathsf{be}\ M\ \mathsf{in}\ N] & = \mathsf{let}\ x \otimes y\ \mathsf{be}\ M\ \mathsf{in}\ C[N] \\ C[\mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ N] & = \mathsf{let}\ !x\ \mathsf{be}\ M\ \mathsf{in}\ C[N] \end{array}$$

where $C[-]$ is a linear context (no ! binds $[-]$).

B μDCLL

B.1 The System μDCLL

Types and Terms

$$\begin{array}{rl} \sigma & ::= b \mid \sigma \to \sigma \mid \sigma \multimap \sigma \mid \bot \\ M & ::= x \mid \boldsymbol{\lambda} x^{\sigma}.M \mid M \mathbin{@} M \mid \lambda x^{\sigma}.M \mid M\, M \mid [\alpha]M \mid \mu\alpha^{\sigma}.M \end{array}$$

Typing

$$\frac{}{\Gamma_1, x : \sigma, \Gamma_2\ ;\ \emptyset \vdash x : \sigma \mid \Sigma}\ (\text{Int-Ax}) \qquad \frac{}{\Gamma\ ;\ x : \sigma \vdash x : \sigma \mid \emptyset}\ (\text{Lin-Ax})$$

$$\frac{\Gamma, x : \sigma_1\ ;\ \Delta \vdash M : \sigma_2 \mid \Sigma}{\Gamma\ ;\ \Delta \vdash \boldsymbol{\lambda} x^{\sigma_1}.M : \sigma_1 \to \sigma_2 \mid \Sigma}\ (\to\text{I}) \qquad \frac{\Gamma\ ;\ \Delta \vdash M : \sigma_1 \to \sigma_2 \mid \Sigma \quad \Gamma\ ;\ \emptyset \vdash N : \sigma_1 \mid \emptyset}{\Gamma\ ;\ \Delta \vdash M \mathbin{@} N : \sigma_2 \mid \Sigma}\ (\to\text{E})$$

$$\frac{\Gamma\ ;\ \Delta, x : \sigma_1 \vdash M : \sigma_2 \mid \Sigma}{\Gamma\ ;\ \Delta \vdash \lambda x^{\sigma_1}.M : \sigma_1 \multimap \sigma_2 \mid \Sigma}\ (\multimap\text{I}) \qquad \frac{\Gamma\ ;\ \Delta_1 \vdash M : \sigma_1 \multimap \sigma_2 \mid \Sigma_1 \quad \Gamma\ ;\ \Delta_2 \vdash N : \sigma_1 \mid \Sigma_2}{\Gamma\ ;\ \Delta_1 \sharp \Delta_2 \vdash MN : \sigma_2 \mid \Sigma_1 \sharp \Sigma_2}\ (\multimap\text{E})$$

$$\frac{\Gamma\ ;\ \Delta \vdash M : \sigma \mid \Sigma}{\Gamma\ ;\ \Delta \vdash [\alpha]M : \bot \mid \alpha : \sigma, \Sigma}\ (\bot\text{I}) \qquad \frac{\Gamma\ ;\ \Delta \vdash M : \bot \mid \alpha : \sigma, \Sigma}{\Gamma\ ;\ \Delta \vdash \mu\alpha^{\sigma}.M : \sigma \mid \Sigma}\ (\bot\text{E})$$

Axioms

$$\begin{array}{lll} (\boldsymbol{\lambda} x.M) \mathbin{@} N & = M[N/x] & \\ \boldsymbol{\lambda} x.M \mathbin{@} x & = M & (x \notin FV(M)) \\ (\lambda x.M)\, N & = M[N/x] & \\ \lambda x.M\, x & = M & \\ L\,(\mu\alpha^{\sigma}.M) & = M\left[{}^{L(-)}/_{[\alpha](-)}\right] & (L : \sigma \multimap \bot) \\ \mu\alpha.[\alpha]M & = M & \end{array}$$

where $M\left[{}^{L(-)}/_{[\alpha](-)}\right]$ is obtained by replacing the (unique) subterm of the form $[\alpha]N$ by $L\,N$ in the capture-free way.

Lemma 5. *The following equations are provable in μDCLL.*

- $L\,(\mu\alpha^{\sigma}.M) = \mu\beta^{\tau}.M\left[{}^{[\beta]L(-)}/_{[\alpha](-)}\right]$ *where* $L : \sigma \multimap \tau$
- $[\alpha'](\mu\alpha^{\sigma}.M) = M[\alpha'/\alpha]$
- $\mu\alpha^{\perp}.M = M\left[{}^{(-)}/_{[\alpha](-)}\right]$
- $\mu\gamma^{\sigma\to\tau}.M = \boldsymbol{\lambda}x^{\sigma}.\mu\beta^{\tau}.M\left[{}^{[\beta](-)\circledast x}/_{[\gamma](-)}\right]$
- $\mu\gamma^{\sigma\multimap\tau}.M = \lambda x^{\sigma}.\mu\beta^{\tau}.M\left[{}^{[\beta](-)x}/_{[\gamma](-)}\right]$ □

B.2 DCLL vs. μDCLL

We first note that the combinator C_{σ} is easily represented in μDCLL by

$$\mathsf{C}_{\sigma} \;=\; \lambda m^{(\sigma\multimap\perp)\multimap\perp}.\mu\alpha^{\sigma}.m\,(\lambda x^{\sigma}.[\alpha]x) \;:\; ((\sigma \multimap \perp) \multimap \perp) \multimap \sigma.$$

Let us write M° for the induced translation of a DCLL-term M in μDCLL by this encoding.

Lemma 6. *If* $\Gamma\ ;\ \Delta \vdash M : \sigma$ *is derivable in DCLL,* $\Gamma\ ;\ \Delta \vdash M^{\circ} : \sigma \mid \emptyset$ *is derivable in* μ*DCLL.* □

Proposition 2. *If* $\Gamma\ ;\ \Delta \vdash M = N : \sigma$ *is provable in DCLL,* $\Gamma\ ;\ \Delta \vdash M^{\circ} = N^{\circ} : \sigma \mid \emptyset$ *is provable in* μ*DCLL.* □

Conversely, there is a translation $(-)^{\bullet}$ from μDCLL to DCLL given by

$$\begin{aligned}([\alpha]M)^{\bullet} &= [\alpha]M^{\bullet}\\ (\mu\alpha^{\sigma}.M)^{\bullet} &= \mathsf{C}_{\sigma}\left(\lambda k.M^{\bullet}\left[{}^{k(-)}/_{[\alpha](-)}\right]\right)\end{aligned}$$

and so on; for this $(-)^{\bullet}$ we have

Lemma 7. *If* $\Gamma\ ;\ \Delta \vdash M : \sigma \mid \alpha_1 : \sigma_1, \ldots, \alpha_n : \sigma_n$ *is derivable in* μ*DCLL,* $\Gamma\ ;\ \Delta, k_n : \sigma_n \multimap \perp, \ldots, k_1 : \sigma_1 \multimap \perp \vdash M^{\bullet}\left[{}^{k_1(-)}/_{[\alpha_1](-)}, \ldots, {}^{k_n(-)}/_{[\alpha_n](-)}\right] : \sigma$ *is derivable in DCLL. In particular, if* $\Gamma\ ;\ \Delta \vdash M : \sigma \mid \emptyset$ *is derivable in* μ*DCLL,* $\Gamma\ ;\ \Delta \vdash M^{\bullet} : \sigma \mid \emptyset$ *is derivable in DCLL.* □

Proposition 3. *If* $\Gamma\ ;\ \Delta \vdash M = N : \sigma \mid \emptyset$ *is provable in* μ*DCLL,* $\Gamma\ ;\ \Delta \vdash M^{\bullet} = N^{\bullet} : \sigma$ *is provable in DCLL.* □

Proposition 4. *For* $\Gamma\ ;\ \Delta \vdash M : \sigma$ *we have* $\Gamma\ ;\ \Delta \vdash M = M^{\circ\bullet} : \sigma$ *in DCLL. For* $\Gamma\ ;\ \Delta \vdash M : \sigma \mid \emptyset$ *we have* $\Gamma\ ;\ \Delta \vdash M = M^{\bullet\circ} : \sigma \mid \emptyset$ *in* μ*DCLL.* □

Thus we conclude that DCLL is identical to the single conclusion-fragment of μDCLL as typed equational theories.

B.3 Categorical Semantics

The interpretation of a typing judgement of the form

$$x_1 : \sigma_1, \ldots, x_m : \sigma_m\ ;\ y_1 : \tau_1, \ldots, y_n : \tau_n \vdash M : \sigma \mid \alpha_1 : \theta_1, \ldots, \alpha_k : \theta_k$$

is given as an arrow from $![\sigma_1]\otimes\ldots\otimes![\sigma_m]\otimes[\tau_1]\otimes\ldots\otimes[\tau_n]$ to $[\sigma] \parr [\theta_1] \parr \ldots \parr [\theta_k]$, by routinely extending the case of DCLL. The soundness and completeness of μDCLL with respect to the same class of categorical models immediately follow.

C Formulation without C

As noted in Sec. 2, we can formalize DCLL using just lambda terms and five axioms, if there is no base type. The same is true for MLL, for which just three axioms are sufficient.

C.1 DCLL

Types and Terms

$$\sigma ::= \sigma \to \sigma \mid \sigma \multimap \sigma \mid \bot \qquad M ::= x \mid \boldsymbol{\lambda} x^{\sigma}.M \mid M \mathbin{@} M \mid \lambda x^{\sigma}.M \mid M\,M$$

Typing

$$\frac{}{\Gamma_1, x:\sigma, \Gamma_2 \;;\; \emptyset \vdash x:\sigma}\ (\text{Int-Ax}) \qquad \frac{}{\Gamma \;;\; x:\sigma \vdash x:\sigma}\ (\text{Lin-Ax})$$

$$\frac{\Gamma, x:\sigma_1 \;;\; \Delta \vdash M:\sigma_2}{\Gamma \;;\; \Delta \vdash \boldsymbol{\lambda} x^{\sigma_1}.M : \sigma_1 \to \sigma_2}\ (\to \mathrm{I}) \quad \frac{\Gamma \;;\; \Delta \vdash M:\sigma_1 \to \sigma_2 \quad \Gamma \;;\; \emptyset \vdash N:\sigma_1}{\Gamma \;;\; \Delta \vdash M \mathbin{@} N : \sigma_2}\ (\to \mathrm{E})$$

$$\frac{\Gamma \;;\; \Delta, x:\sigma_1 \vdash M:\sigma_2}{\Gamma \;;\; \Delta \vdash \lambda x^{\sigma_1}.M : \sigma_1 \multimap \sigma_2}\ (\multimap \mathrm{I}) \frac{\Gamma \;;\; \Delta_1 \vdash M:\sigma_1 \multimap \sigma_2 \quad \Gamma \;;\; \Delta_2 \vdash N:\sigma_1}{\Gamma \;;\; \Delta_1 \sharp \Delta_2 \vdash M\,N : \sigma_2}\ (\multimap \mathrm{E})$$

Axioms

$$\begin{array}{lll}
(\boldsymbol{\lambda} x.M) \mathbin{@} N & = M[N/x] & \\
\boldsymbol{\lambda} x.M \mathbin{@} x & = M & (x \notin FV(M)) \\
(\lambda x.M)\,N & = M[N/x] & \\
\lambda x.M\,x & = M & \\
L\,(\lambda x^{\sigma}.M\,(\lambda f^{\sigma \multimap \bot}.f\,x)) & = M\,L & \begin{pmatrix} L : (\sigma \multimap \bot) \multimap \bot \\ M : ((\sigma \multimap \bot) \multimap \bot) \multimap \bot \end{pmatrix}
\end{array}$$

C.2 MLL

Types and Terms

$$\sigma ::= \sigma \multimap \sigma \mid \bot \qquad M ::= x \mid \lambda x^{\sigma}.M \mid M\,M$$

Typing

$$\frac{}{x:\sigma \vdash x:\sigma}\ (\text{Ax}) \quad \frac{\Delta, x:\sigma_1 \vdash M:\sigma_2}{\Delta \vdash \lambda x^{\sigma_1}.M : \sigma_1 \multimap \sigma_2}\ (\multimap \mathrm{I}) \quad \frac{\Delta_1 \vdash M:\sigma_1 \multimap \sigma_2 \quad \Delta_2 \vdash N:\sigma_1}{\Delta_1 \sharp \Delta_2 \vdash M\,N : \sigma_2}\ (\multimap \mathrm{E})$$

Axioms

$$\begin{array}{lll}
(\lambda x.M)\,N & = M[N/x] & \\
\lambda x.M\,x & = M & \\
L\,(\lambda x^{\sigma}.M\,(\lambda f^{\sigma \multimap \bot}.f\,x)) & = M\,L & \begin{pmatrix} L : (\sigma \multimap \bot) \multimap \bot \\ M : ((\sigma \multimap \bot) \multimap \bot) \multimap \bot \end{pmatrix}
\end{array}$$

Higher-Order Positive Set Constraints*

Jean Goubault-Larrecq

LSV/CNRS UMR 8643, ENS Cachan
61, av. du président-Wilson, 94235 Cachan Cedex, France

Abstract. We introduce a natural notion of positive set constraints on simply-typed λ-terms. We show that satisfiability of these so-called positive higher-order set constraints is decidable in 2-NEXPTIME. We explore a number of subcases solvable in 2-DEXPTIME, among which higher-order definite set constraints, a.k.a., emptiness of higher-order pushdown processes. This uses a first-order clause format on so-called shallow higher-order patterns, and automated deduction techniques based on ordered resolution with splitting. This technique is then applied to the task of approximating success sets for a restricted subset of λ-Prolog, à la Frühwirth et al.

1 Introduction

It is well-known that a certain form of positive set constraints are subsets of the monadic class [3]. In turn, the monadic class can be decided by resolution [17]. More precisely, ordered resolution with splitting decides the satisfiability of positive set constraints in NEXPTIME, and this is optimal [18].

The point of this paper is to note that a similar construction adapts directly to define notions of positive set constraints for typed λ-terms up to $\beta\eta$-conversion. In particular we show that the satisfiability of positive higher-order set constraints is decidable. This hinges on the use of a clausal format with terms replaced by higher-order patterns [19], limited to depth one—the so-called *shallow patterns*. A natural application is computing upper approximations of success sets for a restricted class of λ-Prolog programs (*descriptive typing*), following a construction of [12].

Outline. We give a few preliminary definitions in Section 2 on λ-terms and Miller's higher-order patterns, including our *shallow* patterns: these will be the terms that we shall allow in clauses defining positive higher-order set constraints. We recall and adapt the form of ordered resolution we shall use in Section 3. The meat of the paper is Section 4, where we introduce higher-order automata, higher-order pushdown systems, and higher-order positive set constraints. We show that the latter are decidable in 2-NEXPTIME, and investigate several special cases of lower complexity. We apply this technique to typing a restricted class of λ-Prolog programs in Section 5, and conclude in Section 6.

* Partially supported by the ACI VERNAM, the RNTL project EVA and the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions".

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 473–489, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Related Work. Set constraints traditionally denote relations between sets of ground first-order terms. Their main application is set-based analysis and type inference for functional, imperative and logic programming languages. See [21] for a recent survey—this is a very active field, and it would be too long to list all relevant papers. Even the number of variants of set constraints is daunting: definite, co-definite, positive or negative set constraints, with or without projection notably. The great majority is decidable in NEXPTIME, most of them are NEXPTIME-complete, while some of them, e.g. definite set constraints are DEXPTIME-complete [8]. Set constraints have even been generalized to deal with sets of terms modulo some equational theories [7], where decidability is obtained in the case of linear, shallow equational first-order theories. Our work can be seen as one addition to this category of set constraints, dealing with the theory of $\beta\eta$-equality in the typed λ-calculus—a theory that is far from shallow.

One particular relevant piece of work is [3], which shows that positive set constraints (with a limited form of projection, and with equality) are in close correspondence to monadic first-order formulas. We take this as a starting point to define a higher-order analogue. (In particular, we won't care to define a syntax resembling set constraints for the higher-order case, and will be content with just a clausal format.)

To decide clausal forms representing positive higher-order set constraints, we shallmake extensive use of resolution theorem proving techniques. A comprehensive reference is the handbook [22]. Using resolution to decide subclasses of first-order logic formulas was pioneered by Joyner [17] and by Maslov, see [10]. Standard refinements of resolution used in this area are hyperresolution and ordered refinements.

2 Preliminaries

Simple types, or types for short in this paper, are given by the grammar $\tau ::= b|\tau \to \tau$
where b ranges over a non-empty collection of *base types*. A *signature* Σ is a map from so-called *constants* $a, b, c, \ldots$ to types. A signature is *finite* iff its domain is finite. Fix a countably infinite set $Var_\forall$ of *universal variables* $x, y, z, \ldots$, each equipped with a unique type (let $\tau(x)$ be the type of x). Also, fix a countably infinite set $Var_\exists$ of *existential variables* $X, Y, Z, \ldots$, each equipped with a unique type (let $\tau(X)$ be the type of X). The set $T_\tau(\Sigma)$ of *preterms* $s,\ t,\ u,\ \ldots,$ *of type* τ *on the signature* Σ is defined inductively by the rules:

$$\frac{\tau(X) = \tau}{X \in T_\tau(\Sigma)} \quad \frac{\tau(x) = \tau}{x \in T_\tau(\Sigma)} \quad \frac{\Sigma(c) = \tau}{c \in T_\tau(\Sigma)}$$

$$\frac{s \in T_{\tau_1 \to \tau_2}(\Sigma) \quad t \in T_{\tau_1}(\Sigma)}{st \in T_{\tau_2}(\Sigma)} \quad \frac{\tau(x) = \tau_1 \quad t \in T_{\tau_2}(\Sigma)}{\lambda x \cdot t \in T_{\tau_1 \to \tau_2}(\Sigma)}$$

Abbreviate $(\ldots((st_1)t_2)\ldots t_n)$ as $st_1 \ldots t_n$, and $\lambda x_1 \cdot \lambda x_2 \cdot \ldots \lambda x_n \cdot t$ as $\lambda x_1, \ldots, x_n \cdot s$.

The set of λ-*terms* of type τ is the set of all preterms in $T_\tau(\Sigma)$ whose free variables are existential. (This does not restrict generality, as we can always add λs in front in order to bind all universal variables.) A *ground* λ-term has no free existential variable.

All λ-terms that are α-equivalent (i.e., differ only in the name of bound variables) will be dealt with as though they were equal, using Barendregt's naming convention [4]. We consider the following rewrite rules:

$$(\beta)\ (\lambda x \cdot s)t \to s[x := t] \qquad\qquad (\eta)\ \lambda x \cdot tx \to t\ (x \text{ not free in } t)$$

where $s[x := t]$ denotes the standard capture-avoiding substitution. We write $\to_\beta$, $\to_\eta$, $\to_{\beta\eta}$ the corresponding one-step rewrite relations; if $\to$ is a rewrite relation, we write $\to^*$ its reflexive-transitive closure, $\to^+$ its transitive closure. We write $\approx_\beta$, $\approx_\eta$, $\approx_{\beta\eta}$ for the appropriate congruences.

The relations $\to_\beta$, $\to_\eta$, $\to_{\beta\eta}$ terminate on simply-typed terms [14]. Moreover, any (β)-normal preterm if of the form $\lambda x_1, \ldots, x_n \cdot ht_1 \ldots t_m$, where the *head* h is a constant, an existential variable or one of $x_1, \ldots, x_n$, and $t_1, \ldots, t_m$ are (β)-normal. If h is an existential variable, then $\lambda x_1, \ldots, x_n \cdot ht_1 \ldots t_m$ is *flexible*, otherwise it is *rigid*.

Define the η-long normal form $\eta_\tau[t]$ of any (β)-normal preterm $t \in T_\tau(\Sigma)$ by

$$\eta_\tau[\lambda x_1, \ldots, x_n \cdot ht_1 \ldots t_m] \mathrel{\hat{=}} \lambda x_1, \ldots, x_n, x_{n+1}, \ldots, x_p \cdot \\ h\ \eta_{\tau'_1}[t_1] \ldots \eta_{\tau'_m}[t_m]\ \eta_{\tau_{n+1}}[x_{n+1}] \ldots \eta_{\tau_p}[x_p]$$

where $\tau = \tau_1 \to \ldots \to \tau_n \to \tau_{n+1} \to \ldots \to \tau_p \to b$, b a base type, $x_{n+1}, \ldots, x_p$ are fresh universal variables of types $\tau_{n+1}, \ldots, \tau_p$ respectively, and t_1 has type $\tau'_1, \ldots, t_m$ has type τ'_m. Then it is well-known [16] that any two λ-terms of the same type are $\beta\eta$-equal if and only if they have identical η-long β-normal forms; and that if s is η-long β-normal, and σ is a substitution mapping variables to η-long β-normal terms of the same types, then the η-long β-normal form of $s\sigma$ is its β-normal form (no need to perform η-expansion). This allows us to reason on η-long β-normal forms only, reasoning up to (β)-reduction and ignoring the (η)-rule entirely. From now on, we shall even abuse language and take terms to denote their η-long β-normal forms. In particular, when we talk about a variable x of type τ, we really mean $\eta_\tau[x]$.

Higher-order unification [16] is the following problem: given two λ-terms of the same type, find whether there is a substitution σ mapping variables to λ-terms of the same types such that $s\sigma \approx_{\beta\eta} t\sigma$. By the above remarks, taking s and t to be η-long β-normal, and restricting ourselves to substitutions mapping variables to η-long β-normal terms, it is equivalent to ask that $s\sigma$ and $t\sigma$ have the same β-normal form.

Miller's *patterns* are λ-terms where existential variables are only applied to distinct universal variables. For example, $\lambda x_1, x_2, x_3 \cdot Xx_3x_1$ is a pattern, but $\lambda x_1, x_2, x_3 \cdot X(Xx_3x_1)$ and $\lambda x_1, x_2, x_3 \cdot Xx_1x_1x_2$ are not. It is well-known that higher-order unification of patterns is decidable in polynomial time, and that there is a most general unifier (*mgu*) if any unifier exists at all [19].

For convenience, we shall adopt Snyder and Gallier's convention [23] that $\overline{s}_m$ abbreviates the sequence $s_1 s_2 \ldots s_m$, or $s_1, s_2, \ldots, s_m$ depending on context. If π is a one-to-one mapping from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$, write $\overline{s}_{|\pi}$ the sequence $s_{\pi(1)} s_{\pi(2)} \ldots s_{\pi(k)}$.

To define higher-order automata, we shall need patterns that are not too deep:

Definition 1. *A* variable pattern *is a λ-term of the form $\lambda \overline{x}_m \cdot X \overline{x}_{|\pi}$, where π is a one-to-one mapping from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$.*

A shallow pattern *is either a variable pattern, or a* rigid shallow pattern*, i.e., a pattern of the form $\lambda \overline{x}_m \cdot h \overline{u}_n$, with h rigid, and where for every i, $1 \leq i \leq n$, $\lambda \overline{x}_m \cdot u_i$ is a variable pattern.*

The value of shallow patterns is given by Lemma 2 below. The following two lemmas are mechanical consequences of Miller's unification algorithm [19].

Lemma 1. *Let E be a finite set of pairs (s_i, t_i) of terms of the same type, $1 \leq i \leq n$. If every s_i and every t_i is a variable pattern, then the simultaneous mgu of each pair, if any, maps variables to variable patterns.*

For example, the mgu of $(\lambda x_1, x_2, x_3 \cdot X x_3 x_1, \lambda x_1, x_2, x_3 \cdot X x_2 x_1)$ is $[X := \lambda y_1, y_2 \cdot Y y_2]$ where Y is a fresh free existential variable, and the mgu of $(\lambda x_1, x_2, x_3 \cdot X x_3 x_1, \lambda x_1, x_2, x_3 \cdot X' x_1 x_2)$ (where X and X' are distinct) is $[X := \lambda y_1, y_2 \cdot Y y_2, X' := \lambda y_1, y_2 \cdot Y y_1]$.

Lemma 2. *The mgus, if any, of two shallow patterns s and t are substitutions mapping variables to shallow patterns. Moreover, if both s and t are variable patterns, or both of them are rigid shallow patterns, then the mgus maps variables to variable patterns.*

For example, the mgu of $(\lambda x_1 \cdot x_1(\lambda x_2, x_3 \cdot X x_3 x_1)(\lambda x_2, x_3 \cdot X x_3 x_1), \lambda x_1 \cdot x_1(\lambda x_2, x_3 \cdot X x_2 x_1)(\lambda x_2, x_3 \cdot X' x_1 x_2))$ is $[X := \lambda y_1, y_2 \cdot Y y_2, X' := \lambda y_1, y_2 \cdot Y y_1]$.

3 Ordered Resolution in a First-Order Logic of Higher-Order Patterns

The technical tool we shall use in the sequel is resolution in a first-order logic with higher-order patterns. This is in the spirit of Joyner [17]. Although it would be possible to define a Tarskian semantics for this logic (use domains of individuals indexed by types, forming a Henkin applicative structure [2], then build a Tarskian semantics for first-order formulas atop these domains), we shall only be interested in Herbrand semantics here, where the domain of individuals of type τ is the set of ground λ-terms of type τ, up to $\beta\eta$-conversion—alternatively, the set of η-long β-normal forms of type τ. In fact, we will only consider clausal formats, where existential quantifiers are absent and universal quantifiers are implicit.

As far as syntax is concerned, fix a set of *predicate symbols* P, Q, R, ..., each coming with an *arity*, which is a sequence of types. (Sometimes we shall

call arity, ambiguously, just the number of arguments to predicates, constants or variables.) The *atoms* A, B, ..., are $P(t_1, \ldots, t_n)$, where P is a predicate symbol of arity $\tau_1, \ldots, \tau_n$, t_1 is a λ-term of type τ_1, ..., t_n is a λ-term of type τ_n. *Literals* L are either atoms A or negations of atoms $\neg A$. We also write $+A$ for A, $-A$ for $\neg A$. *Clauses* C are finite disjunctions $L_1 \vee \ldots \vee L_p$ of literals. *Clause sets* S are conjunctions of clauses (possibly infinite, although our interest is in finite ones).

The semantics is as follows. Let the *Herbrand universe of type* τ, D_τ, be the set of all ground η-long normal forms of type τ. A *Herbrand interpretation* I is just a set of ground atoms. Herbrand interpretations are ordered by inclusion. A *valuation* ρ, giving values to each variable, is a substitution mapping each variable of type τ to a ground term of type τ (that is, in D_τ). The *value* of a term t under ρ is $t\rho$. We define the satisfaction relations $\models$ by:

$$I, \rho \models P(t_1, \ldots, t_n) \text{ iff } P(t_1\rho, \ldots, t_n\rho) \in I \tag{1}$$

$$I, \rho \models \neg A \text{ iff } I, \rho \not\models A \tag{2}$$

$$I \models L_1 \vee \ldots \vee L_p \text{ iff for every } \rho, \text{ for some } i,\ 1 \leq i \leq p,\ I, \rho \models L_i \tag{3}$$

$$I \models S \text{ iff for every } C \text{ in } S,\ I \models C \tag{4}$$

We say that a clause set S is *satisfiable* if and only if $I \models S$ for some Herbrand interpretation I. It is *unsatisfiable* otherwise.

Let us now restrict the set of terms we consider to higher-order patterns, so that every pair s, t of terms has exactly one mgu, as soon as they unify. Denote this mgu by $\mathrm{mgu}(s, t)$. Define the *resolution rule* [6] by:

$$\text{(Binary resolution)}\ \frac{C \vee A \qquad \neg A' \vee C'}{(C \vee C')\sigma}\ \sigma = \mathrm{mgu}(A, A')$$

$$\text{(Factoring)}\ \frac{C \vee L \vee L'}{(C \vee L)\sigma}\ \sigma = \mathrm{mgu}(L, L')$$

where it is understood that, in binary resolution, the clauses $C \vee A$ and $\neg A' \vee C'$ are renamed so that they have no common free existential variable, and in factoring, two literals L and L' unify provided they have the same signs and the underlying atoms unify. Resolution is a *sound* deduction calculus, in the sense that if we can derive the empty clause $\Box$ from S by resolution, then S is unsatisfiable. In fact, every conclusion of the rules above is logically implied by the premises.

An ordering $>$ on η-long β-normal atoms is *stable* if and only if $A > B$ implies that the β-normal form of $A\sigma$ is greater than, in the sense of $>$, to the β-normal form of $B\sigma$, for every well-typed substitution σ mapping variables to η-long β-normal forms. *Ordered resolution* is the refinement of resolution where: in binary resolution, A is a $>$-maximal atom in $C \vee A$, A' is a $>$-maximal atom in $\neg A' \vee C'$; in factoring, letting $L \hat{=} \pm A$, then A is $>$-maximal in $C \vee L \vee L'$. The following is standard.

Proposition 1 (Completeness). *Ordered resolution w.r.t. $>$ is* complete*, provided that $>$ is stable. That is, given a finite set S of clauses, S is unsatisfiable if and only if the empty clause $\Box$ can be derived from S by ordered resolution.*

Proof. The if direction is soundness. Conversely, assume S unsatisfiable. Then by construction the (in general infinite) set S_0 of ground instances of clauses in S is unsatisfiable. This is equivalent to the fact that S_0 is propositionally unsatisfiable. By the compactness of propositional logic, S_0 contains a finite unsatisfiable subset S_1. Since propositional ordered resolution is complete, there is a propositional ordered resolution deduction of $\Box$ from S_1. This can then be lifted to a corresponding ordered resolution deduction of $\Box$ from S. $\Box$

Similarly, every form of, say, hyper-resolution is complete.

Given a clause C, we say that C is a *block* if and only if every pair of atoms in C has a common free existential variable. We can always write clauses as a disjunction $B_1 \vee \ldots \vee B_k$ of non-empty blocks that pairwise do not share any free existential variable. Moreover, this decomposition is unique [17].

In our decision procedures, we shall use an additional rule to split clauses into their blocks (on the right),

$$\text{(Splitting)} \qquad \frac{C \vee C'}{C \mid C'}$$

where C and C' do not share any free existential variable, and are non-empty. This means that we shall split the current set of clauses in 2 sets, adding C to the first, and C' to the second. In other words, we define a tableau calculus in the following way. A *branch* is a finite clause set, and a *tableau* is a finite set of branches. A branch is *closed* if and only if it contains the empty clause $\Box$. A tableau is *closed* if and only if all its branches are closed. We read a tableau as the disjunction of its branches.

As far as deduction is concerned, our tableau rules are as follows. We may either add a new clause to some branch by using resolution on this branch, or use splitting to replace some branch S of the tableau such that S contains $B_1 \vee \ldots \vee B_k$ by k branches $S \cup \{B_1\}, \ldots, S \cup \{B_k\}$. We write $T \Longrightarrow T'$ if we can go from tableau T to T' by applying one of these rules. This calculus is clearly sound, in the sense that if $T \Longrightarrow T'$ then T implies T'. So, if we can close some tableau by these deduction rules, then it is unsatisfiable: no Herbrand interpretation satisfies any of its branches. It is also clear that this tableau calculus is complete, even under some stable ordering restriction, because already the calculus without splitting is (Proposition 1).

Splitting can in fact be applied eagerly. That is, we may use the resolution rule on just those branches that contain only blocks without losing completeness. While this is well-known (see e.g., [10]), let us say quickly that the reason is that, just like Joyner's less efficient rule of *condensation*, ordered resolution, even in our higher-order setting, is complete by *semantic trees* [17]. If S is unsatisfiable, there is a finite closed semantic tree based on a finite subset of the Herbrand universe (the set of closed atoms); completeness of ordered resolution follows because this closed semantic tree is finite and can be shrinked by adding ordered resolvents between clauses that fail at leaves of the tree (see [6] for details). Since

splitting replaces a failed clause C by subclauses that are failed at or above C, completeness is preserved.

4 Higher-Order Automata, Pushdown Systems and Positive Set Constraints

We define higher-order automata, higher-order pushdown systems, and higher-order positive set constraints as particular sets of clauses. The idea can be traced to [12].

We consider clauses built from unary predicate symbols and shallow patterns. Consider first Horn clauses of the form:

$$P_1(\lambda \overline{y}^1_{n_1} \cdot X_1 \overline{y}^1_{|\pi_1}), \ldots, P_k(\lambda \overline{y}^k_{n_k} \cdot X_k \overline{y}^k_{|\pi_k}) \supset P(\lambda \overline{x}_m \cdot h\overline{u}_n) \tag{5}$$

where $\overline{u}_n$ are variable patterns, h is rigid, and every X_i, $1 \leq i \leq k$, is free in $P(\lambda \overline{x}_m \cdot h\overline{u}_n)$. In the first order case, i.e. when $n_1 = \ldots = n_k = 0$, and if each u_i is X_i, (5) simplifies to

$$P_1(X_1), \ldots, P_k(X_k) \supset P(h(X_1, \ldots, X_k)) \tag{6}$$

This is a transition of a tree automaton: if t_1 is recognized at state P_1, ..., and t_n is recognized at state P_k, then $h(t_1, \ldots, t_k)$ is recognized at state P.

It seems that people familiar with classical automata theory are puzzled by this definition, and in particular by the fact that no definition of a *run* of a term against an automaton is given; we invite the puzzled reader to check that *positive hyper-resolution* derivations [6] (which are also unit derivations in the case of Horn clauses) are exactly *bottom-up runs* [13]: for every ground term $P(t)$, the positive hyper-resolution derivations of the unit clause $P(t)$ are exactly the runs of t that abut to state P, against the given tree automaton, considered bottom-up. On the other hand, *negative hyper-resolution* derivations are exactly the top-down runs. The theory of resolution theorem proving enables us to replace any complete deduction procedure (positive, negative hyper-resolution) by any other complete procedure; it seems that ordered resolution is the most powerful refinement of resolution in many practical cases.

Returning to clauses of the form (6), in case some X'_i occurs twice as an argument of h, we get *tree automata with equality constraints between brothers* [5]. The higher-order case enables us to write two arguments to h with the same head X'_i, but with permuted arguments to X'_i; for example, we may write transitions such as:

$$P_1(X_1) \supset P(\lambda x_1, x_2 \cdot h(X_1 x_1 x_2)(X_1 x_2 x_1))$$

which means that to be recognized at t, the term $\lambda x_1, x_2 \cdot h t_1 t_2$ should be such that $t_1 = t_2[x_1 := x_2, x_2 := x_1]$. This properly generalizes equality constraints.

In case some X'_i does not occur on the left-hand side of the implication, then X'_i is a don't care: any term of the same type as X'_i can instantiate X'_i. In the first-order case, it is always possible to describe the set of all terms by an

automaton, and this would provide no added expressive power. In the higher-order case, these don't cares are a proper extension of tree automata, since there is no automaton recognizing all (ground) terms of a given type [9].

Sets of clauses of the form (5) will be called *higher-order automata*.

These can be enriched by, say, Horn clauses of the form:

$$P_1(\lambda\overline{y}^1_{n_1} \cdot X\overline{y}^1_{|\pi_1}), \ldots, P_k(\lambda\overline{y}^k_{n_k} \cdot X\overline{y}^k_{|\pi_k}) \supset P(\lambda\overline{y}_n \cdot X\overline{y}_{|\pi}) \tag{7}$$

with the same variable X in each atom. This corresponds to clauses of the form $P_1(X), \ldots, P_k(X) \supset P(X)$ in the first-order case. If $k = 1$, these are ϵ-transitions ("every term recognized at state P_1 must be recognized at state P, too"). If $k \geq 2$, we get conjunctive transitions ("if a term is recognized at states $P_1, \ldots, P_k$ simultaneously, then it must be recognized at P"). Disjunctive transitions are handled naturally by having several ϵ-transitions reach the same state.

Sets of clauses of the form (5) or (7) will be called *alternating higher-order automata*. Notice again that the use of one-to-one mappings that shuffle bound variables around allows us to do a few more tricks than just intersections and unions.

Third, we may also consider Horn clauses of the form:

$$P(\lambda\overline{x}_m \cdot h\overline{u}_n) \supset P_1(\lambda\overline{y}_n \cdot X\overline{y}_{|\pi}) \tag{8}$$

where X is free in the rigid shallow term $\lambda\overline{x}_m \cdot h\overline{u}_n$. In the first-order case, this would simplify to $P(h(X_1, \ldots, X_n)) \supset P_1(X_i)$: this is a *pushdown transition*, which allows us to state that if some functional term $h(X_1, \ldots, X_n)$ is recognized at state P, then its ith argument must be recognized at state P_1. Again, the use of bound variables allows us to state slightly more in the higher-order case.

In general, we consider clauses of the following form:

Definition 2. *An* automatic clause *is any clause of the form*

$$\neg P_1(t_1) \vee \ldots \vee \neg P_m(t_m) \vee P_{m+1}(t_{m+1}) \vee \ldots \vee P_n(t_n) \tag{9}$$

where $0 \leq m \leq n$, *and* t_i, $1 \leq i \leq n$, *are shallow patterns such that:*

(*i*) *if every* t_i *is a variable pattern, then they all have the same head, say* X*;*
(*ii*) *otherwise, all the* t_i*'s that are not variable patterns are rigid shallow patterns* $\lambda\overline{x_i}_m \cdot h_i\overline{u_i}_n$*, which contain every free existential variable in the clause.*

In the first case, we call the clause an ϵ-block*. In the second case, it is a* complex clause*.*

A higher-order pushdown system *is a finite set of Horn automatic clauses. Finite sets of (non-Horn) automatic clauses are called* higher-order positive set constraints*.*

The reason why finite sets of automatic clauses are called higher-order set constraints is by analogy with the first-order case [3]. (Ordinary, first-order) set

constraints are defined as follows. Let the *set expressions* be defined by the grammar:

$$e ::= \xi \mid 0 \mid 1 \mid e \cap e \mid e \cup e \mid \complement e \mid f(e_1, \ldots, e_n) \mid f_i^{-1}(e)$$

where f ranges over all function symbols (of arity n), and ξ ranges over a set of so-called *set variables*. In expressions of the form $f_i^{-1}(e)$, we require $1 \leq i \leq n$. Each set expression is interpreted, under a valuation that maps each set variable to a set of ground terms, as a set of ground terms. $\complement e$ denotes the complement of e, $f(e_1, \ldots, e_n)$ denotes the set of terms $f(t_1, \ldots, t_n)$ where t_1 is in e_1, ..., t_n is in e_n, and $f_i^{-1}(e)$ denotes the set of terms t_i such that $f(t_1, \ldots, t_i, \ldots, t_n)$ is in e for some terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.

The *elementary constraints* are of the forms listed in the first column of the table on the right. Their translation as clauses is given in the second column—this may in fact be taken as the semantics of set constraints.

Set constraint	Automatic clause
$\xi \subseteq \eta$	$-\xi(X) \vee +\eta(X)$
$\xi \subseteq \eta \cup \zeta$	$-\xi(X) \vee +\eta(X) \vee +\zeta(X)$
$\xi \cap \eta \subseteq \zeta$	$-\xi(X) \vee -\eta(X) \vee +\zeta(X)$
$\xi \subseteq \complement\eta$	$-\xi(X) \vee -\eta(X)$
$\complement\xi \subseteq \eta$	$+\xi(X) \vee +\eta(X)$
$\xi \subseteq f(\xi_1, \ldots, \xi_n)$	$\begin{cases} -\xi(f(X_1, \ldots, X_n)) \vee +\xi_1(X_1) \\ \ldots \\ -\xi(f(X_1, \ldots, X_n)) \vee +\xi_n(X_n) \\ -\xi(g(X_1, \ldots, X_m)) \quad \text{(for all } g \neq f) \end{cases}$
$f(\xi_1, \ldots, \xi_n) \subseteq \xi$	$\bigvee_{i=1}^{n} -\xi_i(X_i) \vee +\xi(f(X_1, \ldots, X_n))$
$f_i^{-1}(\xi) \subseteq \eta$	$-\xi(f(X_1, \ldots, X_n)) \vee +\eta(X_i)$

This format of set constraints is *positive*—only inclusions $\subseteq$ can be dealt with, not negated inclusions $\not\subseteq$—and handles projections f_i^{-1} only partially—constraints $\xi \subseteq f_i^{-1}(\eta)$ require an extension of our format. This is just as in the first-order case investigated in [3]. Dealing with negative constraints and projections can be done by adding special constraints expressing that some variables ξ must be non-empty. This can be dealt with in a resolution format by considering clauses with additional *rigid existential* variables, which can be instantiated only once, just like the variables used in V-resolution [6] or in ordinary free-variable tableaux [11]; this will be treated elsewhere.

Finally, note that (higher-order) pushdown processes are just the higher-order analogue of *definite* set constraints.

4.1 Deciding Satisfiability of Higher-Order Positive Set Constraints

We now show that the satisfiability of higher-order positive set constraints is decidable.

To this end, we first need a stable ordering $>$ on shallow patterns such that any rigid shallow pattern s with X free in it is strictly greater than any variable pattern $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ with head X. (This is the natural extension of the subterm ordering in the first-order case.) Take $s > t$ if and only if the *rigid depth* $d(s)$ is greater than $d(t)$, where rigid depth is defined by: $d(\lambda\overline{x}_m \cdot h\overline{t}_n) =$

$1 + \max_{1 \leq i \leq n} d(t_i)$ if h is rigid (the maximum being 0 in case $n = 0$), and zero if h is a free existential variable. In the sequel, we shall do ordered resolution w.r.t. $>$, as defined in Section 3.

Lemma 3. *Every factor of an automatic clause is an automatic clause.*

Proof. Consider the clause $C \vee P(t) \vee P(t')$, and its factor $C\sigma \vee P(t\sigma)$, where $\sigma \hat{=} \mathrm{mgu}(t, t')$. (The case $C \vee \neg P(t) \vee \neg P(t')$ is entirely analogous.) If one of t, t' is a variable pattern, say with head X, and the other is a rigid shallow pattern, then by condition (ii) X is free in the rigid shallow term, hence this case is impossible (while this occurs-check test is correct in unifying higher-order patterns, it would not be in general higher-order unification). So by Lemma 2 σ maps variables to variable patterns. Therefore, the factor is an ϵ-block if the original clause was, and it is a complex clause otherwise. □

Lemma 4. *Every ordered binary resolvent of automatic clauses is either an automatic clause or a disjunction of ϵ-blocks that pairwise do not share free variables.*

Proof. Consider two automatic clauses $C \vee P(t)$ and $\neg P(t') \vee C'$.

If t and t' are both variable patterns, or both rigid shallow patterns, then the mgu σ if any of t and t' maps variables to variable patterns by Lemma 2. If C or C' contains any non-variable pattern at all, then it is easy to check that $(C \vee C')\sigma$ is a complex clause. Otherwise, $(C \vee C')\sigma$ is a disjunction of literals $\pm P(u)$ where u is a variable pattern, hence can be written as a disjunction of ϵ-blocks that pairwise do not share free variables. It may be the case that we do not have just a single ϵ-block, e.g. already in the first-order case, resolving on $-P_1(X_1) \vee -P_2(X_2) \vee +P(f(X_1, X_2))$ and $-P(f(X_1, X_2)) \vee +P_3(X_1)$ yields $-P_1(X_1) \vee -P_2(X_2) \vee +P_3(X_1)$.

If t is a variable pattern $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ and t' is a rigid shallow pattern $\lambda\overline{x}_m \cdot h\overline{u}_n$, then the mgu σ, if any, of t and t' maps X to some rigid shallow pattern $\lambda\overline{x}_m \cdot h\overline{v}_n$, and the free variables in C' to variable patterns, as shown in the proof of Lemma 2. Examining carefully this proof reveals that, additionally, each free variable of t' occurs as the head of some v_i, $1 \leq i \leq n$. Since $\neg A' \vee C'$ is a complex clause, by (ii) $X\sigma$ not only has head h, but also contains the heads of every v_i, $1 \leq i \leq n$, therefore every free variable of $C'\sigma$. Moreover, since t is a variable pattern, by the ordering condition every atom in C has a variable pattern as argument, so by (i) C is an ϵ-block. It follows that every literal of $C\sigma$ is of the form $\pm P(t)$ with t some rigid shallow pattern with head h containing every free variable of $C'\sigma$. So, if C is not empty or if C' contains some rigid shallow pattern, then the resolvent $(C \vee C')\sigma$ is a complex clause. Otherwise, it is trivially a disjunction of ϵ-blocks that pairwise do not share free variables, as above. The case where t' is a variable pattern and t is a rigid shallow pattern is analogous. □

Lemma 5. *Up to renaming of free existential variables, there are only finitely many automatic clauses on any given finite set of predicate symbols and constants.*

Proof. Let p be the number of predicate symbols, k the number of constants. There is an upper bound α on the number n such that $\tau_1 \to \ldots \to \tau_n \to b$ is a subtype of the arity of predicate symbols, or of the types of constants.

Let us first compute an upper bound $\psi(\alpha)$ on the number $\psi_X(m)$ of variable patterns $\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}$ with head X. Letting X apply to at most k arguments, $\psi_X(m)$ is the number of one-to-one functions from $\{1, \ldots, k\}$ to $\{1, \ldots, m\}$, namely $m!/(m-k)!$. This is always at most $m!/(\lfloor m/2 \rfloor)! \leq m^m \leq \alpha^\alpha$. That is, we may take $\psi(\alpha) = \alpha^\alpha$.

Then there are at most $4^{p\psi(\alpha)}$ ϵ-blocks, and at most $16^{p(\alpha+k)(\alpha\psi(\alpha))^\alpha}$ complex clauses. So there are only finitely many automatic clauses: their number is at most doubly exponential in α, and simply exponential in p and k. □

Because η-long forms have size greater than or equal to p, k and α, it follows that:

Theorem 1 (Decidability). *The satisfiability of higher-order positive set constraints is decidable in 2-NEXPTIME.*

4.2 Subcases of Smaller Complexity

A first slightly less complex subcase is that of *extended unary* higher-order positive set constraints, i.e., when all automatic clauses in S have at most one free existential variable. This includes the case of *unary* higher-order positive set constraints, where all rigid heads have arity at most 1. In turn this generalizes the case of unary set constraints [1] to the higher-order case.

Theorem 2. *The satisfiability of extended unary higher-order positive set constraints is decidable in* deterministic *double exponential time (2-DEXPTIME).*

Proof. Ordered resolution only produces clauses with at most one free existential variable again. Then splitting never occurs. □

Another remarkable subcase is that of *alternation-free* higher-order positive set constraints: this is defined as the case where every ϵ-block has at most 2 literals, and in every complex clause, every free existential variable X occurs at most once in some rigid shallow pattern, and at most once in some variable pattern. For example,

$$-P(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi}) \vee +Q(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi'}) \tag{10}$$

$$-P_1(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_2}) \tag{11}$$

$$\begin{aligned} -P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{y}_{m_2} \cdot X_2\overline{y}_{|\pi_2}) \\ \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1})(\lambda\overline{y}'_{m'_2} \cdot X_2\overline{z}_m\overline{y}'_{m'_2})) \end{aligned} \tag{12}$$

with $X_1 \neq X_2$, are alternation-free, but the following are not:

$$P_1(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_1}), P_2(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi_2}) \supset Q(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi'}) \tag{13}$$

$$-P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee -P_2(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_2}) \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1}) \tag{14}$$

$$-P_1(\lambda\overline{x}_{m_1} \cdot X_1\overline{x}_{|\pi_1}) \vee +Q(\lambda\overline{z}_m \cdot h(\lambda\overline{x}'_{m'_1} \cdot X_1\overline{z}_m\overline{x}'_{m'_1})(\lambda\overline{y}'_{m'_1} \cdot X_1\overline{z}_m\overline{y}'_{m'_1}) \tag{15}$$

Theorem 3. *The satisfiability of alternation-free higher-order positive set constraints is decidable in* deterministic *double exponential time (2-DEXPTIME).*

Proof. We first show that every clause that we get by resolution with eager splitting is alternation-free. Resolving two ϵ-blocks of at most 2 literals yields again an ϵ-block of at most 2 literals. When we resolve two complex clauses, alternation-freeness implies that these clauses are of the form:

$$\pm_1 P_1(\lambda\overline{x}^1_{n_1} \cdot X_1\overline{x}^1_{|\pi_1}) \vee \ldots \vee \pm_k P_k(\lambda\overline{x}^k_{n_k} \cdot X_k\overline{x}^k_{|\pi_k}) \vee +P(\lambda\overline{x}_m \cdot h\overline{u}_n) \quad (16)$$

$$\pm'_1 P'_1(\lambda\overline{y}^1_{n'_1} \cdot X'_1\overline{y}^1_{|\pi'_1}) \vee \ldots \vee \pm'_{k'} P'_{k'}(\lambda\overline{y}^{k'}_{n'_{k'}} \cdot X'_{k'}\overline{y}^{k'}_{|\pi'_{k'}}) \vee -P(\lambda\overline{x}_m \cdot h\overline{u}'_n) \quad (17)$$

Then the mgu σ of the rigid shallow patterns $\lambda\overline{x}_m \cdot h\overline{u}_n$ and $\lambda\overline{x}_m \cdot h\overline{u}'_n$ maps free variables X of the first clause to variable patterns, in such a way that no two free variables are mapped to variable patterns with the same head. The resolvent then splits as ϵ-blocks with at most two literals ($\pm_i P_i(\ldots) \vee \pm'_j P'_j(\ldots)$ if $X_i\sigma$ has head X'_j, $\pm_i P_i(\ldots)$ if $X_i\sigma$ has a head that is none of the X'_js, $\pm'_j P'_j(\ldots)$ if X'_j is the head of no $X_i\sigma$).

Finally, when we resolve an ϵ-block C with a complex clause, say (16) (the case (17) is symmetric) then either C is of the form $-P(\lambda\overline{x}_m \cdot X\overline{x}_{|\pi})$, so the resolvent splits as unit clauses $\pm_i P_i(\lambda\overline{x}^i_{n_i} \cdot X_i \ldots)$; or C is a 2-literal ϵ-block, then the resolvent is again an alternation-free complex clause.

Now splitting only produces ϵ-blocks with at most two literals, and there are only exponentially many of them. Provided we remove subsumed clauses [6], this means that every branch of the tableau only splits exponential many times. Moreover, every split produces only polynomially many clauses. Implementing this double exponential time procedure with exponentially many splits by backtracking on a deterministic machine then produces a 2-DEXPTIME algorithm. □

A final 2-DEXPTIME subcase is the Horn case, which we explore in the next section. We believe that all the upper bounds we have given in this paper are tight.

4.3 Deciding Emptiness of Higher-Order Pushdown Systems, Or: the Horn Case

Let S be a higher-order pushdown system. Since S is a set of Horn clauses, if S has a model—a Herbrand interpretation I such that $I \models S$—then it has a least one. The argument is standard: if $(I_i)_{i\in\mathcal{I}}$ is a family of models, then $\bigcap_{i\in\mathcal{I}} I_i$ is a model again. Notice that if S is a set of *definite clauses*—with exactly one positive atom—, then S is satisfiable: the Herbrand interpretation containing every ground term is a model.

Definition 3 (Language). *Given a satisfiable higher-order pushdown system S, and a finite set F of unary predicates $P_1, \ldots, P_n$ (the* final states*), the* language $\mathcal{L}(S;F)$ *defined by S and F is the set of ground terms t such that $P_i(t)$ is in the least model of S for some i, $1 \leq i \leq n$.*

This language can be generated as the set of unit clauses that are ground instances of unit clauses $P_i(t)$ obtained by positive hyper-resolution (equivalently, by Prolog's T_P operator). Recall that positive hyper-resolution derivations are just bottom-up runs of the automaton S.

It is easy to see that $\mathcal{L}(S;F)$ is empty if and only if S plus the clauses $-P_1(X), \ldots, -P_n(X)$ is satisfiable (where X stands for its η-long form): if it is satisfiable, then its least model satisfies $-P_i(X)$ for every i, hence cannot contain any ground term of the form $P_i(t)$, $1 \leq i \leq n$. Conversely, if $\mathcal{L}(S;F)$ is empty then no atom $P_i(t)$ is in its least Herbrand model, i.e., this model satisfies all the clauses $-P_i(X)$.

Theorem 4. *The satisfiability of sets of Horn automatic higher-order clauses is decidable in 2-DEXPTIME.*

Proof. Recall that a *negative* clause is a non-empty clause containing no atom of sign + (i.e., its head is false). Let S be any fixed set of non-negative Horn automatic higher-order clauses. This is satisfiable: let I be its least Herbrand model, and let S^- be the set of all negative ϵ-blocks C (on the given signature) such that $S \cup \{C\}$ is unsatisfiable—equivalently, such that $I \not\models C$. We may compute S^- by noticing first that any splitting in any derivation from $S \cup \{C\}$, $C \in S^-$, must produce one non-negative ϵ-block C_0 (possibly $\Box$), plus negative ϵ-blocks $C_1, \ldots, C_n$. Branches stemming from C_i, $1 \leq i \leq n$, will be unsatisfiable iff $I \not\models C_i$, iff $C_i \in S^-$ (or $C \in S^-$). Consider then the S'*-splitting* rule, for each set S' of negative ϵ-blocks: this derives C_0 when splitting would derive the non-negative ϵ-block C_0 and the negative ϵ-blocks $C_1, \ldots, C_n$, provided the latter are in S'; otherwise it does not apply. Let $F(S')$ be the set of negative ϵ-blocks C that are either in S' or such that ordered resolution with S'-splitting derives $\Box$ from $S \cup \{C\}$. The discussion above shows that S^- is a fixpoint of F. In fact it is easy to see that S^- is the least fixpoint of F.

Let N be an upper bound on the number of Horn automatic clauses (this is doubly exponential). Since S^- contains at most N clauses, this fixpoint can be computed in at most N calls to F. F tests whether $S \cup \{C\}$ is unsatisfiable for at most N clauses C. Then this test is deterministic since S'-splitting is, and proceeds by generating at most N clauses. Hence computing S^- can be done in time $O(N^3)$.

Now given any set S_0 of Horn automatic higher-order clauses, by the same token ordered resolution with S^--splitting is complete, where S^- is computed as above (in a preprocessing step) from the subset S of non-negative clauses in S_0. This takes additional time $O(N)$. $\Box$

Corollary 1. *Given a satisfiable higher-order pushdown system S, and states $P_1, \ldots, P_n$, it is decidable in 2-DEXPTIME whether the language of $(S; P_1, \ldots, P_n)$ is empty.*

Again, recall that this is only simply exponential in the number p of predicate symbols and k of constants, when α is fixed.

In the first-order case every pushdown process is equivalent to (recognizes the same language) as an ordinary tree automaton that we may compute in exponential time. Analogously, every satisfiable set S of Horn automatic clauses is equivalent to an *up-tree automaton*, that is, a set of *up-clauses*, of the form (5) or of the form $+P(\lambda\overline{x}^m \cdot X\overline{x}_{|\pi})$: saturate S by ordered resolution with S^--splitting as in the proof of Theorem 4, getting a set S' of clauses, then keep only the up-clauses in S'. This rests on the fact that whenever there is a positive hyperresolution derivation of the unit clause $P(t)$ from S', there is a positive hyperresolution derivation of some unit clause $P(s)$ using only *up-clauses* from S', with t an instance of s. (Exercise, using induction on the length of the derivation. Hint: take the first resolution step with some non-up-clause C_2; this must be preceded by a resolution step with some up-clause C_1; then this sequence of two steps may be replaced by one resolution step with a resolvent of C_1 and C_2, and this step derives a more general unit clause in general.)

5 Application: Towards Typing λ-Prolog Programs

Following [12], a natural use of our higher-order format is in computing upper approximations of success sets (*descriptive types*) of λ-Prolog programs [20]. This in fact works also for sets of non-Horn clauses over higher-order terms, but we don't consider this here. On the other hand, we consider for simplicity only a restricted subset of λ-Prolog programs, consisting of Horn clauses instead of general hereditary Harrop formulas, and using only rigid heads. (The fact that our typing discipline is simple types is inessential, since higher-order patterns are actually type-independent [19].) We are confident that the case of general λ-Prolog programs can be reduced to this simpler case, drawing inspiration from early Prolog implementations of λ-Prolog to define a translation from λ-Prolog to Horn clauses operating over typed λ-terms. However, existential quantifications cause some headaches, and probably require some approximation already. We prefer to leave this subject for future work.

Consequently, consider any set S of clauses, i.e., finite disjunctions of atoms $\pm P(t_1, \ldots, t_k)$, where P is a constant predicate of arity $\tau_1, \ldots, \tau_k$, and t_i is any λ-term of type τ_i, $1 \leq i \leq k$. Recall that, in this case at least, the success set of a logic program is its least Herbrand model.

We first make every predicate unary. Let o be a fresh base type. For every predicate P of arity $\tau_1, \ldots, \tau_k$ with $k \neq 1$, create a fresh constant f_P of type $\tau_1 \to \ldots \to \tau_k \to o$, and a fresh unary predicate $\tilde{P}$ of arity o. Then replace every atomic formula $P(u_1, \ldots, u_k)$ by $\tilde{P}(f_P(u_1, \ldots, u_k))$. Clearly Herbrand models of the original set of clauses are in one-to-one correspondence with Herbrand models of the transformed set.

We now define a series of transformations on sets of clauses with only unary predicates. While there is a term t that is not a shallow pattern in some clause $C = (C_0 \vee \pm P(t))$ of S:

1. if t is of the form $\lambda\overline{x}_m \cdot h\overline{u}_n$ with h rigid, x_i of type τ'_i, $1 \leq i \leq m$, and u_j of type τ''_j, $1 \leq j \leq n$, create n fresh unary predicates P_j and n free variables X_j

of respective types $\tau'_1 \to \ldots \to \tau'_m \to \tau''_j$, $1 \leq j \leq n$, replace C by the $1+n$ clauses:

$$C_0 \vee \pm P_1(\lambda \overline{x}_m \cdot u_1) \vee \ldots \vee \pm P_n(\lambda \overline{x}_m \cdot u_n) \tag{18}$$

$$\pm P(t') \vee \mp P_j(X_j) \qquad (1 \leq j \leq n) \tag{19}$$

where t' is the rigid shallow pattern $\lambda \overline{x}_m \cdot h(X_1 \overline{x}_m) \ldots (X_n \overline{x}_m)$, and $\mp$ is the sign opposite to $\pm$ (each occurrence of $\pm$ denoting the *same* sign; recall also that we write terms up to η-expansion for brevity and clarity);

2. if t is of the form $\lambda \overline{x}_m \cdot X \overline{u}_n$ with X a free variable, let $x_{\pi(1)}, \ldots, x_{\pi(k)}$ be the free variables in the sequence $\overline{u}_n$, $1 \leq \pi(1) < \ldots < \pi(k) \leq m$, create a fresh variable Y, then replace C by the clause:

$$C_0 \vee \pm P(\lambda \overline{x}_m \cdot Y \overline{x}_{|\pi}) \tag{20}$$

If S' is obtained as above, write $S \rightsquigarrow S'$. The $\rightsquigarrow$ relation terminates: define a measure of atoms A by $\mu(P(t)) = 0$ if t is a shallow pattern, $\mu(P(t)) = d(t)+1$, the rigid depth of t plus 1, otherwise; define $\mu(\pm_1 A_1 \vee \ldots \vee \pm_n A_n) = \sum_{i=1}^{n} \mu(A_i)$. Then $S \rightsquigarrow S'$ implies that the multiset of all $\mu(C)$, $C \in S$, is greater than that of all $\mu(C)$, $C \in S'$.

It is easy to check that if $S \rightsquigarrow S'$, then S' implies S: in case 1 this is because clause C is a (non-ordered) resolvent of the $1+n$ generated clauses, in case 2 this is because C is an instance of clause (20). The interested reader may check that we can in fact improve slightly on items 1 and 2 above: in item 1 notably, we may produce any set of clauses that together produce C as a resolvent (in particular, we may take $X_i = X_j$ when $u_i = u_j$). The distinctive feature of this process compared to the first-order case is item 2, which involves some unavoidable loss of precision; the corresponding case in first-order Prolog programs would be when t is just a variable X and not a shallow pattern already, which is impossible.

Given any $\rightsquigarrow$-normal form S of a given set of clauses. S may fail to be a higher-order pushdown process: there might be a clause C in S of the form $C_0 \vee L_1 \vee L_2$, where L_1 and L_2 are two literals with distinct but non-disjoint sets of free existential variables. Then replace C by $C_0 \vee L_1 \vee C'_0 \vee L'_2$, where $C'_0 \vee L'_2$ is a renamed version of $C_0 \vee L_2$ whose free existential variables are not free in L_1. This process terminates, and results in a set of Horn clauses that are variable-disjoint disjunctions of automatic clauses. By a slight extension of the remark at the end of Section 4.3, this can be converted to a higher-order up-tree automaton (consisting only of up-clauses) in doubly exponential time. As in the first-order case, this automaton is a good candidate for a descriptive type of the values of free existential variables in succeeding goals.

6 Conclusion

We have defined a natural extension of positive set constraints to the case of higher-order terms. While the main idea is elementary (extend ordered resolution techniques used for the monadic class [17] to a higher-order analogue), one of

the subtleties of the approach is to define a restriction of higher-order terms not only to a subcase where unification is decidable (Miller's patterns fit well), but also where applying most general unifiers to produce resolvents will only produce finitely many clauses.

A nice feature of this approach is that it yields as a by-product a natural notion of higher-order automata (up-automata), and a natural notion of approximation (typing) for the Horn, rigid-head subset of all λ-Prolog programs. This is in the line of [12].

We do not claim that any of these ideas alone is novel, however combining them appears to be new. On the other hand, this suggests numerous extensions. First, to other equational theories $\mathcal{E}$—just use $\mathcal{E}$-unification; this might be deceiving, however, and $\beta\eta$-equality of λ-terms is the only meaningful example that we know of where just ordered resolution with splitting provides a decision procedure—see [15] to realize how formidable a challenge just the case of associativity and commutativity is. A second and easier extension is to deal with higher-order analogues of other decidable classes of first-order formulas: just replace the shallow terms of [17] by our shallow patterns. This is probably easy; due to its relationship with set constraints and descriptive typing of logic programs, the higher-order analogue of the monadic class we have dealt with here is certainly the most useful such class.

References

[1] A. Aiken, Dexter Kozen, Moshe Vardi, and E. L. Wimmers. The complexity of set constraints. In *CSL'93*, pages 1–17. Springer-Verlag LNCS 832, 1993. 483

[2] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth through Proof*. Computer Science and Applied Mathematics. Academic Press, 1986. 476

[3] Leo Bachmair, Harald Ganzinger, and Uwe Waldmann. Set constraints are the monadic class. In *LICS'93*, pages 75–83. IEEE Computer Society Press, 1993. 473, 474, 480, 481

[4] Henk Barendregt. *The Lambda Calculus, Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984. 475

[5] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on direct subterms in tree automata. In Alain Finkel and Matthias Jantzen, editors, *STACS'92*, pages 161–172. Springer Verlag LNCS 577, 1992. 479

[6] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic Logic and Mechanical Theorem Proving*. Computer Science Classics. Academic Press, 1973. 477, 478, 479, 481, 484

[7] Witold Charatonik. Set constraints in some equational theories. *Inf. and Computation*, 142(1):40–75, 1998. 474

[8] Witold Charatonik and Andreas Podelski. Set constraints with intersection. In Glynn Winskel, editor, *LICS'97*, pages 362–372, 1997. 474

[9] Hubert Comon and Yan Jurski. Higher-order matching and tree automata. In M. Nielsen and W. Thomas, editors, *CSL'97*, pages 157–176. Springer-Verlag LNCS 1414, 1997. 480

[10] Christian Fermüller, Alexander Leitsch, Ulrich Hustadt, and Tamel Tammet. *Resolution Decision Procedures*, chapter 25, pages 1791–1849. Volume II of Robinson and Voronkov [22], 2001. 474, 478

[11] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer Verlag, 1990. 481

[12] Thom Frühwirth, Ehud Shapiro, Moshe Y. Vardi, and Eyal Yardeni. Logic programs as types for logic programs. In *LICS'91*, 1991. 473, 479, 486, 488

[13] Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer Verlag, 1997. 479

[14] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types*, volume 7. Cambridge University Press, 1989. 475

[15] Jean Goubault-Larrecq and Kumar Neeraj Verma. Alternating two-way AC-tree automata. Submitted, 2002. 488

[16] Gérard P. Huet. A unification algorithm for typed λ-calculus. *TCS*, 1:27–57, 1975. 475

[17] William H. Joyner Jr. Resolution strategies as decision procedures. *J. ACM*, 23(3):398–417, 1976. 473, 474, 476, 478, 487, 488

[18] Harry R. Lewis. Complexity results for classes of quantificational formulas. *J. Comp. Sys. Sciences*, 21:317–353, 1980. 473

[19] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *J. Logic and Computation*, 1(4):497–536, 1991. 473, 475, 476, 486

[20] Gopalan Nadathur and Dale Miller. An overview of λ-Prolog. In R. Kowalski and K. Bowen, editors, *5th Intl. Conf. Logic Programming*, pages 810–827. MIT Press, 1988. 486

[21] Leszek Pacholski and Andreas Podelski. Set constraints—a pearl in research on constraints. In Gert Smolka, editor, *CP'97*. Springer Verlag LNCS 1330, 1997. 474

[22] J. Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning*. North-Holland, 2001. 474, 489

[23] Wayne Snyder and Jean Gallier. Higher order unification revisited: Complete sets of tranformations. *J. Symb. Comp.*, 8(1 & 2):101–140, 1989. 476

A Proof Theoretical Account of Continuation Passing Style

Ichiro Ogata

Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 JAPAN
i.ogata@aist.go.jp
http://staff.aist.go.jp/i.ogata
phone. +81 298 61 5906 fax. +81 298 61 5909

Abstract. We study the "classical proofs as programs" paradigm in Call-By-Value (CBV) setting. Specifically, we show the CBV normalization for **CND** (Parigot 92) can be simulated by the cut-elimination procedure for **LKQ** (Danos-Joinet-Schellinx 93), namely the q-protocol. We use a proof-term assignment system to prove this fact. The term calculus for **CND** we use follows Parigot's $\lambda\mu$-Calculus and is closely related to Ong-Stewart's(Ong-Stewart 97). A new term calculus for **LKQ** is presented as a variant of λ-calculus with a let-construct. We then define a translation from **CND** into **LKQ** and prove simulation theorem. We also show the translation we use can be thought of a familiar CBV CPS-translation without translation on types.

Keywords: Classical Logic, Classical Natural Deduction, **LKQ**, Call-By-Value, CPS-translation, classical proof theory.

1 Introduction

Classical Natural Deduction: It has long been thought that classical logic cannot be put to use for computational purposes. It is because, in general, the normalization procedure for the the proof of classical logic has a lot of critical pairs. Hence classical logic in general, as a rewrite system, is not Church-Rosser(CR). Church's λ-calculus is widely accepted as the logical basis of functional programming. It is also well known that typed λ-calculus has Curry-Howard correspondence with natural deduction-style intuitionistic logic. Parigot extends this idea to a classical logic. Its computational interpretation is a natural extension of Call-By-Name (CBN) λ-calculus, called $\lambda\mu$-Calculus.

We develop a CBV variant of Parigot's $\lambda\mu$-Calculus, namely $\lambda\mu_{\mathrm{V}}$. Our $\lambda\mu_{\mathrm{V}}$ is a general CBV language in the sense that one can simulate CBV λ-calculus with continuations(catch/throw) and exception handling(handle/raise) by our $\lambda\mu_{\mathrm{V}}$. However these investigations are not new, since Ong and Stewart describe these in [16]. What we do here is to improve Ong-Stewart's CBV $\lambda\mu$-Calculus to be compatible to the q-protocol. Specifically, we introduce only two symmetric

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 490–505, 2002.
© Springer-Verlag Berlin Heidelberg 2002

reduction rules, namely β_{V} and ζ_{V}. λ-variables are substituted by **values** in β_{V}, while μ-names are substituted by **evaluation contexts** in ζ_{V}. That is, both values and evaluation contexts are first class (i.e., functional) objects. Moreover, with the help of this refinement, we also get a simple, intuitive proof of the CR-property by using standard parallel reduction method[24]. Our $\lambda\mu_{\mathrm{V}}$, being different from Ong-Stewart's, does not contain CBV λ-calculus as a sub-calculus. Instead we have a simple encoding of CBV λ-calculus into our $\lambda\mu_{\mathrm{V}}$ which is given elsewhere. Also our $\lambda\mu_{\mathrm{V}}$ does not model η-conversion, while Ong-Stewart's does. This is because η-conversion seems to have no relevancy to the cut-elimination procedure.

LKQ: **LKQ** is a variant of Gentzen's sequent-style classical logic **LK**. Gentzen's *Hauptsatz* states that any **LK** proof with cuts can be reduced into a cut-free proof. Numerous cut-elimination procedures have been described in the literature. However, all of them have problems in common — they intrinsically have non-deterministic choices which lead us to critical pairs. **LKQ** is an answer. It is equipped with SN and CR cut-elimination procedure, called the q-protocol[5]. CR property is recovered by adding some restrictions on logical rules of **LK**. Despite of these restrictions, soundness and completeness w.r.t. classical provability is still retained. What we do here is to develop a term calculus for **LKQ**, namely $\overline{\lambda\mu}_{\mathrm{let}}$. The set of reduction rules of $\overline{\lambda\mu}_{\mathrm{let}}$ are set to be compatible to the q-protocol. It is presented as a classically typed λ-calculus with a let-construct.

Translation and Simulation: The main result of this paper is the simulation theorem; the CBV normalization procedure for Classical Natural Deduction (**CND**)[17] is shown to be simulated by the q-protocol. First, we define a translation from $\lambda\mu_{\mathrm{V}}$ to $\overline{\lambda\mu}_{\mathrm{let}}$. This translation can be considered as a variant of CBV CPS-translation *without* translation on types. In previously known CPS-translations, there are so called *administrative reductions* in the target language[19]. That is, some superfluous redexes are produced by the translation, and they have nothing to do with any redexes in the source language. We develop a neat translation such that unnecessary redexes are not produced. This leads us to establish a quite tight reduction relation between normalization and cut-elimination.

We can recover the Hofmann-Streicher-style [9] by considering an intuitionistic decoration of **LKQ** (i.e., an embedding of classical types into intuitionistic types). With the help of this translation, CBV $\lambda\mu_{\mathrm{V}}$ is shown to be simulated by λ-calculus with CBN strategy. This exactly is the Plotkin's CPS simulation theorem [19]. Our CPS-translation is general in the sense that we can also recover Plotkin-style [19] and Fischer-style [6] CPS-translations by considering different intuitionistic decorations of **LKQ**. Furthermore, considering the linear decoration of **LKQ**, one can even use a proof of linear logic (with its cut-elimination procedure) as a target language of CPS-translation.

Since Griffin's pioneering work[8], it is known that there is a connection between the CPS and classical logic. Our work directly relates the classical logic

(**CND** and **LKQ**) and the CPS.

Related Works: First, we briefly summarize our previous works. In [12], we show that an intuitionistic decoration of **LKT** (**LKQ**) can be thought of a target language of a Plotkin-style CBN (CBV, respectively) CPS. In [13], we choose Parigot's $\lambda\mu$-Calculus as a source language; and we show the normalization of $\lambda\mu$-Calculus can be simulated by the t-protocol of **LKT**. In [14], the source language is λ-calculus with various non-local exit operators; and the target is an intuitionistic decoration of **LKQ**.

Curien and Herbelin develops a term calculus for **LKQ** which they call $\overline{\lambda}\tilde{\mu}$-calculus[3]. However, they only establish the isomorphism between the (intuitionistic fragment of) $\overline{\lambda}\tilde{\mu}$-calculus and the CBV λ-calculus with a let-construct (which they call $\lambda\tilde{\mu}$-calculus). Instead, we establish a direct Curry-Howard isomorphism between *full* **LKQ** and λ-calculus with a let-construct($\overline{\lambda\mu}_{\mathrm{let}}$). What is new here is that we extend the λ-calculus with a let-construct to classically typed (i.e., typed by **LKQ** as a classical logic) language. As far as we know, the correspondence between (intuitionistic decorations of) **LKT** and **LKQ** and the target language of CPS-translation first appeared in [12] and [13].

Building a term calculus on Gentzen's sequent-style intuitionistic logic (i.e., **LJ**) is investigated by Zucker[25], Pottinger[20], and recently Mints[10]. We extend these to classical case, by using **LKQ** and the q-protocol. In fact, we can present a term calculus on **LK** (using our $\overline{\lambda\mu}_{\mathrm{let}}$) which will be given elsewhere.

As for the relation between classical logic and CPS-translation, Murthy's pioneering work is also noteworthy[11]. He shows that one can interpret Girard's **LC**[7] (of which the negative fragment is **LKT**) by means of CPS with "intuitionistic extract" method. We conclude how our approach confronts to the Selinger's work on co-control category[23] in the last section.

2 Background

In this section, we recall necessary definitions and notations for our presentation. Basically, we follow the notion of *indexed logical system* according to Parigot[18]. It first appeared in Zucker's pioneering work[25].

2.1 Indexed Logical Systems

In the following, we use the word **derivation**, instead of *proof*, for a tree of derivation rules. **Formulas** are that of second order propositional logic constructed from $\rightarrow$. We use $A, B, C, \ldots$ for formulas and $X, Y, \ldots$ for propositional variables.

We use the notion of **Indexed formula**. In order to relate a term and a derivation, we need some way to specify formulas. For this, we change the notion of context. We interpret a context as a set of indexed formulas. An indexed formula is an ordered pair of a formula and an index. We assume there are

denumerably many **λ-indices** (resp. **μ-indices**) ranged over by $x, y, z \ldots$ (resp. $\alpha, \beta, \gamma, \ldots$). We write an indexed formula (A, x) as A^x and (A, α) as A^α.

As we interpret contexts as sets, occurrences of formulas with the same index are automatically contracted. One can interpret this that binary rules are always followed by appropriate explicit contractions which rename the indices to the same name. We also interpret axiom rules contain appropriate weakenings as contexts. Therefore, we say structural rules are implicit in our formulation of classical logic.

Initial index is an index which appears for the first time in whole derivation. We assume all initial indices are distinct unless they are truly related (i.e., subject of further implicit contraction). This is possible, by introducing the "concatenation" on indices on every binary rules. See Zucker[25]. We use this convention because we'd like to skirt off the fruitless discussion about capture avoiding substitution.

2.2 Classical Natural Deduction

As the name implies, **CND** is a natural deduction system (i.e., formation rules take the form of introduction ($\rightarrow\mathcal{I}$) and elimination ($\rightarrow\mathcal{E}$)) but formulated in a Gentzen-style sequent. Sequents of **CND** are of the form: $\Gamma \Rightarrow \Delta, \Xi$, where $\Rightarrow$ is the **entailment sign** of the calculus. Γ is a **λ-context** which is a *set* of λ-indexed formulas. Similarly, Δ is a **μ-context** which is a set of μ-indexed formulas. Ξ denotes exactly one un-indexed formula. Comma means taking union as sets. Thus, the set $\Gamma_0 \cup \Gamma_1$ is denoted by "Γ_0, Γ_1" and $\{A^x\} \cup \Gamma$ by "A^x, Γ".

2.3 Gentzen's Sequent-Style Constructive Classical Logic

LKQ is a variant of Gentzen's sequent-style classical logic. That is, formation rules take the form of left and right introduction. Sequents of **LKQ** are of the form: $\Gamma \Rightarrow \Delta\,;\ \Pi$, where Π denotes at most one un-indexed formula. The right most place where Π lives is called **stoup**. Roughly speaking, stoup is a place where newly created formula goes. Please pay attention to the fact that application of structural rules are restricted within Γ and Δ. Specifically Π can not be introduced by weakening. We use $\Gamma \Rightarrow \Delta\,;\ \emptyset$ to indicate that the stoup is empty.

2.4 Multiplicative Rules

In both **CND** and **LKQ**, we only handle **multiplicative** rules. That is, λ-contexts (μ-contexts) in the conclusion is the union of λ-contexts (μ-contexts respectively) in the premises. For example, in L$\rightarrow$ of **LKQ**:

$$\frac{\Gamma_0 \Rightarrow \Delta_0\,;\ A \qquad B^y, \Gamma_1 \Rightarrow \Delta_1\,;\ \emptyset}{(A \rightarrow B)^z, \Gamma_0, \Gamma_1 \Rightarrow \Delta_0, \Delta_1\,;\ \emptyset}\ \mathrm{L}\rightarrow$$

Hereafter, for readability, we only write **active** and **main** formulas, and omit contexts as follows:

$$\frac{\Rightarrow ;\ A \qquad B^y \Rightarrow ;\ \emptyset}{(A \to B)^z \Rightarrow ;\ \emptyset}\ \mathrm{L}\!\to$$

In the above, we say A and B^y are **active** formulas, while $(A \to B)^z$ is a **main** formula.

2.5 Restrictions for Propositional Variables

Usual restrictions for propositional variables apply. For example, in case of introduction of $\forall$ in **CND**:

$$\frac{\Gamma \Rightarrow \Delta, A[X := Y]}{\Gamma \Rightarrow \Delta, \forall X.A}\ \forall^2\mathcal{I}^*$$

In the above, the propositional variable Y has no free occurrence in the contexts Γ and Δ. We use $()^*$ to indicate these restrictions.

3 Calculi for Call-by-Value Classical Natural Deduction

3.1 A Call-by-Value Calculus: $\lambda\mu_{\mathbf{V}}$

In this subsection, we shall introduce a Call-By-Value $\lambda\mu$-Calculus, namely $\lambda\mu_{\mathbf{V}}$. The $\lambda\mu_{\mathbf{V}}$-terms includes two sub-categories, namely **values** and μ**-renames**.

Definition 1 ($\lambda\mu_{\mathbf{V}}$-terms).

1. *We define* $\lambda\mu_{\mathbf{V}}$**-values**, *ranged over by* v, *as follows:*

$$\begin{array}{rll} v := & x, y, z, \ldots & \lambda\text{-variables} \\ \mid & \lambda x^A.p & \text{abstraction} \\ \mid & \Lambda X.p & \text{universal-abstraction} \end{array}$$

2. *We define* $\lambda\mu_{\mathbf{V}}$**-**μ**-renames**, *ranged over by* p, q, *are defined as follows:*

$$p, q := \mu\alpha^A.[\beta]\, M$$

3. *We define* $\lambda\mu_{\mathbf{V}}$**-terms**, *ranged over by* L, M, N, *etc., are defined as follows:*

$$\begin{array}{rll} L, M, N := & v & \text{value} \\ \mid & p & \mu\text{-rename} \\ \mid & MN & \text{application} \\ \mid & MB & \text{universal-application} \end{array}$$

Table 1. $\lambda\mu_{\mathrm{V}}$-term Assignment for **CND**

$$\frac{}{x: \quad A^x \Rightarrow A}\,\mathrm{Ax} \qquad \frac{N: \quad \Rightarrow A, \Delta}{\mu\beta^B.[\alpha]\,N: \quad \Rightarrow B, ((A^\alpha, \Delta) \setminus B^\beta)}\,\mathrm{rename}$$

$$\frac{p: \quad A^x \Rightarrow B}{\lambda x^A.p: \quad \Rightarrow A \to B}\,{\to}\mathcal{I} \qquad \frac{M: \quad \Rightarrow A \to B \qquad N: \quad \Rightarrow A}{M\,N: \quad \Rightarrow B}\,{\to}\mathcal{E}$$

$$\frac{p[X := Y]: \quad \Rightarrow A[X := Y]}{\Lambda X.p: \quad \Rightarrow \forall X.A}\,\forall^2\mathcal{I}^* \qquad \frac{M: \quad \Rightarrow \forall X.A}{M\,B: \quad \Rightarrow A[X := B]}\,\forall^2\mathcal{E}$$

$\alpha, \beta, \gamma, \ldots$ are called as **μ-names** and $A, B, \ldots$ are called as **types**. Application associates to left, i.e., we write "LMN" instead of "$(LM)N$". The rules of **term assignment judgment** are displayed in Table 3.1. In the table, λ-variables and μ-names are identified with λ-indices and μ-indices respectively. Moreover types are identified with formulas and type variables are identified with propositional variables. Observe that a body of abstractions must be a μ-rename. This prevents us to include CBV λ-calculus as a sub-calculus of $\lambda\mu_{\mathrm{V}}$. However we have an encoding of CBV λ-calculus into our $\lambda\mu_{\mathrm{V}}$.

As we show above, we use Church-style typing (i.e., every variable have types as superscripts). We will occasionally abbreviate types because in most case types of variables are clear from the context. The set of **free μ-names** of $\lambda\mu_{\mathrm{V}}$-term M, denoted by $\mathrm{FN}(M)$, is defined as follows: $\mathrm{FN}(x) = \emptyset$, $\mathrm{FN}(\lambda x.p) = \mathrm{FN}(\Lambda X.p) = \mathrm{FN}(p)$, $\mathrm{FN}(\mu\alpha.[\beta]\,M) = (\mathrm{FN}(M) \cup \{\,\beta\,\}) \setminus \{\,\alpha\,\}$, $\mathrm{FN}(MB) = \mathrm{FN}(M)$, $\mathrm{FN}(MN) = \mathrm{FN}(M) \cup \mathrm{FN}(N)$. The set of **free λ-variables** , denoted by $\mathrm{FV}(M)$, is defined in the same way with λ-calculus.

Definition 2 (CBV Singular Evaluation Context). *We define* **CBV singular evaluation contexts**, *ranged over by K, as follows:* $K := [\text{-}]N \mid [\text{-}]B \mid v[\text{-}]$.

Definition 3 (CBV Evaluation Context). *We define* **CBV evaluation contexts**, *ranged over by E, as follows:* $E := [\text{-}] \mid EN \mid EB \mid vE$. *Note that a CBV evaluation context E has exactly one hole.*

CBV evaluation context E can be defined as a sequence of singular evaluation contexts such as $E = K_0 \circ K_1 \circ \ldots \circ K_{n-1}$, where $\circ$ is the context composition which is defined by $(K_0 \circ K_1)[\text{-}] = K_0[K_1[\text{-}]]$. The composition is associative. Note that one can *parse* every μ-rename in the form of: $\mu\alpha.[\beta]\,E[N]$, where N is either value: v or μ-rename: p.

Definition 4 ($\lambda\mu_{\mathrm{V}}$ as a reduction system). *The* **reduction relation** $\longrightarrow_{\lambda\mu_v}$ *of $\lambda\mu_v$, viewed as a rewrite system, is defined to be the compatible (i.e. contextual) closure of the notion of reduction defined by three redex rules, namely, β_v, ζ_v and polymorphic.* $\twoheadrightarrow_{\lambda\mu_v}$ *is the reflexive, transitive closure of* $\longrightarrow_{\lambda\mu_v}$.

$$
\begin{array}{lll}
(\beta_{\mathrm{V}}) & (\lambda x.L)v & \longrightarrow_{\lambda\mu_{\mathrm{V}}} \; L\,[x := v] \\
(\zeta_{\mathrm{V}}) & \mu\alpha.[\beta]\,E[\mu\gamma.[\delta]\,M] & \longrightarrow_{\lambda\mu_{\mathrm{V}}} \; \mu\alpha.([\delta]\,M)\,[\gamma := [\beta]\,E[\text{-}]] \\
(\text{polymorphic}) & (\Lambda X.L)B & \longrightarrow_{\lambda\mu_{\mathrm{V}}} \; L\,[X := B]
\end{array}
$$

We shall refer to the above as **$\lambda\mu_{\mathbf{V}}$-redex rules** and terms on the left-hand-side of the redex rules as **$\lambda\mu_{\mathbf{V}}$-redexes**. Three kinds of substitutions can be distinguished in $\lambda\mu_{\mathrm{V}}$. The first **λ-substitution** of the form: $L\,[x := v]$ means the standard substitution as meta-operation. It is the result of substituting v for the free occurrences of x (of the same type as v) in L. The second **μ-substitution** of the form: $M\,[\gamma := [\beta]\,E[\text{-}]]$ means "in M, replace all subterms of the form $[\gamma]\,L$ by the term $[\beta]\,E[L]$". The third **type-substitution** of the form: $M\,[X := B]$ means "in M, replace all occurrences of the type variable X by the type B".

Remark 1 (evaluation context). Traditionally, evaluation contexts are devised so that every non-normal closed term M can be written uniquely as $E[R]$, where R is a redex. It is used to extract an unique redex according to evaluation strategy. Bierman develops operational theory for $\lambda\mu$-Calculus using this idea[2]. Instead, we use the notion of evaluation context to uniquely define a ζ-redex in every μ-rename. In particular we do not specify the order of reduction. Every β-, ζ- and polymorphic redexes can be reduced in any order. Clearly the Church-Rosser property is only meaningful in this setting. Our point here is that the concept of CBV is *not* build on the reduction system as an evaluation order.

3.2 Relation to Ong-Stewart's CBV $\lambda\mu$-Calculus

Now we demonstrate how our $\lambda\mu_{\mathrm{V}}$ is different from Ong-Stewart's CBV $\lambda\mu$-Calculus[15, 16]. In a word, we pack $n+1$ length of "reduction sequence" into single reduction. Consider our general ζ_{V}-redex: $\mu\alpha.[\beta]\,E[\mu\gamma.[\delta]\,M]$. We assume E consists of n-fold singular contexts, i.e., $E = K_0 \circ K_1 \circ \ldots \circ K_{n-1}$. In the style of Ong-Stewart's ζ-reduction rule, the reduction proceeds as follows:

$$
\begin{array}{r}
K_{n-1}[\mu\gamma.[\delta]\,M] \to \mu\beta_{n-1}.[\delta]\,M\,[\gamma := [\beta_{n-1}]\,K_{n-1}[\text{-}]] \\
K_{n-2}[\mu\beta_{n-1}.[\delta]\,M\,[\gamma := [\beta_{n-1}]\,K_{n-1}[\text{-}]]] \to \\
\mu\beta_{n-2}.[\delta]\,M\,[\gamma := [\beta_{n-2}]\,(K_{n-2} \circ K_{n-1})[\text{-}]] \\
\vdots \\
K_0[\mu\beta_1.[\delta]\,M\,[\gamma := [\beta_1]\,(K_1 \circ \ldots \circ K_{n-1})[\text{-}]]] \to \mu\beta_0.[\delta]\,M\,[\gamma := [\beta_0]\,E[\text{-}]] \\
\mu\alpha.[\beta]\,(\mu\beta_0.[\delta]\,M\,[\gamma := [\beta_0]\,E[\text{-}]]) \to \mu\alpha.[\delta]\,M\,[\gamma := [\beta]\,E[\text{-}]]
\end{array}
$$

The last reduction rule is called as μ-β reduction. Observe that each ζ reduction *always* produces another ζ (or μ-β) redex. Hence there always is a $n+1$ length of **sequential reduction**, where n is the size of E. Because of this one cannot apply the standard Tait-Martin-Löf-Takahashi's parallel reduction method for Church-Rosser property[24] to Ong-Stewart's $\lambda\mu$-Calculus. This is simply because the

diamond property for parallel reduction does not hold in this situation. This phenomenon was first observed by Baba et al.[1] in slightly different settings. Full proof of CR-property for our $\lambda\mu_{\mathrm{V}}$ will be given elsewhere.

Remark 2. Our $\lambda\mu_{\mathrm{V}}$ does not model η reduction, while Ong-Stewart's $\lambda\mu$-Calculus does(i.e., it has η and μ-η reductions).

4 Calculi for Gentzen's Sequent-Style Classical Logic: LKQ

In this section, we introduce $\overline{\lambda\mu}_{\mathrm{let}}$, a variant of λ-calculus with a let-construct, as a term calculus for **LKQ**. The $\overline{\lambda\mu}_{\mathrm{let}}$-terms are classified exclusively in the three categories, namely **values**, **contexts** and **$\overline{\mu}$-abstractions**.

Definition 5 ($\overline{\lambda\mu}_{\mathbf{let}}$-terms).

1. *$\overline{\lambda\mu}_{let}$-values, ranged over by V, are defined as follows:*

$$\begin{array}{rll} V := & x, y, z, \ldots & \lambda\text{-variables} \\ \mid & \overline{\lambda} x^A.P & \text{right-term} \\ \mid & \overline{\Lambda} X.P & \text{universal-right-term} \end{array}$$

2. *$\overline{\lambda\mu}_{let}$-contexts, ranged over by S, T, U, etc., are defined as follows:*

$$\begin{array}{rll} S,T,U := & [\alpha]\, V & \text{derelict-term} \\ \mid & \mathit{let}\ \ x = V\ \ \mathit{in}\ \ U & \text{tail-term} \\ \mid & \mathit{let}\ \ x = P\ \ \mathit{in}\ \ T & \text{mid-term} \\ \mid & \mathit{let}\ \ y = zV\ \ \mathit{in}\ \ U & \text{left-term} \\ \mid & \mathit{let}\ \ x = zB\ \ \mathit{in}\ \ T & \text{universal-left-term} \end{array}$$

3. *$\overline{\lambda\mu}_{let}$-$\overline{\mu}$-abstractions, ranged over by P, are defined as follows:*

$$P := \overline{\mu}\alpha.S$$

The rules of **term assignment judgment** are displayed in Table 4. Observe that contexts are assigned to **LKQ**-sequents with empty stoup. On the other hand, values are assigned to sequents which have a formula in the stoup. $\overline{\mu}$-abstractions are not assigned to any **LKQ**-sequents; they only appear as a subterms of values or contexts. In the table, the letter L/R stands for **Left** and **Right** introduction, and D for **Dereliction**.

We have two additional term assignment judgment rules which allows us to express intermediate state between S2-step and L-step of the q-protocol.

$$\frac{V: \quad \Rightarrow\ ;\ A \quad U: \quad B^y \Rightarrow ;\ \emptyset \quad T: \quad A^x \Rightarrow B^\beta\ ;\ \emptyset}{\text{let}\ \ y = (\overline{\lambda} x^A.\overline{\mu}\beta^B.T)V\ \ \text{in}\ \ U: \quad \Rightarrow\ ;\ \emptyset}\ \beta_{\mathrm{V}}$$

Table 2. $\overline{\lambda\mu}_{\mathrm{let}}$-term Assignment for **LKQ**

$$\frac{}{x: \quad A^x \Rightarrow ; A}\,\mathrm{Ax} \qquad \frac{V: \quad \Rightarrow ; A}{[\alpha]\,V: \quad \Rightarrow A^\alpha ; \emptyset}\,\mathrm{D}$$

$$\frac{V: \quad \Rightarrow ; A \qquad T: \quad A^x \Rightarrow ; \emptyset}{\text{let } x = V \text{ in } T: \quad \Rightarrow ; \emptyset}\,\mathrm{tail} \qquad \frac{S: \quad \Rightarrow A^\alpha ; \emptyset \qquad T: \quad A^x \Rightarrow ; \emptyset}{\text{let } x = \overline{\mu}\alpha.S \text{ in } T: \quad \Rightarrow ; \emptyset}\,\mathrm{mid}$$

$$\frac{V: \quad \Rightarrow ; A \qquad U: \quad B^y \Rightarrow ; \emptyset}{\text{let } y = zV \text{ in } U: \quad (A \to B)^z \Rightarrow ; \emptyset}\,\mathrm{L}\!\to \qquad \frac{T: \quad A^x \Rightarrow B^\beta ; \emptyset}{\overline{\lambda} x^A.\overline{\mu}\beta^B.T: \quad \Rightarrow ; A \to B}\,\mathrm{R}\!\to$$

$$\frac{U: \quad (A[X := B])^x \Rightarrow ; \emptyset}{\text{let } x = zB \text{ in } U: \quad (\forall X.A)^z \Rightarrow ; \emptyset}\,\mathrm{L}\forall^2 \qquad \frac{T[X := Y]: \quad \Rightarrow (A[X := Y])^\alpha ; \emptyset}{\overline{\Lambda} X.\overline{\mu}\alpha^A.T: \quad \Rightarrow ; \forall X.A}\,\mathrm{R}\forall^{2*}$$

$$\frac{T[X := Y]: \quad \Rightarrow (A[X := Y])^\alpha ; \emptyset \qquad U: \quad (A[X := B])^x \Rightarrow ; \emptyset}{\text{let } x = (\overline{\Lambda} X.\overline{\mu}\alpha^A.T)B \text{ in } U: \quad \Rightarrow ; \emptyset}\,\beta_{\mathrm{univ}}$$

This idea first appeared in [22] in J.E. Santo's study about intuitionistic fragment of **LKT**.

Definition 6 ($\overline{\lambda\mu}_{\mathbf{let}}$ as a reduction system). *The* **reduction relation** $\longrightarrow_{\overline{\lambda\mu}_{let}}$ *of* $\overline{\lambda\mu}_{let}$*, viewed as a rewrite system, is defined to be the compatible (i.e. contextual) closure of the notion of reduction defined by four redex rules, namely, S1,S2,*$L_\to$ *and* $L_\forall$*.* $\longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{let}}$ *is the reflexive, transitive closure of* $\longrightarrow_{\overline{\lambda\mu}_{let}}$*. We use* $M \longrightarrow_{\overline{\lambda\mu}_{let}} \longrightarrow_{\overline{\lambda\mu}_{let}} N$ *to mean that* $M \longrightarrow_{\overline{\lambda\mu}_{let}} L \longrightarrow_{\overline{\lambda\mu}_{let}} N$ *holds for some* L*.*

(S1)	let $x = \overline{\mu}\alpha.S$ in T	$\longrightarrow_{\overline{\lambda\mu}_{\mathrm{let}}} S\,[\alpha := (\text{let } x = \_ \text{ in } T)]$
(S2)	let $x = V$ in S	$\longrightarrow_{\overline{\lambda\mu}_{\mathrm{let}}} S\,[x := V]$
($\mathrm{L}_\to$)	$(\overline{\lambda} x^A.\overline{\mu}\beta^B.T)V$	$\longrightarrow_{\overline{\lambda\mu}_{\mathrm{let}}} \overline{\mu}\beta^B.(\text{let } x = V \text{ in } T)$
($\mathrm{L}_\forall$)	$(\overline{\Lambda} X.\overline{\mu}\alpha^A.T)B$	$\longrightarrow_{\overline{\lambda\mu}_{\mathrm{let}}} \overline{\mu}\alpha^A.T\,[X := B]$

These redex rules are set to be compatible to the reduction step of the q-protocol(i.e., S1-step, S2-step and L-step). Three kinds of substitutions can be distinguished in $\overline{\lambda\mu}_{\mathrm{let}}$. **$\overline{\lambda}$-substitution** of the form: $T\,[x := V]$ means the standard substitution as meta-operation. Note that one can only substitute λ-variable for $\overline{\lambda\mu}_{\mathrm{let}}$-value, like β_{V} of $\lambda\mu_{\mathrm{V}}$. **$\overline{\mu}$-substitution** of the form: $U\,[\alpha := (\text{let } x = \_ \text{ in } T)]$ means "in U, replace all subterms of the form $[\alpha]\,V$ by the term $(\text{let } x = V \text{ in } T)$". The third **type-substitution** of the form: $M\,[X := B]$ means the standard one.

*Remark 3 (***LKQ***).* We refer to [5] for "technical terms" in this remark. Strictly speaking, our presentation of **LKQ** is a "q-fragment of $\mathbf{LK}^{\eta}$ where all formulas are coloured q". That is, all formulas in the stoup of **LKQ** have "flat m-a-interspaces". This constraint can be rephrased as follows: the main formula introduced in the stoup by Ax or L→ must be an active formula of the previous derivation rule. Of course, by the "stability lemma", this property is preserved under the q-protocol. This definition is slightly different from the one presented in earlier literature[4].

5 Translation

5.1 Simulation of $\lambda\mu_{\mathrm{V}}$ by $\overline{\lambda\mu}_{\mathbf{let}}$

First, we define the translation from $\lambda\mu_{\mathrm{V}}$-terms to $\overline{\lambda\mu}_{\mathrm{let}}$-terms. Clearly, an endsequent of **CND**: $\Gamma \Rightarrow B, \Delta$ corresponds to an endsequent of **LKQ**: $\Gamma \Rightarrow B, \Delta\,;\ \emptyset$. The latter is not a proper **LKQ** sequent. It is introduced by the extra non-logical derivation rule, namely $\overline{\mu}$-abstraction. At the same time, $\lambda\mu_{\mathrm{V}}$-term must be μ-rename in order to specify the μ-name in $\overline{\mu}$-abstraction. So the last derivation rules must be μ-rename and $\overline{\mu}$-abstraction respectively. That is, one can only define the translation from $\lambda\mu_{\mathrm{V}}$-μ-renames to $\overline{\lambda\mu}_{\mathrm{let}}$-$\overline{\mu}$-abstractions. We can assume this without loss of generosity, since we always have $\mu\alpha.[\alpha]\,M$ ($\alpha \notin \mathrm{FN}(M)$) for arbitrary $\lambda\mu_{\mathrm{V}}$-term M. The situation is illustrated as follows:

$$\dfrac{N: \quad \Rightarrow A, \Delta}{\mu\beta^B.[\alpha]\,N: \quad \Rightarrow B, ((A^{\alpha}, \Delta) \setminus B^{\beta})}\ \text{rename} \qquad \dfrac{S: \quad \Gamma \Rightarrow B^{\beta}, \Delta\,;\ \emptyset}{\overline{\mu}\beta^B.S: \quad \Gamma \Rightarrow B, \Delta\,;\ \emptyset}\ \overline{\mu}\text{-abstraction}$$

Definition 7 (Translation from $\lambda\mu_{\mathrm{V}}$ to $\overline{\lambda\mu}_{\mathbf{let}}$).

1. *The translation* $\overline{(\_)}$, $\lambda\mu_v$-μ-*renames* $\mapsto$ $\overline{\lambda\mu}_{let}$-$\overline{\mu}$-*abstractions is defined as follows:*
$$\overline{\mu\beta^B.[\alpha]\,N} = \overline{\mu}\beta^B.(N : let\ \ x = \_\ \ in\ \ [\alpha]\,x)$$
2. *The infix operator* :, $\lambda\mu_v$-*terms* $\times$ $\overline{\lambda\mu}_{let}$-*contexts* $\mapsto$ $\overline{\lambda\mu}_{let}$-*contexts is defined as follows:*
$$\begin{gathered} v : let\ \ y = \_\ \ in\ \ S = S\,[y := \Psi(v)] \\ \mu\alpha.[\beta]\,M : let\ \ y = \_\ \ in\ \ S = let\ \ y = \overline{\mu\alpha.[\beta]\,M}\ \ in\ \ S \\ MN : let\ \ y = \_\ \ in\ \ S = M : let\ \ z = \_\ \ in\ \ (N : let\ \ x = \_\ \ in\ \ (let\ \ y = zx\ \ in\ \ S)) \\ MB : let\ \ y = \_\ \ in\ \ S = M : let\ \ z = \_\ \ in\ \ (let\ \ y = zB\ \ in\ \ S) \end{gathered}$$
3. *An auxiliary function* Ψ, $\lambda\mu_v$-*values* $\mapsto$ $\overline{\lambda\mu}_{let}$-*values, is defined as follows:*
$$\Psi(x) = x; \qquad \Psi(\lambda x.p) = \overline{\lambda} x.\overline{p}; \qquad \Psi(\Lambda X.p) = \overline{\Lambda} X.\overline{p}$$

Proof theoretically, this translation is based on Prawitz's observation to simulate natural deduction-style by Gentzen's sequent style logic[21]. For example, the application pq can be written in **LKQ** as follows:

$$\dfrac{U:\ \Rightarrow (A\to B)^{\gamma}\,;\,\emptyset \quad \dfrac{S:\ \Rightarrow A^{\alpha}\,;\,\emptyset \quad \dfrac{x:\ \ A^{x}\Rightarrow\,;\,A \quad [\beta]\,y:\ \ B^{y}\Rightarrow B^{\beta}\,;\,\emptyset}{\text{let } y=zx \text{ in } [\beta]\,y:\ \ A^{x},(A\to B)^{z}\Rightarrow B^{\beta}\,;\,\emptyset}\,\mathrm{L}\!\to}{\text{let } x=\overline{\mu}\alpha^{A}.S \text{ in } (\text{let } y=zx \text{ in } [\beta]\,y):\ \ (A\to B)^{z}\Rightarrow B^{\beta}\,;\,\emptyset}\,\text{mid}}{\text{let } z=\overline{\mu}\gamma^{A\to B}.U \text{ in } (\text{let } x=\overline{\mu}\alpha^{A}.S \text{ in } (\text{let } y=zx \text{ in } [\beta]\,y)):\ \ \Rightarrow B^{\beta}\,;\,\emptyset}\,\text{mid}$$

where $\overline{p}=\overline{\mu}\gamma.U$ and $\overline{q}=\overline{\mu}\alpha.S$.

Remark 4. A $\overline{\lambda\mu}_{\text{let}}$-$\overline{\mu}$-abstraction: $\overline{p}$ (for some $\lambda\mu_{\text{V}}$-μ-rename p) contains no S2 redex. Instead it contains $\mathrm{L}_{\to}$ and/or $\mathrm{L}_{\forall}$ redexes.

Remark 5. In the above derivation, the order of two mid-cuts does matter. This situation is called the "**q/t** dilemma" in [5]; implication is the dilemmatic logical operator in the q-protocol. To say that the order matters is just to say we have already made a choice. Of course, another choice is possible. See subsection 5.2.

Our main theorem below can be seen as a proof theoretical explanation for Plotkin's CPS simulation theorem.

Theorem 1. *If* $p \longrightarrow_{\lambda\mu_v} q$ *then* $\overline{p} \longrightarrow_{\overline{\lambda\mu}_{let}} \longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{let}} \overline{q}$

We devote the rest of the subsection for this proof.

Proposition 1 (λ-substitution and $\overline{\lambda}$-substitution).

$$\overline{p}\,[x:=\Psi(v)] = \overline{p\,[x:=v]}$$

Proof. by induction on L.

Proposition 2 (An image of an evaluation context). *An image of μ-rename: $\overline{\mu\alpha.[\beta]\,E[M]}$ always has the form of: $\overline{\mu}\alpha.(M : \mathit{let}\ \ y = \_\ \ \mathit{in}\ \ S_{[\beta]E})$.*

Proof. By induction on the construction of evaluation context.

Proposition 3 (μ-substitution and $\overline{\mu}$-substitution). *One only uses S1 in the following reduction relation.*

(1) $\overline{p}\,[\gamma := \mathit{let}\ \ y' = \_\ \ \mathit{in}\ \ S_{[\beta]E}] \longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{let}} \overline{p\,[\gamma := [\beta]\,E[\text{-}]]}$

(2) $(M : \mathit{let}\ \ y = \_\ \ \mathit{in}\ \ S)\,[\gamma := \mathit{let}\ \ y' = \_\ \ \mathit{in}\ \ S_{[\beta]E}]$

$\longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{let}} (M\,[\gamma := [\beta]\,E[\text{-}]]) : (\mathit{let}\ \ y = \_\ \ \mathit{in}\ \ S\,[\gamma := \mathit{let}\ \ y' = \_\ \ \mathit{in}\ \ S_{[\beta]E}])$

Proof. By mutual induction on p and M.

(1) Assume $p = \mu\alpha.[\gamma]\,N$.

$$\begin{aligned}
& \overline{\mu\alpha.[\gamma]\,N}\,[\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}] \\
= \ & \overline{\mu}\alpha.(N : \text{let}\ \ y'' = _\ \ \text{in}\ \ [\gamma]\,y'')\,[\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}] \\
\longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{\text{let}}} \ & \overline{\mu}\alpha.(N\,[\gamma := [\beta]\,E[\text{-}]]) : ((\text{let}\ \ y'' = _\ \ \text{in}\ \ [\gamma]\,y'')\,[\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}]) \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} \ & \overline{\mu}\alpha.(N\,[\gamma := [\beta]\,E[\text{-}]] : \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}) \\
= \ & \overline{\mu\alpha.([\beta]\,E[N\,[\gamma := [\beta]\,E[\text{-}]]])} \\
= \ & \overline{(\mu\alpha.[\gamma]\,N)\,[\gamma := [\beta]\,E[\text{-}]]}
\end{aligned}$$

We use (2) from second to third, S1 from third to fourth.

(2) We only consider the base case: $M = \mu\alpha.[\eta]\, N$.

$$\begin{array}{rl}
 & (\mu\alpha.[\eta]\, N : \text{let}\ \ y = _\ \ \text{in}\ \ S)\, [\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}] \\
= & (\text{let}\ \ y = \overline{\mu\alpha.[\eta]\, N}\ \ \text{in}\ \ S)\, [\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}] \\
\longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{\text{let}}} & \text{let}\ \ y = \overline{\mu\alpha.[\eta]\, N\, [\gamma := [\beta]\, E[\text{-}]]}\ \ \text{in}\ \ (S\, [\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}]) \\
= & (\mu\alpha.[\eta]\, N)\, [\gamma := [\beta]\, E[\text{-}]] : (\text{let}\ \ y = _\ \ \text{in}\ \ S\, [\gamma := \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}])
\end{array}$$

We use (1) from second to third.

In the proofs below, we use the abbreviation $\overline{[\delta]\, L} = L : \text{let}\ \ y = \_\ \ \text{in}\ \ [\delta]\, y$. With this notion, $\overline{\mu\alpha.[\delta]\, L} = \overline{\mu}\alpha.\overline{[\delta]\, L}$.

Proposition 4. *If $p \longrightarrow_{\lambda\mu_v} q$ by β_v then $\overline{p} \longrightarrow_{\overline{\lambda\mu}_{let}} \longrightarrow_{\overline{\lambda\mu}_{let}} \overline{q}$. The two $\longrightarrow_{\overline{\lambda\mu}_{let}}$ are $L_{\rightarrow}$ and S2 respectively.*

Proof. Assume the β_{V} under consideration being $(\lambda x.\mu\gamma.[\delta]\, L)v$ and it appears within context let $y = \_$ in S.

$$\begin{array}{rll}
 & ((\lambda x.\mu\gamma.[\delta]\, L)v) : \text{let}\ \ y = _\ \ \text{in}\ \ S & \\
= & \text{let}\ \ y = (\lambda x.\overline{\mu\gamma.[\delta]\, L})\Psi(v)\ \ \text{in}\ \ S & \text{translation} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \text{let}\ \ y = \overline{\mu}\gamma.(\text{let}\ \ x = \Psi(v)\ \ \text{in}\ \ \overline{[\delta]\, L})\ \ \text{in}\ \ S & \text{L}_{\rightarrow} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \text{let}\ \ y = \overline{\mu\gamma.[\delta]\, L\, [x := \Psi(v)]}\ \ \text{in}\ \ S & \text{S2} \\
= & \text{let}\ \ y = \overline{\mu\gamma.[\delta]\, L\, [x := v]}\ \ \text{in}\ \ S & \text{proposition 1} \\
= & \mu\gamma.[\delta]\, L\, [x := v] : \text{let}\ \ y = _\ \ \text{in}\ \ S & \text{translation}
\end{array}$$

Proposition 5. *If $p \longrightarrow_{\lambda\mu_v} q$ by ζ_v then $\overline{p} \longrightarrow_{\overline{\lambda\mu}_{let}} \longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{let}} \overline{q}$. One only uses S1 in these reduction relations.*

Proof. If $p = \mu\alpha.[\beta]\, E[\mu\gamma.[\delta]\, L]$, then

$$\begin{array}{rll}
 & \overline{\mu\alpha.[\beta]\, E[\mu\gamma.[\delta]\, L]} & \\
= & \overline{\mu}\alpha.(\mu\gamma.[\delta]\, L : \text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E}) & \text{proposition 2} \\
= & \overline{\mu}\alpha.(\text{let}\ \ y' = \overline{\mu}\gamma.\overline{[\delta]\, L}\ \ \text{in}\ \ S_{[\beta]E}) & \text{translation} \\
\longrightarrow_{\overline{\lambda\mu}_{\text{let}}} & \overline{\mu}\alpha.(\overline{[\delta]\, L}\, [\gamma := (\text{let}\ \ y' = _\ \ \text{in}\ \ S_{[\beta]E})]) & \text{S1} \\
\longrightarrow\!\!\!\!\rightarrow_{\overline{\lambda\mu}_{\text{let}}} & \overline{\mu\alpha.[\delta]\, L\, [\gamma := [\beta]\, E[\text{-}]]} & \text{proposition 3 (2)}
\end{array}$$

Proposition 6. *If $p \longrightarrow_{\lambda\mu_v} q$ by polymorphic then $\overline{p} \longrightarrow_{\overline{\lambda\mu}_{let}} \overline{q}$ by $L_{\forall}$.*

This proof is easy, and concludes the proof of the simulation theorem.

Corollary 1. *$\lambda\mu_v$ is Strongly Normalizable.*

Proof. Simulation theorem says that if there is an infinite reduction sequence in $\lambda\mu_{\text{V}}$, then there also is in $\overline{\lambda\mu}_{\text{let}}$. This contradicts the SN property of **LKQ**.

Please note that p being normal does not mean $\overline{p}$ being normal. Consider the normal $\lambda\mu_{\mathrm{V}}$-term $(\lambda x.p)(zw)$. Then

$$\begin{array}{rcl}(\lambda x.p)(zw) : \mathrm{let}\ \ y = \_\ \ \mathrm{in}\ \ S & = & (\mathrm{let}\ \ x' = zw\ \ \mathrm{in}\ \ (\mathrm{let}\ \ y = (\overline{\lambda} x.\overline{p})x'\ \ \mathrm{in}\ \ S)) \\ & \longrightarrow_{\overline{\lambda\mu}_{\mathrm{let}}} & (\mathrm{let}\ \ x' = zw\ \ \mathrm{in}\ \ (\mathrm{let}\ \ y = \overline{p}\,[x := x']\ \ \mathrm{in}\ \ S))\end{array}$$

That is, we can extract "hidden" redexes by translating a $\lambda\mu_{\mathrm{V}}$-μ-rename into a $\overline{\lambda\mu}_{\mathrm{let}}$-$\overline{\mu}$-abstraction. The familiar trick to avoid this obstacle was to extend the syntax of λ-calculus to include a let-construct. What is new here is that we revise and extend this syntax to the classically typed language(i.e., it is typed by **LKQ** sequents).

Claim. A familiar λ-calculus with a let-construct, as a sub-calculus of $\overline{\lambda\mu}_{\mathrm{let}}$, is a target language of CBV CPS-translation. Complex reduction rules related to a let-construct (e.g., see [3]) can be unified into single, simple S1 reduction rule. It also is isomorphic to (a sub-calculus of) λ-calculus which is a target language of Hofmann-Streicher-style CPS-translation.

Remark 6. Prawitz's conversion sends **CND** normal derivations to cut-free **LK** derivations. However the conversion from **LK** into **LKQ**, in general, does not send cut-free **LK** derivations to cut-free **LKQ** derivations. That is why $\overline{p}$ being non-normal in case p being normal.

5.2 There are Two Ways to Map CND into LKQ

We choose the ζ_{V}-redex in the application from left-to-right(LR) order. Of course, the opposite order should be studied in its own right. This phenomena is known in the previous study of CPS; the CBV right-to-left(RL) evaluation method. This kind of CPS-translation was shown, for example, by Murthy[11]. One can adopt the RL evaluation method in our $\lambda\mu_{\mathrm{V}}$. For this, we first modify the evaluation context as follows:

$$E := [\text{-}]\ \mid\ ME\ \mid\ Ev$$

This modification leads us the RL version of our $\lambda\mu_{\mathrm{V}}$. Then, we modify the translation (in order to keep the simulation theorem) as follows:

$$MN : \mathrm{let}\ \ y = \_\ \ \mathrm{in}\ \ S = N : \mathrm{let}\ \ x = \_\ \ \mathrm{in}\ \ (M : \mathrm{let}\ \ z = \_\ \ \mathrm{in}\ \ (\mathrm{let}\ \ y = zx\ \ \mathrm{in}\ \ S))$$

Danos-Joinet-Schellinx's theory give a proof theoretical explanation to this phenomenon – they say there are two ways to map **LK** derivations to **LKQ** derivations. However, Selinger seems to overlook this in his paper[23].

6 Conclusions and Further Directions

We formulate second-order Call-By-Value $\lambda\mu$-Calculus as a yet another term calculus for Parigot's Classical Natural Deduction. We show it is Church-Rosser

and Strongly Normalizable. We also show that the translation from $\lambda\mu_{\mathrm{V}}$ to $\overline{\lambda\mu}_{\mathrm{let}}$ can be thought of a proof theoretical counterpart of a familiar CPS-translation.

Our proof theoretical approach confronts semantical work in some point. Actually there are some advantages of using proof theory as a syntax of the calculus. Recall that the SN-property is proved as a corollary of SN-property of **LKQ**. We also know the class of functions representable in our second-order $\lambda\mu_{\mathrm{V}}$; it exactly is the class of provably total functions in second-order Peano Arithmetic $\mathbf{PA}_2$(i.e., Π_2^0 statements). Since our $\lambda\mu_{\mathrm{V}}$ can encode Girard's system F, so it includes, at least, all provably total functions in $\mathbf{PA}_2$. At the same time, the functions representable in second-order **LKQ** are exactly the provably total functions in $\mathbf{PA}_2$. This fact can also be understood from the fact that the intuitionistic decoration of **LKQ** can be simulated by system F.

In Selinger's co-control category, the two inputs of the application map are connected via a pretensor $\otimes$. This amounts to say that the order of compositions of morphisms does matter. Composition of morphisms corresponds to cut-elimination in proof theory. Hence this means the order of two-cuts matters in implication elimination. If we made a choice of morphisms in co-control category such that every $\otimes$ are bifunctorial, one gets a sub co-control category called the "center" of the category. On the other hand, we have made a choice (LR or RL) in order to map **CND** into **LKQ**. This observation of close resemblance between category theory and proof theory deserves further study.

Our simulation theorem says that **CND** and **LKQ** share denotation under specific reduction rules and translation. Also **LKQ** has its own denotational semantics which is invariant under the q-protocol. Specifically **LKQ** inherits the denotation in linear logic's coherent space semantics. It is shown by considering linear decoration method. Moreover, through intuitionistic decoration, we also know that one can map a center of co-control category to (sub) cartesian-closed category. Obviously, the relation between the semantics of **LKQ** and the center of the co-control category should be investigated in future work. Our conjecture is that **LKQ** is an internal language of the center of the co-control category.

References

[1] K. Baba, S. Hirokawa, and K.Fujita. Parallel reduction in type-free $\lambda\mu$-calculus. *Electronic Notes in Theoretical Computer Science*, 42, 2001. 497

[2] G. M. Bierman. A computational interpretation of the $\lambda\mu$-calculus. In *Proceedings of Symposium on Mathematical Foundations of Computer Science 98*, pages 336–345. Springer-Verlag LNCS 1450, August 1998. 496

[3] P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of ICFP*. World Scientific, September 2000. 492, 502

[4] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. Sequent calculi for second order logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 211–224. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993. 499

[5] Vincent Danos, Jean-Baptiste Joinet, and Harold Schellinx. A new deconstructive logic: linear logic. *Journal of Symbolic Logic*, 62(3), September 1997. 491, 499, 500

[6] Michael J. Fischer. Lambda-calculus schemata. *Lisp and Symbolic Computation*, 6(3/4):259–287, November 1993. 491

[7] Jean-Yves Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1:255–296, 1991. 492

[8] Timothy Griffin. A formulae-as-types notion of control. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, January 1990. 491

[9] Martin Hofmann and Thomas Streicher. Continuation models are universal for $\lambda\mu$-calculus. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*, june 1997. 491

[10] Grigori Mints. Normal forms for sequent derivations. In Piergiorgio Odifreddi, editor, *Kreiseliana – About and Around George Kreisel*. A K Peters Ltd., March 1996. 492

[11] Chetan R. Murthy. A computational analysis of Girard's translation and LC. In *Proceedings, Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 90–101, Santa Cruz, California, 22–25 June 1992. IEEE Computer Society Press. 492, 502

[12] Ichiro Ogata. Cut elimination for classical proofs as continuation passing style computation. In *Proceedings of the Asian Computing Science Conference 98*, pages 61–78, Manila, Philippines, December 1998. Springer-Verlag LNCS 1538. 492

[13] Ichiro Ogata. Gentzen-style classical proofs as $\lambda\mu$-terms. In *Proceedings of the Asian Computing Science Conference 99*, pages 266–280, Phuket, Thailand, December 1999. Springer-Verlag LNCS 1742. 492

[14] Ichiro Ogata. Constructive classical logic as cps-calculus. *International Journal of Foundations of Computer Science*, 11(1):89–112, March 2000. 492

[15] C.-H. L. Ong. A semantic view of classical proofs: type-theoretic, categorical, and denotational characterizations (preliminary extended abstract). In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 230–241, New Brunswick, New Jersey, 27–30 July 1996. IEEE Computer Society Press. 496

[16] C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 215–227, Paris, France, 15–17 January 1997. 490, 496

[17] Michel Parigot. Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. of LPAR'92*, pages 190–201. Springer-Verlag LNCS 624, 1992. 491

[18] Michel Parigot. Strong normalization for second order classical natural deduction. In *Proceedings, Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 39–46, Montreal, Canada, 19–23 June 1993. IEEE Computer Society Press. 492

[19] G. D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125–159, December 1975. 491

[20] G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977. 492

[21] D. Prawitz. *Natural Deduction, a Poof-Theoretical Study*. Almquist and Wiksell, Stockholm, 1965. 499

[22] José Espírito Santo. Revisiting the correspondence between cut elimination and normalization. In *Proc. of ICALP 2000*, pages 600–611. Springer-Verlag LNCS 1853, 2000. 498

[23] Peter Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11:207–260, 2001. 492, 502

[24] Masako Takahashi. Parallel reductions in λ-calculus. *Information and Computation*, 118(1):120–127, April 1995. 491, 496

[25] J. I. Zucker. Correspondence between cut-elimination and normalization, part i and ii. *Annals of Mathematical Logic*, 7:1–156, 1974. 492, 493

Duality between Call-by-Name Recursion and Call-by-Value Iteration

Yoshihiko Kakutani

Research Institute for Mathematical Sciences, Kyoto University
kakutani@kurims.kyoto-u.ac.jp

Abstract. We investigate the duality between call-by-name recursion and call-by-value iteration in the $\lambda\mu$-calculi and their models. Semantically, we consider that iteration is the dual notion of recursion. Syntactically, we extend the call-by-name $\lambda\mu$-calculus and the call-by-value one with a fixed-point operator and an iteration operator, respectively. This paper shows that the dual translations between the call-by-name $\lambda\mu$-calculus and the call-by-value one, which is constructed by Selinger, can be expanded to our extended $\lambda\mu$-calculi. Another result of this study provides uniformity principles for those operators.

1 Introduction

1.1 Background

In this paper, we study the duality between recursion and iteration in functional programming languages with first-class continuations.

The duality between recursion and iteration is induced by the duality between call-by-name and call-by-value, which was first formalized by Filinski in [2]. The duality between call-by-name and call-by-value is based on the duality between a direct semantics and a continuation semantics. In a direct semantics, a term $F : A \to B$ usually represents a function f which accepts a value x of the type A and returns a computation $f(x)$ of the type B. In a continuation semantics, we can consider $F : A \to B$ to transform a B-accepting continuation k to an A-accepting continuation $k \circ f$. This implies that the exchange of the value paradigm for the continuation paradigm reverses the directions of computations.

In [2], Filinski introduced the symmetric λ-calculus, which is an extension of the simply typed λ-calculus with control operators. Since $\lambda\mu$-calculi [10] also include control operators, the duality between call-by-name and call-by-value can be expanded to $\lambda\mu$-calculi. Indeed, in [12] Selinger has given categorical models to the call-by-name $\lambda\mu$-calculus (which we call the $\lambda\mu_{\mathrm{n}}$-calculus) and the call-by-value $\lambda\mu$-calculus (the $\lambda\mu_{\mathrm{v}}$-calculus). The class of models of the $\lambda\mu_{\mathrm{n}}$-calculus consists of the opposite categories of models of the $\lambda\mu_{\mathrm{v}}$-calculus. This semantical duality induces the syntactic duality.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 506–521, 2002.
© Springer-Verlag Berlin Heidelberg 2002

1.2 Recursion and Iteration

Recursion is indispensable for programming languages and has been studied extensively. However, most of such widely-known studies, which include uniformity, dinaturality, and diagonal property [1], are for call-by-name languages rather than for call-by-value ones. Therefore, it is natural to add a recursion operator to the $\lambda\mu_{\mathrm{n}}$-calculus. The aim of this work is to make it explicit what computation in the $\lambda\mu_{\mathrm{v}}$-calculus is the dual of recursion in the $\lambda\mu_{\mathrm{n}}$-calculus.

By the duality between values and continuations, we can get recursion on continuations in call-by-value languages from call-by-name recursion. Recursion on continuations is just iteration because fixed-point operators on negative types and iteration operators have bijective correspondence in the $\lambda\mu_{\mathrm{v}}$-calculus [5].

The categorical investigation leads us to the duality between call-by-name recursion and call-by-value iteration more directly, which is informally suggested by Filinski in [3]. Namely a fixed-point operator on a control category is exactly dual for an iteration operator on a co-control category. In this paper, we investigate the duality between recursion and iteration along this line, and extend the $\lambda\mu_{\mathrm{n}}$-calculus and $\lambda\mu_{\mathrm{v}}$-calculus with a fixed-point operator and an iteration operator.

On the other hand, in [3], Filinski also proposed the uniformity principles for call-by-value fixed-point operators and iteration operators. (We refined and justified the uniformity principle in [5].) This uniformity principle of call-by-value iteration is induced from effect-freeness (centrality [14]). So, we can introduce the uniformity principle for call-by-name recursion in the same way as call-by-value iteration.

1.3 Overview

In this paper, we recall the $\lambda\mu$-calculi and their categorical semantics in Section 2. Section 3 investigates the duality between call-by-name recursion and call-by-value iteration from the categorical point of view. We also introduce a fixed-point operator for the $\lambda\mu_{\mathrm{n}}$-calculus and an iteration operator for the $\lambda\mu_{\mathrm{v}}$-calculus in Section 3. In Section 4. we extend the dual translations between the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus to the recursion operator and the iteration one. Last, we propose the uniformity axioms based on effect-freeness in Section 5.

2 The $\lambda\mu$-Calculi

2.1 Syntax and Axioms

The $\lambda\mu$-calculus was first introduced by Parigot in [10]. $\lambda\mu$-calculi are extensions of λ-calculi with the notion of continuations. In this subsection, we define the syntax of the $\lambda\mu$-calculi, both the call-by-name calculus and the call-by-value one. Our version of the $\lambda\mu$-calculi is based on Selinger's [12]: including conjunction types and disjunction types. Disjunction types are the dual notion of conjunction

$$\frac{x : A \in \Gamma}{\Gamma \vdash x : A \mid \Delta} \qquad \frac{}{\Gamma \vdash * : \top \mid \Delta}$$

$$\frac{\Gamma, x : A \vdash M : B \mid \Delta}{\Gamma \vdash \lambda x^A.\, M : A \to B \mid \Delta} \qquad \frac{\Gamma \vdash M : A \to B \mid \Delta \quad \Gamma \vdash N : A \mid \Delta}{\Gamma \vdash MN : B \mid \Delta}$$

$$\frac{\Gamma \vdash M : A \mid \Delta \quad \Gamma \vdash N : B \mid \Delta}{\Gamma \vdash \langle M, N \rangle : A \wedge B \mid \Delta} \qquad \frac{\Gamma \vdash M : A_1 \wedge A_2 \mid \Delta}{\Gamma \vdash \pi_i M : A_i \mid \Delta}$$

$$\frac{\Gamma \vdash M : \bot \mid \alpha : A, \Delta}{\Gamma \vdash \mu\alpha^A.\, M : A \mid \Delta} \qquad \frac{\Gamma \vdash M : A \mid \Delta \quad \alpha : A \in \Delta}{\Gamma \vdash [\alpha]\, M : \bot \mid \Delta}$$

$$\frac{\Gamma \vdash M : \bot \mid \beta : B, \alpha : A, \Delta}{\Gamma \vdash \mu(\alpha^A, \beta^B).\, M : A \vee B \mid \Delta} \qquad \frac{\Gamma \vdash M : A \vee B \mid \Delta \quad \alpha : A, \beta : B \in \Delta}{\Gamma \vdash [\alpha, \beta]\, M : \bot \mid \Delta}$$

Fig. 1. The deduction rules of the $\lambda\mu$-calculi

types, *i.e.*, call-by-name disjunctions play the role of call-by-value conjunctions, and call-by-value disjunctions play the role of call-by-name conjunctions.

The formal syntax is the following:

$$\begin{aligned}
A, B &::= \sigma \mid A \to B \mid \top \mid A \wedge B \mid \bot \mid A \vee B, \\
M, N &::= x \mid * \mid \lambda x^A.\, M \mid MN \mid \langle M, N \rangle \mid \pi_1 M \mid \pi_2 M \\
&\quad \mid \mu\alpha^A.\, M \mid [\alpha]\, M \mid \mu(\alpha^A, \beta^B).\, M \mid [\alpha, \beta]\, M, \\
V, W &::= x \mid * \mid \lambda x^A.\, M \mid \langle V, W \rangle \mid \pi_1 V \mid \pi_2 V \\
&\quad \mid \mu(\alpha^A, \beta^B).[\alpha]\, V \quad \text{(neither } \alpha \text{ nor } \beta \text{ occurs in } V \text{ freely)} \\
&\quad \mid \mu(\alpha^A, \beta^B).[\beta]\, V \quad \text{(neither } \alpha \text{ nor } \beta \text{ occurs in } V \text{ freely)},
\end{aligned}$$

where x ranges over variables, α and β range over names, and σ ranges over base types. A, M and V are called types, terms and values, respectively. Values make sense only for the call-by-value calculus. The symbol $*$ denotes a special constant with the type $\top$. We assume the usual associative strength among connectives ($\mu\alpha^A.\,(-)$ and $[\alpha]\,(-)$ is as strong as $\lambda x^A.\,(-)$), and the set denoted by $\mathrm{FV}(-)$ of free variables on terms is defined as for λ-calculi. We also define $\mathrm{FN}(-)$, the set of free names on terms, where μ-abstractions bind names. For abbreviation, we may write $\neg A$ for $A \to \bot$ and in the call-by-value calculus we use $\texttt{let}\, x^A\, \texttt{be}\, M\, \texttt{in}\, N$ as syntactic sugar for $(\lambda x^A.\, N)M$.

Every judgment takes the form $\Gamma \vdash M : A \mid \Delta$, where Γ denotes a sequence of pairs $x : A$, and Δ denotes a sequence of pairs $\alpha : A$. The typing rules, which are applied to both the call-by-name $\lambda\mu$-calculus and the call-by-value one, are given in Figure 1. In this paper, we consider only derivable judgments and we may confuse a judgment itself with the predicate that means the judgment is deducible.

The axioms of the call-by-name $\lambda\mu$-calculus are in Figure 2. In that figure, an expression of the form $[N\,/\,x]$ means a usual substitution for free variables or names, and an expression $[\mathbf{C}[(-)]\,/\,[\alpha](-)]$, called a mixed substitution, not

$$
\begin{array}{lll}
(\beta_{\to})\ (\lambda x^A.\, M)N = M\,[N\,/x] & : B & \\
(\eta_{\to})\ \lambda x^A.\, Mx = M & : B & x \notin \mathrm{FV}(M) \\
(\beta_{\wedge})\ \pi_i\langle M_1, M_2\rangle = M_i & : A_i & \\
(\eta_{\wedge})\ \langle \pi_1 M, \pi_2 M\rangle = M & : A \wedge B & \\
(\eta_{\top})\ * = M & : \top & \\
(\beta_{\mu})\ [\beta]\mu\alpha^A.\, M = M\,[\beta\,/\alpha] & : \bot & \\
(\eta_{\mu})\ \mu\alpha^A.[\alpha]\, M = M & : A & \alpha \notin \mathrm{FN}(M) \\
(\beta_{\vee})\ [\gamma,\delta]\mu(\alpha^A,\beta^B).\, M = M\,[\gamma\,/\alpha, \delta\,/\beta] & : \bot & \\
(\eta_{\vee})\ \mu(\alpha^A,\beta^B).[\alpha,\beta]\, M = M & : A \vee B & \alpha, \beta \notin \mathrm{FN}(M) \\
(\beta_{\bot})\ [\beta]\, M = M & : \bot & \\
(\zeta_{\to})\ (\mu\alpha^{A\to B}.\, M)N = \mu\beta^B.\, M\,[[\beta](-)N\,/[\alpha](-)] & : B & \\
(\zeta_{\wedge})\ \pi_i(\mu\alpha^{A_1\wedge A_2}.\, M) = \mu\beta^{A_i}.\, M\,[[\beta]\pi_i(-)\,/[\alpha](-)] & : A_i & \\
(\zeta_{\vee})\ [\gamma,\delta]\mu\alpha^{A\vee B}.\, M = M\,[[\gamma,\delta](-)\,/[\alpha](-)] & : \bot &
\end{array}
$$

Fig. 2. The axioms of the $\lambda\mu_{\mathrm{n}}$-calculus

only replaces all free $[\alpha]\, M$ by $\mathbf{C}[M]$ but also replaces $[\alpha,\beta]\, M$ and $[\beta,\alpha]\, M$ by $\mathbf{C}[\mu\alpha^A.[\alpha,\beta]\, M]$ and $\mathbf{C}[\mu\alpha^A.[\beta,\alpha]\, M]$ respectively. We call this call-by-name $\lambda\mu$-calculus **the $\boldsymbol{\lambda\mu_{\mathrm{n}}}$-calculus**, and we call the call-by-value $\lambda\mu$-calculus **the $\boldsymbol{\lambda\mu_{\mathrm{v}}}$-calculus**. The axioms of the the $\lambda\mu_{\mathrm{v}}$-calculus are given by Figure 3. The $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus are variants of Parigot's $\lambda\mu$-calculi [10] and Ong-Stewart's $\lambda\mu_{\mathrm{v}}$-calculus [9]. Especially we note that the $\lambda\mu_{\mathrm{v}}$-calculus is an extension of Moggi's λ_{c}-calculus [8].

2.2 Control Categories

According to Selinger, the $\lambda\mu_{\mathrm{n}}$-calculus has a complete class of models called **control categories** [12], while the $\lambda\mu_{\mathrm{v}}$-calculus has a complete class of models called **co-control categories**. A co-control category is the opposite category of a control category. So it is natural that there exists the dual correspondence between the $\lambda\mu_{\mathrm{v}}$-calculus and the $\lambda\mu_{\mathrm{n}}$-calculus.

Following Selinger, we shall characterize control categories by response categories. Let $\mathcal{C}$ be a category that has distributive finite products and coproducts and distinguished object R such that R^A exists for any A. We call $\mathcal{C}$ **a response category** (and call R its object of responses), if $\mathcal{C}$ satisfies the mono requirement, *i.e.*, for any A, the canonical morphism $\partial_A : A \to R^{R^A}$ is monic. Given a response category $\mathcal{C}$, we define its **category of continuations** $R^{\mathcal{C}}$, which has the same objects as $\mathcal{C}$ and the morphisms defined by $R^{\mathcal{C}}(A,B) = \mathcal{C}(R^A, R^B)$. Here we remark that the opposite category of continuations $(R^{\mathcal{C}})^{\mathrm{op}}$ can be considered as Kleisli category of the continuation monad $R^{R^{(-)}}$ on $\mathcal{C}$.

It can be seen that a category of continuations has a cartesian closed structure. Indeed, in terms of $\mathcal{C}$,

$$1 \cong R^0, \qquad R^A \times R^B \cong R^{A+B}, \qquad (R^B)^{R^A} \cong R^{B\times R^A}$$

hold. Moreover $R^{\mathcal{C}}$ has a premonoidal structure [11] $\mathfrak{R}$:

$$
\begin{array}{rll}
(\beta_{\to})\ \mathtt{let}\ x^A\ \mathtt{be}\ V\ \mathtt{in}\ M = M\,[V\,/x] & :B & \\
(\eta_{\to})\ \lambda x^A.\,Vx = V & :A \to B & x \notin \mathrm{FV}(V) \\
(\beta_{\wedge})\ \pi_i\langle V_1, V_2\rangle = V_i & :A_i & \\
(\eta_{\wedge})\ \langle \pi_1 V, \pi_2 V\rangle = V & :A \wedge B & \\
(\eta_{\top})\ * = V & :\top & \\
(\beta_{\mu})\ [\beta]\mu\alpha^A.\,M = M\,[\beta\,/\alpha] & :\bot & \\
(\eta_{\mu})\ \mu\alpha^A.[\alpha]\,M = M & :A & \alpha \notin \mathrm{FN}(M) \\
(\beta_{\vee})\ [\gamma,\delta]\mu(\alpha^A,\beta^B).\,M = M\,[\gamma\,/\alpha, \delta\,/\beta] & :\bot & \\
(\eta_{\vee})\ \mu(\alpha^A,\beta^B).[\alpha,\beta]\,M = M & :A \vee B & \alpha,\beta \notin \mathrm{FN}(M) \\
(\beta_{\bot})\ [\beta]\,M = M & :\bot & \\
(\mathrm{let}_{\to})\ MN = \mathtt{let}\ x^{A\to B}\ \mathtt{be}\ M\ \mathtt{in}\ \mathtt{let}\ y^A\ \mathtt{be}\ N\ \mathtt{in}\ xy & :B & x \notin \mathrm{FV}(N) \\
(\mathrm{let}_{\wedge})\ \langle M, N\rangle = \mathtt{let}\ x^A\ \mathtt{be}\ M\ \mathtt{in}\ \mathtt{let}\ y^B\ \mathtt{be}\ N\ \mathtt{in}\ \langle x, y\rangle & :A \wedge B & x \notin \mathrm{FV}(N) \\
(\mathrm{let}_{\pi})\ \pi_i M = \mathtt{let}\ x^{A_1\wedge A_2}\ \mathtt{be}\ M\ \mathtt{in}\ \pi_i x & :A_i & \\
(\mathrm{let}_{\bot})\ [\alpha]\,M = \mathtt{let}\ x^A\ \mathtt{be}\ M\ \mathtt{in}\ [\alpha]\,x & :\bot & \\
(\mathrm{let}_{\vee})\ [\alpha,\beta]\,M = \mathtt{let}\ x^{A\vee B}\ \mathtt{be}\ M\ \mathtt{in}\ [\alpha,\beta]\,x & :\bot & \\
(\mathrm{comp})\ \mathtt{let}\ y^B\ \mathtt{be}\ (\mathtt{let}\ x^A\ \mathtt{be}\ M\ \mathtt{in}\ N)\ \mathtt{in}\ L & & \\
\quad = \mathtt{let}\ x^A\ \mathtt{be}\ M\ \mathtt{in}\ \mathtt{let}\ y^B\ \mathtt{be}\ N\ \mathtt{in}\ L & :C & x \notin \mathrm{FV}(L) \\
(\mathrm{id})\ \mathtt{let}\ x^A\ \mathtt{be}\ M\ \mathtt{in}\ x = M & :A & \\
(\zeta)\ V(\mu\alpha^A.M) = \mu\beta^B.\,M\,[[\beta]\,V(-)\,/[\alpha](-)] & :B &
\end{array}
$$

Fig. 3. The axioms of the $\lambda\mu_{\mathrm{v}}$-calculus

$\bot$ can be defined by R^1 and $R^A \parr R^B$ can be defined by $R^{A\times B}$.

Proposition 1 (Selinger [12]). *Let $\mathcal{C}$ be a response category with R. The category of continuations $R^{\mathcal{C}}$ is a control category.*

The proposition above claims that a category of continuations is an example of a control category, but Selinger has shown that any control category essentially arises as a category of continuations.

Theorem 1 (Selinger [12]). *Any control category is equivalent to a category of continuations.*

2.3 Models of the $\lambda\mu$-Calculi

In this subsection, we outline the interpretation of the $\lambda\mu_{\mathrm{n}}(\lambda\mu_{\mathrm{v}})$-calculus in a (co-)control category.

The type interpretation of the $\lambda\mu_{\mathrm{n}}$-calculus is defined by

$$
\begin{array}{rlrl}
[\![\sigma]\!]_{\mathrm{n}} = \underline{\sigma}, & & [\![A \to B]\!]_{\mathrm{n}} = [\![B]\!]_{\mathrm{n}}^{[\![A]\!]_{\mathrm{n}}}, & \\
[\![A \wedge B]\!]_{\mathrm{n}} = [\![A]\!]_{\mathrm{n}} \times [\![B]\!]_{\mathrm{n}}, & & [\![\top]\!]_{\mathrm{n}} = 1, & \\
[\![A \vee B]\!]_{\mathrm{n}} = [\![A]\!]_{\mathrm{n}} \parr [\![B]\!]_{\mathrm{n}}, & & [\![\bot]\!]_{\mathrm{n}} = \bot, &
\end{array}
$$

while the type interpretation of the $\lambda\mu_{\mathrm{v}}$-calculus is defined by

$$\begin{array}{rlrl} [\![\sigma]\!]_{\mathrm{v}} = \underline{\sigma}, & & [\![A \to B]\!]_{\mathrm{v}} = [\![A]\!]_{\mathrm{v}} \multimap [\![B]\!]_{\mathrm{v}}, \\ [\![A \wedge B]\!]_{\mathrm{v}} = [\![A]\!]_{\mathrm{v}} \otimes [\![B]\!]_{\mathrm{v}}, & & [\![\top]\!]_{\mathrm{v}} = \top, \\ [\![A \vee B]\!]_{\mathrm{v}} = [\![A]\!]_{\mathrm{v}} + [\![B]\!]_{\mathrm{v}}, & & [\![\bot]\!]_{\mathrm{v}} = 0, \end{array}$$

where $\underline{\sigma}$ is an object assigned to each base type σ. The operators are defined in [12]. ($+$ forms coproducts. $\otimes$ is the dual operator of $\parr$. $A \multimap B$ is the dual of $\bot^{A^B}$.)

A $\lambda\mu_{\mathrm{n}}$-judgment $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is interpreted by a morphism from $[\![B_1]\!]_{\mathrm{n}} \times [\![B_2]\!]_{\mathrm{n}} \times \ldots \times [\![B_n]\!]_{\mathrm{n}}$ to $[\![A]\!]_{\mathrm{n}} \parr [\![A_1]\!]_{\mathrm{n}} \parr [\![A_2]\!]_{\mathrm{n}} \parr \ldots \parr [\![A_m]\!]_{\mathrm{n}}$ in a control category. On the other hand, a $\lambda\mu_{\mathrm{v}}$-judgment $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$ is interpreted to a morphism from $[\![B_1]\!]_{\mathrm{v}} \otimes [\![B_2]\!]_{\mathrm{v}} \otimes \ldots \otimes [\![B_n]\!]_{\mathrm{v}}$ to $[\![A]\!]_{\mathrm{v}} + [\![A_1]\!]_{\mathrm{v}} + [\![A_2]\!]_{\mathrm{v}} \ldots + [\![A_m]\!]_{\mathrm{v}}$ in a co-control category. We shall omit the details of the interpretations, which are given as the CPS translations (the reader is referred to [12]).

2.4 Centrality

In call-by-value languages, values are considered to represent effect-free computations, but effect-free computations should not be characterized by values. Centrality represents a sort of effect-freeness in a control category.

Definition 1. *A morphism $f : A \to B$ in a control category $\mathcal{P}$ is* ***central*** *if for every morphism $g \in \mathcal{P}(C, D)$, $(B \parr g) \circ (f \parr C) = (f \parr D) \circ (A \parr g)$ and $(g \parr B) \circ (C \parr f) = (D \parr f) \circ (g \parr A)$.*

The subcategory formed by the central morphisms of a control category $\mathcal{P}$ is called the **center** of $\mathcal{P}$ and denoted by $\mathcal{P}^{\bullet}$.

Some properties of central morphisms in a control category (for example, any central morphism is discardable and copyable) are found in [12].

3 Duality between Recursion and Iteration

3.1 Recursion and Iteration

Because the call-by-name λ-calculus is the subcalculus of the $\lambda\mu_{\mathrm{n}}$-calculus, models of the $\lambda\mu_{\mathrm{n}}$-calculus are also models of the call-by-name λ-calculus. Properties of fixed-point operators on the call-by-name λ-calculus have been studied widely, for example, uniformity, dinaturality, diagonal property and so on (see [13] for recent results). So it is natural to start our investigation by studying recursion in the call-by-name $\lambda\mu$-calculus and its models.

Definition 2. ***A parameterized fixed-point operator*** *on a control category $\mathcal{P}$ is a family of functions*

$$(-)^{\dagger} : \mathcal{P}(X \times A, A \parr Y) \to \mathcal{P}(X, A \parr Y)$$

satisfying the following:

– *It is natural in X in $\mathcal{P}$ and natural in Y in $\mathcal{P}^\bullet$:*
for any $f \in \mathcal{P}(X \times A, A \parr Y)$, $g \in \mathcal{P}(X', X)$, $h \in \mathcal{P}^\bullet(Y, Y')$. $f^\dagger \circ g = (f \circ (g \times A))^\dagger$ and $(A \parr h) \circ f^\dagger = ((A \parr h) \circ f)^\dagger$.
– *For any $f \in \mathcal{P}(X \times A, A \parr Y)$,*

$$\begin{tikzcd} X \arrow[r, "\langle w^{\mathrm{l}}_{X,Y}, f^\dagger \rangle"] \arrow[rr, bend right, "f \parr Y"'] & (X \parr Y) \times (A \parr Y) \arrow[r, "d_{X,A,Y}"] & (X \times A) \parr Y \arrow[r, "A \parr \nabla_Y"'] & A \parr Y \parr Y \arrow[r] & A \parr Y \end{tikzcd}$$

and $f^\dagger : X \to A \parr Y$ agree [1].

Filinski has claimed in [2] that call-by-value iteration is the dual notion of call-by-name recursion. Below we show the case for the $\lambda\mu$-calculi in a control category and its opposite category.

Let $\mathcal{P}$ be a control category and $(-)^\dagger$ is a parameterized fixed-point operator on $\mathcal{P}$. We introduce $(-)_\dagger$ as the dual operator of $(-)^\dagger$ in the opposite category of $\mathcal{P}$. If the parameterization is trivialized, the following dual equations are induced:

$$\frac{f \in \mathcal{P}(A, A)}{f^\dagger \in \mathcal{P}(\top, A),} \qquad \frac{f \in \mathcal{P}^{\mathrm{op}}(A, A)}{f_\dagger \in \mathcal{P}^{\mathrm{op}}(A, \bot),}$$

$$f^\dagger = f \circ f^\dagger = f \circ \cdots \circ f \circ f^\dagger, \qquad f_\dagger = f_\dagger \circ f = f_\dagger \circ f \circ \cdots \circ f.$$

The duality seems to turn programs inside-out, and $f_\dagger$ seems to iterate the computation f. So we call the dual of recursion iteration.

Here, we confirm the definition of the exactly dual notion of parameterized fixed-point operators in co-control categories.

Definition 3. ***A parameterized iteration operator*** *on a co-control category $\mathcal{D}$ is a family of functions*

$$(-)_\dagger : \mathcal{D}(Y \otimes A, A + X) \to \mathcal{D}(Y \otimes A, X)$$

which is a parameterized fixed-point operator on the control category $\mathcal{D}^{\mathrm{op}}$.

Example 1. A non-trivial example is given in the following. We consider the category of ω-cpos and ω-continuous maps as a response category $\mathcal{C}$. ($R^{\mathcal{C}}$ is a control category.) Let R be an ω-cpo that has a bottom element. Because each R^A has a bottom, every $f \in (R^A)^{R^A}$ has a least fixed-point. Then we can get a fixed-point operator on $R^{\mathcal{C}}$ via the natural isomorphism

$$R^{\mathcal{C}}(X \times A, A \parr Y) \cong \mathcal{C}(Y \times R^X \times R^A, R^A).$$

[1] The canonical morphisms w, d, ∇ are defined in [12] by Selinger.

3.2 Fixed-Point Operators and Iteration Operators

Following the semantic insight in the previous subsection, we shall consider the duality syntactically. We add a family of constants $\{\mathtt{fix}^A \mid A \text{ is a type}\}$ to the $\lambda\mu_{\mathrm{n}}$-calculus, and a family of constants $\{\mathtt{loop}^A \mid A \text{ is a type}\}$ to the $\lambda\mu_{\mathrm{v}}$-calculus, where $\{\mathtt{fix}^A\}$ and $\{\mathtt{loop}^A\}$ are called **a fixed-point operator** and **an iteration operator** respectively.

The typing rules and the equality axioms are standard ones:

$$\overline{\Gamma \vdash \mathtt{fix}^A : (A \to A) \to A \mid \Delta}, \qquad \overline{\Gamma \vdash \mathtt{loop}^A : (A \to A) \to A \to \bot \mid \Delta},$$

and

$$\begin{array}{rll} \text{(fix)} & \Gamma \vdash \mathtt{fix}^A =_{\mathrm{n}} \lambda m^{A\to A}.\, m(\mathtt{fix}^A\, m) & : (A \to A) \to A \quad \mid \Delta, \\ \text{(loop)} & \Gamma \vdash \mathtt{loop}^A =_{\mathrm{v}} \lambda f^{A\to A}.\lambda x^A.\, \mathtt{loop}^A\, f\, (f x) & : (A \to A) \to A \to \bot \mid \Delta. \end{array}$$

It follows that $\mathtt{fix}\, M =_{\mathrm{n}} M(\mathtt{fix}^A\, M)$ holds for any $\lambda\mu_{\mathrm{n}}$-term $M : A \to A$, and $\mathtt{loop}\, F =_{\mathrm{v}} (\mathtt{loop}^A\, F) \circ F$ holds for any $\lambda\mu_{\mathrm{v}}$-value $F : A \to A$.

Remark 1. Despite of its restricted type, `loop` has enough expressive power. Indeed, we can define a general feedback operator `feedback` from `loop`:

$$\begin{aligned} &\mathtt{feedback} : (A \to B \vee A) \to A \to B \\ &\equiv \lambda f^{A\to B\vee A}.\lambda x^A.\mu\beta^B.\, \mathtt{loop}(\lambda y^A.\mu\alpha^A.[\beta, \alpha]\,(f y))\, x\ . \end{aligned}$$

$\mathtt{fix}^A$ and $\mathtt{loop}^A$ in the $\lambda\mu$-calculi represent exactly a parameterized fixed-point operator in a control category and a parameterized iteration operator in a co-control category.

Theorem 2. *Control categories with parameterized fixed-point operators provide a sound and complete class of models of the $\lambda\mu_{\mathrm{n}}$-calculus with a fixed-point operator.*

Theorem 3. *Co-control categories with parameterized iteration operators provide a sound and complete class of models of the $\lambda\mu_{\mathrm{v}}$-calculus with an iteration operator.*

Remark 2. We can also extend the CPS translations (defined in [12]) to the $\lambda\mu_{\mathrm{n}}$-calculus with a fixed-point operator and the $\lambda\mu_{\mathrm{v}}$-calculus with an iteration operator. The original CPS target calculus is the simply typed λ-calculus with finite products and finite coproducts. So, we cannot extend the target calculus with a generic fixed-point combinator because a distributive category that has a fixed-point operator is trivial [6]. However, since our extended CPS target calculus requires only a fixed-point combinator on negative types, we can extend the CPS translation validly.

$$
\begin{array}{rcl}
(\!|\sigma|\!) & \equiv & \sigma \\
(\!|A \wedge B|\!) & \equiv & (\!|A|\!) \vee (\!|B|\!) \\
(\!|A \vee B|\!) & \equiv & (\!|A|\!) \wedge (\!|B|\!)
\end{array}
\qquad
\begin{array}{rcl}
(\!|A \to B|\!) & \equiv & ((\!|B|\!) \to (\!|A|\!)) \to \bot \\
(\!|\top|\!) & \equiv & \bot \\
(\!|\bot|\!) & \equiv & \top
\end{array}
$$

$$
\begin{array}{rcll}
(\!|x|\!) & \equiv & \lambda\kappa^{(\!|A|\!)}.[x]\,\kappa & x : A \\
(\!|*|\!) & \equiv & \lambda\kappa^{(\!|\top|\!)}.\,\kappa & \\
(\!|\lambda x^A.\,M|\!) & \equiv & \lambda\kappa^{(\!|A\to B|\!)}.\,\kappa(\lambda\beta^{(\!|B|\!)}.\mu x^{(\!|A|\!)}.\,(\!|M|\!)\beta) & M : B \\
(\!|MN|\!) & \equiv & \lambda\kappa^{(\!|B|\!)}.\,(\!|M|\!)(\lambda\gamma^{(\!|B|\!)\to(\!|A|\!)}.\,(\!|N|\!)(\gamma\kappa)) & M : A \to B,\ N : A \\
(\!|\langle M, N\rangle|\!) & \equiv & \lambda\kappa^{(\!|A\wedge B|\!)}.\,(\!|M|\!)(\mu x^{(\!|A|\!)}.\,(\!|N|\!)(\mu y^{(\!|B|\!)}.[x,y]\,\kappa)) & M : A,\ N : B \\
(\!|\pi_i M|\!) & \equiv & \lambda\kappa^{(\!|A_i|\!)}.\,(\!|M|\!)(\mu(x_1^{A_1}, x_2^{A_2}).[x_i]\,\kappa) & M : A_1 \wedge A_2 \\
(\!|\mu\alpha^A.\,M|\!) & \equiv & \lambda\kappa^{(\!|A|\!)}.\,(\lambda\alpha^{(\!|A|\!)}.\,(\!|M|\!)\diamond)\kappa & \\
(\!|[\alpha]\,M|\!) & \equiv & \lambda\kappa^{(\!|\bot|\!)}.\,(\!|M|\!)\alpha & \\
(\!|\mu(\alpha^A, \beta^B).\,M|\!) & \equiv & \lambda\kappa^{(\!|A\vee B|\!)}.\,(\lambda\alpha^{(\!|A|\!)}.\lambda\beta^{(\!|B|\!)}.\,(\!|M|\!)\diamond)(\pi_1\kappa)\,(\pi_2\kappa) & \\
(\!|[\alpha,\beta]\,M|\!) & \equiv & \lambda\kappa^{(\!|\bot|\!)}.\,(\!|M|\!)\langle\alpha,\beta\rangle &
\end{array}
$$

Fig. 4. The dual translation from the $\lambda\mu_{\mathrm{v}}$ to the $\lambda\mu_{\mathrm{n}}$

4 Duality on the $\lambda\mu$-Calculi

4.1 The Dual Translations

In order to deal with the duality plainly, we assume that the set of names in the $\lambda\mu_{\mathrm{n}}$-calculus is the same as the set of variables in the $\lambda\mu_{\mathrm{v}}$-calculus, while the set of variables in the $\lambda\mu_{\mathrm{n}}$-calculus is the same as the set of names in the $\lambda\mu_{\mathrm{v}}$-calculus. We also assume in the $\lambda\mu_{\mathrm{n}}$-calculus $\diamond$ denotes the constant that plays the role of $*$ because $*$ in the $\lambda\mu_{\mathrm{n}}$-calculus is not the dual of $*$ in the $\lambda\mu_{\mathrm{v}}$-calculus. We will define translations between the $\lambda\mu_{\mathrm{n}}$-calculus and the $\lambda\mu_{\mathrm{v}}$-calculus. These dual translations are just the syntactic incarnation of the categorical duality. Now especially we shall pick up the recursion part:

$$
\begin{array}{rcl}
(\!|\,\mathtt{loop}^A\,|\!) & \equiv & \lambda\kappa^{(\!|(A\to A)\to A\to\bot|\!)}.\,\kappa(\lambda\gamma^{\neg(\top\to(\!|A|\!))}.\lambda\phi^{(\!|A|\!)\to(\!|A|\!)}.\,\gamma(\lambda\tau^{\top}.\,\mathtt{fix}^{(\!|A|\!)}\,\phi)), \\
(\!|\,\mathtt{fix}^A\,|\!) & \equiv & \lambda k^{(\!|(A\to A)\to A|\!)}.\,\mathtt{loop}^{(\!|A|\!)}(\lambda x^{(\!|A|\!)}.\mu\beta^{(\!|A|\!)}.\,\pi_1 k\,\langle\lambda y^{(\!|A|\!)}.[\beta]\,y, x\rangle)\,(\pi_2 k).
\end{array}
$$

Rest of the definitions, due to Selinger [12], are in Figure 4 and 5.

The following propositions guarantee that these translations are sound for the typing and for the equational theories.

Proposition 2. *For any $\lambda\mu_{\mathrm{v}}$-judgment* $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,

$$\alpha_1 : (\!|A_1|\!), \ldots, \alpha_m : (\!|A_m|\!), \kappa : (\!|A|\!) \vdash (\!|M|\!)\kappa : \bot \mid x_1 : (\!|B_1|\!), \ldots, x_n : (\!|B_n|\!),$$

and for any $\lambda\mu_{\mathrm{n}}$-judgment $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$x_1 : (\!|A_1|\!), \ldots, x_m : (\!|A_m|\!), k : (\!|A|\!) \vdash (\!|M|\!)k : \bot \mid \alpha_1 : (\!|B_1|\!), \ldots, \alpha_n : (\!|B_n|\!).$$

Proposition 3. *Each of the translations $(\!|(-)|\!)$ and $(\!|(-)|\!)$ preserves the equality.*

$$
\begin{aligned}
\llparenthesis \sigma \rrparenthesis &\equiv \sigma & \llparenthesis A \to B \rrparenthesis &\equiv (\llparenthesis A \rrparenthesis \to \bot) \wedge \llparenthesis B \rrparenthesis \\
\llparenthesis A \wedge B \rrparenthesis &\equiv \llparenthesis A \rrparenthesis \vee \llparenthesis B \rrparenthesis & \llparenthesis \top \rrparenthesis &\equiv \bot \\
\llparenthesis A \vee B \rrparenthesis &\equiv \llparenthesis A \rrparenthesis \wedge \llparenthesis B \rrparenthesis & \llparenthesis \bot \rrparenthesis &\equiv \top
\end{aligned}
$$

$$
\begin{aligned}
\llparenthesis \alpha \rrparenthesis &\equiv \lambda k^{\llparenthesis A \rrparenthesis}.[\alpha]\, k && \alpha : A \\
\llparenthesis \diamond \rrparenthesis &\equiv \lambda k^{\llparenthesis \top \rrparenthesis}.\, k \\
\llparenthesis \lambda \alpha^A.\, M \rrparenthesis &\equiv \lambda k^{\llparenthesis A \to B \rrparenthesis}.\, \pi_1 k\, (\mu \alpha^{\llparenthesis A \rrparenthesis}.\, \llparenthesis M \rrparenthesis (\pi_2 k)) && M : B \\
\llparenthesis M N \rrparenthesis &\equiv \lambda k^{\llparenthesis B \rrparenthesis}.\, \llparenthesis M \rrparenthesis \langle \llparenthesis N \rrparenthesis, k \rangle && M : A \to B,\ N : A \\
\llparenthesis \langle M, N \rangle \rrparenthesis &\equiv \lambda k^{\llparenthesis A \wedge B \rrparenthesis}.\, \llparenthesis M \rrparenthesis (\mu \alpha^{\llparenthesis A \rrparenthesis}.\, \llparenthesis N \rrparenthesis (\mu \beta^{\llparenthesis B \rrparenthesis}.[\alpha, \beta]\, k)) && M : A,\ N : B \\
\llparenthesis \pi_i M \rrparenthesis &\equiv \lambda k^{\llparenthesis A_i \rrparenthesis}.\, \llparenthesis M \rrparenthesis (\mu (\alpha_1^{\llparenthesis A_1 \rrparenthesis}, \alpha_2^{\llparenthesis A_2 \rrparenthesis}).[\alpha_i]\, k) && M : A_1 \wedge A_2 \\
\llparenthesis \mu x^A.\, M \rrparenthesis &\equiv \lambda k^{\llparenthesis A \rrparenthesis}.\, (\lambda x^{\llparenthesis A \rrparenthesis}.\, \llparenthesis M \rrparenthesis *) k \\
\llparenthesis [x]\, M \rrparenthesis &\equiv \lambda k^{\llparenthesis \bot \rrparenthesis}.\, \llparenthesis M \rrparenthesis x \\
\llparenthesis \mu (x^A, y^B).\, M \rrparenthesis &\equiv \lambda k^{\llparenthesis A \vee B \rrparenthesis}.\, (\lambda x^{\llparenthesis A \rrparenthesis}.\lambda y^{\llparenthesis B \rrparenthesis}.\, \llparenthesis M \rrparenthesis *)(\pi_1 k)\, (\pi_2 k) \\
\llparenthesis [x, y]\, M \rrparenthesis &\equiv \lambda k^{\llparenthesis \bot \rrparenthesis}.\, \llparenthesis M \rrparenthesis \langle x, y \rangle
\end{aligned}
$$

Fig. 5. The dual translation from the $\lambda\mu_{\mathrm{n}}$ to the $\lambda\mu_{\mathrm{v}}$

Proof. Check all the equality axioms. For example, the (loop) case follows from $(\!|\, \mathtt{loop}^A\, f\, x \,|\!)\kappa =_{\mathrm{n}} [f](\lambda\phi^{(\!|A|\!)\to(\!|A|\!)}.[x]\,(\mathtt{fix}^{(\!|A|\!)}\, \phi))$ and the equation (fix). □

Moreover, from the semantic point of view, the translations are mutually inverse up to natural isomorphisms. For example, we can check the translations preserve the CPS transforms up to some simple isomorphisms. In the following subsections, we formalize and demonstrate it syntactically in the $\lambda\mu$-calculi.

4.2 From $\lambda\mu_{\mathrm{v}}$ to $\lambda\mu_{\mathrm{v}}$ via $\lambda\mu_{\mathrm{n}}$

Our claim is that the composite $\mu\kappa^A.\, \llparenthesis (\!|(-)|\!)\kappa \rrparenthesis *$ is equivalent to the identity up to natural isomorphisms. By the definition of the type translations, we can easily see that the composite $\llparenthesis (\!|(-)|\!) \rrparenthesis$ is not the identity generally, but the type $\llparenthesis (\!|A|\!) \rrparenthesis$ looks isomorphic to A. Indeed the following isomorphism exists in a co-control category:

$$
\begin{aligned}
&[\![\llparenthesis (\!|A|\!) \rrparenthesis \to \llparenthesis (\!|B|\!) \rrparenthesis]\!]_{\mathrm{v}} = [\![\llparenthesis (\!|A|\!) \rrparenthesis]\!]_{\mathrm{v}} \multimap [\![\llparenthesis (\!|B|\!) \rrparenthesis]\!]_{\mathrm{v}} \\
&\cong (((\![\llparenthesis (\!|B|\!) \rrparenthesis]\!]_{\mathrm{v}} \multimap 0) \otimes [\![\llparenthesis (\!|A|\!) \rrparenthesis]\!]_{\mathrm{v}}) \multimap 0) \otimes \top = [\![\llparenthesis (\!|A \to B|\!) \rrparenthesis]\!]_{\mathrm{v}}.
\end{aligned}
$$

According to this categorical consideration, we construct the terms $\mathrm{I}_0^{A\to B}$: $(A \to B) \to (((B \to \bot) \wedge A) \to \bot) \wedge \top$ and $\mathrm{J}_0^{A\to B}$: $(((B \to \bot) \wedge A) \to \bot) \wedge \top \to A \to B$ in the $\lambda\mu_{\mathrm{v}}$-calculus:

$$
\begin{aligned}
\mathrm{I}_0^{A\to B} &\equiv \lambda f^{A\to B}.\, \langle \lambda k^{\neg B \wedge A}.\, (\pi_1 k)(f(\pi_2 k)), * \rangle, \\
\mathrm{J}_0^{A\to B} &\equiv \lambda m^{\neg(\neg B \wedge A) \wedge \top}.\lambda x^A.\mu\beta^B.\, (\pi_1 m)\langle \lambda y^B.[\beta]\, y, x \rangle.
\end{aligned}
$$

Indeed $\mathrm{I}_0^{A\to B}$ and $\mathrm{J}_0^{A\to B}$ are isomorphisms, that is, $\mathrm{J}_0^{A\to B}(\mathrm{I}_0^{A\to B} f) =_{\mathrm{v}} f$ and $\mathrm{I}_0^{A\to B}(\mathrm{J}_0^{A\to B} m) =_{\mathrm{v}} m$ hold. The isomorphisms $\mathrm{I}_A : A \to \llparenthesis (\!|A|\!) \rrparenthesis$ and $\mathrm{J}_A : \llparenthesis (\!|A|\!) \rrparenthesis \to A$, which are mutually inverse, are recursively defined from these $\{\mathrm{I}_0^{A\to B}\}$ and

$\{\mathrm{J}_0^{A\to B}\}$. Hence we get the following proposition by fitting $\mu\kappa.\,\langle\!|(\!|M|\!)|\kappa\rangle\!|*$ with $\{\mathrm{I}_A\}$ and $\{\mathrm{J}_A\}$.

Proposition 4. *In the $\lambda\mu_{\mathrm{v}}$-calculus without* `loop`*,*
for any judgment $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,

$$\begin{aligned}&x_1 : B_1, \ldots, x_n : B_n \vdash \\ &\mathrm{J}_A(\mu\kappa^{(\!|A|\!)}.\,\langle\!|(\!|M|\!)|\kappa\rangle\!|*)\,[\mathrm{I}_{B_1}x_1\,/x_1, \ldots,\, [\alpha_1]\,\mathrm{J}_{A_1}(-)\,/[\alpha_1]\,(-), \ldots] =_{\mathrm{v}} M \\ &: A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m.\end{aligned}$$

Remark 3. One would expect the substitutions in the foregoing proposition to mean *parallel* substitutions. However, $[[\alpha_i]\,\mathrm{J}_{A_i}(-)/[\alpha_i]\,(-),\, [\alpha_j]\,\mathrm{J}_{A_j}(-)/[\alpha_j]\,(-)]$ is a problematic substitution if $[\alpha_j, \alpha_i]$ occurs in the target term freely. Here we define the multi-substitution as a sequential composition of single-substitutions. Comparing a term $M[\ldots,\, [\alpha_i]\,\mathrm{J}_{A_i}(-)\,/[\alpha_i]\,(-),\, [\alpha_j]\,\mathrm{J}_{A_j}(-)\,/[\alpha_j]\,(-), \ldots]$ with a term $M[\ldots,\, [\alpha_j]\,\mathrm{J}_{A_j}(-)\,/[\alpha_j]\,(-),\, [\alpha_i]\,\mathrm{J}_{A_i}(-)\,/[\alpha_i]\,(-), \ldots]$, we see that these two terms are equal to each other in the $\lambda\mu_{\mathrm{v}}$-theory, and furthermore, we also get the term equal to them even if we apply the substitution replacing $[\alpha_i, \alpha_j](-)$ by $[\alpha_i, \alpha_j]\,\mathrm{J}_{A_i\vee A_j}(-)$, because $\mathrm{J}_{A_i\vee A_j}$ is an isomorphism. Thus, the multi-substitution is well-defined and there is no need to take care whether some substitutions conflict in a parallel substitution.

If the iterator $\mathtt{loop}^A$ occurs in M, then $\mathtt{loop}^{(\!|A|\!)}$ occurs in $\mu\kappa.\langle\!|(\!|M|\!)|\kappa\rangle\!|*$. Therefore we have to extend the substitutions to include the replacement of $\mathtt{loop}^{(\!|A|\!)}$ by $\lambda f.\lambda x.\ \mathtt{loop}^A(\lambda z.\,\mathrm{J}_A(f(\mathrm{I}_A z)))\,(\mathrm{J}_A x)$.

We define

$$\begin{aligned}\mathrm{I}^A_{\mathrm{loop}} &\equiv \lambda l^{(A\to A)\to\neg A}.\lambda f^{(\!|A|\!)\to(\!|A|\!)}.\lambda x^{(\!|A|\!)}.\,l(\lambda z^A.\,\mathrm{J}_A(f(\mathrm{I}_A z)))\,(\mathrm{J}_A x) \qquad \text{and} \\ \mathrm{J}^A_{\mathrm{loop}} &\equiv \lambda l^{((\!|A|\!)\to(\!|A|\!))\to\neg(\!|A|\!)}.\lambda f^{A\to A}.\lambda x^A.\,l(\lambda z^{(\!|A|\!)}.\,\mathrm{I}_A(f(\mathrm{J}_A z)))\,(\mathrm{I}_A x).\end{aligned}$$

$\mathrm{J}^A_{\mathrm{loop}} \circ \mathrm{I}^A_{\mathrm{loop}}$ and $\mathrm{I}^A_{\mathrm{loop}} \circ \mathrm{J}^A_{\mathrm{loop}}$ are not identities but because $\mathtt{loop}^A\, f : A \to \bot$ is a value, the terms applied to $\mathtt{loop}^A$ are equal to $\mathtt{loop}^A$, *i.e.*, $\mathrm{J}^A_{\mathrm{loop}}(\mathrm{I}^A_{\mathrm{loop}}\,\mathtt{loop}^A) =_{\mathrm{v}} \mathtt{loop}^A$ and $\mathrm{I}^A_{\mathrm{loop}}(\mathrm{J}^A_{\mathrm{loop}}\,\mathtt{loop}^{(\!|A|\!)}) =_{\mathrm{v}} \mathtt{loop}^{(\!|A|\!)}$.

Theorem 4. *In the $\lambda\mu_{\mathrm{v}}$-calculus with* `loop`
for any judgment $x_1 : B_1, \ldots, x_n : B_n \vdash M : A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m$,

$$\begin{aligned}&x_1 : B_1, \ldots, x_n : B_n \vdash \\ &\mathrm{J}_A(\mu\kappa^{(\!|A|\!)}.\,\langle\!|(\!|M|\!)|\kappa\rangle\!|*)\,[\ldots,\, \mathrm{I}^{D_i}_{\mathrm{loop}}\,\mathtt{loop}^{D_i}\,/\mathtt{loop}^{(\!|D_i|\!)}, \ldots] =_{\mathrm{v}} M \\ &: A \mid \alpha_1 : A_1, \ldots, \alpha_m : A_m,\end{aligned}$$

where D_i ranges over all types D such that $\mathtt{loop}^D$ occurs in M.

4.3 From $\lambda\mu_{\mathrm{n}}$ to $\lambda\mu_{\mathrm{n}}$ via $\lambda\mu_{\mathrm{v}}$

In the similar way of the previous case, we define in the $\lambda\mu_{\mathrm{n}}$-calculus the isomorphisms $\mathrm{G}_A : A \to (\!|\langle\!|A|\!\rangle|\!)$ and $\mathrm{H}_A : (\!|\langle\!|A|\!\rangle|\!) \to A$, and also define the type translators

$\mathrm{G}^A_{\mathrm{fix}} : ((A \to A) \to A) \to (\!|\!\langle (A \to A) \to A |\!\rangle\!)$ and $\mathrm{H}^A_{\mathrm{fix}} (\!|\!\langle (A \to A) \to A |\!\rangle\!) \to ((A \to A) \to A)$. Unlike the call-by-value case, $\mathrm{H}^A_{\mathrm{fix}}$ is exactly the inverse of $\mathrm{G}^A_{\mathrm{fix}}$ in the $\lambda\mu_{\mathrm{n}}$-calculus.

Proposition 5. *In the $\lambda\mu_{\mathrm{n}}$-calculus without* `fix`*,*
for any judgment $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$\begin{aligned}&\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash \\ &\mathrm{H}_A(\mu k^{(\!|\!\langle A |\!\rangle\!)}.\, (\!|\!\langle M |\!\rangle\!) k) \diamond)\, [\mathrm{G}_{B_1} \alpha_1 / \alpha_1, \ldots, [x_1]\, \mathrm{H}_{A_1}(-) / [x_1]\,(-), \ldots] =_{\mathrm{n}} M \\ &: A \mid x_1 : A_1, \ldots, x_m : A_m.\end{aligned}$$

Theorem 5. *In the $\lambda\mu_{\mathrm{n}}$-calculus with* `fix`*,*
for any judgment $\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash M : A \mid x_1 : A_1, \ldots, x_m : A_m$,

$$\begin{aligned}&\alpha_1 : B_1, \ldots, \alpha_n : B_n \vdash \\ &\mathrm{H}_A(\mu k^{(\!|\!\langle A |\!\rangle\!)}.\, (\!|\!\langle M |\!\rangle\!) k) \diamond)\, [\ldots, \mathrm{G}^{D_i}_{\mathrm{fix}}\, \mathtt{fix}^{D_i} / \mathtt{fix}^{(\!|\!\langle D_i |\!\rangle\!)}, \ldots] =_{\mathrm{n}} M \\ &: A \mid x_1 : A_1, \ldots, x_m : A_m,\end{aligned}$$

where D_i ranges over all types D such that $\mathtt{fix}^D$ *occurs in M.*

5 Uniformity

5.1 Uniform Iteration Operators

In this section, we investigate uniformity principles for recursors and iterators introduced above. First, we consider the $\lambda\mu_{\mathrm{v}}$-calculus with `loop`.

Under the condition $F \circ H =_{\mathrm{v}} H \circ G$ ($F \circ H$ is the abbreviation of $\lambda x^B.\, F(Hx)$),

$$(\mathtt{loop}^A\, F) \circ H =_{\mathrm{v}} (\mathtt{loop}^A\, F) \circ F \circ H =_{\mathrm{v}} (\mathtt{loop}^A\, F) \circ H \circ G$$

holds. So, $(\mathtt{loop}^A\, F) \circ H$ is expected to behave in the same way as $\mathtt{loop}^B\, G$.

However, if H does not satisfy appropriate conditions, for example, in the case that F, G and H are id_A, id_A and $\lambda x^A.\mu\alpha^A.[\beta]\, x$ respectively, $(\mathtt{loop}^A\, F) \circ H$ is not expected to be equal to $\mathtt{loop}^B\, G$. Therefore, we require a *strictness* condition for the uniformity principle.

Definition 4. *A $\lambda\mu_{\mathrm{v}}$-value $H : B \to A$ is* ***total*** [2] *if*

$$\Gamma, x : B \vdash \mathtt{let}\, y^A\, \mathtt{be}\, Hx\, \mathtt{in}\, \lambda t^\top.\, y =_{\mathrm{v}} \lambda t^\top.\, Hx : \top \to A \mid \Delta.$$

Remark 4. $H : B \to A$ is total if and only if

$$\mathtt{let}\, y^A\, \mathtt{be}\, Hx\, \mathtt{in}\, \mathtt{let}\, z^C\, \mathtt{be}\, N\, \mathtt{in}\, L =_{\mathrm{v}} \mathtt{let}\, z^C\, \mathtt{be}\, N\, \mathtt{in}\, \mathtt{let}\, y^A\, \mathtt{be}\, Hx\, \mathtt{in}\, L$$

[2] The word 'total' is due to Filinski [3]. This usage of 'total' may not be standard, but we put our priority on compatibility with [5].

holds for any $N : C$ and $L : D$ such that y is not free in N and z is not free in Hx. So, the totality of H implies that Hx, such a term is called a central one, is free from computational effects. (Detailed analysis of effect-freeness are found in [4].) Central terms correspond to semantic central morphisms in a co-control category.

The totality plays the role of strictness in the uniformity principle for call-by-value iterators. We propose the following uniformity axiom [5].

Definition 5. *An iteration operator* $\{\mathtt{loop}^A\}$ *on the* $\lambda\mu_{\mathrm{v}}$*-calculus is* ***uniform*** *if* $(\mathtt{loop}^A\, F) \circ H =_{\mathrm{v}} \mathtt{loop}^B\, G$ *holds for any values* $F : A \to A$, $G : B \to B$ *and any total value* $H : B \to A$ *such that* $F \circ H =_{\mathrm{v}} H \circ G$.

5.2 Uniform Fixed-Point Operators

In the $\lambda\mu_{\mathrm{n}}$-calculus, the dual notion of call-by-value uniform iteration operators exists.

Definition 6. *A* $\lambda\mu_{\mathrm{n}}$*-term* $H : A \to B$ *is* ***total*** *if*

$$\Gamma, k : \neg\neg A \vdash H(\mu\alpha^A.\, k(\lambda x^A.[\alpha]\, x)) =_{\mathrm{n}} \mu\beta^B.\, k(\lambda x^A.[\beta]\, Hx) : B \mid \Delta.$$

While a total $\lambda\mu_{\mathrm{v}}$-value is interpreted to a curried form of a central morphism in a co-control category, a total $\lambda\mu_{\mathrm{n}}$-term is also interpreted to a curried form of a central morphism in a control category. So both the notions of totality coincide in the models.

The uniformity principle for call-by-name fixed-point operators is symmetric with the call-by-value case.

Definition 7. *A fixed-point operator* $\{\mathtt{fix}^A\}$ *on the* $\lambda\mu_{\mathrm{n}}$*-calculus is* ***uniform*** *if* $H(\mathtt{fix}^A\, F) =_{\mathrm{n}} \mathtt{fix}^B\, G$ *holds for any terms* $F : A \to A$, $G : B \to B$ *and any total term* $H : A \to B$ *such that* $H \circ F =_{\mathrm{n}} G \circ H$.

If we give an appropriate definition of parameterized uniformity, control categories with uniform parameterized fixed-point operators provide a sound and complete class of models of the $\lambda\mu_{\mathrm{n}}$-calculus with a uniform fixed-point operator. This fact is a uniform operator version of Theorem 2. (The definition and the proof are omitted for the lack of space.) On the other hand, we can extend Theorem 3 with uniform parameterized iteration operators: co-control categories with uniform parameterized iteration operators provide a sound and complete class of models of the $\lambda\mu_{\mathrm{v}}$-calculus with a uniform iteration operator.

Moreover, uniform parameterized iteration operators and uniform parameterized fixed-point operators are categorically dual. Hence, we can say that uniform iteration operators in the $\lambda\mu_{\mathrm{v}}$-calculus are the exact dual of uniform fixed-point operators in the $\lambda\mu_{\mathrm{v}}$-calculus.

Remark 5. An extra bonus of uniformity is that a uniform parameterized fixed-point operator can be reduced to a uniform non-parameterized one, *i.e.*, uniform parameterized fixed-point operators and uniform non-parameterized fixed-point operators are in bijective correspondence (*cf.* [5],[13]). Therefore uniformity principles are helpful in simplifying our semantic consideration. This observation also suggests a general approach to dealing with parameterized operators on control categories. This topic will be studied in details at a forthcoming paper.

5.3 Call-by-Value Fixed-Point Operators

Though we have discussed about iteration in call-by-value languages, iteration is less familiar than recursion with functional languages. However, Filinski demonstrated in [3] that iteration operators have bijective correspondence with recursion operators under *a uniformity condition* in a call-by-value calculus with first-class continuations.

In [5], we proposed an axiomatization of fixed-point operators for the call-by-value $\lambda\mu$-calculus, and demonstrated Filinski's construction in the $\lambda\mu_{\mathrm{v}}$-calculus. (This axiomatization does not require the existence of control operators.) Our axiomatization consists of three axioms, which is the call-by-value fixed-point axiom, the stability axiom and the uniformity axiom. One can see call-by-value uniform iterators defined above have bijective correspondence with call-by-value uniform stable fixed-point operators. So, our uniform iterators are justified in the same sense as stable uniform fixed-point operators.

Concatenating the correspondence between call-by-value recursion and call-by-value iteration with the duality between call-by-name and call-by-value, we can get the correspondence between call-by-name recursion and call-by-value recursion.

$$\begin{array}{ccc} \boxed{\text{Recursion in call-by-value}} & & \\ \Updownarrow & & \\ \boxed{\text{Iteration in call-by-value}} & \Leftrightarrow & \boxed{\text{Recursion in call-by-name}} \end{array}$$

6 Conclusion

6.1 Summary

In this paper, we have investigated the duality between call-by-name recursion and call-by-value iteration in the $\lambda\mu$-calculi.

In [12], Selinger has shown the model-theoretic duality between the call-by-name $\lambda\mu$-calculus and the call-by-value one, and induced the syntactic duality from it. Along the line that Selinger has taken, we studied the duality between call-by-name recursion and call-by-value iteration extending the $\lambda\mu$-calculi with a call-by-name fixed-point operator and a call-by-value iteration one.

Because the syntactic translations have dealt with recursion and iteration, there are some possibilities that they are applied to practical programs. Especially, we expect that the translations may be used for verification of programs or compiling.

6.2 Further Duality

Data structures are also important and necessary for programming languages. The natural numbers type is a typical example of important data structures. In call-by-value languages, the natural numbers type is considered as the coproduct of countably infinite $\top$'s. Hence, applied to the duality, the call-by-name list type of infinite $\bot$'s is induced from the call-by-value natural numbers type.

This duality of inductive data-types and co-inductive data-types may combine with the duality between call-by-name and call-by-value. Further discussion and examples are in [7].

Acknowledgment

I wish to thank Masahito Hasegawa for supervising this work, and thank anonymous referees for helpful suggestions.

References

[1] S. Bloom and Z. Ésik. *Iteration Theories*. EATC Monographs on Theoretical Computer Science. Springer-Verlarg, 1993. 507

[2] A. Filinski. Declarative continuations: an investigation of duality in programming language semantics. In *Category Theory and Computer Science*, volume 389 of *LNCS*, pages 224–249. Springer-Verlag, 1989. 506, 512

[3] A. Filinski. Recursion from iteration. *Lisp and Symbolic Computation*, 7:11–38, 1994. 507, 517, 519

[4] C. Führmann. Varieties of effects. In *Foundations of Software Science and Computation Structures*, volume 2303 of *LNCS*, pages 144–158. Springer-Verlag, 2002. 518

[5] M. Hasegawa and Y. Kakutani. Axioms for recursion in call-by-value. In *Foundations of Software Science and Computation Structures*, volume 2030 of *LNCS*, pages 246–260. Springer-Verlag, 2001. 507, 517, 518, 519

[6] H. Huwig and A. Poigne. A note on inconsistencies caused by fixpoints in a cartesian closed category. *Theoretical Computer Science*, 73(1):101–112, 1990. 513

[7] Y. Kakutani. Duality between call-by-name recursion and call-by-value iteration. Master's thesis, Research Institute for Mathematical Sciences, Kyoto University, 2001. 520

[8] E. Moggi. Computational lambda-calculus and monads. In *4th LICS Conference*. IEEE, 1989. 509

[9] C. H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Proceedings of ACM SIGPLAN-SIGACT Symposium on Principle of Programming Languages, Paris, January 1997*. ACM Press, 1997. 509

[10] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Logic Programming and Automated Reasoning*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, 1992. 506, 507, 509

[11] A. J. Power and E. P. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7(5):453–468, 1997. 509

[12] P. Selinger. Control categories and duality: on the categorical semantics of the lambda-mu calculus. *Mathematical Structures in Computer Science*, 11(2):207–260, 2001. 506, 507, 509, 510, 511, 512, 513, 514, 519

[13] A. K. Simpson and G. D. Plotkin. Complete axioms for categorical fixedpoint operators. In *Proceedings of 15th Annual Symposium on Logic and Computer Science*, 2000. 511, 519

[14] H. Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, 1997. Also available as technical report ECS-LFCS-97-376. 507

Decidability of Bounded Higher-Order Unification

Manfred Schmidt-Schauß[1] and Klaus U. Schulz[2]

[1] Institut für Informatik, J.-W.-Goethe-Universität
Postfach 11 19 32, D-60054 Frankfurt, Germany
schauss@ki.informatik.uni-frankfurt.de
Tel: (+49)69-798-28597, Fax: (+49)69-798-28919
[2] CIS, University of Munich
Oettingenstr 67, D-80538 München, Germany
schulz@cis.uni-muenchen.de
Tel: (+49)89-2178-2700, Fax: (+49)89-2178-2701

Abstract. It is shown that unifiability of terms in the simply typed lambda calculus with β and η rules becomes decidable if there is a bound on the number of bound variables and lambdas in a unifier in η-long β-normal form.

1 Introduction

First-order unification [BS94] is a fundamental operation in several areas of computer science, e.g. automated deduction, term rewriting, logic programming and type-checking. The generalization to higher-order unification increases the expressiveness, the applicability and improves the level of abstraction. This explains the interest in various kinds of higher-order systems (e.g., [And86, Pau94, Pfe01, Bar90, Bir98, Mil91, HKMN95], [Nip91, Klo92, DJ90, Hue75], [Dow01]).

It is well-known that second-order unification – hence higher-order unification – is undecidable ([Gol81, Far91, LV00, Vea00]). In order to introduce natural restrictions that lead to decidable unification problems, at least two orthogonal directions can be followed. On the one hand, we may try to restrict the syntactic form of the input unification problems. A well-known syntactic restriction that leads to a decidable unification problem is the unification of higher-order patterns [Mil91]. On the other hand, we may also impose restrictions on substitutions that may be used to solve unification problems. In [SS99a, SS01] it was shown that second-order unification becomes decidable if an upper bound on the number of occurrences of bound variables in the substitution terms is fixed, which has as a corollary the well-known result that second-order unification with monadic function symbols is decidable [Hue75, Zhe79, Far88].

In this paper we generalize the latter result to higher-order unification in the simply typed lambda calculus with β and η rules [Bar84, HS86]. We show that solvability of higher-order unification problems is decidable if for any variable a bound on the number of lambda-binders and occurrences of bound variables

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 522–537, 2002.
© Springer-Verlag Berlin Heidelberg 2002

in the image of the variable under a unifier is given. The given algorithm is non-elementary. Each term $\sigma(x)$ for a unifier σ is assumed to be in η-expanded β-normal form. Note that the bound does not imply a bound on the size of a unifier.

The result implies that undecidability proofs for higher-order unification require an unbounded number of lambda-bound variables or lambdas in a unifier in η-expanded β-normal form. It can be used to define a semi-decision procedure for ordinary higher-order unification where we start with a given bound b for the variables and lambdas in the unifier and increase b as long as we have an unsolvable problem. From a practical point of view, the bound E on the exponent of periodicity used in the decision algorithm also has to be increased iteratively, since E depends nonelementary on b.

The result obtained in this paper is a new and non-trivial decidability result for higher-order unification without any syntactic restrictions on the input problems. It is a generalization of the result in the second order case [SS01], where the bound on the lambdas can be omitted.

This result can also be seen as a parameterized decidability of higher-order unification in the sense of [DF99].

Due to space limitations, only the central ideas behind the decision algorithm can be described. All details can be found in a technical report [SSS01].

2 Technical Preliminaries

We assume that readers are familiar with the usual notions and notational conventions of (unification in the) simply typed lambda-calculus. See, e.g., [Bar84, Wol93, Hin97] and the full version. Elementary and complex types are introduced as usual. Symbols ι, ι_1 etc. are used for elementary types. The *arity* of a type $\tau_1 \to \ldots \to \tau_n \to \iota$ is n. The background signature Σ for building higher-order terms contains for each type τ a countably infinite set of function symbols of type τ. For every type τ we use in addition a countably infinite set of variables, $ar(x)$ denotes the arity of the type of variable x. $\mathcal{V}$ denotes the set of all variables. Complex terms (i.e., abstractions and applications) are defined as usual. With $t{\downarrow}_{\beta\overline{\eta}}$ we denote the $\beta\overline{\eta}$-normal form (also called the η-long β-normal form) of the term t. $FV(\kappa)$ denotes the set of free variables of an expression (term, set of terms,..) κ.

A *first-order variable* is a variable of elementary type. A *first-order function symbol* is a function symbol of type $\iota_1 \to \ldots \to \iota_m \to \iota$ where $m \geq 0$. A *first-order term* is a term generated by the grammar $FOT ::= x^\iota \mid f(FOT_1, \ldots, FOT_n)$ where f denotes a first-order function symbol of arity $n \geq 0$. *Contexts*, as usual, are meta-expressions with exactly one occurrence of a "hole" $[\cdot]^\tau$, a special constant denoting a missing argument of type τ. Since we mainly use a special kind of context, we define first order contexts. A *first-order context* is defined using the grammar $FOC ::= [\cdot]^\iota \mid f(t_1, \ldots, t_{i-1}, FOC, t_{i+1}, \ldots, t_n)$ where f is a first-order function symbol of arity $n \geq 1$ and the t_i are first-order terms. If C is a first-order context (of type ι) with hole $[\cdot]^\iota$, and if t is a term of type ι, then C^n

(resp. $C^n[t]$) is the first-order context (resp. term) that is obtained by replacing $n-1$ times in C hole $[\cdot]^\iota$ by C (and replacing the last occurrence of $[\cdot]^\iota$ by t).

The *size* of an elementary type ι is $size(\iota) := 1$. The size of a type of the form $\tau = \alpha_1 \to \ldots \alpha_n \to \iota$ is $size(\tau) = 1 + n + \sum_{i=1}^{n} size(\alpha_i)$. The *size* of a term t is $size(t) := 1$ for each $t \in \Sigma \cup \mathcal{V}$, $size(\lambda x.t) = size(t) + 2$, and $size(s\ t) = 1 + size(s) + size(t)$. The *order* of an elementary type ι is $ord(\iota) = 1$. If $\tau = \alpha_1 \to \ldots \alpha_n \to \iota$, then $ord(\tau) = 1 + max\{ord(\alpha_1), \ldots, ord(\alpha_n)\}$. The *degree* of a term t is $deg(t) := max\{(ord(\tau) - 1) \mid \tau$ is a subtype of a subterm of $t\}$. [1]

There are estimations on the maximal length of reduction sequences for various lambda-calculi (see [Bec01, Gan80, Sch82, Sch91]). We adapt this to our purposes and prove that there is a computable upper bound on the size of a $\beta\overline{\eta}$-normal form of a term t. In the sequel, let $2_0(n) := n$ and $2_m(n) = 2^{2_{m-1}(n)}$ for $m > 0$. Let $maxtypesize(t)$ be the maximal size of a type of a subterm of t.

Theorem 2.1. *Let t be a term. Then the size of the $\overline{\eta}$-normal form of t is at most $seqnf(t) := 3 \cdot size(t) \cdot maxtypesize(t)$. The size of the $\beta\overline{\eta}$-normal form of t is at most $sbeqnf(t) := seqnf(t)^a$ where $a = 2_{deg(t)+1}(seqnf(t))$.*

3 Bounded Higher-Order Unification Problems

Let Σ_0 denote a subsignature of Σ. A *higher-order unification problem (HOUP)* is a finite set S of (symmetric) equations $\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$ where s_i, t_i are terms with $type(s_i) = type(t_i)$ for all i. A closed (Σ_0-) substitution σ such that $\sigma(s_i) =_{\beta\eta} \sigma(t_i)$ for $1 \leq i \leq n$ is called a (Σ_0-) *unifier* of S.

For a term t, we define $\#bvl(t)$ to be the number of occurrences of bound variables in t plus the number of lambda-binders in t. For example, $\#bvl(\lambda x.f(\lambda y.(x\ y\ z))) = 4$. If we use a compressed notation like $\lambda x, y, z.t$, then we apply $\#bvl(\cdot)$ to the expanded expression.

Definition 3.1. *Let S be a HOUP, let $b : FV(S) \to \mathbb{N}_0$ be a function. Then the pair (S, b) is called a* bounded HOUP (BHOUP). *A closed (Σ_0-) substitution σ is a (Σ_0-)* unifier *of (S, b) iff all terms in the codomain of σ are in $\beta\overline{\eta}$-normal form, σ is a (Σ_0-) unifier of S and for every variable $x \in FV(S)$ the inequation $\#bvl(\sigma(x)) \leq b(x)$ holds.*

Note that in a BHOUP the size of unifiers is *not* bounded, since for example for $t = \lambda x.\underbrace{f(\ldots(f}_{k}\ x)\ldots)$ we have $\#bvl(t) = 2$, but the size of t grows with k.

If for some unifier σ, the term $t = \sigma(x)$ has more than b occurrences of a symbol f, then this symbol can only be an elementary constant, or a first order symbol. If f is a higher-order symbol, then due to η-expansion, every occurrence of f requires at least one lambda in an argument, so the number of occurrences of f cannot exceed b.

[1] In [Bec01], the degree of a term is defined similarly as the order in papers on unification, however, $degree = order - 1$.

Thus the remaining part is to treat the unbounded number of first order symbols in a unifier. There appears to be no possibility to bound this number. Similar to the approach to string unification, it is possible to bound the number of periodic nested occurrences of first order symbols in a minimal unifier. Since function symbols may have arity greater than 1, this periodicity means periodic occurrences of first order contexts. The aperiodic occurrences are not bounded yet. The described algorithm given in this paper works well with aperiodic but unbounded occurrences of first order function symbols, since in this case the unification algorithm terminates.

The purpose of this paper is to show the following result.

Theorem 3.2 (Main Theorem). *Unifiability of BHOUPs is decidable.*

In the decidability proof the following notion plays an important role.

Definition 3.3. *The* exponent of periodicity *of a unifier σ of a BHOUP (S, b) is the maximal number n such that for some variable x occurring in S the image $\sigma(x)$ contains a subterm of the form $C^n[t]$, where $C \neq [\cdot]$ is a ground first-order context.*

4 Survey of the Decision Algorithm

The algorithm for deciding unifiability of BHOUPs is transformation based. Transformation steps are non-deterministic and assign to a given BHOUP a finite number of possible successor BHOUPs. A well-founded measure μ is given such that each transformation step reduces μ. By König's Lemma, iterated transformation of an input BHOUP (S_0, b_0) defines a finite search tree $\mathcal{T}(S_0, b_0)$.

In each branch of $\mathcal{T}(S_0, b_0)$, the transformation stops if a BHOUP of a special kind (called "xy") is found (these are the presolved systems of equations). BHOUPs of kind "xy" are always unifiable. In a sense to be made precise below, each transformation rule is sound and complete. It follows that the input BHOUP (S_0, b_0) is unifiable if and only if a BHOUP of type "xy" is found in some branch of $\mathcal{T}(S_0, b_0)$. Since $\mathcal{T}(S_0, b_0)$ is finite, for each unifiable input BHOUP (S_0, b_0) such a successful branch can be effectively found.

A needed assumption for the decision algorithm is that each transformation step is finitely branching. In order to achieve this goal, the algorithm does not try to generate a complete set of unifiers for the input BHOUP (S_0, b_0). Instead, the unifiers σ that are taken into account by the transformation steps satisfy two characteristic restrictions:

1. The terms in the codomain of σ are built over a *finite* signature Σ_0 that is determined by (S_0, b_0).
2. The exponent of periodicity of σ is bound by a constant E determined by (S_0, b_0).

Given the input BHOUP (S_0, b_0), the finite subsignature Σ_0 and bound E are determined on the basis of the following observation.

Theorem 4.1. *There exists a computable function that assigns to each BHOUP (S_0, b_0) a natural number $E = E(S_0)$ with the following property: Let $\Sigma_0 \subseteq \Sigma$ be any subsignature that contains all function symbols occurring in S_0 and in addition at least one elementary constant a^ι for each elementary type ι occurring as a subtype of a subterm of S_0. If (S_0, b_0) is solvable, then (S_0, b_0) has a Σ_0-unifier where the exponent of periodicity does not exceed E.*

Since in $\mathcal{T}(S_0, b_0)$ only (potential) unifiers of BHOUPs that satisfy the above restrictions are taken into account, the notions of soundness and completeness have to be adapted. A non-deterministic transformation rule $\mathcal{R}$ that transforms a BHOUP (S, b) occurring in $\mathcal{T}(S_0, b_0)$ into another BHOUP (S', b'), offering a finite number of alternatives, is called

- *sound for* Σ_0 if whenever (S, b) is transformed by $\mathcal{R}$ into (S', b'), and (S', b') is unifiable using a Σ_0-unifier, then (S, b) is unifiable using a Σ_0-unifier.
- *complete for bound* E *and* Σ_0, iff the following holds: If (S, b) has a Σ_0-unifier σ with exponent of periodicity not greater than E, then $\mathcal{R}$ can transform (S, b) into a BHOUP (S', b') that has a Σ_0-unifier with exponent of periodicity not greater than E.

All transformation rules are sound and complete in this specific sense. For every BHOUP that is not of type "xy", a rule can be applied to transform it further. Moreover, each BHOUP (S', b') of type "xy" that is generated in the search tree $\mathcal{T}(S_0, b_0)$ has a Σ_0-unifier σ. On this basis it is simple to see that in fact a BHOUP (S_0, b_0) is unifiable iff a BHOUP of type "xy" is generated in some branch of $\mathcal{T}(S_0, b_0)$ as described above. Theorem 3.2 is obtained as a consequence.

5 A Bound for the Exponent of Periodicity

Theorem 4.1 represents an important and original result of the present paper that is of independent interest. In this part we show how to prove the theorem, ignoring, for simplicity, restrictions on the signature.

Definition 5.1. *Let t be a ground term in $\beta\overline{\eta}$-normal form. Assume that we color in t the positions of each of the n lambda-binders in expressions $\lambda x_1, \ldots, x_n$ occurring in t, each occurrence of a bound variable in t, as well as each occurrence of a function symbol f in an expression $f(t_1, \ldots, t_n)$ where either*

1. *f contains an argument of non-elementary type, i.e. f is not first-order, or*
2. *there are at least two subterms t_{i_1}, t_{i_2} $(i_1 \neq i_2)$ such that t_{i_j} for $j = 1, 2$ contains an occurrence of a variable or a lambda.*

The uncolored positions of t can be considered as the nodes of a graph where links correspond to immediate subterm relationship. Each maximal connected uncolored component either defines a ground first-order term or a ground first-order context. They are called the maximal first-order subterms/subcontexts *of t. The* representation size *of t, repsize(t) is defined similarly as the size of t, but each maximal first-order subterm/subcontext yields a uniform contribution of* 1.

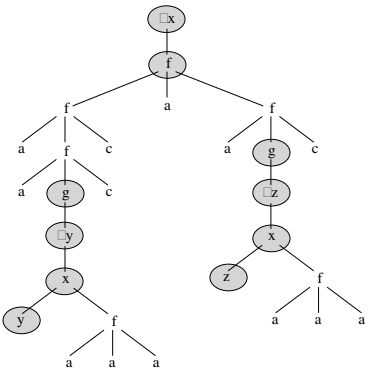


Fig. 1. Colored positions and maximal first-order subterms and subcontexts

Intuitively, in the *repsize*-measure, maximal first-order subterms/subcontexts are treated as primitive symbols.

Example 5.2. The ground term t depicted in Figure 1 is colored in the above sense. Maximal first-order subterms are $f(a, a, a)$ (two occurrences) and a (one occurrence). Maximal first-order contexts are $f(a, f(a, [\cdot], c), c)$ and $f(a, [\cdot], c)$. Hence the maximal first-order subterms/subcontexts yield a total contribution of 5 to $repsize(t)$.

Definition 5.3. *Let (S, b) be a BHOUP. Let* $\mathrm{maxar}(S)$ *denote the maximal arity of a type representing a subtype of a subterm of S, let* maxb *be the maximal value $b(x)$ for variables in $FV(S)$. Then the number* $\mathrm{repn}(S, b) := 6 \cdot \mathrm{maxb} \cdot \mathrm{maxar}(S) + 22 \cdot \mathrm{maxb} + 2$ *is called the* representation number *of (S, b).*

In the following lemma, with a *minimal* unifier of a BHOUP (S, b) we mean a unifier σ such that $\sum_{x \in FV(S)} size(\sigma(x))$ is minimal with respect to all unifiers of the problem.

Lemma 5.4. *Let (S, b) be a BHOUP, and σ be a minimal unifier of (S, b). Then the representation size of any term in the codomain of σ is at most* $\mathrm{repn}(S, b)$.

The important point to note is that the above estimate for the representation size does not depend on σ. In the sequel we use some of the previously introduced measuring functions also for BHOUPs S as follows. If $S = \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$, then $terms(S)$ is the multiset of all terms s_i and t_i $(i = 1, \ldots, n)$. Now we can use the functions ord, deg, $size$, $maxtypesize$, $seqnf$, $sbeqnf$ also for S by applying them to $terms(S)$, and use the obvious operators for extending the functions to multisets.

Lemma 5.5. *There is a positive real constant c_0 such that for every unifiable BHOUP (S, b) the exponent of periodicity of a minimal unifier of (S, b) is less than $2^{(c_0 + 2{,}14 \cdot finsize(S))}$, where* $\mathrm{finsize}(S) := 2_{deg(S)+1}(\mathrm{repn}(S, b) \cdot sbeqnf(S))$.

Proof (Sketch). Let (S, b) be a BHOUP and let σ be a minimal unifier of (S, b). Let $\mathit{terms}(\sigma(S))$ denote the multiset of image terms $\{\sigma(s) \mid s \in \mathit{terms}(S)\}$. In each term $\sigma(s)$ we consider the occurrences of codomain terms $\sigma(x)$ that represent the images of the variables occurring in s under σ. For each such occurrence we consider the maximal first-order subterms/subcontexts of the respective codomain term as primitive symbols. Each such subterm/subcontext will be called an *inner codomain subterm/subcontext*, stressing their origin in codomain terms. By Lemma 5.4, the sum of the sizes of all terms in $\mathit{terms}(\sigma(S))$ with respect to this representation is bound by $\mathit{size}(S) \cdot \mathit{repn}(S, b)$.

When we compute the $\beta\overline{\eta}$-normal form of the terms in $\mathit{terms}(\sigma(S))$, the inner codomain subterms/subcontexts are not destroyed. For the reduction they can be considered as primitive symbols as well. Hence it follows from Theorem 2.1 that the corresponding representation for the normalized image terms in the set $\{\sigma(s)\downarrow_{\beta\overline{\eta}} \mid s \in \mathit{terms}(S)\}$ has representation size not exceeding $\mathit{finsize}(S)$ as defined in the lemma.

Now we use the fact that the $\beta\overline{\eta}$-normal forms of the left- and right-hand side of equations are α-equal to extract a context unification problem [SS99b, SSS02] CUP. This can be done by equating the following:

- the maximal ground first-order terms in equations $\sigma(s)\downarrow_{\beta\overline{\eta}} =_\alpha \sigma(t)\downarrow_{\beta\overline{\eta}}$ at corresponding positions,
- the maximal ground first-order contexts in equations $\sigma(s)\downarrow_{\beta\overline{\eta}} =_\alpha \sigma(t)\downarrow_{\beta\overline{\eta}}$ at corresponding positions,

for $s \doteq t \in S$. Note that all the inner codomain subterms/subcontexts are contained in some maximal first-order term/context. CUP is formed from the equations $\sigma(s)\downarrow_{\beta\overline{\eta}} =_\alpha \sigma(t)\downarrow_{\beta\overline{\eta}}$ $(s \doteq t \in S)$ as follows: The inner codomain contexts are replaced consistently by context variables, and the inner codomain terms are consistently replaced by first-order variables.

The total number of occurrences of variables and function symbols in CUP does not exceed $\mathit{finsize}(S)$. The results in [SSS98] show that there exists a fixed real constant c_0 such that the exponent of periodicity of a minimal unifier for CUP is smaller than $2^{c_0+2.14\cdot finsize(S)}$.

It is easy to see that each unifier of CUP can be backtranslated into a unifier for (S, b) with the same exponent of periodicity, and that the smaller context unifiers translate into smaller unifiers of (S, b). This shows that the exponent of periodicity of σ is smaller than $2^{c_0+2.14*finsize(S)}$. □

6 Transformation of BHOUPs

The remaining part of this abstract is used for an informal description of the transformation rules. We consider a fixed input BHOUP (S_0, b_0) of the decision algorithm. In the sequel, if all terms occurring in a BHOUP are in $\beta\overline{\eta}$-normal form, then we say that the BHOUP is in $\beta\overline{\eta}$-normal form. We assume that (S_0, b_0) is in $\beta\overline{\eta}$-normal form. Given (S_0, b_0), a finite subsignature Σ_0 and a bound E are chosen as explained above.

6.1 Decomposition

The transformation rules operate on so-called *decomposed* BHOUPs. Decomposition is a terminating procedure that is first applied to the initial input BHOUP (S_0, b_0), later as a subprocedure at the end of each transformation step. The input BHOUPs for decomposition are always in $\beta\overline{\eta}$-normal form, and each decomposition step preserves this property. We present two characteristic decomposition rules, other rules can be found in [SSS01]:

$$\frac{\{f\ s_1\ \ldots\ s_n \doteq f\ t_1\ \ldots\ t_n\} \cup S}{\{s_1 \doteq t_1, \ldots, s_n \doteq t_n\} \cup S} \qquad \frac{\{\lambda u^\tau.s \doteq \lambda u^\tau.t\} \cup S}{\{s[f/u] \doteq (t[f/u])\} \cup S}$$

where in the right-hand side rule $f \in \Sigma \setminus \Sigma_0$ is a fresh function symbol of type τ. Soundness of this rule only holds in the restricted sense defined above, as we explain in the full version. The two rules guarantee the following:

For any equation $s \doteq t$ in a decomposed BHOUP (S, b), the terms s and t have (the same) *elementary* type. Furthermore, each side s or t of an equation $s \doteq t$ of S has the form $x(t_1, \ldots, t_n)$ or $f(t_1, \ldots, t_n)$, and there is at least one side of the form $x(t_1, \ldots, t_n)$. Here x (resp. f) is a variable (function constant) of arity n.

6.2 Four Different Types of BHOUPs

For the transformation, four kinds of BHOUPs are distinguished. In order to define the four classes, the following notions are needed.

Definition 6.1. *Let t be a term. The* surface positions *of t are defined as follows:*

- *ε, if t is elementary*[2]*.*
- *if t is elementary, and $t = f(t_1, \ldots, t_n)$, then for every surface position p of t_i: $i.p$ is a surface position of t.*
  In this case we also say f is on the surface of t.
- *if t is elementary, and $t = x\ t_1\ \ldots\ t_{ar(x)}$, then 0, the position of x, is a surface position of t.*

The depth *of a surface position p is the length of p.*

We use the notation $t\lfloor s \rfloor$ to indicate that t has a surface occurrence of the term s. In the sequel, to simplify index notation we use expressions $i\ \texttt{mod*}\ n$ where

$$i\ \texttt{mod*}\ n = \begin{cases} i\ \texttt{mod}\ n & \text{if}\ \ i\ \texttt{mod}\ n \neq 0, \\ n & \text{if}\ \ i\ \texttt{mod}\ n = 0. \end{cases}$$

[2] ε denotes the empty sequence.

Definition 6.2. *Let (S, b) be a BHOUP. A* cycle *is a sequence $s_1 \doteq t_1, \ldots, s_h \doteq t_h$ of length $h \geq 1$ of equations from S, such that for all $1 \leq i \leq h$: $s_i \equiv x_i\ r_{i,1}\ \ldots\ r_{i,m_i}$, and x_i occurs on the surface of $t_{i-1 \texttt{ mod* } h}$. Moreover, there should be at least one term t_i of the form $f(t_{i,1}, \ldots, t_{i,n})$ and at least one term s_i of the form $x_i\ r_{i,1}\ \ldots\ r_{i,m_i}$ with $\mathrm{ar}(x_i) \geq 1$.*

A cycle is path-unique *if for every $1 \leq i \leq h$ there is only one occurrence of x_i on the surface of $t_{(i-1) \texttt{ mod* } h}$.*

Let L be a cycle in (S, b) of the form $s_1 \doteq t_1, \ldots, s_h \doteq t_h$, For each of the terms t_i, $1 \leq i \leq h$, let C_i be the context determined as follows: Let q_i be the smallest subterm of t_i such that all surface occurrences of $x_{(i+1\ mod* h)}$ from t_i are also contained in q_i. The relevant context *C_i of equation i is uniquely determined by $t_i = C_i\lfloor q_i \rfloor$.*

The length of a cycle *is the number of equations in it. If for some cycle L, there is no other cycle in S with a smaller length, then we say L is a* minimal-length *cycle.*

A cycle $s_1 \doteq t_1, \ldots, s_h \doteq t_h$ is called compressed, *iff there is no i such that s_i or t_i is a first-order variable.*

Example 6.3. We give some examples for cycles and non-cycles.
The sequence $x \doteq h(y\ s_1), y\ s_2 \doteq x$ is a (non-compressed) cycle of length 2, provided $x, y\ s_1, y\ s_2$ are terms of elementary type. When instantiating x by $h(y\ s_1)$ we receive from the second equation a shorter cycle of the form $y\ s_2' \doteq h(y\ s_1)$ which is compressed. The sequence $x_1\ s_1 \doteq f(x_1\ s_2, x_2(x_1\ s_3))$ is a path-unique and compressed cycle of length 1, provided that $x_1\ s_1, x_1\ s_2$ are elementary. The sequence $x_1\ y_1 \doteq x_2\ x_1\ y_1$ is not a cycle.

Definition 6.4. *The lexicographic measure $\psi(L) = (\psi_1(L), \psi_2(L), \psi_3(L))$ of a cycle L of a BHOUP (S, b) has the following three components:*

$\psi_1 =$ *the length h of L.*

$\psi_2 =$ 0, *if L is non-path-unique,* 1, *if L is path-unique.*

$\psi_3 =$
- *if L is non-path-unique, then the minimal main depth of the relevant contexts C_j of L where t_j contains at least two different surface-occurrences of $x_{(j+1) \texttt{ mod* } h}$.*
- *if L is path-unique, then the number of indices $1 \leq i \leq h$ such that C_i is not trivial.*

Definition 6.5. *A decomposed BHOUP (S, b) is of*

- type "xy" *if S does not have any cycles, and if there is no function symbol f on the surface of S, (also called pre-unified in the literature on higher-order unification)*
- type "nocycle" *if S does not have any cycles, and if there exists a function symbol f on the surface of S,*
- type "amb" *if S contains a cycle and if there is a ψ-minimal cycle that is non-path-unique,*
- type "unique" *if S contains a cycle and if all ψ-minimal cycles are path-unique.*

6.3 Reduction Rules

The well-founded measure μ that is used to prove termination of transformation is based on six component measures $\mu_1, \ldots, \mu_6$ that are ordered lexicographically. The first two components are $\mu_1(S, b) := \{b(x) \mid x \in FV(S), b(x) > ar(x)\}$ and $\mu_2(S, b) := \{b(x) - ar(x) \mid x \in FV(S), b(x) > ar(x)\}$. Both components are ordered by the multiset ordering. When transforming a BHOUP (S, b), it is often possible (as one choice among several alternatives to be considered) to instantiate one of the free variables x of S in such a way that the lexicographic order (μ_1, μ_2) is reduced. Instantiations of this form, enriched with suitable normalization and decomposition steps, are collected in three (optimistic) "reduction rules": (reduce-bv), (reduce-split) and (reduce-binder). For example, one reduction rule has the following form.

Definition 6.6. *(reduce-bv)*
The input is a decomposed BHOUP (S, b) together with a variable $x \in FV(S)$ with $b(x) > \mathrm{ar}(x) = m$.

(a) Select some $1 \leq i \leq m$ and instantiate x by the $\beta\overline{\eta}$-normal form of

$$\lambda y_1, \ldots, y_m.y_i\ (x_1\ y_1 \ldots y_m)\ \ldots\ (x_k\ y_1\ \ldots\ y_m)$$

where x_j for $j = 1, \ldots, k = \mathrm{ar}(y_i)$ are fresh variables of the appropriate type.
(b) Select bounds $b'(x_j)$ for $j = 1, \ldots, k$ such that $m \leq b'(x_j) < b(x)$ for all variables x_j and furthermore $\sum_{j=1}^{k} (b'(x_j) - m) \leq b(x) - m - 1$.
(c) Beta-reduce the terms until a $\beta\overline{\eta}$-normal form is reached.
(d) Decompose the resulting BHOUP.

The rule "guesses" a value $\sigma(x)$ of x under a (hypothetical) unifier σ of the form $\lambda y_1, \ldots, y_m.y_i(t_1, \ldots, t_k)$. Two further reduction rules refer to a value for x of the form $\lambda y_1, \ldots, y_m.f(t_1, \ldots, t_k)$. Here we have to guess the function symbol $f \in \Sigma_0$. At this point, finiteness of Σ_0 becomes essential.

The three reduction rules are used in the transformation rules for defining a subset of the set of all possible successor systems. Reduction rules are sound for Σ_0 in the sense explained above.

Soundness of these transformation rules is shown in the full paper [SSS01] and requires a careful analysis of the situation after instantiation and $\beta\overline{\eta}$-normalization, since unifiers are assumed to be in $\beta\overline{\eta}$-normal form

6.4 Transformation of BHOUPs of Type "amb"

The third component of the termination order μ is $\mu_3(S, b) := min\{\psi(L) \mid L \text{ is a cycle in } S\}$ if S has a cycle, otherwise, $\mu_3(S, b) := \infty$. The rule (solve-ambiguous-cycle) that is used for BHOUPs of type "amb" decreases (ignoring reduction cases) μ_3, components μ_1, μ_2 are not affected.

Let (S, b) denote a problem of type "amb". Recall that S is decomposed and has a ψ-minimal cycle L that is non-path-unique. We may assume that L has

the form $x_1\ \overrightarrow{s_1}\ \doteq t_1, \ldots, x_h\ \overrightarrow{s_h}\ \doteq t_h$. The cycle could as well be represented as $x_1\ \overrightarrow{s_1} \doteq C_1[t'_1], \ldots, x_h\ \overrightarrow{s_h} \doteq C_h[t'_h]$, where C_i are the relevant contexts (see Definition 6.2). With (repvt) we denote the following rule:

$$\frac{\{x \doteq t\} \cup S}{\{x \doteq t\} \cup S'}$$

where S' is constructed from S by replacing all surface occurrences of x by t. The variable x must be a first-order variable.

Definition 6.7. **(solve-ambiguous-cycle)**. *The input is the decomposed BHOUP (S, b) of type "amb" with a ψ-minimal cycle L as described above. Select one of the following two alternatives.*

1. *Apply one of the three reduction rules (reduce-bv), (reduce-split) or (reduce-binder) using a variable $x \in \{x_1, \ldots, x_h\}$ with $b(x) > \mathrm{ar}(x)$.*
2. *Select an index j such that $x_j \overrightarrow{s_j} \doteq t_j$ is an equation in L where $x_{(j+1 \texttt{ mod* } h)}$ occurs at least twice on the surface of $t_j = f\ t_{j,1}\ \ldots\ t_{j,k}$ and the main depth of the relevant context C_j is minimal in L. If f has an argument with non-elementary type, then fail. Now apply the following steps:*
   (a) *Select an index $r \in \{1, \ldots, k\}$. In the special situation where $h = 1$, the selection of r is subject to the following condition: all surface occurrences of x_1 in $f(t_{1,1}, \ldots, t_{1,k})$ have to be in $t_{1,r}$. If this is not possible since C_1 is trivial, then stop with fail.*
   (b) *Instantiate x_j by $\lambda \overrightarrow{y}.f(z_1, \ldots, z_{r-1}, x'_j\ \overrightarrow{y\downarrow_{\beta\overline{\eta}}}, z_{r+1}\ \ldots\ z_k)$ where the z_i are fresh first-order variables ($1 \leq i \leq k, i \neq r$), and x'_j is a fresh variable.*
   (c) *Define $b'(z_i) := 0$ and $b'(x'_j) := b(x_j)$.*
   (d) *Use β-reduction until a $\beta\overline{\eta}$-normal form is reached for every term in the system.*
   (e) *Apply rule (decomp) to the equation that is obtained from the equation $x_j \overrightarrow{s_j} \doteq t_j$ in Step (d).*
   (f) *Apply (repvt) for all the new equations $z_i \doteq t_{j,i}$ ($1 \leq i \leq k, i \neq r$) that are obtained from the previous step.*
   (g) *Then decompose the resulting BHOUP.*

Lemma 6.8. *Application of the the rule (solve-ambiguous-cycle) to a BHOUP (S, b) of type "amb" either fails or results in a BHOUP (S^*, b^*), such that $\mu(S^*, b^*) < \mu(S, b)$. The rule is sound for Σ_0 and complete for E and Σ_0.*

6.5 Transformation of BHOUPs of Type "nocycle"

Component μ_4 of the termination order has the form $\mu_4(S, b) := \{\mathit{size}(t) \mid t$ is a top-level term in S that is not a first-order variable$\}$ (ordered by the multiset ordering), the fifth component $\mu_5(S, b)$ is the number of occurrences of function symbols in S on surface positions. The imitation-rule that is used

for BHOUPs of type "nocycle" (ignoring reductions) reduces the components μ_4 and μ_5 while μ_1, μ_2 and μ_3 remain unchanged.

Let (S, b) denote a BHOUP of type "nocycle", with a set of variables $\mathcal{V}_S := FV(S)$. Let the relations "$\sim_1$" and "$>_1$" on $\mathcal{V}_S$ be defined as follows: if there exists an equation $x\ s_1\ \ldots s_n \doteq y\ t_1 \ldots\ t_m \in S$, then $x \sim_1 y$. If there exists an equation $x\ s_1\ \ldots s_n \doteq t \in S$, and t has some function symbol f as head, y is on the surface of t, then $x >_1 y$.

Let "$\sim$" denote the equivalence relation in $\mathcal{V}_S$ generated by $\sim_1$. Denote the equivalence class of a variable x by $[x]_\sim$. For equivalence classes D_1, D_2 of $\mathcal{V}_S/\sim$ define $D_1 \rhd_1 D_2$ if there exist $x_i \in D_i$ for $i = 1, 2$ such that $x_1 >_1 x_2$. Let "$\rhd$" denote the transitive closure of "$\rhd_1$".

Lemma 6.9. *If the decomposed BHOUP (S, b) is of type "nocycle", then the relation "$\rhd$" is an irreflexive partial order on $\mathcal{V}_S/\sim$.*

Definition 6.10. (Imitation) *Let (S, b) be a decomposed BHOUP of type "nocycle". Select a $\rhd$-maximal $\sim$-equivalence class D and a function symbol f according to the following conditions: There must be an equation $z\ \ldots \doteq f \ldots$ in S where $z \in D$. Let $k := \mathrm{ar}(f)$. Select one of the following two alternatives. The second alternative is only possible if $f \in \Sigma_0$ has arity $k \geq 1$ and if all arguments of f have elementary type, i.e. f is a first-order function symbol.*

1. *Apply a reduction rule using a variable $x \in D$ with $b(x) > \mathrm{ar}(x)$.*
2. *Apply the following steps:*
   (a) *For every variable $x \in D$ select an index j_x with $1 \leq j_x \leq k$. Instantiate x by the $\beta\overline{\eta}$-normal form of*

$$\lambda y_1, \ldots, y_{ar(x)}.f(z_1, \ldots, z_{j_x-1}, (x'\ y_1 \ldots y_{ar(x)}), z_{j_x+1} \ldots, z_k)$$

   where the $z_i, i = 1, \ldots k, i \neq j_x$ are fresh first-order variables and x' is a new variable of appropriate type. Define $b'(x') := b(x)$ and $b(z_i) = 0$ for $i \in \{1, \ldots, k\}, i \neq j_x$.
   (b) *Use (β)-reduction to transform the terms into $\beta\overline{\eta}$-normal form.*
   (c) *Decompose the resulting BHOUP.*

Lemma 6.11. *Application of the the rule (imitation) to a decomposed BHOUP (S, b) of type "nocycle" either fails or results in a BHOUP (S^*, b^*), such that $\mu(S^*, b^*) < \mu(S, b)$. The rule is sound for Σ_0 and complete for E and Σ_0.*

6.6 Transformation of BHOUPs of Type "unique"

We give an informal description of the non-deterministic rules to treat BHOUPs of type "unique". We ignore in the description the reduction rules which make progress by an optimistic guess. Note that we describe the most pessimistic path of performance of the rules. Of course the non-deterministic nature of guessing always allows to make some intermediate optimistic guess, such that the order decreases.

The starting point of the rules for type "unique" is a length-minimal and path-unique cycle L. The rules operate on the cycle L and the variables that are responsible for the cycle, and try to transform the BHOUP, thereby using the cycle L. After one application of a rule, the next application is in general on the descendant L' of the cycle L.

The first sequence of rule applications is intended to modify step by step the cycle L, such that a cycle is of the form $x_1\ \overrightarrow{s_1} \doteq x_2 \overrightarrow{t_1}, \ldots, x_h\ \overrightarrow{s_h} \doteq C_h[t_h]$.

The second sequence of steps keeps this form of the cycle, and should guarantee that the relevant context C_h permits only instances under any unifier which are first-order contexts. Now the cycle is called "special path-unique".

The last part is to perform iterated parallel "imitations" on the variables of the special path-unique cycle L. It is clear that the number of iterations in the direction of the cycle is bounded by the bound E on the exponent of periodicity, since C_h is a first-order context in any instance. One possibility to stop the instantiation is to guess that after some instantiations, a reduction rule can be applied. If after several "imitations" in the direction of the cycle, there is one "imitation" not in the direction of the cycle, then it is possible to show that the descendant cycle L' is shorter than L, hence the measure μ_3 becomes strictly smaller.

All the rules that are applicable for the case of "unique" BHOUPs are sound and complete for E and keep the BHOUP in $\beta\overline{\eta}$-normal form. Moreover, the rules either fail or strictly reduce the measure μ.

7 Conclusion

The algorithm given in this paper shows that BHOUPs have a decidable unification problem. The complexity of the algorithm is nonelementary which is mainly due to the bound on the exponent of periodicity. We conjecture that the algorithm only adds an NP-complexity to this non-elementary bound.

A recent paper on a variant of higher-order matching [Wie02] shows that so-called k-linear higher-order matching, which imposes a bound on the number of the occurrences of every bound variable, is decidable and that there is also a non-elementary lower complexity bound. Though the problems are similar, the Wierzbicki-restriction is not a special case of bounded higher-order unification, since Wierzbicki has no bound on the overall number of bound variables. Hence the non-elementary lower bound for k-linear higher-order matching does not apply to bounded higher-order unification.

We leave for future research the investigation whether bounded higher unification remains decidable, if the k-linearity condition is used instead of the global bound on the number of occurrences of bound variables and lambdas.

Perhaps the encoding ala Wierzbicki may also show a non-elementary lower bound for bounded higher-order unification.

References

[And86] Peter Andrews. *An introduction to mathematical logic and type theory: to truth through proof*. Academic Press, 1986 522

[Bar84] Henk P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland, Amsterdam, New York, 1984 522, 523

[Bar90] Henk P. Barendregt. Functional programming and lambda calculus. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 7, pages 321–363. Elsevier, 1990 522

[Bec01] Arnold Beckmann. Exact bounds for lengths of reductions in typed λ-calculus. *J. Symbolic Logic*, 66:1277–1285, 2001 524

[Bir98] Richard Bird. *Introduction to Functional Programming using Haskell*. Prentice Hall, 1998 522

[BS94] Franz Baader and Jörg Siekmann. Unification theory. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 41–125. Oxford University Press, 1994 522

[DF99] Rodney.G. Downey and Michael R. Fellows. *Parametrized Complexity*. Springer, 1999 523

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science: Formal Models and Semantics*, volume B, chapter 6, pages 243–320. Elsevier, 1990 522

[Dow01] Gilles Dowek. Higher-order unification and matching. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 16, pages 1009–1062. North-Holland, 2001 522

[Far88] W. A. Farmer. A unification algorithm for second order monadic terms. *Annals of Pure and Applied Logic*, 39:131–174, 1988 522

[Far91] W. A. Farmer. Simple second-order languages for which unification is undecidable. *J. Theoretical Computer Science*, 87:173–214, 1991 522

[Gan80] Robin O. Gandy. Proofs of strong normalization. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 457–477. Academic Press, 1980 524

[Gol81] Warren. D. Goldfarb. The undecidability of the second-order unification problem. *Theoretical Computer Science*, 13:225–230, 1981 522

[Hin97] J.Roger Hindley. *Basic simple type theory*. Cambridge tracts in theoretical computer science. Cambridge University Press, 1997 523

[HKMN95] M. Hanus, H. Kuchen, and J. J. Moreno-Navarro. Curry: A truly functional logic language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995 522

[HS86] J.Roger Hindley and Jonathan P. Seldin. *Introduction to combinators and λ-calculus*. Cambridge University Press, 1986 522

[Hue75] Gérard Huet. A unification algorithm for typed λ-calculus. *Theoretical Computer Science*, 1:27–57, 1975 522

[Klo92] Jan Willem Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E.Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 2–116. Oxford University Press, 1992 522

[LV00] Jordi Levy and Margus Veanes. On the undecidability of second-order unification. *Information and Computation*, 159:125–150, 2000 522

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables and simple unification. *J. of Logic and Computation*, 1(4):497–536, 1991 522

[Nip91] Tobias Nipkow. Higher-order critical pairs. In *Proc. 6th IEEE Symp. LICS*, pages 342–349, 1991 522

[Pau94] Lawrence C. Paulson. *Isabelle*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994 522

[Pfe01] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 17, pages 1063–1147. North-Holland, 2001 522

[Sch82] Helmut Schwichtenberg. Complexity of normalization in the pure typed λ-calculus. In A. S. Troelstra and D. van Dalen, editors, *The L. E. J. Brouwer Centenary Symposium. Proceedings of the Conference hold in Noordwijkerhout, 8–13 June, 1981*, volume 110 of *Studies in Logic and the Foundations of Mathematics*, pages 453–458. North Holland, 1982 524

[Sch91] Helmut Schwichtenberg. An upper bound for reduction sequences in the typed λ-calculus. *Archive for Mathematical Logic*, 30:405–408, 1991. Dedicated to Kurt Schütte on the occasion of his 80th birthday 524

[SS99a] Manfred Schmidt-Schauß. Decidability of bounded second order unification. Frank report 11, FB Informatik, J. W. Goethe-Universität Frankfurt am Main, 1999. available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html` 522

[SS99b] Manfred Schmidt-Schauß. A decision algorithm for stratified context unification. Frank-Report 12, Fachbereich Informatik, J. W. Goethe-Universität Frankfurt, Frankfurt, Germany, 1999. accepted for publication in J. Logic and Computation, available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html` 528

[SS01] Manfred Schmidt-Schauß. Decidability of bounded second order unification, 2001. submitted for publication 522, 523

[SSS98] Manfred Schmidt-Schauß and Klaus U. Schulz. On the exponent of periodicity of minimal solutions of context equations. In *Proceedings of the 9th Int. Conf. on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 61–75, 1998 528

[SSS01] Manfred Schmidt-Schauß and Klaus U. Schulz. Decidability of bounded higher order unification. Frank report 15, Institut für Informatik, J. W. Goethe-Universität Frankfurt am Main, 2001. also appeared as: Forschungsbericht, Centrum für Informations- und Sprachverarbeitung, Universitèt München. The paper is available at `http://www.ki.informatik.uni-frankfurt.de/papers/articles.html` 523, 529, 531

[SSS02] Manfred Schmidt-Schauß and Klaus U. Schulz. Solvability of context equations with two context variables is decidable. *Journal of Symbolic Computation*, 33(1):77–122, 2002 528

[Vea00] Margus Veanes. Farmer's theorem revisited. *Information Processing Letters*, 74:47–53, 2000 522

[Wie02] ToMasz Wierzbicki. A decidable variant of the higher order matching. In *Proc. RTA'02*, 2002 to appear. 534

[Wol93] David A. Wolfram. *The clausal theories of types*. Number 21 in Cambridge tracts in theoretical computer science. Cambridge University Press, 1993 523

[Zhe79] A.P Zhezherun. Decidability of the unification problem for second order languages with unary function symbols. *Kibernetika (Kiev)*, 5:120–125, 1979. Translated as *Cybernetics* 15(5): 735-741,1980 522

Open Proofs and Open Terms: A Basis for Interactive Logic

Herman Geuvers[1] and Gueorgui I. Jojgov[2]

[1] University of Nijmegen, The Netherlands
herman@cs.kun.nl
[2] Eindhoven University of Technology, The Netherlands
G.I.Jojgov@tue.nl

Abstract. When proving a theorem, one makes intermediate claims, leaving parts temporarily unspecified. These 'open' parts may be proofs but also terms. In interactive theorem proving systems, one prominently deals with these 'unfinished proofs' and 'open terms'. We study these 'open phenomena' from the point of view of logic. This amounts to finding a correctness criterion for 'unfinished proofs' (where some parts may be left open, but the logical steps that have been made are still correct). Furthermore we want to capture the notion of 'proof state'. Proof states are the objects that interactive theorem provers operate on and we want to understand them in terms of logic.
In this paper we define 'open higher order predicate logic', an extension of higher order logic with unfinished (open) proofs and open terms. Then we define a type theoretic variant of this open higher order logic together with a formulas-as-types embedding from open higher order logic to this type theory. We show how this type theory nicely captures the notion of 'proof state', which is now a type-theoretic context.

Keywords: interactive theorem proving, type theory, open terms, meta-variables, formulas-as-types

1 Introduction

Logic is about *finished* proofs and not about the process of *finding* a proof. The derivation rules of a logic define inductively what is *derivable*. The rules do not tell us how we should *find* or *construct* such a derivation, but they give us a procedure of *checking* whether an alleged proof is indeed well-formed. Of course, the derivation rules are chosen (by Gentzen) in such a way that they represent 'obviously correct' reasoning steps, but that does not mean that mathematicians actually reason in this way. When proving a mathematical theorem, one makes intermediates claims, leaving parts temporarily unspecified and exploring the possibilities. When the proof is 'finished', it is written up in a style that corresponds - at least in spirit - to natural deduction. Looking more closely at the process of proof finding, one observes that also in that phase, the proof-steps are

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 537–552, 2002.
© Springer-Verlag Berlin Heidelberg 2002

intended to be correct in terms of natural deduction. So, there should be a correctness criterion for 'unfinished proofs', where some parts may be left open or unspecified, but the steps that have been made are correct.

Unfinished proofs appear prominently in systems for interactive theorem proving, where the computer assists the user in finding proofs: the user types in tactics that guide the system through the proof-construction. An important issue for interactive systems is how to communicate to the user what the present 'proof state' (the state of the 'unfinished proof') is, in order for the user to make a sensible next step. To describe precisely what these interactive theorem provers actually operate on, we want to give a precise meaning to 'unfinished proofs' and 'proof states'. The following issues arise:

- Can we give a correctness criterion for unfinished proofs? (In such a way that many of the existing 'open proofs' are captured.)
- Can we give a correctness criterion for operations on unfinished proofs? (In such a way that known tactics are instances of such operations.)

So, we first have to answer the questions what an *unfinished proof* and what a *proof state* are. The way mathematicians (and others) give their proofs closely represents – at least in spirit – natural deduction. Hence, if we want to formalize the notion of unfinished proof, natural deduction is a good starting point. So, then the question is: what is an *unfinished natural deduction*? And what are correct *operations* on these unfinished natural deductions? In this paper we will be answering the first question, taking inspiration from the second one, because we know – intuitively and from experience with interactive theorem provers – quite well what we want to be able to do.

Most of the work in the area of incomplete constructions is done in type theory where a number of systems of open terms in (dependently) typed λ-calculus exist [16, 10, 4, 12, 9, 6] . They have evolved from existing typing systems (the Barendregt cube [2], ECC [7], Martin-Löf type theory, etc.) when their application in (interactive) theorem proving required formalization of the notion of incomplete term. TypeLab [16] is based on ECC and represents unknown terms by meta-variables that are equipped with explicit substitutions. Each meta-variable is given a context and a type in that context and the idea is that the meta-variable stands for a well-typed term of the given type in that context. The approach in OLEG [10] is to treat meta-variables declarations as part of the term. This is done by introducing special binders that locally declare meta-variables. In this way the position of the binder naturally expresses the context in which the meta-variable should be solved. Computations with terms containing meta-variable declarations are limited as such terms are not allowed to leak into types. Bognar [4] generalizes the concept of context as used in the untyped λ-calculus [3] and introduces the $\lambda[\ ]$-cube. Along with the local declarations of meta-variables, these systems have explicit operators for instantiation.

For other related work the reader is referred to the papers on λProlog, Isabelle and Twelf and the work of Miller [11], Paulson [13] and Pfenning [14].

The rest of this paper is organized as follows: In Section 2 we treat a number of examples of 'open proofs'. The examples have been chosen to be quite trivial,

which is done deliberately to keep the exposition small and to be able to pinpoint the crucial issues. In Section 3 we define *open higher order predicate logic*, a version of higher order predicate logic where we allow unfinished (open) proofs and open terms. Open proofs are represented by means of unfinished parts of a deduction, a "hole in the derivation tree". Open terms are represented via a kind of "meta-level Skolem functions" of the form $m[x_1, \ldots, x_n]$ which we call *meta-variables*. A meta-variable can only occur in "fully applied form": $m[t_1, \ldots, t_n]$, where $t_1, \ldots, t_n$ are terms. In the process of filling in the holes of a proof, we seek *instantiations* of these meta-variables. This use of meta-variables avoids the use of explicit substitutions that occur in various other treatments of open terms. Finally in Section 4 we define a type theoretic variant of this open higher order predicate logic. In the type theory both open proofs and open terms are represented as *meta-variables*, in the way mentioned before. Again the instantiation mechanism for meta-variables avoids the use of explicit substitutions. We extend the well-known formulas-as-types embedding to include open proofs and open terms. Then we show how this type theory captures the notion of *proof state*.

2 Motivating Examples

1. An Unfinished Proof with *Backward* Proof Construction.

We start with the goal of proving $A{\to}C$ from hypotheses $A{\to}B{\to}C$ and $A{\to}B$ (1). We solve this goal by the rule for introduction of implication (2). This introduces a new hypothesis A. In (3) we have used the hypothesis $A{\to}B{\to}C$ to deduce C by implication-elimination of the new goals A and B. The first one is solved in (4) by the assumption A and the second by introducing a new goal A and eliminating the assumption $A{\to}B$. Finally (5) we solve A trivially by the hypothesis A and we have a complete derivation of $A{\to}C$ from $A{\to}B{\to}C$ and $A{\to}B$.

1. $$A{\to}B{\to}C \quad A{\to}B \qquad \frac{?}{A{\to}C}$$

2. $$A{\to}B{\to}C \quad A{\to}B \quad [A]^i \qquad \frac{\dfrac{?}{C}}{A{\to}C}\,{}^i$$

3. $$A{\to}B{\to}C \quad A{\to}B \quad [A]^i \qquad \frac{\dfrac{A{\to}B{\to}C \quad \dfrac{?}{[A]^i}}{B{\to}C} \quad \dfrac{?}{B}}{\dfrac{C}{A{\to}C}\,{}^i}$$

4. $$A{\to}B{\to}C \quad A{\to}B \quad [A]^i \qquad \frac{\dfrac{A{\to}B{\to}C \quad [A]^i \quad \dfrac{A{\to}B \quad \dfrac{?}{A}}{B}}{B{\to}C}}{\dfrac{C}{A{\to}C}\,{}^i}$$

5. $$A{\to}B{\to}C \quad A{\to}B \quad [A]^i \qquad \frac{\dfrac{A{\to}B{\to}C \quad [A]^i \quad \dfrac{A{\to}B \quad [A]^i}{B}}{B{\to}C}}{\dfrac{C}{A{\to}C}\,{}^i}$$

2. An Unfinished Proof with a *Forward* Proof Construction.

We proceed forward by using elimination rules on the hypotheses. In (2) we have used the A and $A{\to}B$ to obtain B which is used in (3) to deduce $B{\to}C$. Then we must infer B again and use it to derive C at step (4). Note that in step (4) we would like to be able to *reuse* the already proven result B instead of having to derive it again, but natural deduction does not allow this.

1. $$\dfrac{B{\to}B{\to}C \quad A{\to}B \quad A}{C}\,?$$

2. $$\dfrac{B{\to}B{\to}C \quad \dfrac{A{\to}B \quad A}{B}}{C}\,?$$

3. $$\dfrac{\dfrac{B{\to}B{\to}C \quad \dfrac{A{\to}B \quad A}{B}}{B{\to}C}}{C}\,?$$

4. $$\dfrac{\dfrac{B{\to}B{\to}C \quad \dfrac{A{\to}B \quad A}{B}}{B{\to}C} \quad \dfrac{A{\to}B \quad A}{B}}{C}$$

3. An Unfinished Proof with *Open Terms*

In this example we have a transitive relation $R(x,y)$ and we want to prove $R(a,c)$. The question is *what to take for y?* We don't know (yet), so we want to leave y *open*. From this example we see that open terms arise quite naturally in interactive theorem proving if we want to postpone the specific choice of a value for a variable.

1. $$\dfrac{\forall x,y,z.R(x,y){\to}R(y,z){\to}R(x,z)}{R(a,c)}\,?$$

2. $$\dfrac{\dfrac{R(a,\underline{y}){\to}R(\underline{y},c){\to}R(a,c) \quad \dfrac{}{R(a,\underline{y})}\,?}{R(\underline{y},c){\to}R(a,c)} \quad \dfrac{}{R(\underline{y},c)}\,?}{R(a,c)}$$

The 'open place' $\underline{y}$ in the example has a different role than a variable: we seek an value for it and we will not abstract over it. We will call these open places *meta-variables*. A term containing a meta-variable will be called an *open term*.

Convention 1 *To clearly distinguish variables from meta-variables, we will underline meta-variables, so $\underline{y}$ denotes a meta-variable and y is different from $\underline{y}$.*

4. Delaying the Choice of the *Witness* for an Existential Quantifier and *Computing* with Open Terms.

In order to prove an existential formula $\exists x.A(x)$ constructively one usually needs to find a term t (called also a *witness*) and prove $A(t)$. Often the choice of the term is not obvious and one may want to leave it open while continuing with the proof. This can be achieved by using a metavariable for t.

In this example, the witness meta-variable $\underline{n}$ should actually *depend* on y, because we want to be able to instantiate n with y. If we do that in the last proof (4), y becomes an unbound variable, so that is not correct. Hence we have to be careful with the definition of instantiation. As we can see, the problem occurs because *reduction* and *instantiation* do not *commute*.

To prove the correctness of instantiation, we would need that instantiation must commute with the derivation rules (Lemma 13). This property depends essentially on the commutation of instantiation and reduction. It is depicted in the diagram below, together with its instance to the above example (where it fails).

1. $$\dfrac{?}{\exists f \forall x.f(x) = x}$$

2. $$\dfrac{\dfrac{?}{\forall x.\underline{f}(x) = x}}{\exists f \forall x.f(x) = x}$$

3. $$\dfrac{\dfrac{?}{\forall x.(\lambda y.\underline{n})(x) = x}}{\exists f \forall x.f(x) = x}$$

4. $$\dfrac{\dfrac{\dfrac{???}{\forall x.\underline{n} = x}}{\forall x.(\lambda y.\underline{n})(x) = x}}{\exists f \forall x.f(x) = x}$$

$$\begin{tikzcd}
M \arrow[r, "\text{instantiate } \underline{n} := t"] \arrow[d, "\beta"'] & N \arrow[d, "\beta"] \\
P \arrow[r, "\text{instantiate } \underline{n} := t"] & ?? \\
(\lambda y.\underline{n})(x) \arrow[r, "\text{instantiate } \underline{n} := y"] \arrow[d, "\beta"'] & (\lambda y.y)(x) \arrow[d, "\beta"] \\
\underline{n} \arrow[r, "\text{instantiate } \underline{n} := y"] & ??
\end{tikzcd}$$

The solution is to record the dependency of a meta-variable on other variables by writing $\underline{n}[y]$. An alternative solution is to delay substitutions by using *explicit* substitutions. Then we would have, e.g. $x[y := t] = x$, for x a normal variable ($x \neq y$), but $\underline{n}[y := t] \neq \underline{n}$ for a meta-variable. This approach is taken by [12] and [9]. We follow the first approach, also taken by [16]. As an illustration on the right we redo the above example, but now with *dependencies* of meta-variables recorded.

$$\dfrac{?}{\exists f \forall x.f(x) = x} \qquad \dfrac{\dfrac{?}{\forall x.\underline{f}(x) = x}}{\exists f \forall x.f(x) = x}$$

$$\dfrac{\dfrac{?}{\forall x.(\lambda y.\underline{n}[y])(x) = x}}{\exists f \forall x.f(x) = x} \qquad \dfrac{\dfrac{\dfrac{?}{\forall x.\underline{n}[x] = x}}{\forall x.(\lambda y.\underline{n}[y])(x) = x}}{\exists f \forall x.f(x) = x}$$

$$\dfrac{\dfrac{\forall x.x = x}{\forall x.(\lambda y.\underline{n}[y])(x) = x}}{\exists f \forall x.f(x) = x}$$

5. Using Meta-variables to Represent *Unknown Formulas*.

Suppose we are in arithmetics. The 'usual' induction principle is expressed by the formula $Ind_1 = \forall P{:}N{\to}\mathsf{Prop}.P(0) \wedge \forall n.P(n){\to}P(n{+}1) \to \forall n.P(n)$. The 'course-of-value' induction principle is expressed by the formula $Ind_2 = \forall P.(\forall n(\forall k < n.P(k)){\to}P(n)) \to \forall n.P(n)$.

Suppose we want to prove that Ind_1 implies Ind_2. We will show how meta-variables can be used to prove this implication without having to make guesses 'out of the blue'. After an obvious backward step we have the initial open proof shown on the right ($P_<(n)$ abbreviates $\forall k < n.P(k)$).

$$\dfrac{\dfrac{\dfrac{Ind_1 \quad [\forall n P_<(n) \to P(n)]^i}{?}}{\forall n.P(n)}\,i}{Ind_2}$$

It is clear that we need to use the hypothesis Ind_1. To do that we have to eliminate the universal quantifier. Since we do not want to make guesses, we delay the choice and introduce a meta-variable $\underline{B}$ for the unknown predicate.

$$\dfrac{\dfrac{\dfrac{\dfrac{Ind_1}{\underline{B}[0] \wedge (\forall n.\underline{B}[n] \to \underline{B}[n+1]) \to \forall n.\underline{B}[n]} \quad [\forall n.P_<(n) \to P(n)]^i}{?}}{\forall n.P(n)}\,i}{Ind_2}$$

An obvious step towards solving the goal is to reduce it to these three subgoals:

$$(1)\ \dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{?}}{\underline{B}[0]} \qquad (2)\ \dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{?}}{\forall n.\underline{B}[n] \to \underline{B}[n+1]} \qquad (3)\ \dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{?}}{\forall n.\underline{B}[n] \to P(n)}$$

The idea of course is to use (1) and (2) with implication elimination to obtain $\forall n.\underline{B}[n]$ from which using (3) we would derive $\forall n.P(n)$. To discard goal (3), it is sufficient to define $\underline{B}[n] := P(n) \wedge \underline{C}[n]$ where $\underline{C}[n]$ is a fresh meta-variable of type Prop. After the instantiation goals (1) and (2) look like this:

$$(1)\ \dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{?}}{P(0) \wedge \underline{C}[0]} \qquad (2)\ \dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{?}}{\forall n.(P(n) \wedge \underline{C}[n]) \to (P(n+1) \wedge \underline{C}[n+1])}$$

Goal (2) is the hardest to solve. However without much creativity we observe that we can replace it by the following two goals: (2a) $P(n) \wedge \underline{C}[n] \to \underline{C}[n+1]$ and (2b) $\forall m.\underline{C}[m] \to P(m)$. Analyzing goal (2b) shows that we are in the following situation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\forall n.P_<(n) \to P(n)}{P_<(m) \to P(m)} \quad [\underline{C}[m]]^j}{?}}{P(m)}}{\underline{C}[m] \to P(m)}\,j}{\forall m.\underline{C}[m] \to P(m)}$$

and it is now not difficult to see that $\underline{C}[n]$ can be taken to be the formula $P_<(n)$ and the remaining goals (1) and (2a) are easily provable. Hence the final solution for the predicate $\underline{B}[n]$ is $P(n) \wedge P_<(n)$ or equivalently, $\forall k \leq n.P(k)$.

3 Open Higher Order Predicate Logic

We now give a formal definition of higher order predicate logic with open terms and open proofs, o-HOL. As usual, we first define the language, then the derivation rules and then the notion of derivability. We show that o-HOL is conservative over HOL, ordinary higher order predicate logic [2, 5]. This means that, if we have derived the higher order formula A in o-HOL without unfinished subproofs, then A is derivable in HOL.

Most of o-HOL is the same as HOL, but we present it nevertheless.

Definition 2 (Language of o-HOL).

- *The* domains*:* $\mathcal{D} ::= \mathsf{Prop} \mid \mathcal{B} \mid \mathcal{D}{\to}\mathcal{D}$*, where* Prop *is the domain of* propositions*,* $\mathcal{B}$ *is an arbitrary* base *domain. We use currying to represent domains of higher arity. Arbitrary domains will be denoted by* σ, τ*.*
- *The* terms*, Term(o-HOL):*
  - variables*, typed with a domain, notation* x_i^σ *or* $x_i : \sigma$*.*
  - application*:* $(f\,t) : \tau$*, if* $f : \sigma{\to}\tau$ *and* $t : \sigma$*.*
  - abstraction $(\lambda x{:}\sigma.q) : \sigma{\to}\tau$*, if* $q : \tau$*.*
  - formula constructors $A{\wedge}B : \mathsf{Prop}$*,* $A{\to}B : \mathsf{Prop}$*,* $A{\vee}B : \mathsf{Prop}$*,* $\neg A : \mathsf{Prop}$*,* $\forall x{:}\sigma.A : \mathsf{Prop}$*,* $\exists x{:}\sigma.A : \mathsf{Prop}$*, if* $A, B : \mathsf{Prop}$ *and* σ *a domain.*
  - meta-variable applications*:* $\underline{m}[t_1, \ldots, t_n] : \tau$*, if* $t_1 : \sigma_1, \ldots, t_n : \sigma_n$ *and* $\underline{m}[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$ *is a* meta-variable*.*

Remark 3. We will call 'formula' any term from the domain Prop. Note that the definition above allows also metavariables standing for formulas or functions producing formulas.

Remark 4. Meta-variables themselves are *not* terms. There are countably many meta-variables for every $\sigma_1, \ldots, \sigma_n, \tau$. We view the 'assignment' $[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$ as being part of the meta-variable, so, for example $\underline{m}[y : \sigma] : \tau$ and $\underline{m}[y : \sigma] : \sigma$ are different meta-variables (but of course we will use different names as much as possible). Furthermore, α-convertible assignments will are considered identical: e.g. $m[x : \sigma] : \tau$ and $m[y : \sigma] : \tau$ denote the same meta-variable.

As terms with meta-variables are ordinary terms, meta-variables can occur in the arguments of another (or the same) meta-variable. For example, if $\underline{m}[y : \sigma, z : \sigma] : \sigma$ is a meta-variable and $f : \sigma{\to}\sigma$, then e.g. $\underline{m}[(f\,a), \underline{m}[a, (f\,a)]]$ is a well-formed term.

Notation: If the domains that we quantify over are irrelevant, we will write $\forall x.A$ instead of $\forall x{:}\sigma.A$. Also, we will often write $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}] : \tau$ or just $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}]$ or $\underline{m}[\boldsymbol{y}]$ for $\underline{m}[y_1 : \sigma_1, \ldots, y_n : \sigma_n] : \tau$.

Definition 5 (Derivation Rules of o-HOL). *These are the same as for HOL plus an extra rule for representing unknown proofs. We show the rules for* $\to$*,* $\forall$

and $\exists$*, the conversion rule and the new rule (claim).*

$\dfrac{\begin{array}{c}[A]^i\\ \vdots\\ B\end{array}}{A\to B}\,{\scriptstyle i}\ \to\text{-}I$	$\dfrac{A\to B \quad A}{B}\to\text{-}E$	$\dfrac{A[t/x]}{\exists x{:}\sigma.A}\exists\text{-}I$	$\dfrac{\exists x{:}\sigma.A \quad \begin{array}{c}[A]^i\\ \Sigma\\ B\end{array}}{B}\,{\scriptstyle i}\ \exists\text{-}E$
		if $t:\sigma$	if $x \notin FV(\mathcal{A}(\Sigma)\setminus\{A\}\cup\{B\})$
$\dfrac{\begin{array}{c}\Sigma\\ A\end{array}}{\forall x{:}\sigma.A}\ \forall\text{-}I$	$\dfrac{\forall x{:}\sigma.A}{A[t/x]}\ \forall\text{-}E$	$\dfrac{A}{B}\ (conv)$	$\dfrac{B_1,\ldots,B_n}{A}(claim)$
if $x \notin FV(\mathcal{A}(\Sigma))$	if $t:\sigma$	if $A =_\beta B$	

where $\mathcal{A}(\Sigma)$ *is the set of undischarged assumptions of* Σ*.*

The rule (claim) represents an unknown derivation of A from $B_1,\ldots,B_n$. The hypotheses of the unknown derivation need to be specified explicitly, for example, because we need to check side conditions on assumptions in the rest of the rules (and these refer to the leaves of a derivation). This explicit representation of the hypotheses also allows us to represent the *forward steps* that one may want to do. Sometimes in derivations we will use the symbol '?' to denote the (claim) rule.

As usual, in the $\to$-I rule, the A-leaves that are labelled with i (notation $[A]^i$) are *discharged*, so they are no longer assumptions. Similarly, the A-leaves in the $\exists$-E rule are discharged.

In the conversion rule, $=_\beta$ is defined in terms of

$$(\lambda x{:}\sigma.t)q \longrightarrow_\beta t[q/x]$$

The *substitution* used here extends immediately to terms with *meta-variables*:

$$\underline{m}[t_1,\ldots,t_n][q/x] := \underline{m}[t_1[q/x],\ldots,t_n[q/x]]$$

We always work *modulo* α*-conversion*. Hence we adopt the *variable convention* (also called 'Barendregt convention') that we always assume all *bound variables* (BV) to be *different* and *different* from the *free variables* (FV).

A *derivation tree* in o-HOL is the same as a derivation in HOL, except for the fact that we can now also have (claim) nodes in the tree. In the notion of *derivability* we also have to take the 'open parts' of the derivation tree (the (claim) nodes) into account. We will call these *goals*.

It is allowed that variables occur free in the goals. If a variable x occurs free in a specific formula in a derivation Σ, it may be *bound* in Σ (by a $\forall$-I rule or a $\exists$-E rule) or it may be *free* in Σ. We define these notions explicitly, as it is important for our interpretations of goals.

Definition 6 (Bound Occurrences of a Variables in a Derivation). *Let* Σ *be a derivation and* A *a formula occurring in* Σ *with* $x \in FV(A)$*. We say that*

$x \in \mathrm{FV}(A)$ is bound in Σ *in one of the following two situations*

$$\dfrac{\begin{array}{c} A \\ \vdots \\ B \end{array}}{\forall x{:}\sigma.B}\,\forall\text{-}I \qquad\qquad \dfrac{\exists x{:}\sigma.C \quad \dfrac{\begin{array}{c}[C]^i \\ \vdots \\ A \\ \vdots \end{array}}{B}\,i}{B}\,\exists\text{-}E$$

with x free in all the formulas in the derivation between A and B (inclusive).

with x free in all the formulas in the derivation between C and A (inclusive).

So, the notion of '$x \in \mathrm{FV}(A)$ is bound in Σ' is about a specific occurrence of A in the derivation Σ. It is defined by induction on Σ. Note that $x \in \mathrm{FV}(A)$ may be bound for one occurrence of A and free for another.

Definition 7 (Goals in a Derivation).

1. *A* goal *in o-HOL is a judgement of the form*

$$x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n, A_1, \ldots, A_n \rightsquigarrow B,$$

where $A_1, \ldots, A_n, B$ are formulas and $x_1, \ldots, x_n \in FV(A_1, \ldots, A_n, B)$. The goal binds the occurrences of $x_1, \ldots, x_n$ in its formulas.

2. *A goal $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n, A_1, \ldots, A_n \rightsquigarrow B$, is a* goal of the derivation Σ *if Σ contains an application of the claim rule*

$$\dfrac{A_1 \;\; \ldots \;\; A_n}{B}\,(claim)$$

with $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n$ the variables free in $A_1, \ldots, A_n, B$ but bound in Σ.

The problem of managing the free and bound variables and their scopes is crucial for solving the problems of instantiation and computation (see 2.4).

Definition 8 (Derivability in o-HOL). *Given a set of formulas Γ, a set of goals G and a formula B, we say that B is* derivable *from $\Gamma; G$ in o-HOL, notation*

$$\Gamma; G \vdash_i B,$$

if there is a derivation Σ with conclusion B, (non-discharged) assumptions in Γ and all goals of Σ in G.

An important property of HOL is that the derivation rules are *compatible* with *substitution*. Hence derivations and derivability are compatible with substitution:

if $\Gamma \vdash A$ with derivation Σ, then $\Gamma[t/x] \vdash A[t/x]$ with derivation $\Sigma[t/x]$.

For o-HOL we have the same properties, where we have to take note that in a goal $x_1{:}\sigma_1, \ldots, x_n{:}\sigma_n, A_1, \ldots, A_n \rightsquigarrow B$, the variables $x_1, \ldots, x_n$ are *bound* in $A_1, \ldots, A_n, B$. Hence, we do not substitute for these variables but rename them appropriately.

Lemma 9 (Compatibility of Derivability and Substitution in o-HOL). *If $\Gamma; G \vdash_i A$, then $\Gamma[t/x]; G[t/x] \vdash_i A[t/x]$.*

Proof. By induction on the derivation tree Σ, one proves that, if Σ has conclusion A, assumptions Γ and goals G, then $\Sigma[t/x]$ is a well-formed derivation with conclusion $A[t/x]$, assumptions $\Gamma[t/x]$ and goals $G[t/x]$. □

Example 10. Consider the following two derivations on the right, where in the first x occurs bound and in the second, x occurs free. The judgements associated with these two derivations are $A, C; (y{:}\sigma, A) \rightsquigarrow B(y), (y'{:}\sigma, C) \rightsquigarrow B(y') \to D(y') \vdash \forall x{:}\sigma.D(x)$ for the first and $A(x), C; A \rightsquigarrow B(x), C \rightsquigarrow B(x) \to D(x) \vdash D(x)$ for the second. Note what happens if we substitute t for x in the two derivations.

$$\dfrac{\dfrac{\dfrac{A}{B(x)}\,? \qquad \dfrac{C}{B(x)\to D(x)}\,?}{D(x)}}{\forall x{:}\sigma.D(x)}$$

$$\dfrac{\dfrac{A(x)}{B(x)}\,? \qquad \dfrac{C}{B(x)\to D(x)}\,?}{D(x)}$$

An important operation on derivations is *instantiation* (choosing a value for a meta-variable). Therefore, an equally important property for o-HOL is the compatibility of the derivation rules with *instantiation* of meta-variables. We first give a precise definition of instantiation.

Definition 11. *For $\underline{n}[\boldsymbol{y} : \boldsymbol{A}] : B$ a meta-variable and $t : B$ a term, we call*

$$\{\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := t\}$$

an instantiation *(of $\underline{n}[\boldsymbol{y}]$ by t). The instantiation binds the occurrences of $\boldsymbol{y}$ in t and t may contain also variables different from those in $\boldsymbol{y}$.*

Since the variables $\boldsymbol{y}$ are considered bound, the following two instantiations by our convention are considered identical:

$$\{\underline{n}[x : A, y : B] := xy\} \qquad \{\underline{n}[z : A, x : B] := zx\}$$

The application of instantiation is defined immediately for all terms. The only interesting cases are the meta-variable applications.

$$(\underline{n}[\boldsymbol{q}])\{\underline{n}[\boldsymbol{y}] := t\} := t[\boldsymbol{q}\{\underline{n}[\boldsymbol{y}] := t\}/\boldsymbol{y}],$$
$$(\underline{m}[\boldsymbol{q}])\{\underline{n}[\boldsymbol{y}] := t\} := \underline{m}[\boldsymbol{q}\{\underline{n}[\boldsymbol{y}] := t\}] \text{ for } m, n \text{ different meta-variables.}$$

Note that the instantiations have to be applied heriditarily (also to $\boldsymbol{q}$ in the first case), because $\boldsymbol{q}$ may contain $\underline{n}$, so for example

$$\underline{n}[(f\,a), \underline{n}[a, (f\,a)]]\{n[x, y] := g\,x\,y\} \;=\; g\,(f\,a)(g\,a\,(f\,a)).$$

The well-foundness of the instantiation can easily be proved by induction on the structure of the term in which we instantiate. Informally, we can think of the instantiation $M\{\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := t\}$ as (a reduct of) $(\lambda n.M)(\lambda \boldsymbol{y}.t)$, of $n[\boldsymbol{y} : \boldsymbol{A}] : B$ as a meta-level skolem function from $\boldsymbol{A}$ to B and of $n[\boldsymbol{s}]$ as a fully applied skolem function.

Adding parameters to meta-variables is enough to record the relevant substitutions that might be executed over the metavariable (see 2.4). This approach, also used in [16], eliminates the need to introduce *explicit substitutions* as a mechanism for postponing the substitutions over meta-variables.

We sometimes have to rename bound variables in derivations before performing an instantiation. This problem is not really new for o-HOL, because it already appears in HOL (when performing a substitution). To make our point clear we treat the following example.

Example 12. Consider a derivation Σ of $(P\,\underline{n}[\ ])$ and a derivation Θ of $(P\,\underline{n}[x])$, where Θ and Σ do not contain a free x in their assumptions. We can do a $\forall$-introduction and we can perform an instantiation, $\{\underline{n}[\,] := x+y\}$ on Σ, respectively $\{\underline{n}[x] := x+y\}$ on Θ. In the first derivation, to perform the instantiation, we first have to rename the bound variable x to z.

$$\dfrac{\dfrac{\Sigma}{(P\,\underline{n}[\,])}}{\forall x.(P\,\underline{n}[\,])} \quad \overset{\{n[\,]:=x+y\}}{\longmapsto} \quad \dfrac{\dfrac{\Sigma^{\{n[\,]:=x+y\}}}{(P(x+y))}}{\forall z.(P(x+y)}$$

$$\dfrac{\dfrac{\Theta}{(P\,\underline{n}[x])}}{\forall x.(P\,\underline{n}[x])} \quad \overset{\{n[x]:=x+y\}}{\longmapsto} \quad \dfrac{\dfrac{\Theta^{\{n[x]:=x+y\}}}{(P(x+y))}}{\forall x.(P(x+y))}$$

Instantiation is compatible with derivations in o-HOL. The proof is by induction on the structure of the derivation trees:

Lemma 13. *Let * denote an instantiation. If $\Gamma; G \vdash_i A$ with derivation Σ, then $\Gamma^*; G^* \vdash_i A^*$ with derivation Σ^*.*

Corollary 14 (o-HOL is conservative over HOL). *Let Γ and A be a context and a formula in HOL respectively.*

$$\textit{If } \Gamma; \emptyset \vdash_i A, \textit{ then } \Gamma \vdash A$$

Proof. Suppose $\Gamma; \emptyset \vdash_i A$ with derivation Σ. This derivation may still contain meta-variables, say $\underline{n_1}, \ldots, \underline{n_k}$. Let $\{\underline{n_1}[\_] := x_1\}, \ldots, \{\underline{n_k}[\_] := x_k\}$ be instantiations for these meta-variables with fresh variables of appropriate sort. If we perform all these instantiations on Σ, we obtain a derivation Σ' of $\Gamma; \emptyset \vdash_i A$ and this derivation contains no more meta-variables. But then Σ' is also a derivation in HOL, because it contains no applications of the (claim) rule and all the terms occurring in it are HOL-terms. □

Beyond Open Derivations

The logic o-HOL defined above gives us the answer to the problem of what an incomplete derivation is. Interactive theorem proving is however not only about

individual derivations. Often we encounter situations where more advanced applications are needed:

1. **Proof reuse.** Consider example 2 in Section 2. There we had to prove the same formula twice because we needed it in two different places. One would probably want to avoid this unnecessary effort by *reusing* proofs that have already been done.
2. **'Scratch-paper' mechanism.** We may also wish to *explore* our knowledge to come to good instantiations, or to reject potential instantiations.
   For example, suppose we want to prove the formula $\exists x.\varphi(x) \wedge (x < 2)$ from $\forall x.\varphi(x) \rightarrow (0 < x)$ (see (1)). From the assumption and the formula that we want to prove we can derive some properties that $\underline{x}$ must have (2).

$$(1)\quad \dfrac{\dfrac{\forall x.\varphi(x) \rightarrow (0 < x)}{\varphi(\underline{x}) \wedge (\underline{x} < 2)}\,?}{\exists x.\varphi(x) \wedge (x < 2)}$$

   From the conclusion of this extra derivation we may conclude that the only possible instantiation for $\underline{x}$ is $\{\underline{x} := 1\}$ (assuming that x is a natural number).

$$(2)\quad \dfrac{\dfrac{\dfrac{\varphi(\underline{x}) \wedge (\underline{x} < 2)}{\varphi(\underline{x})} \quad \dfrac{\forall x.\varphi(x) \rightarrow (0 < x)}{\varphi(\underline{x}) \rightarrow (0 < \underline{x})}}{(0 < \underline{x})}}{(0 < \underline{x} < 2)}$$

   This simple example illustrates the need to sometimes pause the construction of the 'main' derivation, do some side computations or inferences *within* its scope and then come back with the results.

A general problem that emerges from the examples above is that open derivations do not (yet) capture the notion of *proof state*. The system o-HOL is just about individual open derivations. A proof state is, intuitively, a 'connected' set of derivations. We will use *type theory* to formalize the notion of proof state.

4 The Curry-Howard Formulas-as-Types Embedding

The Curry-Howard formulas-as-types embedding maps derivations of the logic, in our case HOL, to *proof terms* of an appropriate type theory, in our case λHOL. The type system λHOL has two 'universes': Type, the type of all domains ($\mathcal{D}$ in the logic), and Prop, the type of all formulas. (Hence $\mathsf{Prop} : \mathsf{Type}$.) We do not give a definition of the type system λHOL but refer the reader to [5] or [1].

A central point in this mapping is that all elements of the language and all the variables in a HOL derivation can be systematically given bindings that form a context in type theory and that the derivation itself can be coded by a term which is typable in that context.

The type theory λHOL represents the logic HOL *faithfully*, because we have a *soundness* and a *completeness* result, stated as follows. (We use $\vdash_\lambda$ to denote derivability in the type theory and $\vdash_L$ to denote derivability in the logic.)

– **Soundness**: If $\Gamma \vdash_L A$ with derivation Σ, then $\Gamma_{\mathcal{L}}, \Gamma \vdash_\lambda [\![\Sigma]\!] : A$, where $\Gamma_{\mathcal{L}}$ declares the required parts of the language of HOL.

– **Completeness**: If $\Gamma \vdash_\lambda M : A$, then $\Gamma^- \vdash_L A$, where Γ^- selects the $A : \mathsf{Prop}$ for which $h{:}A \in \Gamma$.

For example the trivial derivation of $(Q\,x) \vdash_L (P\,x){\to}(Q\,x)$ maps to

$$D{:}\mathsf{Type}, P, Q{:}D{\to}\mathsf{Prop}, x{:}D, h : (Q\,x) \vdash_\lambda \lambda z{:}(P\,x).h : (P\,x){\to}(Q\,x).$$

We extend the formulas-as-types embedding to o-HOL by defining o-λHOL.

Definition 15. *The type system o-λHOL extends the type system λHOL allowing* meta-variable declarations *in the context of the form*

– $\underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau$ *with* $\boldsymbol{\sigma}, \tau : \mathsf{Type}$, open terms,
– $\underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B$ *with* $\boldsymbol{\sigma} : \mathsf{Type}$, $\boldsymbol{A}, B : \mathsf{Prop}$, open proofs,

The derivation rules are as follows.

$$\frac{\Gamma \vdash_\lambda \boldsymbol{\sigma}{:}\mathsf{Type} \quad \Gamma \vdash_\lambda \tau{:}\mathsf{Type}}{\Gamma, \underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau \vdash_\lambda \mathsf{Ok}} \qquad \frac{\Gamma \vdash_\lambda \boldsymbol{\sigma}{:}\mathsf{Type} \quad \Gamma, \boldsymbol{y} : \boldsymbol{\sigma} \vdash_\lambda \boldsymbol{A} : \mathsf{Prop} \quad \Gamma, \boldsymbol{y} : \boldsymbol{\sigma} \vdash_\lambda B : \mathsf{Prop}}{\Gamma, \underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B \vdash_\lambda \mathsf{Ok}}$$

$$\frac{\Gamma \vdash_\lambda \boldsymbol{t} : \boldsymbol{\sigma} \quad (\underline{n}[\boldsymbol{y} : \boldsymbol{\sigma}] : \tau) \in \Gamma}{\Gamma \vdash_\lambda \underline{n}[\boldsymbol{t}] : \tau} \qquad \frac{\Gamma \vdash_\lambda \boldsymbol{t} : \boldsymbol{\sigma} \quad \Gamma \vdash_\lambda \boldsymbol{r} : \boldsymbol{A}[\boldsymbol{t}/\boldsymbol{y}] \quad (\underline{p}[\boldsymbol{y} : \boldsymbol{\sigma}, \boldsymbol{q} : \boldsymbol{A}] : B) \in \Gamma}{\Gamma \vdash_\lambda \underline{p}[\boldsymbol{t}, \boldsymbol{r}] : B[\boldsymbol{t}/\boldsymbol{y}]}$$

$\Gamma \vdash_\lambda \mathsf{Ok}$ *is the judgement that* Γ *is* well-formed.

The type system o-λHOL enjoys all the nice meta-theoretic properties, like Subject Reduction, Confluence and Strong Normalization.

Lemma 16. *The formulas-as-types embedding from HOL to λHOL extends to a sound and complete formulas-as-types embedding from o-HOL to o-λHOL.*

Proof. Given the derivation Σ of $\Gamma; G \vdash_i A$, the embedding is defined by induction on Σ. We show how $[\![\Sigma]\!]$ is defined for some cases. First we have to define the context in which $[\![\Sigma]\!]$ is well-typed: from $\Gamma = \{A_1, \ldots, A_n\}$, we construct $h_1{:}A_1, \ldots, h_n{:}A_n$, with $h_1, \ldots, h_n$ fresh variables. We denote this context also by Γ. A goal $(\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{A}) \rightsquigarrow B$ is translated to the declaration $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{A}] : B$, with $\underline{m}$ a fresh meta-variable. Thus the set of goals G is translated to a sequence of meta-variable declarations, which we also denote by G. Finally, we need a context to declare all the domain symbols and all free variables and meta-variables that occur in Σ, Γ and G. This yields the context $\Gamma_{\mathcal{L}}$. To show that $[\![\Sigma]\!]$ is indeed a well-typed term of type A in $\Gamma_{\mathcal{L}}, \Gamma, G$ requires some meta-theory of the type system, which we do not provide here. In the following, if we write a derivation Σ with A on top and B below it, we mean that A and B are part of the derivation Σ.

1. If the last rule is (claim), then

$$\frac{\begin{matrix}\Sigma_1 & & \Sigma_n \\ B_1 & \cdots & B_n\end{matrix}}{A}$$

We construct $\Gamma_{\mathcal{L}}$ as the context of declarations for free variables and domains in Σ, Γ, G. For each Σ_i we construct Γ_i and G_i and by induction we find $[\![\Sigma_i]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_i, G_i \vdash_i [\![\Sigma_i]\!] : B_i$. The goal is translated to a meta-variable $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{B}] : A$, with $\boldsymbol{y}$ the variables bound in Σ. We define $[\![\Sigma]\!] := \underline{m}[\boldsymbol{y}, [\![\boldsymbol{\Sigma}]\!]]$ and find that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1, \ldots, \Gamma_n, G_n, \underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{B}] : A \vdash_i [\![\Sigma]\!] : B$.

2. If the last rule is ($\rightarrow$-I), then

$$\dfrac{\begin{array}{c}[A]^i \ldots [A]^i \\ \Sigma_1 \\ B\end{array}}{A \rightarrow B}\, i$$

For Σ_1 we construct $\Gamma_{\mathcal{L}}$, Γ_1 and G_1 and by induction we find $[\![\Sigma_1]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1 \vdash_i [\![\Sigma_1]\!] : B$. The discharged occurrences of A correspond to variable declarations $h_1{:}A, \ldots, h_n{:}A$ in Γ. We take $\Gamma := \Gamma \setminus (h_1{:}A, \ldots, h_n{:}A)$ and $G := G_1$. We define $[\![\Sigma]\!] := \lambda h{:}A.([\![\Sigma_1]\!][h/h_1, \ldots, h/h_n])$ and find that $\Gamma_{\mathcal{L}}, \Gamma, G \vdash_i [\![\Sigma]\!] : A{\rightarrow}B$.

3. If the last rule is ($\forall$-I), then

$$\dfrac{\begin{array}{c}\Sigma_1 \\ B\end{array}}{\forall x{:}\sigma.B}$$

For Σ_1 we construct $\Gamma_{\mathcal{L}}$, Γ_1 and G_1 and by induction we find $[\![\Sigma_1]\!]$ such that $\Gamma_{\mathcal{L}}, \Gamma_1, G_1 \vdash_i [\![\Sigma_1]\!] : B$. The quantified variable x may occur as a declaration in $\Gamma_{\mathcal{L}}$, but it does not occur free in Γ_1. So for Σ, we have $\Gamma_{\mathcal{L}} = \Gamma_{\mathcal{L}} \setminus (x{:}\sigma)$ and $\Gamma = \Gamma_1$. In the goals of Σ_1, x is free, whereas in the goals of Σ, x is bound. So, if $\underline{m}[\boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{C}] : A$ is a meta-variable declaration in G_1 with $x \in \mathrm{FV}(\boldsymbol{C}, A)$, then we replace this with the meta-variable declaration $\underline{m'}[x{:}\sigma, \boldsymbol{y}{:}\boldsymbol{\sigma}, \boldsymbol{h}{:}\boldsymbol{C}] : A$ in G. We define $[\![\Sigma]\!] := \lambda x{:}\sigma.[\![\Sigma_1]\!]\{\underline{m}[\boldsymbol{y}, \boldsymbol{h}] := \underline{m'}[x, \boldsymbol{y}, \boldsymbol{h}]$ and we find that $\Gamma_{\mathcal{L}} \setminus (x{:}\sigma), \Gamma, G \vdash_i [\![\Sigma]\!] : \forall x{:}\sigma.B$.

Proof states can now be represented as well-formed *contexts*. For *reuse* we also introduce *definitions* of (meta-)variables.

Definition 17. *The derivation rule for* definitions *is as follows:*

$$\dfrac{\Gamma, \boldsymbol{y} : \boldsymbol{A} \vdash_\lambda q : B}{\Gamma, (\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := q : B) \vdash_\lambda \mathsf{Ok}} \quad \dfrac{\Gamma \vdash_\lambda q : B}{\Gamma, (n := q : B) \vdash_\lambda \mathsf{Ok}}$$

The *computation rules* for definitions are by *local* instantiation and *local* unfolding. That is because in general we do not want to instantiate all meta-variables at the same time (or unfold all definitions at the same time), but do that one by one. This reduction depends on the context Γ, where the definitions are recorded. If $(\underline{n}[\boldsymbol{y} : \boldsymbol{A}] := q : B) \in \Gamma$, resp. $(n := q : B) \in \Gamma$, the rule reads as follows.

$$t(\underline{n}[\boldsymbol{r}]) \xrightarrow{\Gamma}_\delta t(q[\boldsymbol{r}/\boldsymbol{y}])$$
$$t(n) \xrightarrow{\Gamma}_\delta t[q/n]$$

where $t(n)$ signifies one specific occurrence of n in t (and similarly for $t(\underline{n}[\boldsymbol{r}])$. Details of extensions of type theory with an explicit definition mechanism can be found in [15].

We illustrate how the type-theoretic contexts capture the notion of proof state by the following two examples.

Example 18. Consider the 'scratch-paper' example from Section 3. We can accomodate both the main derivation and the scratch derivation in one context. Let M be the term encoding the scratch derivation. The context now is as follows.

$$\begin{array}{l}
\Gamma_0, \underline{x}[] : _, \\
h_{goal}[p : \forall x.\varphi(x) \rightarrow (0 < x)] : \varphi(\underline{x}) \wedge (\underline{x} < 2), \\
h_{scratch}[p : \forall x.\varphi(x) \rightarrow (0 < x)] := M(\underline{x}, p, h_{goal}) : (0 < \underline{x} < 2), \\
h_{main}[p : \forall x.\varphi(x) \rightarrow (0 < x)] := \langle \underline{x}, h_{goal}[p] \rangle : \exists x.\varphi(x) \wedge (\underline{x} < 2).
\end{array}$$

A tactic transforms proof states. As proof states are formalized as contexts, tactics should be context transformers. As an example we show the 'apply' tactic.

Example 19 (The `Apply` *tactic).* Together with a goal to be proved, this tactic takes as inputs a proof of a universally quantified or implicational formula U and a list of terms/proofs. It applies elimination rules to U with the terms/proofs from the list, until a proof of the current goal B is obtained or no elimination rule is applicable. In the latter case the tactic fails. If the user has not made a decision on which terms/proofs to take, the system uses fresh meta-variables. Suppose Σ is some (open) derivation of $U = \forall x.C_1(x) \rightarrow \forall y.C_2(x,y) \rightarrow B(x)$ and we want to prove $B(s)$.

$$\frac{A_1, \ldots, A_n}{\dfrac{?}{B(s)}} \quad \xrightarrow{\texttt{Apply}\ \Sigma\ s} \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma}{\forall x.C_1(x) \rightarrow \forall y.C_2(x,y) \rightarrow B(x)}}{C_1(s) \rightarrow \forall y.C_2(s,y) \rightarrow B(s)} \quad \dfrac{\dfrac{A_1, \ldots, A_n}{?}}{C_1(s)}}{\forall y.C_2(s,y) \rightarrow B(s)}}{C_2(s, \underline{y}[\,]) \rightarrow B(s)} \quad \dfrac{\dfrac{A_1, \ldots, A_n}{?}}{C_2(s, \underline{y}[\,])}}{B(s)}$$

Note the introduction of the two new goals and the meta-variable $\underline{y}$. We can represent this tactic as a mapping between contexts:

$$\Gamma, \underline{h}[\boldsymbol{p} : \boldsymbol{A}] : B(s), \Delta \quad \xrightarrow{\texttt{Apply}\ M\ s} \quad \begin{array}{l}
\Gamma, \underline{y}[\,] : \sigma, \\
\underline{h}'[\boldsymbol{p} : \boldsymbol{A}] : C_1(s), \\
\underline{h}''[\boldsymbol{p} : \boldsymbol{A}] : C_1(s, \underline{y}[\,]), \\
\underline{h}[\boldsymbol{p} : \boldsymbol{A}] := (M\ s\ \underline{h}'[\boldsymbol{p}]\ \underline{y}[\,]\ \underline{h}''[\boldsymbol{p}]) : B(s), \\
\Delta
\end{array}$$

where $\Gamma \vdash M : \forall x.C_1(x) \rightarrow \forall y.C_2(x,y) \rightarrow B(x)$ represents the derivation Σ. Note the introduction and the use of the three new meta-variables $\underline{h'}$, $\underline{h''}$ and $\underline{y}$.

5 Conclusions and Further Work

In this paper we have formalized incomplete derivations in higher order predicate logic. By extending the Curry-Howard embedding to incomplete proofs we hope

to have filled a gap that results from focusing the studies of incomplete objects exclusively to type theory.

Among the topics that need to be investigated further is the question whether this framework is *flexible* enough to 'freely' do proofs in the way we like. This is a crucial point with respect to the practical applicability of interactive theorem proving. Related issues are the problems of finding a canonical set of *basic tactics* and *tacticals* that generate all (useful) tactics and the problems connected with viewing large proof states.

References

[1] H. Barendregt and H. Geuvers. Proof assistants using dependent type systems. In *Handbook of Automated Reasoning*. Elsevier Science Publishers B. V., 1999. 548

[2] Henk Barendregt. Lambda calculi with types. In Abramsky et al., editor, *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992. 538, 543

[3] H. P. Barendregt. *The λ-calculus: Its syntax and semantics*. North-Holland, 1984. 538

[4] Mirna Bognar. PhD thesis, VU Amsterdam, to appear, 2002. 538

[5] J. H. Geuvers. *Logics and Type systems*. PhD thesis, University of Nijmegen. 543, 548

[6] G. I. Jojgov. Systems for open terms: An overview. Technical Report CSR 01-03, Technische Universiteit Eindhoven, 2001. 538

[7] Zhaohui Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, July 1990. 538

[8] Zhaohui Luo. PAL$^+$: A lambda-free logical framework. *Lournal of Functional Programming*, to appear.

[9] Lena Magnusson. *The Implementation of ALF - a Proof Editor based on Martin-Löf Monomorphic Type Theory with Explicit Substitutions*. PhD thesis, Chalmers University of Technology / Göteborg University, 1995. 538, 541

[10] Conor McBride. *Dependently Typed Functional Programs and their Proofs*. PhD thesis, University of Edinburgh, 1999. 538

[11] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991. 538

[12] César A. Muñoz. *A Calculus of Substitutions for Incomplete-Proof Representation in Type Theory*. PhD thesis, INRIA, November 1997. 538, 541

[13] Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989. 538

[14] Frank Pfenning. Logical frameworks. In *Handbook of Automated Reasoning*, pages 1063–1147. 2001. 538

[15] P. Severi and E. Poll. Pure Type Systems with definitions. In *Proc. of LFCS'94, St. Petersburg, Russia*, number 813 in LNCS, Berlin, 1994. Springer Verlag. 551

[16] M. Strecker. *Construction and Deduction in Type Theories*. PhD thesis, Universität Ulm, 1998. 538, 541, 547

Logical Relations for Monadic Types*

Jean Goubault-Larrecq[1], Slawomir Lasota[1,2], and David Nowak[1]

[1] LSV, CNRS & ENS Cachan, France
[2] Institute of Informatics, Warsaw University, Poland
{goubault,lasota,nowak}@lsv.ens-cachan.fr

Abstract. Logical relations and their generalizations are a fundamental tool in proving properties of lambda-calculi, e.g., yielding sound principles for observational equivalence. We propose a natural notion of logical relations able to deal with the monadic types of Moggi's computational lambda-calculus. The treatment is categorical, and is based on notions of subsconing and distributivity laws for monads. Our approach has a number of interesting applications, including cases for lambda-calculi with non-determinism (where being in logical relation means being bisimilar), dynamic name creation, and probabilistic systems.
Keywords: logical relations, monads, semantics, typed lambda-calculus.

1 Introduction

Motivation and context. Logical relations and their generalizations [13] are a fundamental tool in proving properties of lambda-calculi, e.g., characterizing lambda-definability [19, 9, 2, 4], proving equational completeness [13, 24], studying parametric polymorphism [21, 12, 11] notably. On the other hand, Moggi's computational lambda-calculus [16] has proved useful to define various notions of computations on top of the lambda-calculus: side-effects, input-output, continuations, non-determinism [26], probabilistic computation [20] in particular.

What should then be a natural notion of logical relation for Moggi's computational lambda-calculus? Although there is no unique answer to this question, we propose one that is satisfying in practice. We shall demonstrate the relevance of our approach by illustrating our construction on monads for non-determinism, dynamic name creation, and probabilistic computation.

Moggi's insight is based on categorical semantics: while categorical models of the λ-calculus are cartesian closed categories (CCCs), the computational lambda-calculus requires CCCs with a strong monad $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{t})$. The *monadic types* of the computational lambda-calculus are given by the syntax:

$$\tau ::= b | \tau \to \tau | \tau \times \tau | \boldsymbol{T}(\tau)$$

* The first author acknowledges partial support by the RNTL project EVA. The first and third authors acknowledge partial support by the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions". The second author acknowledges partial support by the post-doc fellowship of the Foundation for Polish Science and by the Polish KBN grant 7 T11C 002 21.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 553–568, 2002.
© Springer-Verlag Berlin Heidelberg 2002

where b ranges over a set B of so-called *base types*, and $\boldsymbol{T}(\tau)$ is meant to denote the type of *computations* of type τ. Compared to the lambda-calculus, Moggi's calculus has an additional **val** operation, of type $\tau \to \boldsymbol{T}(\tau)$, and an additional **let** $x = u$ **in** v construct, of type $\boldsymbol{T}(\tau')$ provided u has type $\boldsymbol{T}(\tau)$ and v has type $\boldsymbol{T}(\tau')$ under the assumption $x : \tau$. Every computational lambda-term has a unique interpretation as a morphism in a CCC with a strong monad. In fact the category $\boldsymbol{Comp}$ whose objects are types and whose morphisms are terms up to $\beta\eta$-conversion is the free CCC-with-a-strong-monad over the set B.

Accordingly, our study will rest on categorical principles. While there is a flurry of generalizations of logical relations (Kripke logical relations [13], lax logical relations [18], pre-logical relations [7], etc.), we use *subscones* [14] as a unifying framework for defining logical relations. Recall that subscones over $\boldsymbol{Set}$ allow us to define logical relations, and subscones over the presheaf category $\boldsymbol{Set}^{\mathcal{I}}$ lead to $\mathcal{I}$-indexed Kripke logical relations [14]. The important property of logical relations is the so-called Basic Lemma [13]: meanings of a lambda-term in different models w.r.t. related environments are related. This is immediate for subscones, and stems from the fact that $\boldsymbol{Comp}$ is the free CCC-with-a-strong-monad on B (a trivial adaptation of Proposition 5.2 in [14]). In particular, that any two closed terms that are in logical relation are observationally equivalent is immediate. Our whole endeavor then reduces to finding appropriate *liftings* of monads on categories $\boldsymbol{C}$ to the subscone category $\mathrm{Subscone}_{\boldsymbol{C}}^{\mathbb{C}}$ (see Section 3).

Outline. We define liftings of monads to *scones* in Section 2; this is simpler than for subscones, and of independent interest. This requires *distributivity laws*, slightly extending [25]. We then lift monads to *sub*scones in Section 3. The important case where the target category $\boldsymbol{C}$ is a product of two categories is investigated in Section 4: this is where *binary* logical relations arise, allowing us to compare two models. We terminate our lifting construction by lifting monad strengths in Section 5. It remains to test the relevance of our construction (Section 6): the logical relations thus defined characterize bisimulations when $\boldsymbol{T}$ is the non-determinism monad (as suggested in [11]), a generalization of Larsen and Skou's [10] probabilistic bisimulations when $\boldsymbol{T}$ is a measure monad [8], and a notion close to Pitts and Stark's logical relations for observational equivalence of programs that create names dynamically [17, 23]. We conclude in Section 7.

Preliminaries. Fix two categories $\boldsymbol{C}$ and $\mathbb{C}$ and a functor $|\_| : \boldsymbol{C} \to \mathbb{C}$. Consider the comma category $(\mathbb{C} \downarrow |\_|)$, whose objects are tuples $\langle S, f, A\rangle$, with $f : S \to |A|$ in $\mathbb{C}$ and whose morphisms are pairs $\langle g, h\rangle : \langle S, f, A\rangle \to \langle S', f', A'\rangle$, $g : S \to S'$ in $\mathbb{C}$ and $h : A \to A'$ in $\boldsymbol{C}$, such that the diagram on the right commutes in $\mathbb{C}$:

$$\begin{array}{ccc} S & \xrightarrow{f} & |A| \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle |h|} \\ S' & \xrightarrow[f']{} & |A'| \end{array} \qquad (1)$$

This category is the *scone of* $\boldsymbol{C}$ *over* $\mathbb{C}$, $\mathrm{Scone}_{\boldsymbol{C}}^{\mathbb{C}}$. The second projection functor $U : (\mathbb{C} \downarrow |\_|) \to \boldsymbol{C}$ maps $\langle S, f, A\rangle$ to A and a morphism $\langle g, h\rangle$ to h.

In the sequel we shall be especially interested in the case where $\mathbb{C} = \boldsymbol{Set}$, and $|\_| = \boldsymbol{C}(1, \_)$ is the *global section* functor, where 1 is terminal in $\boldsymbol{C}$. Another interesting situation arises when $\boldsymbol{C} = \mathbb{C}\times\mathbb{C}$ and $|(A, B)| = A\times B$, assuming that $\mathbb{C}$

has finite products. Objects of the scone then represent binary relations between objects in $\boldsymbol{C}$. In this case, given two functors $|\_|_1 : \boldsymbol{C}_1 \to \mathbb{C}$ and $|\_|_2 : \boldsymbol{C}_2 \to \mathbb{C}$, we may define $|\_| : \boldsymbol{C} \to \mathbb{C}$, for $\boldsymbol{C} = \boldsymbol{C}_1 \times \boldsymbol{C}_2$, by $|\langle A_1, A_2\rangle| = |A_1|_1 \times |A_2|_2$.

Further assume we are given a monad $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu})$ on $\boldsymbol{C}$. When $\boldsymbol{C} = \boldsymbol{C}_1 \times \boldsymbol{C}_2$, the monad $\boldsymbol{T}$ on $\boldsymbol{C}$ will be usually defined pointwise, by two monads $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ on $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$, respectively: $\boldsymbol{T}(\langle A_1, A_2\rangle) = \langle \boldsymbol{T}_1(A_1), \boldsymbol{T}_2(A_2)\rangle$.

Related work. We have already said that there is no unique notion of monad lifting. One of the simplest is the lifting, proposed in [3], $\widetilde{T}$ of $\boldsymbol{T}$, which maps the object $\langle S, f, A\rangle$ of the scone to $\langle S, f; |\boldsymbol{\eta}_A|, \boldsymbol{T}(A)\rangle$. Turi [25] considers lifting monads to the category of coalgebras of a given endofunctor. This is a special case of our framework, when $\boldsymbol{C} = \mathbb{C}$ (and $\boldsymbol{T} = T$) and moreover only objects of the form $S \xrightarrow{f} |S|$ are taken into consideration, and only morphisms of the form $\langle g, g\rangle$. This defines the category of $|\_|$-coalgebras as a proper subcategory of scones. Turi uses a simpler version of the distributivity law: distributivity of a monad over an endofunctor; our law involves two monads and a functor between distinct categories. Neither Pitts nor Turi deal with subscones.

In the same way that we lift a monad to relations, Rutten [22] defines an extension of an endofunctor in ***Set*** to a category of relations. The latter has relations as *morphisms* between sets. An endofunctor extends to relations iff it preserves weak pullbacks, and if so, the extension is unique. The approach taken by Rutten is different from ours, where relations are *objects* rather than morphisms. Hence, Rutten imposes a different functoriality condition: the action of a lifted endofunctor on a composition of two relations must coincide with a composition of actions of the lifted endofunctor on these two relations. This amounts to closedness under composition of relations yielded by the lifted endofunctor.

An approach related to ours is [5], where a *comonad* lifting is defined. This relies on pullbacks, whereas we use mono factorization systems. Nonetheless, the *commutators* of [5] *are* dual to our distributivity laws.

2 Lifting of a Monad to a Scone

By *lifting of a monad* $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu})$ to the scone of $\boldsymbol{C}$ over $\mathbb{C}$ we mean a monad $(\widetilde{T}, \widetilde{\eta}, \widetilde{\mu})$ on $\mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}}$ such that the diagram on the right commutes. That is, $U \circ \widetilde{T} = \boldsymbol{T} \circ U$ and moreover

$$\begin{tikzcd} \mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}} \arrow[r, "\widetilde{T}"] \arrow[d, "U"'] & \mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}} \arrow[d, "U"] \\ \boldsymbol{C} \arrow[r, "\boldsymbol{T}"'] & \boldsymbol{C} \end{tikzcd} \tag{2}$$

$$U\widetilde{\eta} = \boldsymbol{\eta}_U \quad \text{and} \quad U\widetilde{\mu} = \boldsymbol{\mu}_U. \tag{3}$$

By $U\widetilde{\eta}$ and $\boldsymbol{\eta}_U$ we mean the two possible compositions of a natural transformation with U, similarly $U\widetilde{\mu}$ and $\boldsymbol{\mu}_U$. The equations (3) amount to the requirement that the two diagrams on the right commute, for all objects X in $\mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}}$:

$$\begin{tikzcd} & \boldsymbol{T}UX \arrow[d, equal, "(2)"] \\ UX \arrow[ur, "\boldsymbol{\eta}_{UX}"] \arrow[r, "U\widetilde{\eta}_X"'] & U\widetilde{T}X \end{tikzcd} \qquad \begin{tikzcd} & \boldsymbol{T}^2UX \arrow[dl, "\boldsymbol{\mu}_{UX}"'] \arrow[d, equal, "(2)"] \\ \boldsymbol{T}UX \arrow[d, equal, "(2)"] & \boldsymbol{T}U\widetilde{T}X \arrow[d, equal, "(2)"] \\ U\widetilde{T}X & U\widetilde{T}^2X \arrow[l, "U\widetilde{\mu}_X"] \end{tikzcd}$$

In other words, the functor U together with the identity natural transformation is a morphism of monads from $\widetilde{T}$ to $\boldsymbol{T}$. Note that the equations (3) determine the $\boldsymbol{C}$-components of $\widetilde{\eta}$ and $\widetilde{\mu}$ unambiguously. Moreover, diagram (2) determines the $\boldsymbol{C}$-component of the action of $\widetilde{T}$ on objects and morphisms, i.e. $\langle S, f, A\rangle$ is necessarily mapped to $\langle \widetilde{S}, \widetilde{f}, \boldsymbol{T}A\rangle$, for some $\widetilde{S}$, $\widetilde{f}$ and a morphism $\langle g, h\rangle$ is necessarily mapped to $\langle \widetilde{g}, \boldsymbol{T}h\rangle$, for some $\widetilde{g}$.

To be able to give an appropriate lifting we assume another monad (T, η, μ) on $\mathbb{C}$ such that $\boldsymbol{T}$ and T are related by a *distributivity law*, i.e. a natural transformation $\sigma : T|\_| \Rightarrow |\boldsymbol{T}|$ making the two diagrams on the right commute, for each object A in $\boldsymbol{C}$:

$$\begin{tikzcd}
 & T|A| \arrow[d, "\sigma_A"] \\
|A| \arrow[ur, "\eta_{|A|}"] \arrow[r, "|\boldsymbol{\eta}_A|"'] & |\boldsymbol{T}A|
\end{tikzcd}
\qquad
\begin{tikzcd}
 & T^2|A| \arrow[dl, "\mu_{|A|}"'] \arrow[d, "T\sigma_A"] \\
T|A| \arrow[d, "\sigma_A"] & T|\boldsymbol{T}A| \arrow[d, "\sigma_{\boldsymbol{T}A}"] \\
|\boldsymbol{T}A| & |\boldsymbol{T}^2A| \arrow[l, "|\boldsymbol{\mu}_A|"]
\end{tikzcd} \tag{4}$$

Having σ, we define $\widetilde{T}$ on objects by $\langle S, f, A\rangle \mapsto \langle TS, \sigma_X \circ Tf, \boldsymbol{T}A\rangle$ exploiting that if $S \xrightarrow{f} |A|$ then $TS \xrightarrow{Tf} T|A| \xrightarrow{\sigma_A} |\boldsymbol{T}A|$ is an arrow. On morphisms, note that

$$\begin{tikzcd}
TS \arrow[r, "Tf"] \arrow[d, "Tg"'] & T|A| \arrow[r, "\sigma_A"] & |\boldsymbol{T}A| \arrow[d, "|\boldsymbol{T}h|"] \\
TS' \arrow[r, "Tf'"'] & T|A'| \arrow[r, "\sigma_{A'}"'] & |\boldsymbol{T}A'|
\end{tikzcd}
\quad \text{commutes since} \quad
\begin{tikzcd}
S \arrow[r, "f"] \arrow[d, "g"'] & |A| \arrow[d, "|h|"] \\
S' \arrow[r, "f'"'] & |A'|
\end{tikzcd}$$

commutes and σ is natural. So we define $\widetilde{T}$ by $\langle g, h\rangle \mapsto \langle Tg, \boldsymbol{T}h\rangle$. Moreover, we put $\widetilde{\eta}_{\langle S,f,A\rangle} = \langle \eta_S, \boldsymbol{\eta}_A\rangle$ and $\widetilde{\mu}_{\langle S,f,A\rangle} = \langle \mu_S, \boldsymbol{\mu}_A\rangle$.

Checking that this defines a monad is straightforward. First, to check that unit and multiplication are well defined it is sufficient to merge the commuting diagrams (4) and complete them with naturality squares for η and μ as shown on the right.
Unit $\widetilde{\eta}$ and multiplication $\widetilde{\mu}$ are natural since they are defined pointwise and $\boldsymbol{\eta}$, $\boldsymbol{\mu}$, η and μ are. Verifying monad laws is immediate, by the same argument.

$$\begin{tikzcd}
S \arrow[r, "\eta_S"] \arrow[ddd, bend right, "f"'] & TS \arrow[dd, "Tf"'] & T^2S \arrow[l, "\mu_S"'] \arrow[d, "T^2f"] \\
 & & T^2|A| \arrow[dl, "\mu_{|A|}"'] \arrow[d, "T\sigma_A"] \\
 & T|A| \arrow[d, "\sigma_A"] & T|\boldsymbol{T}A| \arrow[d, "\sigma_{\boldsymbol{T}A}"] \\
|A| \arrow[ur, "\eta_{|A|}"] \arrow[r, "|\boldsymbol{\eta}_A|"'] & |\boldsymbol{T}A| & |\boldsymbol{T}^2A| \arrow[l, "|\boldsymbol{\mu}_A|"]
\end{tikzcd}$$

3 Lifting of a Monad to a Subscone

The full subcategory of $\mathrm{Scone}_{\boldsymbol{C}}^{\mathbb{C}}$ consisting of all objects $\langle S, f, A\rangle$ with f a mono, written $S \hookrightarrow^{f} |A|$, we call *the subscone of* $\boldsymbol{C}$ *over* $\mathbb{C}$ and denote by $\mathrm{Subscone}_{\boldsymbol{C}}^{\mathbb{C}}$.
Objects When $\mathbb{C} = \boldsymbol{Set}$ and $|A|$ is given by $\boldsymbol{C}(1, A)$, each object $S \hookrightarrow |A|$ in the subscone represents a subset of global elements of A. In the binary case, i.e. when $\boldsymbol{C} = \boldsymbol{C}_1 \times \boldsymbol{C}_2$ and $|\langle A_1, A_2\rangle| = \boldsymbol{C}_1(1_1, A_1) \times \boldsymbol{C}_2(1_2, A_2)$, $S \hookrightarrow |\langle A_1, A_2\rangle|$

corresponds to a binary relation on global elements of A_1 and A_2—when A_1 and A_2 are the respective denotations of type τ in two given models, this will be the logical relation at type τ.

For technical reasons, we require that $\mathbb{C}$ has a *mono factorization system*. This is essentially an epi-mono factorization [1], except we relax part of the definition: we keep the mono part but do not require the epis in the sequel.

Formally, a mono factorization system is given by two distinguished subclasses of morphisms in $\mathbb{C}$, the so-called *pseudoepis* $\twoheadrightarrow$ and the so-called *relevant monos* $\hookrightarrow$. The latter must be monos, while the former are not required to be epis. Both classes must contain all isomorphisms and be closed under composition with isomorphisms.

Each morphism f in $\mathbb{C}$ must factor as $f = m \circ e$ for some pseudoepi e and some relevant mono m; and each commuting square (5) has a diagonal making both triangles commute as in (6). Note that the diagonal is necessarily unique and that whenever the lower-right triangle commutes, the upper-left triangle does too. Furthermore, the latter property guarantees that the factorization $f = m \circ e$ is unique up to iso.

$$\begin{tikzcd} \cdot \arrow[r, two heads] \arrow[d] & \cdot \arrow[d] \\ \cdot \arrow[r, hook] & \cdot \end{tikzcd} \tag{5}$$

$$\begin{tikzcd} \cdot \arrow[r, two heads] \arrow[d] & \cdot \arrow[d] \arrow[dl, dotted] \\ \cdot \arrow[r, hook] & \cdot \end{tikzcd} \tag{6}$$

Additionally, we assume that functor T preserves pseudoepis, i.e. T maps a pseudoepi to a pseudoepi. This will be needed in diagram (11) below.

Note the following simple and important fact:

Fact 1 *The first component g of a morphism $\langle g, h\rangle$ in a subscone is uniquely determined by the second component h.*

This is because the bottom arrow in (1) is now mono.

Let us define a lifting of the monad to the subscone by analogy with (2) and (3) for the scone. In the binary case mentioned at the beginning of this section, this corresponds to a lifting of a monad to the category of binary relations (as objects) and relation preserving functions (as morphisms).

The lifting $\widetilde{T}$ on objects is given by the mono part of the mono factorization of the lifting of the previous section: $\langle S, f, A\rangle$ is taken to $\langle \widetilde{S}, m, \boldsymbol{T}A\rangle$ given by the diagram on the right:

$$\begin{tikzcd} TS \arrow[r, "Tf"] \arrow[d, "e"'] & T|A| \arrow[d, "\sigma_A"] \\ \widetilde{S} \arrow[r, hook, "m"'] & |\boldsymbol{T}A| \end{tikzcd} \tag{7}$$

Clearly $\widetilde{T}$ is defined only up to iso. Formally, the construction would be unambiguous if we worked with subobjects of $|\boldsymbol{T}A|$, which are determined uniquely.

Given a morphism $\langle g, h\rangle$, the diagram on the right commutes. Then the action of $\widetilde{T}$ on $\langle g, h\rangle$ will be obtained from the unique diagonal guaranteed by (6). We construct diagram (9) below from two copies of (7).

$$\begin{tikzcd} S \arrow[r, hook, "f"] \arrow[dr, "g"'] & {|A|} \arrow[dr, "{|h|}"] & \\ & S' \arrow[r, hook, "f'"'] & {|A'|} \end{tikzcd} \tag{8}$$

All four given faces of the cube commute. Both front and back faces commute by definition of $\widetilde{T}$ on objects: they are copies of diagram (7). The right-hand face is a naturality square of σ; the top face is by application of T to diagram (8), hence commutes by definition of morphisms in the subscone.

$$\begin{tikzcd} TS \arrow[rr, "Tf"] \arrow[dr, "Tg"] \arrow[dd, "e"'] & & T|A| \arrow[dr, "\boldsymbol{T}|h|"] \arrow[dd, "\sigma_A"] & \\ & TS' \arrow[rr, "Tf'"] \arrow[dd, "e'"] & & T|A'| \arrow[dd, "\sigma_{A'}"] \\ \widetilde{S} \arrow[rr, hook, "m"] & & {|\boldsymbol{T}A|} \arrow[dr, "{|\boldsymbol{T}h|}"] & \\ & \widetilde{S}' \arrow[rr, hook, "m'"'] & & {|\boldsymbol{T}A'|} \end{tikzcd} \tag{9}$$

Now, an instance of diagram (5) can be found in (9) by two walks from TS to $|\boldsymbol{T}A'|$: one starts with the pseudoepi $TS \overset{e}{\twoheadrightarrow} \widetilde{S}$, the other ends with the relevant mono $\widetilde{S}' \overset{m'}{\hookrightarrow} |\boldsymbol{T}A'|$. Since all faces commute, there is an arrow $\widetilde{S} \dashrightarrow \widetilde{S}'$ as in diagram (6), making the two newly created faces of the cube commute. This arrow is unique by Fact 1. Now $\widetilde{T}\langle g, h\rangle$ is given by the bottom face. Functoriality follows immediately from uniqueness of the diagonal arrow in (6).

The ($\mathbb{C}$-component of the) unit $\widetilde{\eta}_{\langle S,f,A\rangle}$ is defined by post-composing η_S with the pseudoepi part of the mono factorization in (7). This is well defined since everything in sight in the diagram on the right commutes. Indeed, the right triangle is one of the distributivity law diagrams,

$$\begin{tikzcd} S \arrow[rr, hook, "f"] \arrow[d, "\eta_S"'] & & {|A|} \arrow[dl, "\eta_{|A|}"'] \arrow[dd, "{|\boldsymbol{\eta}_A|}"] \\ TS \arrow[r, "Tf"] \arrow[d, "e"'] & T|A| \arrow[dr, "\sigma_A"] & \\ \widetilde{S} \arrow[rr, hook, "m"'] & & {|\boldsymbol{T}A|} \end{tikzcd} \tag{10}$$

the upper square is the naturality of η while the lower one is a copy of (7). The ($\mathbb{C}$-component of the) multiplication $\widetilde{\mu}_{\langle S,f,A\rangle}$ will be induced by a diagram similar to (9) (below).

Again, all the faces not having the required dotted arrow as edge commute. The front face and the lower half of the back face are instances of (7), defining $\widetilde{T}\langle S, f, A\rangle$ and $\widetilde{T}^2\langle S, f, A\rangle$, respectively. The upper half of the back face is by application of T to the front face. The right-hand face is the other distributivity law diagram, which we had not used yet, while the upper one is a naturality square for μ.

$$\begin{tikzcd} T^2S \arrow[rr, "T^2f"] \arrow[d, "Te"'] \arrow[ddr, "\mu_S"] & & T^2|A| \arrow[d, "T\sigma_A"'] \arrow[ddr, "\mu_{|A|}"] & \\ T\widetilde{S} \arrow[rr, "Tm"] \arrow[dd, "\widetilde{e}"'] \arrow[ddr, dashed] & & T|\boldsymbol{T}A| \arrow[dd, "\sigma_{\boldsymbol{T}A}"] & \\ & TS \arrow[rr, "Tf"] \arrow[dd, "e"] & & T|A| \arrow[dd, "\sigma_A"] \\ \widetilde{\widetilde{S}} \arrow[rr, hook, "\widetilde{m}"] \arrow[dr, dotted, "!"'] & & {|\boldsymbol{T}^2A|} \arrow[dr, "{|\boldsymbol{\mu}_A|}"] & \\ & \widetilde{S} \arrow[rr, hook, "m"'] & & {|\boldsymbol{T}A|} \end{tikzcd} \tag{11}$$

Note that Te is a pseudoepi, since T preserves pseudoepis. Composition $\widetilde{e} \circ Te$ is not necessarily a pseudoepi, hence we will need a diagonal (6) twice. First, similarly as in diagram (9) we find an instance of diagram (5) by two walks from T^2S to $|\boldsymbol{T}A|$, one starting with Te and the other ending with m. Hence, the unique dashed arrow exists and makes the two triangles commute. One of them, involving the pseudoepi Te, is the upper part of the left-hand side. The other one, namely that involving the relevant mono m, allows us to apply (5) again, since the following two walks from $T\widetilde{S}$ to $|\boldsymbol{T}A|$ commute: one starting with the pseudoepi $\widetilde{e}$ and the other consisting of the dashed arrow followed by m. Hence, the unique dotted arrow exists and makes the bottom face as well as the triangle in the left-hand face commute. The multiplication $\widetilde{\mu}_{\langle S,f,A\rangle}$ is then defined by the bottom face of the cube.

Verification of the monad laws is a formality due to the following:

Fact 2 *Given two parallel arrows in* $\mathrm{Subscone}_{\mathbb{C}}^{\boldsymbol{C}}$*, say* $\langle g_1, h_1\rangle$ *and* $\langle g_2, h_2\rangle$*, they are equal whenever the second components* h_1 *and* h_2 *are.*

The proof is immediate by Fact 1. Using this fact, and knowing that second components of $\widetilde{\eta}$ and $\widetilde{\mu}$ satisfy the monad laws (as they are unit and multiplication of $\boldsymbol{T}$, respectively), we deduce immediately that $\widetilde{\eta}$ and $\widetilde{\mu}$ satisfy the monad laws too. Similarly one proves naturality of $\widetilde{\eta}$ and $\widetilde{\mu}$.

It is useful to summarize the ingredients we have used here. To lift a monad $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu})$ on $\boldsymbol{C}$ to $\mathrm{Subscone}_{\boldsymbol{C}}^{\mathbb{C}}$, we need:

(i) a category $\mathbb{C}$ and a functor $\lvert\_\rvert : \boldsymbol{C} \to \mathbb{C}$,	
(ii) a monad (T, η, μ) on $\mathbb{C}$, related to $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu})$ by a distributivity law σ,	
(iii) a mono factorization system on $\mathbb{C}$,	**(iv)** T preserves pseudoepis.

Recall that to lift the CCC structure of $\boldsymbol{C}$ to the subscone, we additionally require $\mathbb{C}$ to be a CCC with pullbacks, and $\lvert\_\rvert$ to preserve finite products [14]. Description of the construction can be found eg. in [5], Section 5.4.

4 Lifting of a Monad to Relations

Recall that we would like to lift monads to categories of binary relations as objects. Hence, assume in this section that $\boldsymbol{C}$ is a product category, $\boldsymbol{C} = \boldsymbol{C}_1 \times \boldsymbol{C}_2$ and that both $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are equipped with monads $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$, and functors $\lvert\_\rvert_1 : \boldsymbol{C}_1 \to \mathbb{C}$ and $\lvert\_\rvert_2 : \boldsymbol{C}_2 \to \mathbb{C}$. A monad $\boldsymbol{T}$ on $\boldsymbol{C}$ can be defined pairwise: $\boldsymbol{T}\langle A_1, A_2\rangle = \langle \boldsymbol{T}_1 A_1, \boldsymbol{T}_2 A_2\rangle$ and similarly we define $\lvert\_\rvert : \boldsymbol{C} \to \mathbb{C}$ by $|\langle A_1, A_2\rangle| = |A_1|_1 \times |A_2|_2$—to this aim we assume binary products in $\mathbb{C}$.

In the same vein distributivity laws for $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ induce a distributivity law for $\boldsymbol{C}$. Assumed two distributivity laws, $\sigma^1 : T|\_|_1 \Rightarrow |\boldsymbol{T}_1|_1$ and $\sigma^2 : T|\_|_2 \Rightarrow |\boldsymbol{T}_2|_2$, we can define $\sigma_{\langle A_1, A_2\rangle} : T(|A_1|_1 \times |A_2|_2) \to |\boldsymbol{T}_1 A_1|_1 \times |\boldsymbol{T}_2 A_2|_2$ by

$$\sigma_{\langle A_1, A_2\rangle} = \langle T\pi_1; \sigma^1_{A_1}, T\pi_2; \sigma^2_{A_2}.\rangle \tag{12}$$

where π_1 and π_2 denote the projections from $|A_1|_1 \times |A_2|_2$.

The situation gets much simpler when $\mathbb{C} = \boldsymbol{Set}$, $|\_|_1 = \boldsymbol{C}_1(1_1, \_)$ and $|\_|_2 = \boldsymbol{C}_2(1_2, \_)$. We assume that $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ have terminal objects, 1_1 and 1_2 respectively. Each object $S \hookrightarrow |\langle A_1, A_2\rangle|$ in the subscone defines a binary relation (again noted S) on global elements of A_1 and A_2. Obviously $\boldsymbol{Set}$ satisfies all requirements from previous sections, with surjections as pseudoepis and injections as relevant monos.

Given two CCCs $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ with respective strong monads $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$, the fact that $\boldsymbol{Comp}$ is the free CCC with strong monad on the set B of base types means that there are two representations of CCCs-with-strong-monads, $[\![\_]\!]_1$ and $[\![\_]\!]_2$, from $\boldsymbol{Comp}$ to $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ respectively: they are the natural meaning functions for monadic types and computational λ-terms.

Our construction of a lifting together with standard constructions on subscones [14] yield another representation of CCCs-with-strong-monads $[\![\_]\!]$ from $\boldsymbol{Comp}$ to $\mathrm{Subscone}^{\boldsymbol{Set}}_{\boldsymbol{C}_1\times\boldsymbol{C}_2}$. That $[\![\_]\!]$ is a lifting means that $U \circ [\![\_]\!] = \langle [\![\_]\!]_1, [\![\_]\!]_2\rangle$, i.e., the diagram on the right commutes. When $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ are concrete categories, this means that

$$\begin{tikzcd} & \mathrm{Subscone}^{\boldsymbol{Set}}_{\boldsymbol{C}_1\times\boldsymbol{C}_2} \arrow[d, "U"] \\ \boldsymbol{Comp} \arrow[ur, "{[\![\_]\!]}"] \arrow[r, "{\langle [\![\_]\!]_1, [\![\_]\!]_2\rangle}"'] & \boldsymbol{C}_1 \times \boldsymbol{C}_2 \end{tikzcd}$$

$$\forall a_1 \in [\![\Gamma]\!]_1, a_2 \in [\![\Gamma]\!]_2 . (a_1, a_2) \in [\![\Gamma]\!] \Rightarrow ([\![t]\!]_1(a_1), [\![t]\!]_2(a_2)) \in [\![\tau]\!] \tag{13}$$

for all terms t of type τ in the context $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$; representations of Γ are taken to be products of the representations of $\tau_1, \ldots, \tau_n$; $[\![\tau]\!]$ is a relation between $[\![\tau]\!]_1$ and $[\![\tau]\!]_2$, defined by induction on types τ (the case where τ is a base type is arbitrary):

$$\begin{gathered} (f_1, f_2) \in [\![\tau \to \tau']\!] \iff \forall (a_1, a_2) \in [\![\tau]\!] . (f_1(a_1), f_2(a_2)) \in [\![\tau']\!] \\ ((a_1, a_1'), (a_2, a_2')) \in [\![\tau \times \tau']\!] \iff (a_1, a_2) \in [\![\tau]\!] \wedge (a_1', a_2') \in [\![\tau']\!] \\ (B_1, B_2) \in [\![\boldsymbol{T}\tau]\!] \iff (B_1, B_2) \in \widetilde{T}[\![\tau]\!] \end{gathered}$$

These equations (except possibly the last one) are the standard definition of a *logical relation*. (13) is the already cited *Basic Lemma*.

Further simplification is gained when $\boldsymbol{C}_1 = \boldsymbol{C}_2 = \boldsymbol{Set}$, the three monads $\boldsymbol{T}_1$, $\boldsymbol{T}_2$ and T are identical and both $|\_|_1$ and $|\_|_2$ are identity functors. The distributivity law reduces to distributivity of the monad T over binary product, and (12) rewrites to $\sigma_{(A_1,A_2)} = \langle T\pi_1, T\pi_2\rangle : T(A_1 \times A_2) \to TA_1 \times TA_2$, where by T we denote a given single monad on $\boldsymbol{Set}$. This is a particularly interesting special case, so we study it in more detail.

Every binary relation $S \subseteq A_1 \times A_2$ has a representation $S \xhookrightarrow{\langle \pi^S{}_1, \pi^S{}_2\rangle} A_1 \times A_2$ where the arrow is the inclusion induced by two projections $\pi^S{}_1 : S \to A_1$ and $\pi^S{}_2 : S \to A_2$. In fact, the full subcategory of subscone consisting exclusively of inclusions instead of all injections is equivalent to the whole subscone, so without loss of generality we consider only inclusions in the rest of this section.

Recall the action of a lifted monad $\widetilde{T}$ on a relation $S \overset{\langle \pi^S{}_1, \pi^S{}_2 \rangle}{\rightarrowtail} A_1 \times A_2$:

$$\begin{tikzcd} TS \arrow[r, "T\langle \pi^S{}_1, \pi^S{}_2 \rangle"] \arrow[d] & T(A_1 \times A_2) \arrow[d, "\sigma_{(A_1, A_2)}"] \\ \widetilde{S} \arrow[r, tail] & TA_1 \times TA_2 \end{tikzcd}$$

The functor $\widetilde{T}$ maps a relation S to the relation between sets TA_1 and TA_2 defined as the direct image of the function $\langle T\pi^S{}_1, T\pi^S{}_2 \rangle : TS \to TA_1 \times TA_2$, since $\langle T\pi^S{}_1, T\pi^S{}_2 \rangle = \langle T\pi_1, T\pi_2 \rangle \circ T\langle \pi^S{}_1, \pi^S{}_2 \rangle$.

5 Lifting of a Strong Monad to a Scone and a Subscone

In this section we assume that both categories $\boldsymbol{C}$ and $\mathbb{C}$ have finite products given explicitly. By the same symbol 1 we denote the terminal object both in $\boldsymbol{C}$ and $\mathbb{C}$. Moreover we assume that natural isomorphisms are given:

$$\begin{array}{rrl} \boldsymbol{r}_A : 1 \times A \to A & \boldsymbol{\alpha}_{A,B,C} : (A \times B) \times C \to A \times (B \times C) & \text{in } \boldsymbol{C} \\ r_S : 1 \times S \to S & \alpha_{S,R,Q} : (S \times R) \times Q \to S \times (R \times Q) & \text{in } \mathbb{C} \end{array} \tag{14}$$

and that $|\_|$ preserves strictly finite products as well as $\boldsymbol{r}$ and $\boldsymbol{\alpha}$, i.e., $|A \times B| = |A| \times |B|$, $|\boldsymbol{r}_A| = r_{|A|}$, $|\boldsymbol{\alpha}_{A,B,C}| = \alpha_{|A|,|B|,|C|}$. We will also need an assumption that products preserve pseudoepis, i.e., if f and g are pseudoepis then $f \times g$ is pseudoepi too. Furthermore we assume that two monads $\boldsymbol{T}$ and T are *strong*, i.e., a *strength* natural transformation $\boldsymbol{t}_{A,B} : A \times \boldsymbol{T}B \to \boldsymbol{T}(A \times B)$ is given such that the diagrams in Definition 3.2 in [16] commute and analogously a strength $t_{S,R} : S \times TR \to T(S \times R)$ for T is assumed.

Our definition of monad lifting $\boldsymbol{T}$ to scones (or subscones) in diagram (2) and equations (3) will be extended to strong monads below. First observe that we do have finite products and natural isomorphisms $\widetilde{r}$ and $\widetilde{\alpha}$ in scones and subscones, if we put $\widetilde{r}_{\langle S,f,A \rangle} = \langle r_S, \boldsymbol{r}_A \rangle$ and $\widetilde{\alpha}_{\langle S,f,A \rangle, \langle R,g,B \rangle, \langle Q,h,C \rangle} = \langle \alpha_{S,R,Q}, \boldsymbol{\alpha}_{A,B,C} \rangle$.

Lemma 1. *Both* $\mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}}$ *and* $\mathrm{Subscone}^{\mathbb{C}}_{\boldsymbol{C}}$ *have finite products, given explicitly. Functor U preserves strictly finite products as well as $\widetilde{r}$ and $\widetilde{\alpha}$.* □

By lifting of $\boldsymbol{T}$ to $\mathrm{Scone}^{\mathbb{C}}_{\boldsymbol{C}}$ we now mean a strong monad, i.e. a monad $(\widetilde{T}, \widetilde{\eta}, \widetilde{\mu})$ together with a strength $\widetilde{t}_{X,Y} : X \times \widetilde{T}Y \to \widetilde{T}(X \times Y)$, such that diagram (2) commutes, equations (3) hold and $U\widetilde{t}_{X,Y} = \boldsymbol{t}_{UX,UY}$, i.e., U preserves strength. To be able to give an appropriate lifting, we extend distributivity law (4) by one more condition, relating strengths $\boldsymbol{t}_{A,B}$ and $t_{|A|,|B|}$:

$$\begin{tikzcd} |A| \times T|B| \arrow[r, "\mathrm{id}_{|A|} \times \sigma_B"] \arrow[d, "t_{|A|,|B|}"] & |A| \times |\boldsymbol{T}B| \arrow[r, equal] & |A \times \boldsymbol{T}B| \arrow[d, "|\boldsymbol{t}_{A,B}|"] \\ T(|A| \times |B|) \arrow[r, equal] & T|A \times B| \arrow[r, "\sigma_{A \times B}"] & |\boldsymbol{T}(A \times B)| \end{tikzcd} \tag{15}$$

Having lifted $\boldsymbol{T}$ to scones and subscones in previous sections, we only need to give a lifting of the strength $\boldsymbol{t}$. For scones this is straightforward—define $\widetilde{t}$ pointwise by $\widetilde{t}_{\langle S,f,A \rangle, \langle R,g,B \rangle} := \langle t_{S,R}, \boldsymbol{t}_{A,B} \rangle$.

Verifying that this is well-defined amounts to pasting together a naturality square for t and a diagram (15):

$$\begin{tikzcd}
S\times TR \arrow[r, "f\times Tg"] \arrow[d, "t_{S,R}"] & |A|\times T|B| \arrow[r, "\mathrm{id}_{|A|}\times\sigma_B"] \arrow[d, "t_{|A|,|B|}"'] & |A\times \boldsymbol{T}B| \arrow[d, "|\boldsymbol{t}_{A,B}|"'] \\
T(S\times R) \arrow[r, "T(f\times g)"] & T(|A|\times|B|) \arrow[r, "\sigma_{A\times B}"] & |\boldsymbol{T}(A\times B)|
\end{tikzcd}$$

Note that the upper side of this diagram is precisely $\langle S,f,A\rangle\times\widetilde{T}\langle R,g,B\rangle$ in the scone while the lower side is $\widetilde{T}(\langle S,f,A\rangle\times\langle R,g,B\rangle)$. Checking naturality of $\widetilde{t}$ and strength laws is immediate since $\widetilde{t}$, $\widetilde{\alpha}$, $\widetilde{r}$, $\widetilde{\eta}$ and $\widetilde{\mu}$ are all defined pointwise.

Now we move to subscones. Call $\widetilde{T}$ the lifted monad defined in (7) and (9). As in previous sections, $\widetilde{t}$ in subscones will differ from the case of scones only in its $\mathbb{C}$-component, and this component will be induced as a unique diagonal guaranteed by diagram (6) in the diagram on the right:

$$\begin{tikzcd}
S\times TR \arrow[rr, "f\times Tg"] \arrow[dr, "t_{S,R}"] \arrow[dd, "\mathrm{id}_S\times e"'] & & |A|\times T|B| \arrow[dr, "t_{|A|,|B|}"] \arrow[dd, "\mathrm{id}_{|_|A}\times\sigma_B"] & \\
& T(S\times R) \arrow[rr, "T(f\times g)"] \arrow[dd] & & T(|A|\times|B|) \arrow[d, equal] \\
S\times\widetilde{R} \arrow[rr, hook, "f\times m"] \arrow[dr, dotted] & & |A|\times|\boldsymbol{T}B| \arrow[d, equal] & T|A\times B| \arrow[dd, "\sigma_{A\times B}"] \\
& & |A\times\boldsymbol{T}B| \arrow[dr, "|\boldsymbol{t}_{A,B}|"] & \\
& \cdot \arrow[rr, "n"] & & |\boldsymbol{T}(A\times B)|
\end{tikzcd} \tag{16}$$

As ingredients of this diagram we have used instances of diagram (7) for $\widetilde{T}\langle R,g,B\rangle$ and $\widetilde{T}(\langle S,f,A\rangle\times\langle R,g,B\rangle)$:

$$\begin{tikzcd}
TR \arrow[r, "Tg"] \arrow[d, "e"'] & T|B| \arrow[d, "\sigma_B"] \\
\widetilde{R} \arrow[r, "m"] & |\boldsymbol{T}B|
\end{tikzcd}
\qquad
\begin{tikzcd}
T(S\times R) \arrow[r, "T(f\times g)"] \arrow[d] & T(|A|\times|B|) \arrow[r, equal] & T|A\times B| \arrow[d, "\sigma_A"] \\
\cdot \arrow[rr, "n"] & & |\boldsymbol{T}(A\times B)|
\end{tikzcd}$$

The right diagram is the front face of (16), while the product of f with the left diagram generates the back face. The upper face of (16) is a naturality square for t and the right-hand face is exactly diagram (15). Note that $\mathrm{id}_{|\_|A}\times\sigma_B$ is marked as pseudoepi due to our assumption that products preserve pseudoepis. Since the bottom (slightly deformed) face commutes, $\widetilde{t}$ is well-defined. And again, checking naturality of $\widetilde{t}$ and strength laws is immediate by Fact 2.

Here is the final set of ingredients for lifting a strong monad $(\boldsymbol{T},\boldsymbol{\eta},\boldsymbol{\mu},\boldsymbol{r},\boldsymbol{\alpha},\boldsymbol{t})$ on category $\boldsymbol{C}$ with explicitly given finite products, to $\mathrm{Subscone}_{\boldsymbol{C}}^{\mathbb{C}}$:

(i) a category $\mathbb{C}$ with explicitly given finite products and natural isos r and α,
(ii) a functor $|\_| : \boldsymbol{C}\to\mathbb{C}$, preserving finite products, $\boldsymbol{r}$ and $\boldsymbol{\alpha}$, strictly,
(iii) a strong monad (T,η,μ,r,α,t) on $\mathbb{C}$, related to $(\boldsymbol{T},\boldsymbol{\eta},\boldsymbol{\mu},\boldsymbol{r},\boldsymbol{\alpha},\boldsymbol{t})$ by a distributivity law σ defined in (4) and (15),
(iv) a mono factorization system on $\mathbb{C}$,
(v) pseudoepis are preserved by T as well as by finite products.

5.1 Building Distributivity Laws from Adjunctions

It is often the case that we have a (strong) monad on $\boldsymbol{C}$, and wish to build another one on $\mathbb{C}$ related to the latter by a distributivity law. The following results are then of some help. For lack of space, proofs are omitted and can be found in the full version [6].

Proposition 1. *Let $\boldsymbol{C}$ and $\mathbb{C}$ be two categories with explicitly given finite products and with natural isomorphisms $\boldsymbol{r}$, $\boldsymbol{\alpha}$ in $\boldsymbol{C}$ and r, α in $\mathbb{C}$ as in (14). Let $|\_| : \boldsymbol{C} \to \mathbb{C}$ be a functor with a left adjoint $\boldsymbol{D} : \mathbb{C} \to \boldsymbol{C}$. Assume that both functors strictly preserve finite products and the natural isomorphisms: $|\boldsymbol{r}_A| = r_{|A|}$, $|\boldsymbol{\alpha}_{A,B,C}| = \alpha_{|A|,|B|,|C|}$, $\boldsymbol{D}(r_E) = \boldsymbol{r}_{\boldsymbol{D}E}$ and $\boldsymbol{D}(\alpha_{E,F,G}) = \boldsymbol{\alpha}_{\boldsymbol{D}E,\boldsymbol{D}F,\boldsymbol{D}G}$. Furthermore assume that the adjunction preserves products: $\dot{\eta}_{E\times F} = \dot{\eta}_E \times \dot{\eta}_F$ (and hence $\dot{\epsilon}_{A\times B} = \dot{\epsilon}_A \times \dot{\epsilon}_B$), where $\dot{\eta}_E : E \to |\boldsymbol{D}(E)|$ is the unit of the adjunction and $\dot{\epsilon}_A : \boldsymbol{D}|A| \to A$ is the counit. Let $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{r}, \boldsymbol{\alpha}, \boldsymbol{t})$ be a strong monad on $\boldsymbol{C}$.*

Define $T = |\_| \circ \boldsymbol{T} \circ \boldsymbol{D}$, $\eta_E = |\boldsymbol{\eta}_{\boldsymbol{D}(E)}| \circ \dot{\eta}_E$, $\mu_E = |\boldsymbol{\mu}_{\boldsymbol{D}(E)} \circ \boldsymbol{T}\dot{\epsilon}_{\boldsymbol{T}\boldsymbol{D}(E)}|$, and $t_{E,F} = |\boldsymbol{t}_{\boldsymbol{D}(E),\boldsymbol{D}(F)}| \circ (\dot{\eta}_E \times \mathrm{id}_{TF})$. Finally, let $\sigma_A = |\boldsymbol{T}\dot{\epsilon}_A| : T|A| \to |\boldsymbol{T}A|$. Then $(T, \eta, \mu, r, \alpha, t)$ is a strong monad on $\mathbb{C}$ and σ is a distributivity law of strong monads from $T|\_|$ to $|\boldsymbol{T}\_|$.

Proposition 1 is surprisingly powerful: in each of the the examples given in the following section, a distributivity law that induces the relevant lifting can be obtained by Proposition 1.

Any category with explicitly given finite products offers a standard choice for $\boldsymbol{r}$ and $\boldsymbol{\alpha}$: let $\boldsymbol{r}_A : 1 \times A \to A$ be the second projection π_2, and $\boldsymbol{\alpha} : (A \times B) \times C \to A \times (B \times C)$ be $\langle \pi_1 \circ \pi_1, \langle \pi_2 \circ \pi_1, \pi_2 \rangle\rangle$. Call these isomorphisms *standard*.

Corollary 1. *Let $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{r}, \boldsymbol{\alpha}, \boldsymbol{t})$ be a strong monad on a category $\boldsymbol{C}$ with explicit finite products, $|\_| : \boldsymbol{C} \to \mathbb{C}$, with left adjoint $\boldsymbol{D} : \mathbb{C} \to \boldsymbol{C}$, where $\boldsymbol{r}$ and $\boldsymbol{\alpha}$ are standard, and $\mathbb{C}$ has explicit finite products. Assume that $|\_|$ and $\boldsymbol{D}$ preserve finite products strictly, and $|\boldsymbol{D}|$ is the identity functor on $\mathbb{C}$ and $\dot{\eta}_E = \mathrm{id}_E$.*

Let T, η, μ, σ be as in Proposition 1 , and $t_{E,F} = |\boldsymbol{t}_{\boldsymbol{D}(E),\boldsymbol{D}(F)}|$.

Then $(T, \eta, \mu, r, \alpha, t)$ is a strong monad on $\mathbb{C}$, where r and α are standard, and σ is a distributivity law of strong monads from $T|\_|$ to $|\boldsymbol{T}\_|$.

It's only for simplicity that we have assumed strict preservation of products in this section – in fact all the development can be done when products are preserved only up to iso, see [6].

6 Examples

As in Section 4, suppose $\mathbf{C}_1 = \mathbf{C}_2 = \mathbb{C} = \boldsymbol{Set}$, and $|\_|_1$ and $|\_|_2 = |\_|$ are identities. Below we summarize the action of $\widetilde{T}$ on a relation $S \longrightarrow A_1 \times A_2$, for different computational monads T of Moggi [16]. This is parameterized by a binary relation R_{St} on states in the state monad $(A \times \mathrm{St})^{\mathrm{St}}$, and by a binary relation $R_{\mathcal{R}}$ in the continuation monad $\mathcal{R}^{\mathcal{R}^A}$.

Monad T	relation $\widetilde{S} \subseteq TA_1 \times TA_2$
$TA = A_\perp = A \cup \{\perp\}$	$\widetilde{S} = S \cup \{\perp, \perp\}$
$TA = (A \times \mathrm{St})^{\mathrm{St}}$	$(f,g) \in \widetilde{S} \iff \forall s_1, s_2 \in \mathrm{St}.$ $(s_1, s_2) \in \mathrm{St} \Rightarrow (\pi_1(fs_1), \pi_1(gs_2)) \in S \wedge (\pi_2(fs_1), \pi_2(gs_2)) \in R_{\mathrm{St}}$
$TA = \mathbb{P}_{\mathrm{fin}}(A)$	$(B_1, B_2) \in \widetilde{S} \iff$ $\forall b_1 \in B_1. \exists b_2 \in B_2. (b_1, b_2) \in S \wedge$ $\forall b_2 \in B_2. \exists b_1 \in B_1. (b_1, b_2) \in S$
$TA = \mathcal{R}^{\mathcal{R}^A}$	$(\alpha_1, \alpha_2) \in \widetilde{S} \iff \big($ $\forall f_1, f_2. (\forall a_1, a_2. (a_1, a_2) \in S \Rightarrow (f_1(a_1), f_2(a_2)) \in R_{\mathcal{R}}) \Rightarrow$ $(\alpha_1(f_1), \alpha_2(f_2)) \in R_{\mathcal{R}} \big)$

Our construction in the case of the finite powerset monad $\mathbb{P}_{\mathrm{fin}}()$ in fact expands to: $(B_1, B_2) \in \widetilde{S}$ iff $B_1 = \{x | (x,y) \in R\}$ and $B_2 = \{y | (x,y) \in R\}$ for some $R \subseteq S$. (Recall that $\widetilde{T}$ maps relations S to the direct image $\widetilde{S}$ of $\langle T\pi_1, T\pi_2 \rangle : TS \to TA_1 \times TA_2$, see the end of Section 4.) This is equivalent to the condition given above, which is the more usual way of defining bisimulations.

Indeed, if $B_1 = \{x | (x,y) \in R\}$ and $B_2 = \{y | (x,y) \in R\}$ for some $R \subseteq S$ then for every $b_1 \in B_1$ by construction there is some $b_2 \in B_2$ such that $(b_1, b_2) \in R$, therefore $(b_1, b_2) \in S$ since $R \subseteq S$, and symmetrically for every $b_2 \in B_2$ there is some $b_1 \in B_1$ such that $(b_1, b_2) \in S$: B_1 and B_2 are bisimilar.

Conversely, if B_1 and B_2 are bisimilar (in the sense just given), then let R be the restriction of S to $B_1 \times B_2$. For every $b_1 \in B_1$, by bisimilarity there is some $b_2 \in B_2$ such that $(b_1, b_2) \in S$, so $(b_1, b_2) \in R$, therefore $b_1 \in \{x | (x,y) \in R\}$; so $B_1 \subseteq \{x | (x,y) \in R\}$. The reverse inclusion is obvious, so $B_1 = \{x | (x,y) \in R\}$. The other equality $B_2 = \{y | (x,y) \in R\}$ is by symmetry.

That logical relations on powersets define bisimulations was conjectured in [11] and, for pre-logical relations, in [7].

6.1 Labelled Transition Systems and Bisimulations

The case $TA = \mathbb{P}_{\mathrm{fin}}(A)$ defines labelled transition systems as elements of $(TA)^{A \times L}$, with labels in L and states in A, as functions mapping states a and labels ℓ to the set of states a' such that $a \xrightarrow{\ell} a'$. Our monad lifting $\widetilde{S}$ in this case is parameterized by a binary relation on R_L on labels and is defined by:

$$(f_1, f_2) \in \widetilde{S} \iff (\forall a_1, a_2, \ell_1, \ell_2 \cdot (a_1, a_2) \in S \wedge (\ell_1, \ell_2) \in R_L \Rightarrow \left\{ \begin{array}{l} \forall b_1 \in f_1(a_1, \ell_1). \exists b_2 \in f_2(a_2, \ell_2). (b_1, b_2) \in S \wedge \\ \forall b_2 \in f_2(a_2, \ell_2). \exists b_1 \in f_1(a_1, \ell_1). (b_1, b_2) \in S \end{array} \right\})$$

In case R_L is the equality relation, the relation $\widetilde{S}$ relates f_1 and f_2 iff S is a *strong bisimulation* between the labelled transition systems f_1 and f_2.

6.2 Logical Relations for Dynamic Name Creation

Consider Moggi's model of dynamic name creation [15]. Let $\mathcal{I}$ be the category of finite sets and injective functions, and $\boldsymbol{Set}^{\mathcal{I}}$ be the category of functors from

$\mathcal{I}$ to $\boldsymbol{Set}$ and natural transformations (the category of *covariant presheaves* over $\mathcal{I}$). For short, write $\boldsymbol{T}As$ for $\boldsymbol{T}(A)(s)$ and similarly for other notations. Let $+$ denote coproduct in $\mathcal{I}$.

We define the strong monad $\boldsymbol{T}$ on $\boldsymbol{Set}^{\mathcal{I}}$ as follows. $\boldsymbol{T}A = \mathrm{colim}_{s'}\, A(\_ + s') : \mathcal{I} \to \boldsymbol{Set}$. On objects, this is given by $\boldsymbol{T}As = \mathrm{colim}_{s'}\, A(s + s')$, i.e., $\boldsymbol{T}As$ is the set of all equivalence classes of pairs (s', a) with $s' \in \mathcal{I}$ and $a \in A(s+s')$ modulo the smallest equivalence relation $\equiv$ such that $(s', a) \equiv (s'', A(\mathrm{id}_s + j)a)$ for every morphism $s' \xrightarrow{j} s''$ in $\mathcal{I}$ (intuitively, given a set of *names* s, elements of $\boldsymbol{T}As$ are formal expressions $(\nu s')a$ where all names in s' are bound and every name free in a is in $s+s'$—modulo the fact that $(\nu s', s'')a \equiv (\nu s')a$ for any additional set of new names s'' not free in a). On morphisms $s_1 \xrightarrow{i} s_2$, $\boldsymbol{T}Ai$ maps the equivalence class of (s', a) to the equivalence class of $(s', A(i + \mathrm{id}_{s'})a)$.

It is important to note how $\equiv$ works. The category $\mathcal{I}$ has pushouts: in particular, if $s_0 \xrightarrow{i_1} s_1$ and $s_0 \xrightarrow{i_2} s_2$ are two morphisms in $\mathcal{I}$, then there is a finite set $s_1 +_{s_0} s_2$ and two morphisms $s_1 \xrightarrow{j_1} s_1 +_{s_0} s_2$, $s_2 \xrightarrow{j_2} s_1 +_{s_0} s_2$ such that $i_1; j_1 = i_2; j_2$—take $s_1 +_{s_0} s_2$ to be the disjoint sum $s_1 + s_2$ modulo the equivalence relation relating $i_1(a_0) = i_2(a_0)$ for every $a_0 \in s_0$. It follows that for every $a_1 \in A(s + s_1)$, $a_2 \in A(s + s_2)$, $(s_1, a_1) \equiv (s_2, a_2)$ if and only if there is a finite set s_{12} and two arrows $s_1 \xrightarrow{j_1} s_{12}$ and $s_2 \xrightarrow{j_2} s_{12}$ such that $A(\mathrm{id}_{s_1} + j_1)a_1 = A(\mathrm{id}_{s_2} + j_2)a_2$.

We take $\boldsymbol{C}_1 = \boldsymbol{C}_2 = \mathbb{C} = \boldsymbol{Set}^{\mathcal{I}}$, hence objects in the subscone give raise to the $\mathcal{I}$-indexed *Kripke logical relations*. Furthermore, $|\_|_1 = |\_|_2 = |\_|$ is the identity functor and T is just $\boldsymbol{T}$. Category $\boldsymbol{Set}^{\mathcal{I}}$ has a mono factorization consisting of pointwise surjections and pointwise injections.

As in Section 4, the distributivity law $\sigma_{\langle A_1, A_2\rangle} s : T(A_1 \times A_2)s \to \boldsymbol{T}A_1 s \times \boldsymbol{T}A_2 s$ is equal to $\langle T\pi_1, T\pi_2\rangle s$. $\widetilde{S}s \hookrightarrow \boldsymbol{T}A_1 s \times \boldsymbol{T}A_2 s$ is thus given by

$$(s_1, a_1)\ \widetilde{S}s\ (s_2, a_2) \iff \begin{array}{l} \exists s_0 \in \mathcal{I} \cdot \exists i_1 : s_1 \to s_0 \in \mathcal{I} \cdot \exists i_2 : s_2 \to s_0 \in \mathcal{I} \cdot \\ (A_1(\mathrm{id}_s + i_1)a_1)\ S(s + s_0)\ (A_2(\mathrm{id}_s + i_2)a_2) \end{array} \tag{17}$$

where $a_1 \in A_1(s + s_1)$ and $a_2 \in A_2(s + s_2)$.

This is similar to the logical relations of [17, 23]. While (17) is roughly similar to the notion of logical relation of [17], this paper does not rest on Moggi's computational λ-calculus. On the other hand [23] does rest on the computational λ-calculus but does not define a suitable notion of logical relation.

6.3 Monads of Measures and Probabilities

Let us consider a natural CCC equipped with a notion of measure: $\boldsymbol{Ipo}$, the category of *inductive partial orders* or *ipos* [8]. The objects (ipos) are partial orders such that every directed subset has a least upper bound; the morphisms are continuous functions. This is cartesian-closed, and has pullbacks.

For every ipo A, Jones observed that the set $\boldsymbol{T}A$ of all continuous evaluations, to be defined next, was again an ipo. An *evaluation* $\boldsymbol{\nu}$ maps Scott opens to reals in $[0, +\infty]$ so that $\boldsymbol{\nu}(O \cup O') = \boldsymbol{\nu}(O) + \boldsymbol{\nu}(O') - \boldsymbol{\nu}(O \cap O')$ (for all opens O, O'),

$\boldsymbol{\nu}(\emptyset) = 0$ and $\boldsymbol{\nu}$ is monotonic, i.e., $O \subseteq O'$ implies $\boldsymbol{\nu}(O) \leq \boldsymbol{\nu}(O')$. A *continuous evaluation* in addition maps unions of directed sets of opens to the sup of their evaluations. This extends to a strong monad $(\boldsymbol{T}, \boldsymbol{\eta}, \boldsymbol{\mu}, \boldsymbol{t})$: see Jones' thesis [8].

Take $\boldsymbol{C} = \boldsymbol{Ipo}$, $\mathbb{C} = \boldsymbol{Set}$, $|\_| : \boldsymbol{Ipo} \to \boldsymbol{Set}$ be the underlying set functor, with left adjoint the *discretization* functor $\boldsymbol{D}$: $\boldsymbol{D}(E)$ is E with equality as ordering. By Corollary 1 we get a pair of strong monads related by a distributivity law.

To lift the monads to the subscone, check that T preserves pseudoepis, i.e., that whenever $f : E \to F$ is surjective, then for every continuous evaluation ξ on F (opens are all subsets), there is a continuous evaluation ν of E such that $\xi(Y) = Tf(\nu)(Y)$, i.e., $\xi(Y) = \nu(f^{-1}(Y))$ for every $Y \subseteq F$. Using the axiom of choice, let γ map every $Z \in \mathbb{P}(E) \setminus \{\emptyset\}$ to some element of Z. Then defining $\nu(X)$ as $\xi(\{y \in Y | \gamma(f^{-1}(y)) \in X\})$ for every $X \subseteq E$ fits the bill.

Turning to binary relations is a matter of taking $\boldsymbol{C} = \boldsymbol{Ipo} \times \boldsymbol{Ipo}$, $\mathbb{C} = \boldsymbol{Set} \times \boldsymbol{Set}$, as in Section 4, and letting the distributivity law be given by (12), where $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ are both the continuous evaluation monad on $\boldsymbol{Ipo}$, $|\_|_1$ and $|\_|_2$ are both the forgetful functor from $\boldsymbol{Ipo}$ to $\boldsymbol{Set}$, and σ^1 and σ^2 are both given by Corollary 1.

Let us spell this out: for any relation $S \subseteq |A_1|_1 \times |A_2|_2$, the lifted relation $\widetilde{S}$ between continuous evaluations $\boldsymbol{\nu}_1 \in |\boldsymbol{T}_1 A_1|_1$ and $\boldsymbol{\nu}_2 \in |\boldsymbol{T}_2 A_2|_2$ is given by:

$$(\boldsymbol{\nu}_1, \boldsymbol{\nu}_2) \in \widetilde{S} \iff \left(\exists \nu \in TS. \begin{cases} \forall O_1 \subseteq A_1 \text{ open}. \boldsymbol{\nu}_1(O_1) = \nu((O_1 \times A_2) \cap S) \wedge \\ \forall O_2 \subseteq A_2 \text{ open}. \boldsymbol{\nu}_2(O_2) = \nu((A_1 \times O_2) \cap S) \end{cases} \right)$$

Interestingly, and analogously with Section 6.1, we may define a *probabilistic* labelled transition system as an element of $(\boldsymbol{T}A)^{A \times L}$. Then two such transition systems f_1 and f_2 are in relation if and only if:

$$\forall a_1 \in |A_1|_1, a_2 \in |A_2|_2 \,,\, \ell_1, \ell_2 \in L. (a_1, a_2) \in S \wedge (\ell_1, \ell_2) \in R_L \Rightarrow \tag{18}$$
$$\left(\exists \nu \in TS \,.\, \begin{cases} \forall O_1 \subseteq A_1 \text{ open}. f_1(a_1, \ell)(O_1) = \nu((O_1 \times A_2) \cap S) \wedge \\ \forall O_2 \subseteq A_2 \text{ open}. f_2(a_2, \ell)(O_2) = \nu((A_1 \times O_2) \cap S) \end{cases} \right)$$

We invite the reader to check that this definition is equivalent to Larsen and Skou's notion of *probabilistic bisimulation* [10] in the case where A_1 and A_2 are *finite* and *discrete*: then relations S as described by our subscone construction have probabilistic bisimulations as reflexive symmetric transitive closures, and probabilistic bisimulations are equivalence relations S obeying (18).

7 Conclusion

The main contribution of this paper is a natural extension of logical relations able to deal with monadic types. We illustrate its naturality and its practical value by demonstrating that various notions of bisimulations and a non-trivial notion of logical relation for dynamic name creation are instances of our construction. Besides, our construction provides a natural integration between notions of simulations between transition systems (possibly probabilistic), higher-order computation (the import of the λ-calculus), and limited forms of side-effects (e.g., dynamic names), yielding streamlined criteria for observational equivalence of those combined systems.

References

[1] J. Adamek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories*. Wiley, New York, 1990. 557

[2] M. Alimohamed. A characterization of lambda definability in categorical models of implicit polymorphism. *Theoretical Computer Science*, 146:5–23, 1995. 553

[3] R. Crole and A. Pitts. New foundations for fixpoint computations: Fix-hyperdoctrines and the fix-logic. *Information and Computation*, 98:171–210, 1992. 555

[4] M. Fiore and A. Simpson. Lambda definability with sums via Grothendieck logical relations. In *TLCA'99*, pages 147–161. Springer Verlag LNCS 1581, 1999. 553

[5] J. Goubault-Larrecq and E. Goubault. On the geometry of intuitionistic S4 proofs. Research Report LSV-01-8, LSV, CNRS & ENS Cachan, 2001. To appear in Homology, Homotopy and Applications. 555, 559

[6] J. Goubault-Larrecq, S. Lasota, and D. Nowak. Logical relations for monadic types. Research Report, LSV, CNRS & ENS Cachan, 2002. 563

[7] F. Honsell and D. Sannella. Pre-logical relations. In *CSL'99*, pages 546–561. Springer Verlag LNCS 1683, 1999. 554, 564

[8] C. Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, 1990. Technical Report ECS-LFCS-90-105. 554, 565, 566

[9] A. Jung and J. Tiuryn. A new characterization of lambda definability. In *TLCA'93*, pages 245–257. Springer Verlag LNCS 664, 1993. 553

[10] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991. 554, 566

[11] R. Lazić and D. Nowak. A unifying approach to data-independence. In *CONCUR'2000*, pages 581–595. Springer Verlag LNCS 1877, 2000. 553, 554, 564

[12] Q. Ma and J. C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. In *MFPS'91*, pages 1–40. Springer-Verlag LNCS 598, 1992. 553

[13] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996. 553, 554

[14] J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In *CSL'92*, pages 352–378. Springer Verlag LNCS 702, 1993. 554, 559, 560

[15] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, LFCS, Department of Computer Science, University of Edinburgh, 1990. 564

[16] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991. 553, 561, 563

[17] A. Pitts and I. Stark. Observable properties of higher order functions that dynamically create local names, or: What's *new*? In *MFCS'93*, pages 122–141. Springer-Verlag LNCS 711, 1993. 554, 565

[18] G. Plotkin, J. Power, D. Sannella, and R. Tennent. Lax logical relations. In *ICALP'2000*, pages 85–102. Springer Verlag LNCS 1853, 2000. 554

[19] G. D. Plotkin. Lambda-definability in the full type hierarchy. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980. 553

[20] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *POPL'02*, pages 154–165, 2002. 553

[21] J. C. Reynolds. Types, abstraction and parametric polymorphism. In *IFIP'83*, pages 513–523. North-Holland, 1983. 553

[22] J. Rutten. Relators and metric bisimulations. In *CMCS'98*, volume 11 of *Electronic Notes in Theoretical Computer Science*, pages 1–7. Elsevier Science, 1998. 555

[23] I. Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998. 554, 565

[24] R. Statman. Logical relations and the typed λ-calculus. *Information and Control*, 65(2–3):85–97, 1985. 553

[25] D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, June 1996. 554, 555

[26] P. Wadler. Comprehending monads. *Mathematical Structures in Computer Science*, 2:461–493, 1992. 553

On the Automatizability of Resolution and Related Propositional Proof Systems*

Albert Atserias and María Luisa Bonet

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona
{atserias,bonet}@lsi.upc.es

Abstract. We analyse the possibility that a system that simulates Resolution is automatizable. We call this notion "weak automatizability". We prove that Resolution is weakly automatizable if and only if Res(2) has feasible interpolation. In order to prove this theorem, we show that Res(2) has polynomial-size proofs of the reflection principle of Resolution (and of any Res(k)), which is a version of consistency. We also show that Resolution proofs of its own reflection principle require slightly subexponential size. This gives a better lower bound for the monotone interpolation of Res(2) and a better separation from Resolution as a byproduct. Finally, the techniques for proving these results give us a new complexity measure for Resolution that refines the width of Ben-Sasson and Wigderson. The new measure and techniques suggest a new algorithm to find Resolution refutations, and a way to obtain a large class of examples that have small Resolution refutations but require relatively large width. This answers a question of Alekhnovich and Razborov related to whether Resolution is automatizable in quasipolynomial-time.

1 Introduction

In several areas of Computer Science there has been important efforts in studying algorithms for satisfiability, despite the problem is NP-complete, and also in studying the complementary problem of verifying tautologies.

By the theorem of Cook and Reckhow [14], there is strong evidence that for every propositional proof system there is a class of tautologies whose shortest proofs are super-polynomial in the size of the tautologies. From this we conclude that given a propositional proof system S, there will not be an algorithm that will produce S-proofs of a tautology in time polynomial in the size of the tautology. This is because in some cases we might require exponential time just to write down the proof. Considering this limitation of proof systems, Bonet, Pitassi and Raz [12] proposed the following definition. A propositional proof system S is *automatizable* if there exists an algorithm that, given a tautology, it produces an S-proof of it in time polynomial in the size of the smallest S-proof of the tautology. The idea behind this definition is that if short S-proofs

* Partially supported by CICYT TIC2001-1577-C03-02, ALCOM-FT IST-99-14186 and HA2000-41.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 569–583, 2002.
© Springer-Verlag Berlin Heidelberg 2002

exist, an automatization algorithm for S should find them quickly. In the sequel of papers [24, 13, 9] it was proved that no proof system that simulates AC^0-Frege is automatizable, unless some widely accepted cryptographic conjecture is violated. Later, Alekhnovich and Razborov [1] proved that Resolution is not automatizable under a reasonable assumption in parameterized complexity. The drawback of this result is that it is weaker than the others in the sense that we do not know whether a system that simulates Resolution can be automatizable. This problem suggests the following definition. We say that a proof system S is *weakly automatizable* if there is a proof system that polynomially simulates S and is automatizable. At this point it is still open whether Resolution is weakly automatizable.

In this paper we characterize the question of whether Resolution is weakly automatizable as whether the extension of Resolution Res(2) (or Res(k) for k constant) has feasible interpolation. This notion will be defined in Section 4. Let us say for the moment, that Resolution, Cutting Planes, Relativized Bounded Arithmetic, Polynomial Calculus, Lovász-Schrijver and Nullstellensatz have feasible interpolation (see [20, 12, 26, 15, 22, 30, 29, 27]). On the other hand, the stronger system Frege, and any system that simulates AC^0-Frege do not have feasible interpolation under a cryptographic conjecture. To obtain this characterization we show that Res(2) has polynomial-size proofs of the reflection principle of Resolution, which is a form of consistency saying that if a CNF formula is satisfiable, then it does not have a Resolution refutation. We also show that Resolution requires almost exponential size to prove its own reflection principle. As a corollary we get an almost exponential lower bound for the monotone interpolation of Res(2) improving over the quasipolynomial lower bound in [4].

Despite the discouraging results in [1] mentioned before, there is still some effort put in finding good algorithms for proof systems such as Resolution. The first implementations were variants of the Davis-Putnam procedure [18, 17] for testing unsatisfiability that consists of either producing a tree-like Resolution refutation (if one exists), or giving a satisfying assignment. For various versions of this algorithm, one can prove that is it not an automatization procedure even for tree-like Resolution. A better algorithm for finding tree-like Resolution refutations was proposed by Beame and Pitassi [5]. They give an algorithm that works in time quasipolynomial in the size of the smallest proof of the tautology. So tree-like Resolution is automatizable in quasipolynomial time, but the algorithm is not a good automatization procedure for general Resolution (see [10, 6, 11]). A more efficient algorithm is the one of Ben-Sasson and Wigderson based on the width of a refutation. This algorithm weakly automatizes tree-like Resolution in quasipolynomial time and automatizes Resolution in subexponential time. On the other hand, Bonet and Galesi gave a class of tautologies for which the algorithm will take subexponential time to finish, matching the upper bound. Using the techniques introduced in this paper, we show that this is not an isolated example. We describe a method to produce tautologies that have small Resolution refutations but require relatively large width, answering an open problem of Alekhnovich and Razborov [1]. As they claim, this is a necessary step towards

proving that Resolution is not automatizable in quasipolynomial-time. Our techniques also suggest a new complexity measure for Resolution that refines the width of Ben-Sasson and Wigderson, and that gives rise to a new algorithm to find Resolution refutations.

2 Definitions

Resolution is a refutational proof system for CNF formulas, that is, conjunctions of clauses. The system has one inference rule, the *resolution rule*:

$$\frac{A \vee l \quad \neg l \vee B}{A \vee B}$$

where l is a literal, and A and B are clauses. The refutation finishes with the empty clause. The *size* of a Resolution refutation is the number of clauses in it. The system *tree-like Resolution* requires that each clause is used at most once in the proof. When this restriction is not fulfilled, we say that the refutation is in *DAG* form.

Following [7] the *width* of a refutation Π is defined as the maximum number of literals of the clauses appearing in Π. The main result in [7] is a relation between the size and the width of Resolution refutations. They show that if a set of 3-clauses has a tree-like Resolution refutation of size S_T, then it has a Resolution refutation of width $\log S_T$. Similarly, if it has a Resolution refutation of size S_R, then it has a Resolution refutation of width $O(\sqrt{n \log S_R})$. Ben-Sasson and Wigderson used this size-width trade-off to obtain an algorithm that finds Resolution refutations. It consists in deriving all posible clauses of increasing width until the empty clause is found. The time of the algorithm is $n^{O(w)}$ where w is the minimal width of a Resolution refutation of the initial set of clauses. Notice that the space used by the algorithm can only be bounded by $n^{O(w)}$ since all derivable clauses of width $v < w$ are needed to obtain the clauses of width w. Recall that the minimal width w is at most $\log S_T$ in the tree-like case, where S_T is the minimal tree-like size to refute the initial set of clauses. Therefore, the algorithm takes time $S_T^{O(\log n)}$ in this case. Also, the minimal width w is at most $\sqrt{n \log S_R}$ in the general case, where S_R is the minimal size to refute the set of clauses in general Resolution. This gives an $n^{O(\sqrt{n \log S_R})}$ bound on the running time.

A *k-term* is a conjunction of up to k literals. A *k-disjunction* is an (unbounded fan-in) disjunction of k-terms. The refutation system Res(k), defined by Krajíček [23], works with k-disjunctions. There are three inference rules in Res(k): Weakening, $\wedge$-Introduction, and Cut.

$$\frac{A}{A \vee B} \qquad \frac{A \vee l_1 \quad B \vee (l_2 \wedge \ldots \wedge l_s)}{A \vee B \vee (l_1 \wedge \ldots \wedge l_s)} \qquad \frac{A \vee (l_1 \wedge \ldots \wedge l_s) \quad B \vee \neg l_1 \vee \ldots \vee \neg l_s}{A \vee B}$$

Here A and B are k-disjunctions and the l_i's are literals. As usual, if l is a literal, $\neg l$ denotes the oposite literal. We also allow the axioms $l \vee \neg l$. Observe

that Res(1) is equivalent to Resolution since the axioms and the weakening rule are easy to eliminate in this case. The size of a Res(k) refutation is the number of k-disjunctions in it. As in Resolution, the tree-like version of Res(k) requires each k-disjunction in the proof to be used only once.

3 Some Simple Lemmas and a New Measure

For every set of literals $l_1, \ldots, l_s$ we define a new variable $z_{l_1,\ldots,l_s}$ meaning $l_1 \wedge \ldots \wedge l_s$. The following clauses define $z_{l_1,\ldots,l_s}$:

$$\neg z_{l_1,\ldots,l_s} \vee l_i \quad \text{for every } i \in \{1, \ldots, s\} \tag{1}$$

$$\neg l_1 \vee \ldots \vee \neg l_s \vee z_{l_1,\ldots,l_s} \tag{2}$$

Let $\mathcal{C}$ be a set of clauses on the variables $v_1, \ldots, v_n$. For every integer $k > 0$, we define $\mathcal{C}_k$ as the union of $\mathcal{C}$ with all the defining clauses for the variables $z_{l_1,\ldots,l_s}$ for all $s \leq k$.

Lemma 1. *If the set of clauses $\mathcal{C}$ has a Res(k) refutation of size S, then $\mathcal{C}_k$ has a Resolution refutation of size $O(kS)$. Furthermore, if the Res(k) refutation is tree-like, then the Resolution refutation is also tree-like.*

Proof of Lemma 1: Let Π be a Res(k) refutation of size S. To get a Resolution refutation of $\mathcal{C}_k$, we will first get a clause for each k-disjunction of Π. The translation consists in substituting each conjunction $l_1 \wedge \ldots \wedge l_s$ for $s \leq k$ in a clause of Π by $z_{l_1,\ldots,l_s}$. Also we have to make sure that we can make this new sequence of clauses into a Resolution refutation so that if Π is tree-like, then the new refutation will also be. We have the following cases:

Case 1: In Π we have the step:

$$\frac{C \vee (l_1 \wedge \ldots \wedge l_s) \qquad D \vee \neg l_1 \vee \ldots \vee \neg l_s}{C \vee D}$$

The corresponding clauses in the translation will be: $C' \vee z_{l_1,\ldots,l_s}$, $D' \vee \neg l_1 \vee \ldots \vee \neg l_s$ and $C' \vee D'$. To get a tree-like proof of $C' \vee D'$ from the two other ones, first obtain $\neg z_{l_1,\ldots,l_s} \vee D'$ in a tree-like way from $D' \vee \neg l_1 \vee \ldots \vee \neg l_s$ and the clauses $\neg z_{l_1,\ldots,l_s} \vee l_i$. Finally resolve $\neg z_{l_1,\ldots,l_s} \vee D'$ with $C' \vee z_{l_1,\ldots,l_s}$ to get $C' \vee D'$.

Case 2: In Π we have the step:

$$\frac{C \vee l_1 \qquad D \vee (l_2 \wedge \ldots \wedge l_s)}{C \vee D \vee (l_1 \wedge \ldots \wedge l_s)}$$

The corresponding clauses in the translation will be: $C' \vee l_1$, $D' \vee z_{l_2,\ldots,l_s}$ and $C' \vee D' \vee z_{l_1,\ldots,l_s}$. Notice that there is a tree-like proof of $\neg l_1 \vee \neg z_{l_2,\ldots,l_s} \vee z_{l_1,\ldots,l_s}$ from the clauses of $\mathcal{C}_k$. Using this clause and the translation of the premises, we get $C' \vee D' \vee z_{l_1,\ldots,l_s}$.

Case 3: The Weakening rule turns into a weakening rule for Resolution which can be eliminated easily.

At this point we have obtained a Resolution refutation of $\mathcal{C}_k$ that may use axioms of the type $l \vee \neg l$. These can be eliminated easily too. □

Lemma 2. *If the set of clauses $\mathcal{C}_k$ has a Resolution refutation of size S, then $\mathcal{C}$ has a Res(k) refutation of size $O(kS)$. Furthermore, if the Resolution refutation is tree-like, then the Res(k) refutation is also tree-like.*

Proof: We first change each clause of the Resolution refutation by a k-disjunction of Res(k) by translating $z_{l_1,\ldots,l_s}$ by $l_1 \wedge \ldots \wedge l_s$ and $\neg z_{l_1,\ldots,l_s}$ by $\neg l_1 \vee \ldots \vee \neg l_s$. At this point the rules of the Resolution refutation turn into valid rules of Res(k).

Now we only need to produce proofs of the defining clauses of the z variables in Res(k) to finish the simulation. The clauses $\neg z_{l_1,\ldots,l_s} \vee l_i$ get translated into $\neg l_1 \vee \ldots \vee \neg l_s \vee l_i$, which is a weakening of the axiom $l_i \vee \neg l_i$. The clause $\neg l_1 \vee \ldots \vee \neg l_s \vee z_{l_1,\ldots,l_s}$ gets translated into $\neg l_1 \vee \ldots \vee \neg l_s \vee (l_1 \wedge \ldots \wedge l_s)$ which can be proved form the axioms $l_i \vee \neg l_i$ using the rule for the introduction of the $\wedge$. □

The next lemmas are essentially Proposition 1.1 and 1.2 of [21].

Lemma 3. *Any Resolution refutation of width k and size S can be translated into a tree-like Res(k) refutation of size $O(kS)$.*

Proof sketch: Let Π be a Resolution refutation of width k and size S. Every non-initial clause C of Π is derived from two other clauses, say C_1 and C_2. Note that the k-disjunction $\neg C_1 \vee \neg C_2 \vee C$, where $\neg C_i$ is the conjunction of the negated literals of C_i, has a very simple tree-like Res(k) proof. The rest of the proof goes as in [21]. □

Lemma 4. *([21, 25, 19]) Any tree-like Res(k) refutation of size S can be translated into a Resolution refutation of size $O(S^2)$*

These lemmas suggest a refinement of the width mesure that we discuss next. Following [7], for an unsatisfiable set of clauses $\mathcal{C}$, let $w(\mathcal{C})$ be the minimal width of the Resolution refutations of $\mathcal{C}$. We define $k(\mathcal{C})$ to be the minimal k such that $\mathcal{C}$ has a tree-like Res(k) refutation of size n^k, where n is the number of variables of $\mathcal{C}$. We will prove that $k(\mathcal{C})$ is at most linear in $w(\mathcal{C})$, and that in some cases, $k(\mathcal{C})$ is significantly smaller than $w(\mathcal{C})$.

Lemma 5. $k(\mathcal{C}) = O(w(\mathcal{C}))$.

Proof: Let $w = w(\mathcal{C})$. Then $\mathcal{C}$ has a Resolution refutation of size $n^{O(w)}$ and width w since there are less than $n^{O(w)}$ clauses of width at most w and each clause needs to be derived only once since we are in the dag-like case. By Lemma 3, $\mathcal{C}$ a tree-like Res(w) refutation of size $O(wn^{O(w)})$. Taking $k = O(w)$, we see that $k(\mathcal{C}) = O(w(\mathcal{C}))$. □

Lemma 6. *There are sets of 3-clauses $\mathcal{F}_n$ such that $k(\mathcal{F}_n) = O(1)$ but $w(\mathcal{F}_n) = \Omega(\log n/\log\log n)$.*

Proof: Let $\mathcal{F}_n$ be the set of 3-clauses $E\text{-}PHP^m_{m'}$ where $m' = \log m/\log\log m$. Let n be the number of variables of $E\text{-}PHP^m_{m'}$. Dantchev and Riis [16] proved that $\mathcal{F}_n$ has tree-like Resolution refutations of size $2^{O(m'\log m')}$ which in this

case is $n^{O(1)}$. Therefore, $k(\mathcal{F}_n) = O(1)$. On the other hand, a standard width lower bound argument proves that $w(\mathcal{F}_n) = \Omega(m')$ which in this case is $\Omega(\log n/\log\log n)$. □

These Lemmas give rise to an algorithm to find Resolution refutations that improves the width algorithm of Ben-Sasson and Wigderson. Due to space limitations, we omit the precise description of this algorithm (see [3] instead). In a nutshell, the algorithm consists in using the algorithm of Beame and Pitassi [5] to find tree-like Resolution refutations of $\mathcal{C}_k$ of size n^k for increasing values of k until one is found. By Lemma 6, this algorithm improves Ben-Sasson and Wigderson in terms of space usage, and by Lemma 5 its running time is never worse for sets of clauses with relatively small (subexponential) Resolution refutations.

4 Reflection Principles and Weak Automatizability

Let $\mathcal{S}$ be a refutational proof system. Following Razborov [30] (see also [28]), let $REF(\mathcal{S})$ be the set of pairs $(\mathcal{C}, m)$, where $\mathcal{C}$ is a CNF formula that has an $\mathcal{S}$-refutation of size m. Furthermore, let SAT^* be the set of pairs $(\mathcal{C}, m)$ where $\mathcal{C}$ is a satisfiable CNF. Observe that when m is given in unary, both $REF(\mathcal{S})$ and SAT^* are in the complexity class NP. Pudlák called $(REF(\mathcal{S}), SAT^*)$ the *canonical NP-pair* of $\mathcal{S}$. Note also that $REF(\mathcal{S}) \cap SAT^* = \emptyset$ since $\mathcal{S}$ is supposed to refute unsatisfiable CNF formulas only. Interestingly enough, there is a tight connection between the complexity of the canonical NP-pair of $\mathcal{S}$ and the weak automatizability of $\mathcal{S}$. Namely, Pudlák [28] showed that $\mathcal{S}$ is weakly automatizable if and only if the canonical NP-pair of $\mathcal{S}$ is polynomially separable, which means that a polynomial-time algorithm returns 0 on every input from $REF(\mathcal{S})$ and returns 1 on every input from SAT^*. We will use this connection later.

The disjointness of the canonical NP-pair for a proof system $\mathcal{S}$ is often expressible as a contradictory set of clauses. Suppose that one is able to write down a CNF formula $SAT^n_r(x, z)$ meaning that "z encodes a truth assignment that satisfies the CNF encoded by x. The CNF is of size r and the underlying variables are $v_1, \ldots, v_n$". Similarly, suppose that one is able to write down a CNF formula $REF^n_{r,m}(x, y)$ meaning that "y encodes an $\mathcal{S}$-refutation of the CNF encoded by x. The size of the refutation is m, the size of the CNF is r, and the underlying variables are $v_1, \ldots, v_n$". Under these two assumptions, the disjointness of the canonical NP-pair for $\mathcal{S}$ is expressible by the contradictions $REF^n_{r,m}(y, z) \wedge SAT^n_r(x, z)$. This collection of CNF formulas is referred to as the *Reflection Principle* of $\mathcal{S}$. Notice that $REF^n_{r,m}(y, z) \wedge SAT^n_r(x, z)$ is a form of consistency of S.

We turn next to the concept of Feasible Interpolation introduced by Krajicek [22] (see also [12, 26]). Suppose that $A_0(x, y_0) \wedge A_1(x, y_1)$ is a contradictory CNF formula, where x, y_0, and y_1 are disjoint sets of variables. Note that for every given truth assignment a for the variables x, one of the formulas $A_0(a, y_0)$ or $A_1(a, y_1)$ must be contradictory by itself. We say that a proof system $\mathcal{S}$ has the *Interpolation Property* in time $T = T(m)$ if there exists an algorithm that, given a truth assignment a for the common variables x, returns an $i \in \{0, 1\}$

such that $A_i(a, y_i)$ is contradictory, and the running time is bounded by $T(m)$ where m is the minimal size of an $\mathcal{S}$-refutation of $A_0(x, y_0) \wedge A_1(x, y_1)$. Whenever $T(m)$ is a polynomial, we say that $\mathcal{S}$ has *Feasible Interpolation*.

The following result by Pudlák connects feasible interpolation with the reflection principle and weak automatizability.

Theorem 1. *[28] If the reflection principle for $\mathcal{S}$ has polynomial-size refutations in a proof system that has the feasible interpolation, then the canonical NP-pair for $\mathcal{S}$ is polynomially separable, and therefore $\mathcal{S}$ is weakly automatizable.*

For the rest of this section, we will need a concrete encoding of the reflection principle for Resolution. We start with the encoding of $SAT_r^n(x, z)$. The encoding of the set of clauses by the variables in x is as follows. There are variables $x_{e,i,j}$ for every $e \in \{0, 1\}$, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, r\}$. The meaning of $x_{0,i,j}$ is that the literal v_i appears in clause j, while the meaning of $x_{1,i,j}$ is that the literal $\neg v_i$ appears in clause j.

The encoding of the truth assignment $a \in \{0, 1\}^n$ by the variables z is as follows. There are variables z_i for every $i \in \{1, \ldots, n\}$, and $z_{e,i,j}$ for every $e \in \{0, 1\}$, $i \in \{1, \ldots, n + 1\}$ and $j \in \{1, \ldots, r\}$. The meaning of z_i is that variable v_i is assigned true under the truth assignment. The meaning of $z_{0,i,j}$ is that clause j is satisfied by the truth assignment due to a literal among $v_1, \neg v_1, \ldots, v_{i-1}, \neg v_{i-1}$. Similarly, the meaning of $z_{1,i,j}$ is that clause j is satisfied by the truth assignment due to a literal among $v_1, \neg v_1, \ldots, v_{i-1}, \neg v_{i-1}, v_i$. We formalize this as a set of clauses as follows:

$$\neg z_{0,1,j} \quad (3) \qquad z_{0,n+1,j} \quad (4)$$
$$z_{0,i,j} \vee \neg x_{0,i,j} \vee z_i \vee \neg z_{1,i,j} \quad (5) \qquad z_{1,i,j} \vee \neg x_{1,i,j} \vee \neg z_i \vee \neg z_{0,i+1,j} \quad (6)$$
$$z_{0,i,j} \vee x_{0,i,j} \vee \neg z_{1,i,j} \quad (7) \qquad z_{1,i,j} \vee x_{1,i,j} \vee \neg z_{0,i+1,j} \quad (8)$$

The encoding of $REF_{r,m}^n(x, y)$ is also quite standard. The encoding of the set of clauses by the variables in $\overline{x}$ is as before. The encoding of the Resolution refutation by the variables in $\overline{y}$ is as follows. There are variables $y_{e,i,j}$ for every $e \in \{0, 1\}$, $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, m\}$. The meaning of $y_{0,i,j}$ is that the literal v_i appears in clause j of the refutation. Similarly, the meaning of $y_{1,i,j}$ is that the literal $\neg v_i$ appears in clause j of the refutation. There are variables $p_{j,k}$ and $q_{j,k}$ for every $j \in \{1, \ldots, m\}$ and $k \in \{r, \ldots, m\}$. The meaning of $p_{j,k}$ (of $q_{j,k}$) is that clause C_k was obtained from clause C_j and some other clause, and C_j contains the resolved variable positively (negatively). Finally, there are variables $w_{i,k}$ for every $i \in \{1, \ldots, n\}$ and $k \in \{r, \ldots, m\}$. The meaning of $w_{i,k}$ is that clause C_k was obtained by resolving upon v_i. We formalize this by the

following set of clauses:

$$\neg x_{e,i,j} \vee y_{e,i,j} \quad (9) \qquad \neg y_{e,i,m} \quad (10)$$
$$\neg y_{0,i,j} \vee \neg y_{1,i,j} \quad (11) \qquad p_{1,k} \vee \ldots \vee p_{k-1,k} \quad (12)$$
$$q_{1,k} \vee \ldots \vee q_{k-1,k} \quad (13) \qquad \neg p_{j,k} \vee \neg q_{j,k} \quad (14)$$
$$\neg p_{j,k} \vee \neg p_{j',k} \quad (15) \qquad \neg q_{j,k} \vee \neg q_{j',k} \quad (16)$$
$$\neg p_{j,k} \vee \neg w_{i,k} \vee y_{0,i,j} \quad (17) \qquad \neg q_{j,k} \vee \neg w_{i,k} \vee y_{1,i,j} \quad (18)$$
$$\neg p_{j,k} \vee w_{i,k} \vee \neg y_{e,i,j} \vee y_{e,i,k} \quad (19) \qquad \neg q_{j,k} \vee w_{i,k} \vee \neg y_{e,i,j} \vee y_{e,i,k} \quad (20)$$
$$w_{1,k} \vee \ldots \vee w_{n,k} \quad (21) \qquad \neg w_{i,k} \vee \neg w_{i',k} \quad (22)$$

Notice that this encoding has the appropriate form for the monotone interpolation theorem.

Theorem 2. *The reflection principle for Resolution* $SAT^n_r(x,z) \wedge REF^n_{r,m}(x,y)$ *has Res(2) refutations of size* $(nr+nm)^{O(1)}$.

Proof: The goal is to get the following 2-disjunction

$$D_k \equiv \bigvee_{i=1}^{n} (y_{0,i,k} \wedge z_i) \vee (y_{1,i,k} \wedge \neg z_i)$$

for every $k \in \{1,\ldots,m\}$. The empty clause will follow by resolving D_m with (10). We distinguish two cases: $k \leq r$ and $r < k \leq m$. Since the case $k \leq r$ is easier but long, we leave it to Appendix A.

For the case $r < k \leq m$, we show how to derive D_k from $D_1,\ldots,D_{k-1}$. First, we derive $\neg p_{j,k} \vee \neg q_{l,k} \vee D_k$. From (18) and (11) we get $\neg q_{l,k} \vee \neg w_{q,k} \vee \neg y_{0,q,l}$. Resolving with D_l on $y_{0,q,l}$ we get

$$\neg q_{l,k} \vee \neg w_{q,k} \vee (y_{1,q,l} \wedge \neg z_q) \vee \bigvee_{\substack{i=1\\ i\neq q}}^{n} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \quad (23)$$

A cut with $z_q \vee \neg z_q$ on $y_{1,q,l} \wedge \neg z_q$ gives

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee \bigvee_{\substack{i=1\\ i\neq q}}^{n} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \quad (24)$$

Let $q' \neq q$. A cut with $z_{q'} \vee \neg z_{q'}$ on $y_{0,q',l} \wedge z_{q'}$ gives

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee z_{q'} \vee (y_{1,q',l} \wedge \neg z_{q'}) \vee \bigvee_{i\neq q,q'} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \quad (25)$$

From (20) and (22) we get $\neg q_{l,k} \vee \neg w_{q,k} \vee \neg y_{0,q',l} \vee y_{0,q',k}$. Resolving with (24) on $y_{0,q',l} \wedge z_{q'}$ gives

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee y_{0,q',k} \vee (y_{1,q',l} \wedge \neg z_{q'}) \vee \bigvee_{i\neq q,q'} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \quad (26)$$

An introduction of conjunction between (25) and (26) gives

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee (y_{0,q',k} \wedge z_{q'}) \vee (y_{1,q',l} \wedge \neg z_{q'}) \vee \bigvee_{i \neq q,q'} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \tag{27}$$

From (20) and (22) we also get $\neg q_{l,k} \vee \neg w_{q,k} \vee \neg y_{1,q',l} \vee y_{1,q',k}$. Repeating the same procedure we get

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee (y_{0,q',k} \wedge z_{q'}) \vee (y_{1,q',k} \wedge \neg z_{q'}) \vee \bigvee_{i \neq q,q'} (y_{0,i,l} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \tag{28}$$

Now, repeating this two-step procedure for every $q' \neq q$, we get

$$\neg q_{l,k} \vee \neg w_{q,k} \vee \neg z_q \vee \bigvee_{i \neq q} (y_{0,i,k} \wedge z_i) \vee (y_{1,i,l} \wedge \neg z_i). \tag{29}$$

A dual argument yould yield $\neg p_{j,k} \vee \neg w_{q,k} \vee z_q \vee \bigvee_{i \neq q} (y_{0,i,k} \wedge z_i) \vee (y_{1,i,k} \wedge \neg z_i)$. A cut with (29) on z_q gives $\neg p_{j,k} \vee \neg q_{l,k} \vee \neg w_{q,k} \vee \bigvee_{i \neq q} (y_{0,i,k} \wedge z_i) \vee (y_{1,i,k} \wedge \neg z_i)$. Weakening gives then $\neg p_{j,k} \vee \neg q_{l,k} \vee \neg w_{q,k} \vee D_k$. Resolving with (21) gives $\neg p_{j,k} \vee \neg q_{l,k} \vee D_k$. Coming to the end, we resolve this with (12) to get $p_{l,k} \vee \neg q_{l,k} \vee D_k$. Then resolve it with (14) to get $\neg q_{l,k} \vee D_k$, and resolve it with (13) to get D_k. □

An immediate consequence of Theorems 2 and 1 is that if Res(2) has feasible interpolation, then Resolution is weakly automatizable. The reverse implication holds too.

Theorem 3. *Resolution is weakly automatizable if and only if Res(2) has feasible interpolation.*

Proof: Suppose Resolution is weakly automatizable. Then by Corollary 10 in [28], the NP-pair of resolution is polynomially separable. We claim that the canonical pair of Res(2) is also polynomially separable. Here is the separation algorithm: Given a set of clauses $\mathcal{C}$ and a number S, we build $\mathcal{C}_2$ and run the separation algorithm for the canonical pair of Resolution on $\mathcal{C}_2$ and $c \cdot 2S$, where c is the hidden constant in Lemma 1. For the correctness, note that if $\mathcal{C}$ has a Res(2) refutation of size S, then $\mathcal{C}_2$ has a Resolution refutation of size $c \cdot 2S$ by Lemma 1, and the separation algorithm for the canonical pair of Resolution will return 0 on it. On the other hand, if $\mathcal{C}$ is satisfiable, so is $\mathcal{C}_2$ and the separation algorithm for Resolution will return 1 on it. Now, for the feasible interpolation of Res(2), consider the following algorithm. Let $A_0(x,y) \wedge A_1(x,z)$ be a contradictory set of clauses with a Res(2) refutation Π of size S. Given a truth assignment a for the variables x, run the separation algorithm for the canonical pair of Res(2) on inputs $A_0(a,y)$ and S. For the correctness, observe that if $A_1(a,z)$ is satisfiable, say by $z = b$, then $\Pi|_{x=a,z=b}$ is a Res(2) refutation of $A_0(a,y)$ of size at most S and the separation algorithm will return 0 on it. On the other hand, if $A_0(a,y)$ is satisfiable, the separation algorithm will return 1, which is correct. If both are unsatisfiable, any answer is fine. □

The previous theorem works for any k constant. If $k = \log n$, then we get that if Resolution is weakly automatizable then Res(log) has feasible interpolation in quasipolynomial time. The positive interpretation of these results is that to show that Resolution is weakly automatizable, then we only have to prove that Res(2) has feasible interpolation. The negative interpretation is that to show that resolution is not weakly automatizable we only have to prove that Res(log) doesn't have feasible interpolation in quasipolynomial time.

It is not clear whether Res(2) has feasible interpolation. We know, however, that Res(2) does not have monotone feasible interpolation (see [4] and Corollary 1 in this paper). On the other hand, tree-like Res(2) has feasible interpolation (even monotone) since Resolution polynomially simulates it by Lemma 4.

A natural question to ask is whether the reflection principle for Resolution has Resolution refutations of moderate size. Since Resolution has feasible interpolation, a positive answer would imply that Resolution is weakly automatizable by theorem 1. Unfortunately, as the next theorem shows, this will not happen. The proof of this result uses an idea due to Pudlak.

Theorem 4. *For some choice of n, r, and m of the order of a quasipolynomial $s^{O(\log s)}$ on the parameter s, every Resolution refutation of $REF^n_{r,m}(x,y) \wedge SAT^n_r(x,z)$ requires size at least $2^{\Omega(s^{1/4})}$.*

Proof: Suppose for contradiction that there is a Resolution refutation of size $S = 2^{o(s^{1/4})}$. Let $k = s^{1/2}$, and let $COL_k(p,q)$ be the CNF formula expressing that q encodes a k-coloring of the graph on s nodes encoded by $\{p_{i,j}\}$. An explicit definition is the following: For every $i \in \{1,\ldots,s\}$, there is a clause of the form $\bigvee_{l=1}^{k} q_{il}$; and for every $i,j \in \{1,\ldots,s\}$ with $i \neq j$ and $l \in \{1,\ldots,k\}$, there is a clause of the form $\neg q_{il} \vee \neg q_{jl} \vee \neg p_{ij}$. Obviously, if G is k-colorable, then $COL_k(G,q)$ is satisfiable, and if G contains a $2k$-clique, then $COL_k(G,q)$ is unsatisfiable. More importantly, if G contains a $2k$-clique, then the clauses of PHP^{2k}_k are contained in $COL_k(G,q)$. Now, for every graph G on s nodes, let $\mathcal{F}(G)$ be the clauses $COL_k(G,q)$ together with all clauses defining the extension variables for the conjunctions of up to $c \log k$ literals on the q-variables. Here, c is a constant so that the $k^{O(\log k)}$ upper bound on PHP^{2k}_k of [25] can be done in Res($c \log k$). From its very definition and Lemma 1, if G contains a $2k$-clique, then $\mathcal{F}(G)$ has a Resolution refutation of size $k^{O(\log k)}$. Finally, for every graph G, let $x(G)$ be the encoding of the formula $\mathcal{F}(G)$. With all this notation, we are ready for the argument.

In the following, let n be the number of variables of $\mathcal{F}(G)$, let r be the number of clauses of $\mathcal{F}(G)$, and let $m = k^{O(\log k)}$. By assumption, the formulas $REF^n_{r,m}(x(G),y) \wedge SAT^n_r(x(G),z)$ have Resolution refutations of size at most S. Let C be the monotone circuit that interpolates these formulas given $x(G)$. The size of C is $S^{O(1)}$. Moreover, if G is k-colorable, then $SAT^n_r(x(G),z)$ is satisfiable, and C must return 0 on $x(G)$. Also, if G contains a $2k$-clique, then $REF^n_{r,m}(x(G),y)$ is satisfiable, and C must return 1 on $x(G)$. Now, an anti-monotone circuit for separating $2k$-cliques from k-colorings can be built as follows: given a graph G, build the formula $x(G)$ (anti-monotonically, see below

for details), and apply the monotone circuit given by the monotone interpolation. The size of this circuit is $2^{o(s^{1/4})}$, and this contradicts Theorem 3.11 of Alon and Boppana [2].

It remains to show how to build an anti-monotone circuit that, on input $G = \{p_{uv}\}$, produces outputs of the form $x_{e,i,j}$ that correspond to the encoding of $\mathcal{F}(G)$ in terms of the x-variables.

- Clauses of the type $\bigvee_{l=1}^{k} q_{il}$: Let t be the numbering of this clause in $\mathcal{F}(G)$. Then, its encoding in terms of the x-variables is produced by plugging the constant 1 to the outputs $x_{1,q_{i1},t}, \ldots, x_{1,q_{ik},t}$. The rest of outputs of clause t get plugged the constant 0.
- Clauses of the type $\neg q_{il} \vee \neg q_{jl} \vee \neg p_{ij}$: Let t be the numbering of this clause in $\mathcal{F}(G)$. The encoding is $x_{0,q_{il},t} = 1$, $x_{0,q_{jl},t} = 1$, $x_{0,p_{ij},t} = \neg p_{ij}$ and the rest are zero. Notice that this encoding is anti-monotone in the p_{ij}'s. Notice also that the encoded $\mathcal{F}(G)$ contains some p-variables (and not only q-variables as the reader might have expected) but this will not be a problem since the main properties of $\mathcal{F}(G)$ are preserved as we show below.
- Finally, the clauses defining the conjunctions of up to $c \log k$ literals are independent of G since only the q-variables are relevant here. Therefore, the encoding is done as in the first case.

The reader can easily verify that when G contains a $2k$-clique, the encoded formula contains the clauses of PHP_k^{2k} and the definitions of the conjunctions up to $c \log k$ literals. Therefore $REF(x(G), y)$ is satisfiable given that PHP_k^{2k} has a small Res($c \log k$) refutation. Similarly, if G is k-colorable, the formula $SAT(x(G), z)$ is satisfiable by setting $z_{p_{ij}} = p_{ij}$ and $q_{il} = 1$ if and only if node i gets color l. Therefore, the main properties of $\mathcal{F}(G)$ are preserved, and the theorem follows. □

An immediate corollary of the last two results is that Res(2) is exponentially more powerful than resolution. In fact, the proof shows a lower bound for the monotone interpolation of Res(2) improving over the quasipolynomial lower bound in [4].

Corollary 1. *Monotone circuits that interpolate Res(2) refutations require size* $2^{\Omega(s^{1/4})}$ *on Res(2) refutations of size* $s^{O(\log s)}$.

Theorem 4 is in sharp contrast with the fact that an appropriate encoding of the reflection principle for Res(2) has polynomial-size proofs in Res(2). This encoding incorporates new z-variables for the truth values of conjunctions of two literals, and new y-variables encoding the presence of conjunctions in the 2-disjunctions of the proof. The resulting formula preserves the form of the feasible interpolation. We leave the tedious details to the interested reader.

Theorem 5. *The reflection principle for Res(2) has Res(2) refutations of size* $(n^2 r + mr)^{O(1)}$. *More strongly, the reflection principle for Res(k) has Res(2) refutations of size* $(n^k r + mr)^{O(1)}$.

We observe that there is *a version* of the reflection principle for Resolution that has polynomial-size proofs in Resolution. Namely, let $\mathcal{C}$ be the CNF formula $SAT_r^n(x,z) \wedge REF_{r,m}^n(y,z)$. Then, $\mathcal{C}_2$ has polynomial-size Resolution refutations by Lemma 1 and Theorem 2. However, this does not imply the weak automatizability of Resolution since the set of clauses does not have the appropriate form for the feasible interpolation theorem.

5 Short Proofs that Require Large Width

Bonet and Galesi [11] gave an example of a CNF expressed in constant width, with small Resolution refutations, and requiring relatively large width (square root of the number of variables). This showed that the size-width trade-off of Ben-Sasson and Wigderson could not be improved. Also it showed that the algorithm of Ben-Sasson and Wigderson for finding Resolution refutations could perform very badly in the worst case. This is because their example requires large width, and the algorithm would take almost exponential time, while we know that there is a polynomial size Resolution refutation.

Alekhnovich and Razborov [1] posed the question of whether more of these examples could be found. They say this is a necessary first step for showing that Resolution is not automatizable in quasipolynomial-time. Here we give a way of producing such bad examples for the algorithm. Basically the idea is finding CNFs that require sufficiently high width in Resolution, but that have polynomial size Res(k) refutations for small k, say $k \leq \log n$. Then the example consists of adding to the formula the clauses defining the extension variables for all the conjunctions of at most k literals. Below we ilustrate this technique by giving a large class of examples that have small Resolution refutations, require large width. Moreover, deciding whether a formula is in the class is hard (no polynomial-time algorithm is known).

Let $G = (U \cup V, E)$ be a bipartite graph on the sets U and V of cardinality m and n respectively, where $m > n$. The $G\text{-}PHP_n^m$, defined by Ben-Sasson and Wigderson [7], states that there is no matching from U into V. For every edge $(u,v) \in E$, let $x_{u,v}$ be a propositional variable meaning that u is mapped to v. The principle is then formalized as the conjunction of the following clauses:

$$x_{u,v_1} \vee \cdots \vee x_{u,v_r} \quad u \in U,\ N_G(u) = \{v_1, \ldots, v_r\}$$
$$\bar{x}_{u,v} \vee \bar{x}_{u',v} \quad v \in V,\ u, u' \in N_G(v),\ u \neq u'.$$

Here, $N_G(w)$ denotes the set of neighbors of w in G. Note that if G has left-degree at most d, then the width of the initial clauses is bounded by d.

Ben-Sasson and Wigderson proved that whenever G is expanding in a sense defined next, every Resolution refutation of $G\text{-}PHP_n^m$ must contain a clause with many literals. We observe that this result is not unique to Resolution and holds in a more general setting. Before we state the precise result, let us recall the definition of expansion:

Definition 1. [7] *Let $G = (U \cup V, E)$ be a bipartite graph where $|U| = m$, and $|V| = n$. For $U' \subset U$, the* boundary *of U, denoted by $\partial U'$, is the set of vertices in V that have exactly one neighbor in U'; that is, $\partial U' = \{v \in V : |N(v) \cap U'| = 1\}$. We say that G is (m, n, r, f)-*expanding *if every subset $U' \subseteq U$ of size at most r is such that $|\partial U'| \geq f \cdot |U'|$.*

The proof of the following statement is the same as in [7] for Resolution.

Theorem 6. [7] *Let $\mathcal{S}$ be a sound refutation system with all rules having fan-in at most two. Then, if G is (m, n, r, f)-expanding, every $\mathcal{S}$-refutation of G-PHP_n^m must contain a formula that involves at least $rf/2$ distinct literals.*

Now, for every bipartite graph G with $m \geq 2n$, let $\mathcal{C}(G)$ be the set of clauses defining G-PHP_n^m together with the clauses defining all the conjunctions up to $c \log n$ literals, where c is a large constant.

Theorem 7. *Let G be an $(m, n, \Omega(n/\log m), \frac{3}{4}\log m)$-expander with $m \geq 2n$ and left-degree at most $\log m$. Then (i) $\mathcal{C}(G)$ has initial width $\log m$, (ii) any Resolution refutation of $\mathcal{C}(G)$ requires width at least $\Omega(n/\log n)$, and (iii) $\mathcal{C}(G)$ has polynomial-size Resolution refutations.*

Proof: Part (i) is obvious. For (ii), suppose for contradiction that $\mathcal{C}(G)$ has a Resolution refutation of width $w = o(n/\log n)$. Then, by the proof of Lemma 2, G-PHP_n^m has a Res$(c \log n)$ refutation in which every $(c \log n)$-disjunction involves at most $wc \log n = o(n)$ literals. This contradicts Theorem 6. For (iii), recall that PHP_n^m has a Res$(c \log n)$ refutation of size $n^{O(\log n)}$ by [25] since $m \geq 2n$. Now, setting to zero the appropriate variables of PHP_n^m, we get a Res$(c \log n)$ refutation of G-PHP_n^m of the same size. By Lemma 1, $\mathcal{C}(G)$ has a Resolution refutation of roughly the same size, which is polynomial in the size of the formula. □

It is known that deciding whether a bipartite graph is an expander (for a slightly different definition than ours) is coNP-complete [8]. Although we have not checked the details, we suspect that deciding whether a bipartite graph is an (m, n, r, f)-expander in the sense of Definition 1 is also coNP-complete. However, we should note that the class of formulas $\{\mathcal{C}(G) : G \text{ expander}, m \geq 2n\}$ is contained in $\{\mathcal{C}(G) : G \text{ bipartite}, m \geq 2n\}$ which is decidable in polynomial-time, and that all formulas of this class have short Resolution refutations that are easy to find. This is so because the proof of PHP_n^{2n} in [25] is given explicitely.

6 Conclusions and Open Problems

We showed that the new measure $k(\mathcal{C})$ introduced in section 3 is a refinement of the width $w(\mathcal{C})$. Actually, we believe that a careful analysis in Lemma 5 could even show that $k(\mathcal{C}) \leq w(\mathcal{C}) + 1$ for sets of clauses $\mathcal{C}$ with sufficiently many variables. On the other hand, we proved a logarithmic gap between $k(\mathcal{C})$ and

$w(\mathcal{C})$ for a concrete class of 3-clauses $\mathcal{C}_n$. We do not know if a larger gap is possible.

It is surprising that the weak pigeonhole principle PHP_n^{2n} has short Resolution proofs when encoded with the clauses defining the extension variables. This suggests that to prove Resolution lower bounds that are robust, one should prove Res(k) lower bounds for relatively large k. In fact, at this point the only robust lower bounds we know are the ones for AC^0-Frege.

Of course, it remains open whether Resolution is weakly automatizable, or automatizable in quasipolynomial-time.

Acknowledgement

We are grateful to Pavel Pudlák for stimulating discussions on the idea of Theorem 4.

References

[1] M. Alekhnovich and A. A. Razborov. Resolution is not automatizable unless W[P] is tractable. In *42nd Annual IEEE Symposium on Foundations of Computer Science*, 2001. 570, 580

[2] N. Alon and R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7:1–22, 1987. 579

[3] A. Atserias and M. L. Bonet. On the automatizability of resolution and related propositional proof systems. ECCC TR02-010, 2002. 574

[4] A. Atserias, M. L. Bonet, and J. L. Esteban. Lower bounds for the weak pigeonhole principle and random formulas beyond resolution. Accepted for publication in *Information and Computation*. A preliminary version appeared in ICALP'01, *Lecture Notes in Computer Science* 2076, Springer, pages 1005–1016., 2001. 570, 578, 579

[5] P. Beame and T. Pitassi. Simplified and improved resolution lower bounds. In *37th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–282, 1996. 570, 574

[6] E. Ben-Sasson, R. Impagliazzo, and A. Wigderson. Near-optimal separation of general and tree-like resolution. To appear, 2002. 570

[7] E. Ben-Sasson and A. Wigderson. Short proofs are narrow–resolution made simple. *J. ACM*, 48(2):149–169, 2001. 571, 573, 580, 581

[8] M. Blum, R. M. Karp, O. Vornberger, C. H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letter*, 13:164–167, 1981. 581

[9] M. L. Bonet, C. Domingo, R. Gavaldà, A. Maciel, and T. Pitassi. Non-automatizability of bounded-depth Frege proofs. In *14th IEEE Conference on Computational Complexity*, pages 15–23, 1999. Accepted for publication in the Journal of Computational Complexity. 570

[10] M. L. Bonet, J. L. Esteban, N. Galesi, and J. Johansen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM Journal of Computing*, 30(5):1462–1484, 2000. A preliminary version appeared in FOCS'98. 570

[11] M. L. Bonet and N. Galesi. Optimality of size-width trade-offs for resolution. *Journal of Computational Complexity*, 2001. To appear. A preliminary version appeared in FOCS'99. 570, 580

[12] M. L. Bonet, T. Pitassi, and R. Raz. Lower bounds for cutting planes proofs with small coefficients. *Journal of Symbolic Logic*, 62(3):708–728, 1997. A preliminary version appeared in STOC'95. 569, 570, 574

[13] M. L. Bonet, T. Pitassi, and R. Raz. On interpolation and automatization for Frege systems. *SIAM Journal of Computing*, 29(6):1939–1967, 2000. A preliminary version appeared in FOCS'97. 570

[14] S. Cook and R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44:36–50, 1979. 569

[15] S. A. Cook and A. Haken. An exponential lower bound for the size of monotone real circuits. *Journal of Computer and System Sciences*, 58:326–335, 1999. 570

[16] S. Dantchev and S. Riis. Tree resolution proofs of the weak pigeon-hole principle. In *16th IEEE Conference on Computational Complexity*, pages 69–75, 2001. 573

[17] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962. 570

[18] M. Davis and H. Putnam. A computing procedure for quantification theory. *J. ACM*, 7:201–215, 1960. 570

[19] J. L. Esteban, N. Galesi, and J. Messner. Personal communication. Manuscript, 2001. 573

[20] R. Impagliazzo, T. Pitassi, and A. Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *9th IEEE Symposium on Logic in Computer Science*, pages 220–228, 1994. 570

[21] J. Krajícek. Lower bounds to the size of constant-depth propositional proofs. *Journal of Symbolic Logic*, 39(1):73–86, 1994. 573

[22] J. Krajícek. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62:457–486, 1997. 570, 574

[23] J. Krajícek. On the weak pigeonhole principle. To appear in *Fundamenta Mathematicæ*, 2000. 571

[24] J. Krajícek and P. Pudlák. Some consequences of cryptographical conjectures for S_2^1 and EF. *Information and Computation*, 140(1):82–94, 1998. 570

[25] A. Maciel, T. Pitassi, and A. R. Woods. A new proof of the weak pigeonhole principle. In *32nd Annual ACM Symposium on the Theory of Computing*, 2000. 573, 578, 581

[26] P. Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, 1997. 570, 574

[27] P. Pudlák. On the complexity of the propositional calculus. In *Sets and Proofs, Invited Papers from Logic Colloquium '97*, pages 197–218. Cambridge University Press, 1999. 570

[28] P. Pudlák. On reducibility and symmetry of disjoint NP-pairs. In *26th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science, pages 621–632. Springer-Verlag, 2001. 574, 575, 577

[29] P. Pudlák and J. Sgall. Algebraic models of computation and interpolation for algebraic proof systems. In P. W. Beame and S. R. Buss, editors, *Proof Complexity and Feasible Arithmetic*, volume 39 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 279–296. American Mathematical Society, 1998. 570

[30] A. A. Razborov. Unprovability of lower bounds on circuit size in certain fragments of bounded arithmetic. *Izvestiya of the RAN*, 59(1):205–227, 1995. 570, 574

Extraction of Proofs from the Clausal Normal Form Transformation

Hans de Nivelle

Max Planck Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany
nivelle@mpi-sb.mpg.de

Abstract. This paper discusses the problem of how to transform a first-order formula into clausal normal form, and to simultaneously construct a proof that the clausal normal form is correct. This is relevant for applications of automated theorem proving where people want to be able to use theorem prover without having to trust it.

1 Introduction

Modern theorem provers are complicated pieces of software containing up to 100,000 lines of code. In order to make the prover sufficiently efficient, complicated datastructures are implemented for efficient maintenance of large sets of formulas ([16]) In addition, they are written in programming languages that do not directly support logical formulas, like C or C++. Because of this, theorem provers are subject to errors.

One of the main applications of automated reasoning is in verification, both of software and of hardware. Because of this, users must be able to trust proofs from theorem provers completely. There are two approaches to obtain this goal: The first is to formally verify the theorem prover (the *internalization* approach), the second is to make sure that the proofs of the theorem prover can be formally verified. We call this the *external approach*.

The first approach has been applied on simple versions of the CNF-transformation with success. In [10], a CNF-transformer has been implemented and verified in ACL2. In [5], a similar verification has been done in COQ.

The advantage of this approach is that once the check of the CNF-transformer is complete, there is no additional cost in using the CNF-transformer. It seems however difficult to implement and verify more sophisticated CNF-transformations, as those in [12], [1], or [8]. As a consequence, users have to accept that certain decision procedures are lost, or that less proofs will be found.

A principal problem seems to be the fact that in general, program verification can be done on only on small (inductive) types. For example in [5], it was necessary to inductively define a type **prop** mimicking the behaviour of Prop in COQ. In [10], it was necessary to limit the correctness proof to finite models. Because of this limitation, the internalization approach seems to be restricted to problems that are strictly first-order.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 584–598, 2002.
© Springer-Verlag Berlin Heidelberg 2002

Another disadvantage of the internalization approach is the fact that proofs cannot be communicated. Suppose some party proved some theorem and wants to convince another party, who is skeptical. The other party is probably not willing to recheck correctness of the theorem prover and rerun it, because this might be very costly. It is much more likely that the other party is willing to recheck a proof.

In this paper, we explore the external approach. The main disadvantage of the external approach is the additional cost of proof checking. If one does the proof generation naively, the resulting proofs can have unacceptible size [6]. We present methods that bring down this cost considerably.

In this paper, we discuss the three main technical problems that appear when one wants to generate explicit type theory proofs from the CNF-transformation. The problems are the following: **(1)** Some of the transformations in the CNF-transformation are not equivalence preserving, but only satisfiability preserving. Because of this, it is in general not possible to prove $F \leftrightarrow \mathrm{CNF}(F)$. The problematic conversions are Skolemization, and subformula replacement. In order to simplify the handling of such transformations, we will define an intermediate proof representation language that has instructions that allow signature extension, and that make it possible to specify the condition that the new symbol must satisfy. When it is completed, the proof script can be tranformed into a proof term.

(2) The second problem is that naive proof construction results in proofs of unacceptible size. This problem is caused by the fact that one has to build up the context of a replacement, which constructs proofs of quadratic size. Since for most transformations (for example the Negation Normal Form transformation), the total number of replacements is likely to be at least linear in the size of the formula, the resulting proof can easily have a size cubic in the size of the formula. Such a complexity would make the external approach impossible, because it is not uncommon for a formula to have 1000 or more symbols. We discuss this problem in Section 3. For many transformations, the complexity can be brought down to a linear complexity.

(3) The last technical problem that we discuss is caused by improved Skolemization methods, see [11], [13]. Soundness of Skolemization can be proven through choice axioms. There are many types of Skolemization around, and some of them are parametrized. We do not want have a choice axiom for each type of Skolemization, for each possible value of the parameter. That would result in far too many choice axioms. In Section 4 we show that all improved Skolemization methods (that the author knows of) can be reduced to standard Skolemization.

In the sequel, we will assume familiarity with type theory. (See [15], [3]) We make use only of standard polymorphic type theory. In particular, we don't make use of inductive types.

2 Proof Scripts

We assume that the goal is find a proof term for $F \to \bot$, for some given formula F. If instead one wants to have a proof instead of rejection, for some G, then one has to first construct a proof of $\neg\neg G \to \bot$, and then transform this into a proof of G.

It is convenient not to construct this proof term directly, but first to construct a sequence of intermediate formulas that follow the derivation steps of the theorem prover. We call such sequence of formulas a *proof script*.

The structure of the proof script will be as follows: First $\Gamma \vdash A_1$, is proven. Next, $\Gamma, A_1 \vdash A_2$, is proven, etc. until $\Gamma, A_1, A_2, \ldots, A_{n-1} \vdash A_n = \bot$ is reached.

The advantage of proof scripts is that they can closely resemble the derivation process of the theorem prover. In particular, no stack is necessary to translate the steps of the theorem prover into a proof script. It will turn out, (Definition 2) that in order to translate a proof script into one proof term, the proof script has to be read backwards. If one would want to construct the proof term at once from the output of the theorem prover, one would have to maintain a stack inside the translation program, containing the complete proof. This should be avoided, because the translation of some of the proof steps alone may already require much memory. (See Section 3) When generating proof scripts, the intermediate proofs can be forgotten once they have been output.

Another advantage is that a sequence of intermediate formulas is more likely to be human readable than a big λ-term. This makes it easier to present the proof or to debug the programs involved in the proof generation.

Once the proof script has been constructed, one can translate the proof script into one proof term of the original formula. Alternatively, one can simply check the proof script itself.

We now define what a proof script is and when it is correct in some context. There are instructions for handling all types of intermediate steps that can occur in resolution proofs. The lemma-instruction proves an intermediate step, and gives a name to the result. The witness-instruction handles signature extension, as is needed for Skolemization. The split-instruction handles reasoning by cases. Some resolution provers have this rule implemented, most notably Spass, [17], see also [18].

Definition 1. *A proof script is a list of commands* $(c_1, \ldots, c_p)$ *with* $p > 0$. *We recursively define when a proof script is correct in some context. We write* $\Gamma \vdash (c_1, \ldots, c_p)$ *if* $(c_1, \ldots, c_p)$ *is correct in context* Γ.

- *If* $\Gamma \vdash x{:}\bot$, *then* $\Gamma \vdash (\mathrm{false}(x))$.
- *If* $\Gamma, a_1{:}X_1, \ldots, a_m{:}X_m \vdash (c_1, \ldots, c_p)$, *and* c *has form*

$$\mathrm{lemma}(a_1, x_1, X_1;\ \ \ldots\ ; a_m, x_m, X_m),\ \textit{with } m \geq 1,$$

the $a_1, \ldots, a_m$ *are distinct atoms, not occurring in* Γ, *and there are* $X'_1, \ldots, X'_m$, *such that for each* k, $(1 \leq k \leq m)$, $\Gamma \vdash x_k{:}X'_k$, *and*

$$\Gamma,\quad a_1 := x_1{:}X_1,\quad \ldots,\quad a_{k-1} := x_{k-1}{:}X_{k-1} \vdash X_k \equiv_{\alpha,\beta,\delta,\eta} X'_k,$$

then $\Gamma \vdash (c, c_1, \ldots, c_p)$.

- *Assume that* $\Gamma, a{:}A,\ h{:}(P\ a) \vdash (c_1, \ldots, c_p)$, *the atoms* a, h *are distinct and do not occur in* Γ. *If* $\Gamma \vdash x{:}(\forall a{:}A\ (P\ a) \to \bot) \to \bot$, *and* c *has form* $\mathrm{witness}(a, A, h, x, (P\ a))$, *then* $\Gamma \vdash (c, c_1, \ldots, c_p)$.
- *Assume that* $\Gamma, a_1{:}A_1 \vdash (c_1, \ldots, c_p)$ *and* $\Gamma, a_2{:}A_2 \vdash (d_1, \ldots, d_q)$. *If atoms* a_1, a_2 *do not occur in* Γ,

$$\Gamma \vdash x{:}(A_1 \to \bot) \to (A_2 \to \bot) \to \bot,$$

and c *has form* $\mathrm{split}(a_1, A_1, a_2, A_2, x)$, *then*

$$\Gamma \vdash (c, c_1, \ldots, c_p, d_1, \ldots, d_q).$$

When the lemma-instruction is used for proving a lemma, one has $m = 1$. Using the Curry-Howard isomorphism, the lemma-instruction can be also used for introducing definitions. The case $m > 1$ is needed in a situation where wants to define some object, prove some of its properties while still remembering its definition, and then forget the definition. Defining the object and proving the property in separate lemma-instructions would not be possible, because the definition of the object is immediately forgotten after the first lemma-instruction.

The witness-instruction is needed for proof steps in which one can prove that an object with a certain property exists, without being able to define it explicitly. This is the case for Skolem-functions obtained with the axiom of choice.

The split-instruction and the witness-instruction are more complicated than intuitively necessary, because we try to avoid using classical principles as much as possible. The formula $(\forall a{:}A\ (P\ a) \to \bot) \to \bot$ is equivalent to $\exists a{:}A\ (P\ a)$ in classical logic. Similarly $(A_1 \to \bot) \to (A_2 \to \bot) \to \bot$ is equivalent to $A_1 \vee A_2$ in classical logic. Sometimes the first versions are provable in intuitionistic logic, while the second versions are not.

Checking correctness of proof scripts is straightforward, and we omit the algorithm. We now give a translation schema that translates a proof script into a proof term. The proof term will provide a proof of $\bot$.

The translation algorithm constructs a translation of a proof script $(c_1, \ldots, c_p)$ by recursion. It breaks down the proof script into smaller proof scripts and calls itself with these smaller proof scripts. There is no need to pass complete proof scripts as argument. It is enough to maintain one copy of the proof script, and to pass indices into this proof script.

Definition 2. *We define a translation function* T. *For correct proof scripts,* $T(c_1, \ldots, c_p)$ *returns a proof of* $\bot$. *The algorithm* $T(c_1, \ldots, c_p)$ *proceeds by analyzing* c_1 *and by making recursive calls.*

- *If* c_1 *equals* $\mathrm{false}(x)$, *then* $T(c_1) = x$.
- *If* c_1 *has form* $\mathrm{lemma}(a_1, x_1, X_1, \ldots, a_m, x_m, X_m)$, *then first construct* $t := T(c_2, \ldots, c_p)$. *After that,* $T(c_1, \ldots, c_p)$ *equals*

$$(\lambda a_1{:}X_1 \cdots\ a_m{:}X_m\ t) \cdot x_1 \cdot \ldots \cdot x_m.$$

– *If* c_1 *has form* witness$(a, A,\ h, x, (P\ a))$, *first compute* $t := T(c_2, \ldots, c_p)$. *Then* $T(c_1, \ldots, c_p)$ *equals*

$$(x\ (\lambda a{:}A\ \ \lambda h{:}(P\ a)\ \ t)\ ).$$

– *If* c_1 *has form* split(a_1, A_1, a_2, A_2, x), *then there are two* false *statements in* $(c_2, \ldots, c_p)$, *corresponding to the left and to the right branch of the case split. Let* k *be the position of the* false-*statement belonging to the first branch. It can be easily found by walking through the proof script from left to right, and keeping track of the* split *and* false-*statements.*
Then compute $t_1 = T(c_2, \ldots, c_k)$, *and* $t_2 = T(c_{k+1}, \ldots, c_p)$.
The translation $T(c_1, \ldots, c_p)$ *equals*

$$(x\ (\lambda a_1{:}A_1\ \ t_1)\ (\lambda a_2{:}A_2\ \ t_2)\ ).$$

The following theorem is easily proven by induction on the length of the proof script.

Theorem 1. *Let the size of a proof script* $(c_1, \ldots, c_p)$ *be defined as* $|c_1| + \cdots + |c_p|$, *where for each instruction* c_i, *the size* $|c_i|$ *is defined as the sum of the sizes of the terms that occur in it.*
Then $|T(c_1, \ldots, c_p)|$ *is linear in* $|(c_1, \ldots, c_p)|$.

Proof. It can be easily checked that in $T(c_1, \ldots, c_p)$ no component of $(c_1, \ldots, c_p)$ is used more than once.

Theorem 2. *Let* $(c_1, \ldots, c_p)$ *be a proof script. If* $\Gamma \vdash (c_1, \ldots, c_p)$, *then* $\Gamma \vdash t{:}\bot$.

3 Replacement of Equals with Proof Generation

We want to apply the CNF-transformation on some formula F. Let the result be G. We want to construct a proof that G is a correct CNF of F. In the previous section we have seen that it is possible to generate proof script commands that generate a context Γ in which F and G can be proven logically equivalent. (See Definition 1) In this section we discuss the problem of how to prove equivalence of F and G.
Formula G is obtained from F by making a sequence of replacements on subformulas. The replacements made are justified by some equivalance, which then have to lifted into a context by functional reflexivity axioms.

Example 1. Suppose that we want to transform $(A_1 \wedge A_2) \vee B_1 \vee \cdots \vee B_n$ into Clausal Normal Form. We assume that $\vee$ is left-associative and binary. First $(A_1 \wedge A_2) \vee B_1$ has to replaced by $(A_1 \vee B_1) \wedge (A_2 \vee B_1)$. The result is $((A_1 \vee B_1) \wedge (A_2 \vee B_1)) \vee B_2 \vee \cdots \vee B_n$. Then $((A_1 \vee B_1) \wedge (A_2 \vee B_1) \vee B_2)$ is replaced by $(A_1 \vee B_1 \vee B_2) \wedge (A_2 \vee B_1 \vee B_2)$. n such replacements result in the CNF $(A_1 \vee B_1 \vee \cdots \vee B_n) \wedge (A_2 \vee B_1 \vee \cdots \vee B_n)$.
The i-th replacement can be justified by lifting the proper instantiation of the axiom $(P \wedge Q) \vee R \leftrightarrow (P \vee R) \wedge (Q \vee R)$ into the context $(\#) \wedge B_i \wedge \cdots \wedge B_n$. This can be done by taking the right instantiation of the axiom $(P_1 \leftrightarrow Q_1) \rightarrow (P_2 \leftrightarrow Q_2) \rightarrow (P_1 \wedge P_2 \leftrightarrow Q_1 \wedge Q_2)$.

The previous example gives the general principle with which proofs are to be generated. In nearly all cases the replacement can be justified by direct instantiation of an axiom. In most cases the transformations can be specified by a rewrite system combined with a strategy, usually outermost replacement.

In order to make proof generation feasible, two problems need to be solved: The first is the problem that in type theory, it takes quadratic complexity to build up a context. This is easily seen from Example 1. For the first step, the functional reflexivity axiom needs to be applied $n-1$-times. Each time, it needs to be applied on the formula constructed so far. This causes quadratic complexity.

The second problem is the fact that the same context will be built up many times. In Example 1, the first two replacements both take place in context $(\#) \vee B_3 \vee \cdots \vee B_n$. All replacements, except the last take place in context $(\#) \vee B_n$. It is easily seen that in Example 1, the total proof has size $O(n^3)$. The size of the result is only $2n$.

Our solution to the problem is based on two principles: Reducing the redundancy in proof representation, and combination of contexts.

Type theory is extremely redundant. If one applies a proof rule, one has to mention the formulas on which the rule is applied, even though this information can be easily derived. In [4], it has been proposed to obtain proof compression by leaving out redundant information. However, even if one does not store the formulas, they are still generated and compared during proof checking, so the order of proof checking is not reduced. (If one uses type theory. It can be different in other calculi) We solve the redundancy problem by introducing abbreviations for repeated formulas. This has the advantage that the complexity of checking the proof is also reduced, not only of storing.

The problem of repeatedly building up the same context can be solved by first combining proof steps, before building up the context. One could obtain this by tuning the strategy that makes the replacements, but that could be hard for some strategies. Therefore we take another approach. We define a calculus in which repeated constructions of the same context can be normalized away. We call this calculus the *replacement calculus*. Every proof has a unique normal form. When a proof is in normal form, there is no repeated build up of contexts. Therefore, it corresponds to a minimal proof in type theory. The replacement calculus is somewhat related to the rewriting calculus of [7], but it is not restricted to rewrite proofs, although it can be used for rewrite proofs. Another difference is that our calculus is not intended for doing computations, only for concisely representing replacement proofs.

Definition 3. *We recursively define what is a valid replacement proof π in a context Γ. At the same time, we associate an equivalence $\Delta(\pi)$ of form $A \equiv B$ to each valid replacement proof, called the* conclusion *of π.*

- *If formula A is well-typed in context Γ, then* $\mathrm{refl}(A)$ *is a valid proof in the replacement calculus. Its conclusion is $A \equiv A$.*
- *If π_1, π_2 are valid replacemet proofs in context Γ, and there exist formulas A, B, C, s.t. $\Delta(\pi_1)$ equals $(A \equiv B)$, $\Delta(\pi_2)$ equals $(B \equiv C)$, then* $\mathrm{trans}(\pi_1, \pi_2)$ *is a valid replacement proof with conclusion $(A \equiv C)$ in Γ.*

- *If* $\pi_1, \ldots, \pi_n$ *are valid replacement proofs in* Γ*, for which* $\Delta(\pi_1) = (A_1 \equiv B_1), \ldots, \Delta(\pi_n) = (A_n \equiv B_n)$*, both* $f(A_1, \ldots, A_n)$ *and* $f(B_1, \ldots, B_n)$ *are well-typed in* Γ*, then* $\mathrm{func}(f, \pi_1, \ldots, \pi_n)$ *is a valid replacement proof with conclusion* $f(A_1, \ldots, A_n) \equiv f(B_1, \ldots, B_n)$ *in* Γ*.*
- *If* π *is a valid replacement proof in a context of form* $\Gamma, x{:}X$*, with* $\Delta(\pi) = (A \equiv B)$*, the formulas* A, B *are well-typed in context* $\Gamma, x{:}X$*, then* $\mathrm{abstr}(x, X, \pi)$ *is a valid replacement proof, with conclusion* $(\lambda x{:}X\ A) \equiv (\lambda x{:}X\ B)$*.*
- *If* $\Gamma \vdash t{:}A \equiv B$*, then* $\mathrm{axiom}(t)$ *is a valid replacement proof in* Γ*, with conclusion* $A \equiv B$

In a concrete implementation, there probably will be additional constraints. For example use of the refl-, trans-rules will be restricted to certain types. Similarly, use of the func-rule will probably be restricted.

The $\equiv$-relation is intended as an abstraction from the concrete equivalence relation being used. In our situation, $\equiv$ should be read as $\leftrightarrow$ on Prop, and it could be equality on domain elements. In addition, one could have other equivalence relations, for which functional reflexivity axioms exist. (Actually not a full equivalence relation is needed. Any relation that is reflexive, transitive, and that satisfies at least one axiom of form $A \succ B \Rightarrow s(A) \succ s(B)$ could be used)

The abstr-rule is intended for handling quantifiers. A formula of form $\forall x{:}X\ P$ is represented in typetheory by (forall $\lambda x{:}X\ P$). If one wants to make a replacement inside P, one first has to apply the abstr-rule, and then to apply the refl-rule on forall. In order to be able to make such replacements, one needs an additional equivalence relation equivProp, such that (equivProp $P\ Q$) $\rightarrow$ (forall P) $\leftrightarrow$ (forall Q). This can be easily obtained by defining equivProp as $\lambda X{:}\mathrm{Set}\ \lambda P, Q{:}X \rightarrow \mathrm{Prop}\ \forall x{:}X\ (P\ x) \leftrightarrow (Q\ x)$.

We now define two translation functions that translate replacement proofs into type theory proofs. The first function is fairly simple. It uses the method that was used in Example 1. The disadvantage of this method is that the size of the constructed proof term can be quadratic in the size of the replacement proof. On the other hand it is simple, and for some applications it may be good enough. The translation assumes that we have for each type of discourse terms of type refl_X, and trans_X available. In addition, we assume availability of terms of type func_f with obvious types.

Definition 4. *The following axioms are needed for translating proofs of the rewrite calculus into type theory.*

- refl_X *is a proof of* $\Pi x{:}X\ X \equiv X$*.*
- trans_X *is a proof of* $\Pi x_1, x_2, x_3{:}X\ \ x_1 \equiv x_2 \rightarrow x_2 \equiv x_3 \rightarrow x_1 \equiv x_3$*.*
- func_f *is a proof of* $\Pi x_1, y_1{:}X_1\ \cdots \Pi x_n, y_n{:}X_n\ \ x_1 \equiv y_1 \rightarrow \cdots \rightarrow x_n \equiv y_n \rightarrow (f\ x_1 \cdots x_n) \equiv (f\ y_1 \cdots y_n)$*. Here* $X_1, \ldots, X_n$ *are the types of the arguments of* f*.*

Definition 5. *Let* π *be a valid replacement proof in context* Γ*. We define translation function* $T(\pi)$ *by recursion on* π*.*

- $T(\mathrm{refl}(A)\ )$ *equals* $(\mathrm{refl}_X\ A)$, *where* X *is the type of* A.
- $T(\mathrm{trans}(\pi_1, \pi_2)\ )$ *equals as* $(\mathrm{trans}_X\ A\ B\ C\ T(\pi_1)\ T(\pi_2)\ )$, *where* A, B, C *are defined from* $\Delta(\pi_1) = (A \equiv B)$ *and* $\Delta(\pi_2) = (B \equiv C)$.
- $T(\mathrm{func}(f, \pi_1, \ldots, \pi_n)\ )$ *is defined as* $(\mathrm{func}_f\ A_1\ B_1 \cdots\ A_n\ B_n\ T(\pi_1) \cdots T(\pi_n)\ )$, *where* A_i, B_i *are defined from* $\Delta(\pi_i) = (A_i \equiv B_i)$, *for* $1 \leq i \leq n$.
- $T(\mathrm{abstr}(x, X, \pi)\ )$ *is defined as* $(\mathrm{abstr}_X\ (\lambda x{:}X\ A)\ (\lambda x{:}X\ B)\ (\lambda x{:}X\ T(\pi)\ )\ )$, *where* A, B *are defined from* $\Delta(\pi) = (A \equiv B)$.
- $T(\mathrm{axiom}(t))$ *is defined simply as* t.

Theorem 3. *Let* π *be a valid replacement proof in context* Γ. *Then* $|T(\pi)| = O(|\pi|^2)$.

Proof. The quadratic upperbound can be shown by induction. That this upperbound is also a lowerbound was demonstrated in Example 1.

Next we define an improved translation function that constructs a proof of size linear in the size of the replacement proof. The main idea is to introduce definitions for all subformulas. In this way, the iterated built-ups of subformulas can be avoided. In order to introduce the definitions, proof scripts with lemma-instructions are constructed simultaneously with the translations.

Definition 6. *Let* π *be a valid replacement proof in context* Γ. *The improved translation function* $T(\pi)$ *returns a quadruple* (Σ, t, A, B), *where* Σ *is a proof script and* t *is a term such that* $\Gamma, \Sigma \vdash t{:}A \equiv B$. *(The notation* Γ, Σ *means:* Γ *extended with the definitions induced by* Σ*)*

- $T(\mathrm{refl}(A)\ )$ *equals* $(\emptyset, (\mathrm{refl}_X\ A), A, A\ )$, *where* X *is the type of* A.
- $T(\mathrm{trans}(\pi_1, \pi_2)\ )$ *is defined as* $(\Sigma_1 \cup \Sigma_2, (\mathrm{trans}_X\ A\ B\ C\ t_1\ t_2), A, C)$, *where* $\Sigma_1, \Sigma_2, t_1, t_2, A, C$ *are defined from*

$$T(\pi_1) = (\Sigma_1, t_1, A, B), \quad T(\pi_2) = (\Sigma_2, t_2, B, C).$$

- $T(\mathrm{func}(f, \pi_1, \ldots, \pi_n)\ )$ *is defined as*

$$(\Sigma_1 \cup \cdots \cup \Sigma_n \cup \Sigma, (\mathrm{func}_f\ A_1\ B_1 \cdots\ A_n\ B_n\ t_1 \cdots\ t_n), x_1, x_2),$$

  where, for i *with* $1 \leq i \leq n$, *the* Σ_i, A_i, B_i, t_i *are defined from*

$$T(\pi_i) = (\Sigma_i, t_i, A_i, B_i).$$

  Both x_1, x_2 *are new atoms, and* Σ *is defined from*

$$\Sigma = \{\mathrm{lemma}(x_1, (f\ A_1\ \cdots\ A_n), X),\ \mathrm{lemma}(x_2, (f\ B_1\ \cdots\ B_n), X)\},$$

  where X *is the common type of* $(f\ A_1 \cdots A_n)$ *and* $(f\ B_1 \cdots B_n)$.
- $T(\mathrm{abstr}(x, X, \pi)\ )$ *is defined as*

$$(\Sigma \cup \Theta, (\mathrm{abstr}_X\ (\lambda x{:}X\ A)\ (\lambda x{:}X\ B)\ (\lambda x{:}X\ t), x_1, x_2),$$

  where Σ, t, A, B *are defined from* $T(\pi) = (\Sigma, t, A, B)$. *The* x_1, x_2 *are new atoms, and*

$$\Theta = \{\mathrm{lemma}(x_1, (\lambda x{:}X\ A), X \to Y),\ \mathrm{lemma}(x_2, (\lambda x{:}X\ B), X \to Y)\}.$$

– $T(\text{axiom } t)$ *is defined as* $(\emptyset, t, A, B)$, *where* A, B *are defined from* $\Gamma \vdash t{:}A \equiv B$.

Definition 7. *We define the following reduction rules on replacement proofs. Applying* trans *on a* refl*-proof does not change the equivalence being proven:*

– $\text{trans}(\pi, \text{refl}(A)) \Rightarrow \pi$,
– $\text{trans}(\text{refl}(A), \pi) \Rightarrow \pi$.

The trans*-rule is associative. The following reduction groups* trans *to the left:*

– $\text{trans}(\pi, \text{trans}(\rho, \sigma)) \Rightarrow \text{trans}(\text{trans}(\pi, \rho), \sigma)$.

If the func*-rule, or the* abstr*-rule is applied only on* refl*-rules, then it proves an identity. Because of this, it can be replaced by one* refl*-application.*

– $\text{func}(f, \text{refl}(A_1), \ldots, \text{refl}(A_n)) \Rightarrow \text{refl}(f(A_1, \ldots, A_n))$.
– $\text{abstr}(x, X, \text{refl}(A)) \Rightarrow \text{refl}(\lambda x{:}X\ A)$.

The following two reduction rules are the main ones. If a trans*-rule, or an* abstr*-rule is applied on two proofs that build up the same context, then the context building can be shared:*

– $\text{trans}(\text{func}(f, \pi_1, \ldots, \pi_n), \text{func}(f, \rho_1, \ldots, \rho_n)) \Rightarrow$
$\text{func}(f, \text{trans}(\pi_1, \rho_1), \ldots, \text{trans}(\pi_n, \rho_n))$.
– $\text{trans}(\text{abstr}(x, X, \pi), \text{abstr}(x, X, \rho)) \Rightarrow \text{abstr}(x, X, \text{trans}(\pi, \rho)\ )$.

Theorem 4. *The rewrite rules of Definition 7 are terminating. Moreover, they are confluent. For every proof* π, *the normal form* π' *corresonds to a type-theory proof of minimal complexity.*

Now a proof can be generated naively in the replacment calculus, after that it can be normalized, and from that, a type theory proof can be generated.

4 Skolemization Issues

We discuss the problem of generating proofs from Skolemization steps. Witness-instructions can be used to introduce the Skolem functions into the proof scripts, see Definition 1. The wittness-instructions can be justified by either a choice axiom or by the ϵ-function.

It would be possible to completely eliminate the Skolem-functions from the proof, but we prefer not to do that for efficiency reasons. Elimination of Skolem-functions may cause hyperexponential increase of the size of the proof, see [2]. This would make proof generation not feasible. However, we are aware of the fact that for some applications, it may be necessary to perform the elimination of Skolem functions. Methods for doing this have been studied in [9] and [14]

It is straightforward to handle standard Skolemization using of a witness-instruction. However, several improved Skolemization methods have been proposed, in particular *optimized Skolemization* [13] and *strong Skolemization.* (see

[11] or [12]) Experiments show that such improved Skolemization methods do improve the chance of finding a proof. Therefore, we need to be able to handle these methods. In order to obtain this, we will show that both strong and optimized Skolemization can be reduced to standard Skolemization. Formally this means the following: For every first-order formula F, there is a first-order formula F', which is first-order equivalent to F, such that the standard Skolemization of F' equals the strong/optimized Skolemization of F. Because of this, no additional choice axioms are needed to generate proofs from optimized or strong Skolemization steps. An additional consequence of our reduction is that the Skolem-elimination techniques of [9] and [14] can be applied to strong and optimized Skolemization as well, without much difficulty.

The reductions proceed through a new type of Skolemization that we call *stratified* Skolemization. Both strong and improved Skolemization can be reduced to stratified Skolemization (in the way that we defined a few lines above). Stratified Skolemization in its turn can be reduced to standard Skolemization. This solves the question that was asked in the last line of [11] whether or not it is possible to unify strong and optimized Skolemization.
We now repeat the definitions of inner and outer Skolemization, which are standard. (Terminology from [12]) After that we give the definitions of strong and optimized Skolemization.

Definition 8. *Let F be a formula in* NNF. *Skolemization replaces an outermost existential quantifier by a new function symbol. We define four types of Skolemization. In order to avoid problems with variables, we assume that F is standardized apart. Write $F = F[\ \exists y{:}Y\ A,]$, where $\exists y{:}Y\ A$ is not in the scope of another existential quantifier. We first define outer Skolemization, after that we define the three other type of Skolemization.*

Outer Skolemization *Let $x_1, \ldots, x_p$ be the variables belonging to the universal quantifiers which have $\exists y{:}Y\ A$ in their scope. Let $X_1, \ldots, X_p$ be the corresponding types. Let f be a new function symbol of type $X_1 \to \cdots \to X_p \to Y$. Then replace $F[\exists y{:}Y\ A]$ by $F[A\ [y := (f\ x_1 \cdots x_p)]\ ]$.*

With the other three types of Skolemization, the Skolem functions depend only on the universally quantified variables that actually occur in A. Let $x_1, \ldots, x_p$ be the variables that belong to the universal quantifiers which have A in their scope, and that are free in A. The $X_1, \ldots, X_p$ are the corresponding types.

Inner Skolemization *Inner Skolemization is defined in the same way as outer Skolemization, but it uses the improved $x_1, \ldots, x_p$.*

Strong Skolemization *Strong Skolemization can be applied only if formula A has form $A_1 \wedge \cdots \wedge A_q$ with $q \geq 2$. For each k, with $1 \leq k \leq q$, we first define the sequence of variables $\overline{\alpha}_k$ as those variables from $(x_1, \ldots, x_p)$ that do not occur in $A_k \wedge \cdots \wedge A_q$. It can be easily checked that for $1 \leq k < q$, sequence $\overline{\alpha}_k$ is a subsequence of $\overline{\alpha}_{k+1}$.*
For each k with $1 \leq k \leq q$, write $\overline{\alpha}_k$ as $(v_{k,1}, \ldots, v_{k,l_k})$. Write $(V_{k,1}, \ldots, V_{k,l_k})$ for the corresponding types. Define the functions $\overline{Q}_k$ from

$$\overline{Q}_k(Z) = \forall v_{k,1}{:}V_{k,1} \cdots\ \forall v_{k,l_k}{:}V_{k,l_k}\ (Z),$$

It is intended that the quantifiers $\forall v_{k,j}{:}V_{k,j}$ *will capture the free atoms of* Z*. Let* f *be new function symbol of type* $X_1 \to \cdots \to X_p \to Y$*. For each* k*, with* $1 \leq k \leq q$*, define* $B_k = A_k[y := (f\ x_1 \cdots x_p)]$*. Finally replace*

$$F[\exists y{:}Y\ (A_1 \wedge A_2 \wedge \cdots \wedge A_q)] \text{ by } F[\overline{Q}_1(B_1) \wedge \overline{Q}_2(B_2) \wedge \cdots \wedge \overline{Q}_q(B_q)].$$

Optimized Skolemization *Formula* A *must have form* $A_1 \wedge A_2$*, and* F *must have form* $F_1 \wedge \cdots \wedge F_q$*, where one of the* $F_k,\ 1 \leq k \leq q$ *has form*

$$F_k = \forall x_1{:}X_1\ \forall x_2{:}X_2 \cdots \forall x_p{:}X_p\ \exists y{:}Y\ A_1.$$

If this is the case, then $F[\ \exists y{:}Y\ (A_1 \wedge A_2)]$ *can be replaced by the formula*

$$F_k[A_2[y := (f\ x_1 \cdots x_p)]\ ],$$

and F_k *can be simultaneously replaced by the formula*

$$\forall x_1{:}X_1\ \forall x_2{:}X_2\ \cdots\ \forall x_p{:}X_p\ A_1[y := (f\ x_1 \cdots x_p)].$$

If F *is not a conjunction or does not contain an* F_k *of the required form, but it does imply such a formula, then optimized Skolemization can still be used. First replace* F *by* $F \wedge \forall x_1{:}X_1\ \forall x_2{:}X_2 \cdots \forall x_p{:}X_p\ \exists y{:}Y\ A_1$*, and then apply optimized Skolemization.*

As said before, choice axioms or ϵ-functions can be used in order to justify the wittness-instructions that introduce the Skolem-funcions. This is straightforward, and we omit the details here.

In the rest of this section, we study the problem of generating proofs for optimized and strong Skolemization. We want to avoid introducing additional axioms, because strong Skolemization has too many parameters. (The number of conjuncts, and the distribution of the $x_1, \ldots, x_p$ through the conjuncts). We will obtain this by reducing strong and optimized Skolemization to inner Skolemization. The reduction proceeds through a new type of Skolemization, which we call *Stratified Skolemization*. We show that Stratified Skolemization can be obtained from inner Skolemization in first-order logic. In the process, we answer a question asked in [11], whether or not there a common basis in strong and optimized Skolemization.

Definition 9. *We define* stratified Skolemization*. Let* F *be some first-order formula in negation normal form. Assume that* F *contains a conjunction of the form* $F_1 \wedge \cdots \wedge F_q$ *with* $2 \leq q$*, where each* F_k *has form*

$$\forall x_1{:}X_1 \cdots\ x_p{:}X_p\ (C_k \to \exists y{:}Y\ A_1 \wedge \cdots \wedge A_k).$$

The C_k *and* A_k *are arbitrary formulas. It is assumed that the* F_k *have no free variables. Furthermore assume that for each* $k,\ 1 \leq k < q$*, the following formula is provable:*

$$\forall x_1{:}X_1 \cdots\ x_p{:}X_p\ (C_{k+1} \to C_k).$$

Then $F[\ F_1 \wedge \cdots \wedge F_q]$ *can be Skolemized into* $F[F'_1 \wedge \cdots \wedge F'_q]$*, where each* $F'_k,\ 1 \leq k \leq q$ *has form*

$$\forall x_1{:}X_1 \cdots\ x_p{:}X_p\ (C_k \to A_k[\ y := (f\ x_1 \cdots\ x_p)\ ]\ ).$$

As with optimized and strong Skolemization, it is possible to Skolemize more than one existential quantifier at the same time. Stratified Skolemization improves over standard Skolemization by the fact that it allows to use the same Skolem-function for existential quantifiers, which is an obvious improvement. In addition, it is allowed to drop all but the last members from the conjunctions on the righthandsides. It is not obvious that this is an improvement.

The $C_1, \ldots, C_q$ could be replaced by any context through a subformula replacement.

We now show that stratified Skolemization can be reduced to inner Skolemization. This makes it possible to use a standard choice axiom for proving the correctness of a stratified Skolemization step.

Theorem 5. *Stratified Skolemization can be reduced to inner Skolemization in first-order logic. More precisely, there exists a formula G, such that F is logically equivalent to G in first-order logic, and the stratified Skolemization of F equals the inner Skolemization of G.*

Proof. Let $F_1, \ldots, F_q$ be defined as in Definition 9. Without loss of generality, we assume that F is equal to $F_1 \wedge \cdots \wedge F_q$. The situation where F contains $F_1 \wedge \cdots \wedge F_q$ as a subformula can be easily obtained from this.

For G, we take

$$\forall x_1{:}X_1 \cdots \forall x_p{:}X_p\ \exists y{:}Y\ (C_1 \rightarrow A_1) \wedge \cdots \wedge (C_q \rightarrow A_q).$$

It is easily checked that the inner Skolemization of G equals the stratified Skolemization of F, because y does not occur in the C_k.
We will show that for all $x_1, \ldots, x_p$, the instantiated formulas are equivalent, so we need to prove for abitrary $x_1, \ldots, x_p$,

$$\bigwedge_{k=1}^{q} C_k \rightarrow \exists y{:}Y\ (A_1 \wedge \cdots \wedge A_k) \Leftrightarrow \exists y{:}Y\ \bigwedge_{k=1}^{q} (C_k \rightarrow A_k).$$

We will use the abbreviation LHS for the left hand side, and RHS for the right hand side.
Define $D_0 = \neg C_1 \wedge \cdots \wedge \neg C_q$.
For $1 < k < q$, define

$$D_k = C_1 \wedge \cdots \wedge C_k \wedge \neg C_{k+1} \wedge \cdots \wedge \neg C_q.$$

Finally, define $D_q = C_1 \wedge \cdots \wedge C_q$.
It is easily checked that $(C_2 \rightarrow C_1) \wedge \cdots \wedge (C_q \rightarrow C_{q-1})$ implies $D_0 \vee \cdots \vee D_q$.

Assume that the LHS holds. We proceed by case analysis on $D_0 \vee \cdots \vee D_q$. If D_0 holds, then RHS can be easily shown for an arbitrary y. If a D_k with $k > 0$ holds, then C_k holds. It follows from the k-th member of the LHS, that there is a y such that the $A_1, \ldots, A_k$ hold. Since $k' > k$ implies $\neg C_{k'}$, the RHS can be proven by chosing the same y.

Now assume that the RHS holds. We do another case analysis on $D_0 \vee \cdots \vee D_q$. Assume that D_k holds, with $0 \leq k \leq q$.

For $k' > k$, we then have $\neg C_{k'}$. There is a $y{:}Y$, such that for all $k' \leq k$, $A_{k'}$ holds. Then the LHS can be easily proven by choosing the same y in each of the existential quantifiers.

Theorem 6. *Optimized Skolemization can be trivially obtained from stratified Skolemization.*

Proof. Take $q = 2$ and take for C_1 the universally true predicate.

Theorem 7. *Strong Skolemization can be obtained from stratified Skolemization in first-order logic.*

Proof. We want to apply strong Skolemization on the following formula

$$\forall x_1{:}X_1 \cdots \forall x_p{:}X_p \ (C \ x_1 \ \cdots \ x_p) \rightarrow \exists y{:}Y \ \ A_1 \wedge \cdots \wedge A_q.$$

For sake of clarity, we write the variables in C explicitly. First reverse the conjunction into

$$\forall x_1{:}X_1 \cdots \forall x_p{:}X_p \ (C \ x_1 \cdots \ x_p) \rightarrow \exists y{:}Y \ \ A_q \wedge \cdots \wedge A_1.$$

Let $\overline{\alpha}_1, \ldots, \overline{\alpha}_q$ be defined as in Definition 8. The fact that A_k does not contain the variables in $\overline{\alpha}_k$ can be used for weakening the assumptions $(C \ x_1 \cdots \ x_p)$ as follows:

$$\bigwedge_{k=q}^{1} \forall x_1{:}X_1 \cdots \forall x_p{:}X_p \ \ [\ \exists \overline{\alpha}_k \ (C \ x_1 \cdots x_p)\ ] \rightarrow \exists y{:}Y \ A_q \wedge \cdots \wedge A_k.$$

Note that k runs backwards from q to 1. Because $\overline{\alpha}_k \subseteq \overline{\alpha}_{k+1}$, we have $\exists \overline{\alpha}_k \ (C \ x_1 \cdots x_p)$ implies $\exists \overline{\alpha}_{k+1} \ (C \ x_1 \cdots x_p)$. As a consequence, stratified Skolemization can be applied. The result is:

$$\bigwedge_{k=q}^{1} \forall x_1{:}X_1 \cdots \forall x_p{:}X_p \ [\ \exists \overline{\alpha}_k \ (C \ x_1 \cdots x_p)\ ] \rightarrow A_k[y := (f \ x_1 \cdots x_p)\ ].$$

For each k with $1 \leq k \leq$, let $\overline{\beta}_k$ be the variables of $(x_1, \ldots, x_p)$ that are not in $\overline{\alpha}_k$. Then the formula can be replaced by

$$\bigwedge_{k=q}^{1} \forall \overline{\alpha}_k \ \forall \overline{\beta}_k \ [\ \exists \overline{\alpha}_k \ (C \ x_1 \cdots x_p)\ ] \rightarrow A_k[y := (f \ x_1 \cdots x_p)\ ].$$

This can be replaced by

$$\bigwedge_{k=q}^{1} \forall \overline{\beta}_k \ [\ \exists \overline{\alpha}_k \ (C \ x_1 \cdots x_p)\ ] \rightarrow \forall \overline{\alpha}_k \ A_k[y := (f \ x_1 \cdots x_p)\ ],$$

which can in turn be replaced by

$$\bigwedge_{k=q}^{1} \forall \overline{\beta}_k \ \forall \overline{\alpha}_k \ (C \ x_1 \cdots x_p) \rightarrow \forall \overline{\alpha}_k \ \ A_k[y := (f \ x_1 \cdots x_p)\ ],$$

The result follows immediately.

It can be concluded that strong and optimized Skolemization can be reduced to Stratified Skolemization, which in its turn can be reduced to inner Skolemization. It is an interesting question whether or not Stratified Skolemization has useful applications on its own. We intend to look into this.

5 Conclusions

We have solved the main problems of proof generation from the clausal normal form transformation. Moreover, we think that our techniques are wider in scope: They can be used everywhere, where explicit proofs in type theory are constructed by means of rewriting, automated theorem proving, or modelling of computation.

We also reduced optimized and strong Skolemization to standard Skolemization. In this way, only standard choice axioms are needed for translating proofs involving these forms of Skolemization. Alternatively, it has become possible to remove applications of strong and optimized Skolemization commpletely from a proof.

We do intend to implement a clausal normal tranformer, based on the results in this paper. The input is a first-order formula. The output will be the clausal normal form of the formula, together with a proof of its correctness.

References

[1] Matthias Baaz, Uwe Egly, and Alexander Leitsch. Normal form transformations. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 5, pages 275–333. Elsevier Science B. V., 2001. 584

[2] Matthias Baaz and Alexander Leitsch. On skolemization and proof complexity. *Fundamenta Informatika*, 4(20):353–379, 1994. 592

[3] Henk Barendregt and Herman Geuvers. Proof-assistents using dependent type systems. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 18, pages 1151–1238. Elsevier Science B. V., 2001. 585

[4] Stefan Berghofer and Tobias Nipkow. Proof terms for simply typed higher order logic. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher-Order Logics, TPHOLS 2000*, volume 1869 of *LNCS*, pages 38–52. Springer Verlag, 2000. 589

[5] Marc Bezem, Dimitri Hendriks, and Hans de Nivelle. Automated proof construction in type theory using resolution. In David McAllester, editor, *Automated Deduction - CADE-17*, number 1831 in LNAI, pages 148–163. Springer Verlag, 2000. 584

[6] Samuel Boutin. Using reflection to build efficient and certified decision procedures. In Martín Abadi and Takayasu Ito, editors, *Theoretical Aspects of Computer Software (TACS)*, volume 1281 of *LNCS*, pages 515–529, 1997. 585

[7] Horatiu Cirstea and Claude Kirchner. The rewriting calculus, part 1 + 2. *Journal of the Interest Group in Pure and Applied Logics*, 9(3):339–410, 2001. 589

[8] Hans de Nivelle. A resolution decision procedure for the guarded fragment. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction- CADE-15*, volume 1421 of *LNCS*, pages 191–204. Springer, 1998. 584

[9] Xiaorong Huang. Translating machine-generated resolution proofs into ND-proofs at the assertion level. In Norman Y. Foo and Randy Goebel, editors, *Topics in Artificial Intelligence, 4th Pacific Rim International Conference on Artificial Intelligence*, volume 1114 of *LNCS*, pages 399–410. Springer Verlag, 1996. 592, 593

[10] William McCune and Olga Shumsky. Ivy: A preprocessor and proof checker for first-order logic. In Matt Kaufmann, Pete Manolios, and J. Moore, editors, *Using the ACL2 Theorem Prover: A tutorial Introduction and Case Studies*. Kluwer Academic Publishers, 2002? preprint: ANL/MCS-P775-0899, Argonne National Labaratory, Argonne. 584

[11] Andreas Nonnengart. Strong skolemization. Technical Report MPI-I-96-2-010, Max Planck Institut für Informatik Saarbrücken, 1996. 585, 593, 594

[12] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 6, pages 335–367. Elsevier Science B. V., 2001. 584, 593

[13] Hans Jürgen Ohlbach and Christoph Weidenbach. A note on assumptions about skolem functions. *Journal of Automated Reasoning*, 15:267–275, 1995. 585, 592

[14] Frank Pfenning. Analytic and non-analytic proofs. In Robert E. Shostak, editor, *7th International Conference on Automated Deduction CADE 7*, volume 170 of *LNCS*, pages 394–413. Springer Verlag, 1984. 592, 593

[15] Frank Pfenning. Logical frameworks. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 17, pages 1065–1148. Elsevier Science B. V., 2001. 585

[16] R. Sekar, I. V. Ramakrishnan, and Andrei Voronkov. Term indexing. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, chapter 26, pages 1853–1964. Elsevier Science B. V., 2001. 584

[17] Christoph Weidenbach. The spass homepage. `http://spass.mpi-sb.mpg.de/`. 586

[18] Christoph Weidenbach. Combining superposition, sorts and splitting. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science B. V., 2001. 586

Resolution Refutations and Propositional Proofs with Height-Restrictions

Arnold Beckmann*

Institute of Algebra and Computational Mathematics
Vienna University of Technology
Wiedner Hauptstr. 8-10/118, A-1040 Vienna, Austria
arnold.beckmann@logic.at

Abstract. Height restricted resolution (proofs or refutations) is a natural restriction of resolution where the height of the corresponding proof tree is bounded. Height restricted resolution does not distinguish between tree- and sequence-like proofs. We show that polylogarithmic-height resolution is strongly connected to the bounded arithmetic theory $S^1_2(\alpha)$. We separate polylogarithmic-height resolution from quasi-polynomial size tree-like resolution.
Inspired by this we will study infinitely many sub-linear-height restrictions given by functions $n \mapsto 2_i\left((\log^{(i+1)} n)^{O(1)}\right)$ for $i \geq 0$. We show that the resulting resolution systems are connected to certain bounded arithmetic theories, and that they form a strict hierarchy of resolution proof systems. To this end we will develop some proof theory for height restricted proofs.

Keywords: Height of proofs; Length of proofs; Resolution refutation; Propositional calculus; Frege systems; Order induction principle; Cut elimination; Cut introduction; Bounded arithmetic.

MSC: Primary 03F20; Secondary 03F07, 68Q15, 68R99.

1 Introduction

In this article, we will focus on two approaches to the study of computational complexity classes, *propositional proof systems* and *bounded arithmetic theories*. Cook and Reckhow in their seminal paper [8] have shown that the existence of "strong" *propositional proof systems* in which all tautologies have proofs of polynomial size is tightly connected to the NP vs. co-NP question. This has been the starting point for a currently very active area of research where one tries to separate all kinds of proof systems by proving super-polynomial lower bounds.

Theories of bounded arithmetic have been introduced by Buss in [6]. They are logical theories of arithmetic where formulas and induction are restricted (bounded) in such a way that provability in those theories can be tightly connected to complexity classes (cf. [6, 12]). A hierarchy of bounded formulas, Σ^b_i,

* Supported by a Marie Curie Individual Fellowship #HPMF-CT-2000-00803 from the European Commission.

J. Bradfield (Ed.): CSL 2002, LNCS 2471, pp. 599–612, 2002.
© Springer-Verlag Berlin Heidelberg 2002

and of theories $S_2^1 \subseteq T_2^1 \subseteq S_2^2 \subseteq T_2^2 \subseteq S_2^3 \ldots$ has been defined (cf. [6]). The class of predicates definable by Σ_i^b formulas is precisely the class of predicates in the ith level Σ_i^p of the polynomial hierarchy. The Σ_i^b-definable functions of S_2^i form precisely the ith level $\square_i^p$ of the polynomial hierarchy of functions, which consists of the functions which are polynomial time computable with an oracle from Σ_{i-1}^p.

It is an open problem of bounded arithmetic whether the hierarchy of theories collapses. This is connected with the open problem of complexity theory whether the polynomial hierarchy PH collapses – the P=?NP problem is a subproblem of this. The hierarchy of bounded arithmetic collapses if and only if PH collapses provably in bounded arithmetic (cf. [14, 7, 18]). The case of relativized complexity classes and theories behaves completely differently. The existence of an oracle A is proven in [1, 17, 9], such that the polynomial hierarchy in this oracle PH^A does not collapse, hence in particular $\mathrm{P}^A \neq \mathrm{NP}^A$ holds. Building on this one can show $T_2^i(\alpha) \neq S_2^{i+1}(\alpha)$ [14]. Here, the relativized theories $S_2^i(\alpha)$ and $T_2^i(\alpha)$ result from S_2^i and T_2^i, resp., by adding a free set variable α and the relation symbol $\in$. Similarly also, $S_2^i(\alpha) \neq T_2^i(\alpha)$ is proven in [10], and separation results for further relativized theories (dubbed $\Sigma_n^b(\alpha)$-$\mathrm{L}^m\mathrm{IND}$) are proven in [16]. Independently of these, and with completely different methods, we have shown separation results for relativized theories of bounded arithmetic using as method called *dynamic ordinal analysis* [2, 3]. Despite all answers in the relativized case, all separation questions continue to be open for theories without set parameters.

Propositional proof systems and bounded arithmetic theories are connected. For example, Paris and Wilkie have shown in [15] that the study of constant-depth propositional proofs is relevant to bounded arithmetic. In particular, the following translations are known for the first two levels of bounded arithmetic $S_2^1(\alpha)$ and $T_2^1(\alpha)$ (a definition of these theories can be found e.g. in [6, 12]). Krajíček has observed (cf. [13, 3.1]) that provability in $T_2^1(\alpha)$ translates to *quasi-polynomial*[1] *size sequence-like* resolution proofs. Furthermore, it is known that provability in $S_2^1(\alpha)$ translates to *quasi-polynomial size tree-like* resolution proofs.[2] It is also known that quasi-polynomial size tree-like resolution proofs are separated from quasi-polynomial size sequence-like resolution proofs (the best known separation can be found in [5]).

An examination of dynamic ordinal analysis (cf. [2, 3]) shows that provability in $S_2^1(\alpha)$ can even be translated to *polylogarithmic*[3]*-height* resolution proofs. We will prove that polylogarithmic-height resolution proofs form a proper subsystem of quasi-polynomial size tree-like resolution proofs. Hence we will obtain the relationships represented in Fig. 1.

In this article we pick up this observation and examine height restricted propositional proofs and refutations. To this end we develop some proof theory

[1] A function $f(n)$ grows quasi-polynomial (in n) iff $f(n) \in 2^{(\log n)^{O(1)}}$.

[2] The author of this paper could not find a reference for this, but it follows by similar calculations as in [13, 3.1].

[3] A function $f(n)$ grows polylogarithmic (in n) iff $f(n) \in (\log n)^{O(1)}$.

$$
\begin{array}{c}
S_2^1(\alpha) \hookrightarrow \text{polylogarithmic-height resolution} \\
\not\cap \\
\text{quasi-polynomial-size tree-like resolution} \\
\not\cap \\
T_2^1(\alpha) \hookrightarrow \text{quasi-polynomial-size sequence-like resolution}
\end{array}
$$

Fig. 1. Translation of $S_2^1(\alpha)$ and $T_2^1(\alpha)$ to resolution

for height restricted propositional proofs. This includes several cut elimination results, and the following so called boundedness theorem (cf. [4]): Any resolution proof of the order induction principle for n, i.e. for the natural ordering of numbers less than n, must have height at least n. On the other hand there are tree-like resolution proofs of the order induction principle for n which have height linear in n and size quadratic in n. This gives us the separation of polylogarithmic-height resolution from quasi-polynomial size tree-like resolution. In particular, we obtain simple proofs of separation results of relativized theories of bounded arithmetic which reprove some separation results mentioned before.

This way we will study infinitely many sub-linear-height restrictions given by functions $n \mapsto 2_i\left((\log^{(i+1)} n)^{O(1)}\right)$ for $i \geq 0$. We will show that the resulting resolution systems are connected to certain bounded arithmetic theories $\Sigma_{i+1}^b(\alpha)$-$\mathrm{L}^{i+1}\mathrm{IND}$ (a definition of these theories can be found e.g. in [2, 3]), and that they form a strict hierarchy of resolution proof systems utilizing the order induction principle.

The paper is organized as follows: In the next section we recall the definition of the proof system LK. We introduce an inductively defined provability predicate for LK which measures certain parameters of proofs. Furthermore, we introduce the order induction principle for n and give suitable resolution proofs of height linear in n and size quadratic in n. We recall the lower bound (linear in n) to the height of resolution *proofs* of the order induction principle for n, and we give a proof for the lower bound to the height of resolution *refutations* of that principle. In section 3 we develop some proof theory for height restricted propositional proofs. This includes several cut elimination techniques. We further recall the translation from bounded arithmetic to height restricted resolution from [2]. We conclude this section by stating the relationship between resulting height restricted resolution systems. The last section gives an attemp to prove simulations between height restricted LK systems with different so called Σ-depths. The Σ-depth of an LK-proof restricts the depth of principle formulas in cut-inferences. Cut elimination lowers the Σ-depth but raises the height of proofs. For the opposite effect (shrinking height by raising Σ-depth) we introduce some form of cut-introduction. We end this section by some final remarks and open problems.

2 The Proof System LK

We recall the definition of language and formulas of LK from [11]. LK consists of constants $0, 1$, propositional variables $p_0, p_1, p_2 \ldots$ (also called atoms; we may use $x, y, \ldots$ as meta-symbols for variables), the connectives negation $\neg$, conjunction $\bigwedge$ and disjunction $\bigvee$ (both of unbounded finite arity), and auxiliary symbols like brackets. Formulas are defined inductively: constants, atoms and negated atoms are formulas (they are called literals), and if φ_i is a formula for $i < I$, so are $\bigwedge_{i<I} \varphi_i$ and $\bigvee_{i<I} \varphi_i$. $\neg\varphi$ is an abbreviation of the formula formed from φ by interchanging $\bigwedge$ and $\bigvee$, 0 and 1, and atoms and their negations. The *logical depth*, or just *depth*, $\mathrm{dp}(\varphi)$ of a formula φ, is the maximal nesting of $\bigwedge$ and $\bigvee$ in it. In particular, constants and atoms have depth 0, the depths of φ and $\neg\varphi$ are equal, and $\mathrm{dp}(\bigvee_{i<I} \varphi_i)$ equals $1 + \max_{i<I} \mathrm{dp}(\varphi_i)$.

In our setting, *cedents* $\Gamma, \Delta, \ldots$ are finite *sets* of formulas, not *sequences* as in [11], and the meaning of a cedent Γ is $\bigvee \Gamma$. Cedents are also called *clauses* (in case of resolution). We often abuse notation by writing Γ, φ or $\Gamma \vee \varphi$ instead of $\Gamma \cup \{\varphi\}$, or by writing $\varphi_1, \ldots, \varphi_k$ instead of $\{\varphi_1, \ldots, \varphi_k\}$.

Our version of LK does not have structural rules as special inferences, they will be available as derivable rules. LK consists of four inference rules: cut-rule, initial cedent rule, and introduction rules for $\bigwedge$ and $\bigvee$. We define a derivability predicate $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$ meaning that there is a proof of Γ which may use axioms from $\mathcal{A}$ such that the height of the proof-tree is bounded by η, the size (= number of occurrences of cedents in it) is bounded by σ, and the number of formulas $|\Gamma|$ of every cedent Γ in it is bounded by λ.

Definition 1. *We inductively define $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$ for $\mathcal{A}$ is a set of cedents consisting only of literals, Γ a cedent, $\mathcal{C}$ a set of formulas and natural numbers η, σ, λ. $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$ holds iff*

(Init) *$\eta \geq 0$, $\sigma \geq 1$, $\lambda \geq |\Gamma|$ and Γ is an initial cedent, i.e. $1 \in \Gamma$, or $x, \neg x \in \Gamma$ for some variable x, or there is some $\Gamma' \subseteq \Gamma$ such that $\Gamma' \in \mathcal{A}$.*

($\bigwedge$) *There are some $\bigwedge_{i<I} \varphi_i \in \Gamma$, $\eta' < \eta$, $\sigma_i \in \mathbb{N}$ for $i < I$ with $\sum_{i<I} \sigma_i < \sigma$ such that $\mathcal{A} \vdash^{\eta',\sigma_i,\lambda}_{\mathcal{C}} \Gamma, \varphi_i$ for all $i < I$.*

($\bigvee$) *There are some $\bigvee_{i<I} \varphi_i \in \Gamma$, $i_0 < I$, $\eta' < \eta$ and $\sigma' < \sigma$ such that $\mathcal{A} \vdash^{\eta',\sigma',\lambda}_{\mathcal{C}} \Gamma, \varphi_{i_0}$.*

(Cut) *There are some $\varphi \in \mathcal{C}$, $\eta' < \eta$, $\sigma_0 + \sigma_1 < \sigma$ such that $\mathcal{A} \vdash^{\eta',\sigma_0,\lambda}_{\mathcal{C}} \Gamma, \varphi$ and $\mathcal{A} \vdash^{\eta',\sigma_1,\lambda}_{\mathcal{C}} \Gamma, \neg\varphi$.*

Parameters which are unimportant are often dropped (if possible) or replaced by $-$. E.g., $\mathcal{A} \vdash^{\eta}_{\mathcal{C}} \Gamma$ abbreviates $(\exists\sigma, \lambda)\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$, and $\mathcal{A} \vdash^{-,\sigma}_{\mathcal{C}} \Gamma$ abbreviates $(\exists\eta, \lambda)$ $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$. $\vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$ means $\emptyset \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$.

If $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \emptyset$ then we call this proof a *refutation proof* of $\mathcal{A}$. Proofs where cut-formulas $\mathcal{C}$ are only variables are called *resolution proofs*, refutations of that kind *resolution refutations*. We denote this by $\vdash^{\eta,\sigma,\lambda}_{Var}$.

Let φ be a CNF-formula, i.e. of the form $\bigwedge_{i<I}\bigvee_{j<J_i}\varphi_{ij}$ with φ_{ij} being literals. Then φ can be viewed as a set of clauses consisting of all $\{\varphi_{ij} : j < J_i\}$ for $i < I$. This gives meaning to the writing $\varphi \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$.

Structural rules are not included in the definition of LK. But we obtain structural rules as derivable rules which is stated in the next proposition. It is readily proven by induction on the height of the derivation η.

Proposition 2 (Structural Rule). *Assume* $\mathcal{A} \subseteq \mathcal{A}'$, $\eta \leq \eta'$, $\sigma \leq \sigma'$, $\mathcal{C} \subseteq \mathcal{C}'$ *and* $|\Gamma'| \leq \lambda'$, *then* $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma$ *implies* $\mathcal{A}' \vdash^{\eta',\sigma',\lambda+\lambda'}_{\mathcal{C}'} \Gamma, \Gamma'$. □

The principle $\mathcal{O}\mathrm{Ind}(n)$ of order induction for n is given by

$$\left(\bigwedge_{i<n}\left(\left(\bigwedge_{j<i} p_j\right) \to p_i\right)\right) \to \bigwedge_{i<n} p_i$$

(of course $A \to B$ is an abbreviation of $\bigvee\{\neg A, B\}$). Let us also fix the set of clauses corresponding to $\neg\,\mathcal{O}\mathrm{Ind}(n)$:

type I	$\neg p_0, \ldots, \neg p_{a-1}, p_a$	for any $a < n$,
type II	$\neg p_0, \ldots, \neg p_{n-1}$.	

We can give upper bounds for certain parameters of shortest proofs of $\mathcal{O}\mathrm{Ind}(n)$.

Theorem 3. *1.* $\vdash^{O(n),O(n^2)}_{\emptyset} \mathcal{O}\mathrm{Ind}(n)$.

2. $\neg\,\mathcal{O}\mathrm{Ind}(n) \vdash^{n,O(n)}_{Var} \emptyset$.

Proof. Ad 1.: We can easily show by induction on k that

$$\vdash^{H(k),S(k)}_{\emptyset} \neg \bigwedge_{i<n}\Big(\bigwedge_{j<i} p_j \to p_i\Big), \bigwedge_{i<n} p_i, \{\neg p_i : i < k\}$$

holds for $k = n, \ldots, 0$, with $H(k) := 3(n+1-k)$, $S(k) := (n+1-k)(n+2)$. The assertion then follows for $k = 0$.

Ad 2.: We can easily show by induction on k that

$$\neg\,\mathcal{O}\mathrm{Ind}(n) \vdash^{H(k),S(k)}_{Var} \{\neg p_i : i < k\}$$

holds for $k = n, \ldots, 0$, with $H(k) := n-k$, $S(k) := 2(n+1-k)$. The assertion then follows for $k = 0$. □

2.1 Lower Bounds on Heights for Resolution

Viewing the "Boundedness Theorem" from [3, 2] (which is adapted from [4]) in the light of resolution we obtain that the principle of order-induction $\mathcal{O}\mathrm{Ind}(n)$ for n gives us lower bounds to the height of resolution *proofs*:

Theorem 4 ([4, 2, 3]). $\vdash^{\eta}_{Var} \mathcal{O}\mathrm{Ind}(n) \quad \Rightarrow \quad \eta \geq n\,.$ □

Together with Theorem 3.1 this gives us a separation of polylogarithmic-height resolution proofs from quasi-polynomial size tree-like resolution proofs.

A similar result holds for resolution *refutations* of $\neg\,\mathcal{O}\mathrm{Ind}(n)$, but with a much simpler proof.

Theorem 5. $\neg\,\mathcal{O}\mathrm{Ind}(n) \vdash^{\eta}_{Var} \emptyset \quad \Rightarrow \quad \eta \geq n\,.$

Proof. Assume for the sake of contradiction that $\neg\,\mathcal{O}\mathrm{Ind}(n) \vdash^{\eta}_{Var} \emptyset$ and $\eta < n$ hold. Let P be such a resolution refutation tree of height bounded by η. The assumption $\eta < n$ implies that the type II axiom of $\neg\,\mathcal{O}\mathrm{Ind}(n)$ does not occur in P, because the size of sequents can only shrink by 1 through an application of **(Cut)**. But the set of axioms of type I is satisfiable (by assigning each variable to 1) and the rules of LK are correct, hence the last sequent in the proof, which is $\emptyset$, must be true under this assignment, too. Contradiction. □

Theorem 3.2 and Theorem 5 together give us a separation of polylogarithmic-height resolution refutations from quasi-polynomial size tree-like resolution refutations.

3 Height Restricted Propositional Proofs

We start this section by proving further properties of height restricted propositional proofs like inversions and different kinds of cut-elimination.

The following propositions on ($\bigwedge$)-Inversion and ($\bigvee$)-Exportation are readily proven by induction on the height of the derivation η.

Proposition 6 (**($\bigwedge$)-Inversion**). *Assume that* $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma, \bigwedge_{i<I} \varphi_i$ *holds, and that* $\bigwedge_{i<I} \varphi_i \notin \Gamma$, *then* $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma, \varphi_i$ *holds for all* $i < I$. □

Proposition 7 (**($\bigvee$)-Exportation**). *Suppose* $\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\mathcal{C}} \Gamma, \bigvee_{i<I} \varphi_i$ *holds, then* $\mathcal{A} \vdash^{\eta,\sigma,\lambda+I}_{\mathcal{C}} \Gamma, \varphi_0, \ldots, \varphi_{I-1}\,.$ □

We define special sets of constant depth formulas.

Definition 8. $\Sigma^{s,t}_d$ *is the set of all formulas* φ *with*

1. $\mathrm{dp}(\varphi) \leq d+1$;
2. *if* $\mathrm{dp}(\varphi) = d+1$, *then the outermost connective of* φ *is* $\bigvee$;
3. *all depth* > 1 *sub-formulas of* φ *have the arity of their outermost connective bounded by* s; *and*
4. *all depth* 1 *sub-formulas of* φ *have the arity of their outermost connective bounded by* t.

A formula is in $\Pi^{s,t}_d$ *iff its negation is in* $\Sigma^{s,t}_d$.

For sets of number-theoretic functions $\Xi, \Sigma, \Lambda, F, G$ and a sequence of cedents Γ_n, $n \in \mathbb{N}$, we write $(\mathcal{A}_n)_n \vdash^{\Xi,\Sigma,\Lambda}_{\Sigma_d^{F,G}} (\Gamma_n)_n$, or sometimes $\mathcal{A}_n \vdash^{\Xi,\Sigma,\Lambda}_{\Sigma_d^{F,G}} \Gamma_n$, to denote that there are some $\eta, \sigma, \lambda, f, g$ from $\Xi, \Sigma, \Lambda, F, G$, resp., such that $\mathcal{A}_n \vdash^{\eta(n),\sigma(n),\lambda(n)}_{\Sigma_d^{f(n),g(n)}} \Gamma_n$ holds for all n. We further use $\Sigma_d^{\mathrm{poly}(n)}$ as an abbreviation for $\bigcup\{\Sigma_d^{f,g} \ : \ f(n) \in 2^{(\log n)^{O(1)}}, g(n) \in (\log n)^{O(1)}\}$. Here $\Sigma_d^{f,g}$ denotes the set of sequences $(\varphi_n)_n$ of formulas such that $\varphi_n \in \Sigma_d^{f(n),g(n)}$ for all $n \in \mathbb{N}$. We often write $\varphi_n \in \Sigma_d^{f,g}$ instead of $(\varphi_n)_n \in \Sigma_d^{f,g}$.

Remark 9. Krajíček in [13] has defined resolution systems R^* and $R(\log)^*$ which correspond to our setting as follows: Let Φ_n be a sequence of clauses. Then $(\Phi_n)_n$ is quasi-polynomial size refutable in R^* (respectively $R(\log)^*$) iff $(\Phi_n)_n \vdash^{-,2^{(\log n)^{O(1)}}}_{Var} \emptyset$ $\left(\text{respectively } (\Phi_n)_n \vdash^{-,2^{(\log n)^{O(1)}}}_{\Sigma_0^{\mathrm{poly}(n)}} \emptyset\right)$.

The next Proposition shows that by controlling heights we also obtain control over sizes and sequent-lengths of proofs. It follows directly by induction on the height.

Proposition 10. $\mathcal{A} \vdash^{\eta}_{\Sigma_d^{s,t}} \Gamma \subset \Sigma_{d'}^{s,t}$ *and* $t \leq s \quad \Rightarrow \quad \mathcal{A} \vdash^{\eta,s^\eta,|\Gamma|+\eta}_{\Sigma_d^{s,t}} \Gamma$. □

Remark 11. Krajíček in [11] has defined a notion called Σ-depth of a proof. This can be expressed in our terms as follows: φ has a Σ-depth d tree-like LK-proof of size σ iff $\vdash^{-,\sigma}_{\Sigma_d^{\sigma,\log\sigma}} \varphi$. Hence, the sequence $(\varphi_n)_n$ has quasi-polynomial-size Σ-depth d tree-like proofs iff $\vdash^{-,2^{(\log n)^{O(1)}}}_{\Sigma_d^{\mathrm{poly}(n)}} (\varphi_n)_n$. The last Proposition shows that $\vdash^{(\log n)^{O(1)}}_{\Sigma_d^{\mathrm{poly}(n)}} (\varphi_n)_n$ implies that $(\varphi_n)_n$ has Σ-depth d tree-like LK-proofs of size quasi-polynomial in n in which every cedent is of length polylogarithmic in n. Similar statements hold for refutations.

The proof of the next Lemma and Proposition follows the standard one which can be found e.g. in [2, 3] – we only have to control additional parameters.

Lemma 12 (Cut Elimination Lemma). *If* $\mathcal{A} \vdash^{\eta_0,\sigma_0,\lambda_0}_{\Sigma_d^{s,t}} \Gamma, \varphi$, $\mathcal{A} \vdash^{\eta_1,\sigma_1,\lambda_1}_{\Sigma_d^{s,t}} \Delta, \neg\varphi$ *and* $\varphi \in \Sigma_{d+1}^{s,t}$, *then* $\mathcal{A} \vdash^{\eta_0+\eta_1,\sigma_0\cdot\sigma_1,\lambda_0+\lambda_1}_{\Sigma_d^{s,t}} \Gamma, \Delta$. □

Proposition 13 (Cut Elimination Theorem).

$$\mathcal{A} \vdash^{\eta,\sigma,\lambda}_{\Sigma_{d+1}^{s,t}} \Gamma \quad \Rightarrow \quad \mathcal{A} \vdash^{2^\eta,\sigma^{2^\eta},2^\eta\cdot\lambda}_{\Sigma_d^{s,t}} \Gamma \ .$$ □

The next Proposition gives a form of cut elimination which makes use of the parameters size and sequent-length (and arity of outermost connective of cut formulas) while at the same time ignoring height of proofs. The one after the next one ignores size and sequent-length and depends only on height (and length of cut formulas).

Proposition 14 (Krajíček's **Cut Elimination** [11, **12.2.1**]).

$$\mathcal{A}\,\vert^{\eta,\sigma,\lambda}_{\Sigma^{s,t}_{d+1}}\,\Gamma \quad\Rightarrow\quad \mathcal{A}\,\vert^{-,\sigma\cdot s^{\lambda}}_{\Sigma^{s,t}_{d}}\,\Gamma\ .$$ □

The following Bounded-Cut Elimination is central for the study of height restricted proof systems. We repeat the proof from [3].

Proposition 15 (**Bounded-Cut Elimination** [2, 3]).

$$\mathcal{A}\,\vert^{\eta}_{\Sigma^{s,t}_{0}}\,\Gamma \quad\Rightarrow\quad \mathcal{A}\,\vert^{\eta\cdot t}_{Var}\,\Gamma\ .$$

Proof. The Proposition follows from the following Bounded-Cut Elimination lemma, which even gives rise to a more general Bounded-Cut Elimination – we keep the proposition in the form we have because that is all we need here.

Let $\mathrm{noa}(\varphi)$ be the number of (occurrences of) atoms in φ.

$$\mathcal{A}\,\vert^{\eta}_{Var}\,\Gamma,\varphi \quad\text{and}\quad \mathcal{A}\,\vert^{\eta}_{Var}\,\Gamma,\neg\varphi \quad\Rightarrow\quad \mathcal{A}\,\vert^{\eta+\mathrm{noa}(\varphi)}_{Var}\,\Gamma\ . \tag{1}$$

We prove (1) by induction on φ. If φ is atomic we just apply **(Cut)**.

Now assume w.l.o.g. that φ has the form $\bigvee_{i<k}\varphi_i$. By $(\bigwedge)$-Inversion and $(\bigvee)$-Exportation from Section 3 we obtain $\mathcal{A}\,\vert^{\eta}_{Var}\,\Gamma,\varphi_0,\ldots,\varphi_{k-1}$ and $\mathcal{A}\,\vert^{\eta}_{Var}\,\Gamma,\neg\varphi_i$ for all $i<k$. By successively applying the induction hypothesis k times we obtain $\mathcal{A}\,\vert^{\eta+\mathrm{noa}(\varphi_0)+\cdots+\mathrm{noa}(\varphi_{k-1})}_{Var}\,\Gamma$. Observe that $\mathrm{noa}(\varphi)=\sum_{i<k}\mathrm{noa}(\varphi_i)$. □

We repeat the translation (also called embedding) of provability in $S^1_2(\alpha)$, $T^1_2(\alpha)$, and more general of $\Sigma^b_{m+1}(\alpha)$-$\mathrm{L}^{m+1}\mathrm{IND}$, to LK from [2, 3]. We do not introduce language and theories of bounded arithmetic. All what we need from bounded arithmetic is that formulas translate in a certain way to the language of LK as described below, and that provability translates in the way described by the next theorem. Readers not familiar with bounded arithmetic simply can view these connections to bounded arithmetic as a motivation for studying the resulting propositional proof systems.

Let $\log^{(k)}(n)$ be the k-times iterated logarithm applied to n, and $2_k(n)$ the k-times iterated exponentiation applied to n.

There exists a canonical translation from the language of bounded arithmetic to the language of LK (see [12, 9.1.1], or [2, 3]). Let φ be a formula in the language of bounded arithmetic in which no individual (i.e. first order) variable occurs free – we call such a formula (first order) closed. Then $[\![\varphi]\!]$ denotes the translation of φ to the language of LK, which for example maps the atom $\alpha(t)$, for t a closed term of value $m_t\in\mathbb{N}$, to the propositional variable p_{m_t}, and bounded quantifiers to connectives $\bigwedge$ resp. $\bigvee$, e.g. $[\![(\forall x\le t)\varphi(x)]\!]=\bigwedge_{i\le m_t}[\![\varphi(i)]\!]$. It follows that a formula $\varphi(x)$ from Σ^b_d (with x being the only variable occurring free in φ) translates to $\left([\![\varphi(n)]\!]\right)_n$ in $\Sigma^{\mathrm{poly}(n)}_d$.

Theorem 16 ([2, 3]). *Let $\varphi(x)$ be a formula in the language of bounded arithmetic, in which at most the variable x occurs free.*

$$
\begin{array}{lcl}
S^1_2(\alpha) & \hookrightarrow & \text{polylogarithmic-height resolution} \\
 & & \not\cap \\
sR^2_2(\alpha) & \hookrightarrow & 2^{(\log\log n)^{O(1)}}\text{-height resolution} \\
 & & \not\cap \\
s\Sigma^b_3(\alpha)\text{-L}^3\text{IND} & \hookrightarrow & 2_2\left((\log^{(3)} n)^{O(1)}\right)\text{-height resolution} \\
 & & \not\cap \\
 & & \vdots
\end{array}
$$

Fig. 2. Translation of $\Sigma^b_{m+1}(\alpha)$-L^{m+1}IND

1. *If* $S^1_2(\alpha) \vdash \varphi(x)$, *then* $\vdash^{O\left(\log^{(2)} n\right)}_{\Sigma^{\mathrm{poly}(n)}_1} [\![\varphi(n)]\!]$.
2. *If* $T^1_2(\alpha) \vdash \varphi(x)$, *then* $\vdash^{O(\log n)}_{\Sigma^{\mathrm{poly}(n)}_1} [\![\varphi(n)]\!]$.
3. *If* $\Sigma^b_{m+1}(\alpha)$-L^{m+1}IND $\vdash \varphi(x)$, *then* $\vdash^{O\left(\log^{(m+2)} n\right)}_{\Sigma^{\mathrm{poly}(n)}_{m+1}} [\![\varphi(n)]\!]$. □

By combining this Theorem first with the Cut Elimination Theorem and afterwards with the Bounded-Cut Elimination we obtain

Theorem 17 ([2, 3]). *Let $\varphi(x)$ be a formula in the language of bounded arithmetic, in which at most the variable x occurs free.*

1. *If* $S^1_2(\alpha) \vdash \varphi(x)$, *then* $\vdash^{(\log n)^{O(1)}}_{Var} [\![\varphi(n)]\!]$.
2. *If* $\Sigma^b_{m+1}(\alpha)$-L^{m+1}IND $\vdash \varphi(x)$, *then* $\vdash^{2_m\left((\log^{(m+1)} n)^{O(1)}\right)}_{Var} [\![\varphi(n)]\!]$. □

If we take Theorem 16, and first apply the Cut Elimination Theorem, then Proposition 10, and finally Krajíček's Cut Elimination, we obtain the following Theorem:

Theorem 18 ([13, 3.1]). *Let $\varphi(x)$ be a formula in the language of bounded arithmetic, in which at most the variable x occurs free.*
If $T^1_2(\alpha) \vdash \varphi(x)$ *or* $\Sigma^b_{m+1}(\alpha)$-L^{m+1}IND $\vdash \varphi(x)$, *then* $\vdash^{-,2^{(\log n)^{O(1)}}}_{\Sigma^{\mathrm{poly}(n)}_0} [\![\varphi(n)]\!]$. □

We represent the last two Theorems together with previously obtained results in Fig. 1 and 2. The separation between quasi-polynomial-size tree-like resolution and quasi-polynomial-size sequence-like resolution is well-known (the best known separation can be found in [5]).

A separation between polylogarithmic-height resolution and quasi-polynomial-size tree-like resolution follows from Theorems 3 and 4: The first Theorem shows that $\mathcal{O}\mathrm{Ind}(n)$ has tree-like resolution proofs of size $O(n^2)$, whereas the second one shows that a resolution proof of this statement must have height $\Omega(n)$ and hence is unprovable in polylogarithmic-height resolution.

Theorems 3 and 4 can also be used to obtain a separation between $2_m\left(\left(\log^{(m+1)} n\right)^{O(1)}\right)$-height resolution and $2_{m+1}\left(\left(\log^{(m+2)} n\right)^{O(1)}\right)$-height resolution: By the first theorem, the formulas $\mathcal{O}\mathrm{Ind}\left(2_{m+1}\left(\left(\log^{(m+2)} n\right)^2\right)\right)$, for m fixed, have resolution proofs of height $2_{m+1}\left(\left(\log^{(m+2)} n\right)^{O(1)}\right)$, whereas the second theorem can be used to show that resolution proofs of these statements must have height $\Omega\left(2_{m+1}\left(\left(\log^{(m+2)} n\right)^2\right)\right)$, again for m fixed, and, therefore, are unprovable in $2_m\left(\left(\log^{(m+1)} n\right)^{O(1)}\right)$-height resolution.

By Theorem 17 and Theorem 18 we obtain translations of provability in $\Sigma^b_m(\alpha)$-L^{m}IND into two propositional proof systems which seem to be incomparable (for $m \geq 2$). We have visualized this for the case $m = 2$ in Fig. 3. Note that in general $2_m\left(\left(\log^{(m+1)} n\right)^{O(1)}\right)$-height resolution proofs have size super-quasi-polynomial in n.

$$sR^2_2(\alpha) \quad \hookrightarrow \quad \vdash^{(\log\log n)^{O(1)}}_{\Sigma_1^{\mathrm{poly}(n)}} \quad \begin{array}{l} \nearrow \; 2^{(\log\log n)^{O(1)}}\text{-height resolution} \\ \searrow \; \text{quasi-polynomial-size } R(\log)^* \end{array}$$

Fig. 3. Differences of translations of derivations

4 Cut Introduction and Simulation

In this section we investigate converses to cut-elimination. Krajíček has used ideas from Spira ([11, 4.3.10]) to reduce the number of cuts on any path through a tree-like proof by adding a special-$\bigwedge$-rule to LK and raising the depth of formulas in the proof.

Here we will study how the height of proofs can be shrinked by raising the depth of cut-formulas. We will obtain the following converse to the Cut Elimination Theorem from Section 3. Recall that $|\Gamma|$ denotes the number of formulas in the cedent Γ.

Theorem 19. *1. Assume* $\vdash^{\gamma}_{\Sigma_d^{s,t}} \Gamma$ *for* $\Gamma \subset \Pi^{s,t}_{d+1}$ *such that* $|\Gamma| \leq \log\gamma$ *and* $d > 0$. *Then* $\vdash^{O((\log\gamma)^2)}_{\Sigma_{d+1}^{s_\gamma,t}} \Gamma$ *for* $s_\gamma := s^{\gamma^{O(1)}}$.

2. Assume $\vdash^{\gamma}_{Var} \Gamma$ *for* $\Gamma \subset \Pi_1^{s,t}$ *such that* $|\Gamma| \leq \log\gamma$. *Then* $\vdash^{O(\log\gamma)}_{\Sigma_1^{2^\gamma, O(t\cdot\gamma)}} \Gamma$.

The proof of this Theorem needs some lemmas. Let ${}^m n$ denote the set of all number-theoretic functions from $\{0,\ldots,m-1\}$ to $\{0,\ldots,n-1\}$. The first lemma reduces heights by introducing intermediate cut formulas from the set $\Sigma^{s,t,\delta}_{d+1}$

given by formulas of the form $\bigvee_s \big(\bigwedge_\delta(\Sigma_d^{s,t} \cup \Pi_d^{s,t})\big)$. We understand $\Sigma_{d+1}^{s,t} \subset \Sigma_{d+1}^{s,t,\delta}$.

Lemma 20. *1. Let $\Gamma \subset \Pi_{d+1}^{s,t}$ and assume $\vdash^{\gamma}_{\Sigma_d^{s,t}} \Gamma$. Then $\vdash^{O(\log\gamma)+|\Gamma|}_{\Sigma_{d+1}^{s^\gamma,t,\gamma+|\Gamma|}} \Gamma$.*

2. In case of $d = 0$ let $\Gamma \subset \Pi_1^{s,t}$ and assume $\vdash^{\gamma}_{Var} \Gamma$. Then $\vdash^{O(\log\gamma)+|\Gamma|}_{\Sigma_1^{2^\gamma,\gamma+t\cdot|\Gamma|}} \Gamma$.

The proof of this lemma is postponed to Appendix A. The second part of the previous Lemma already proves the second part of Theorem 19.

The next Lemma is a propositional variant of sharply bounded collection [11, Def. 5.2.11].

Lemma 21. *Let $\varphi_{ij} \in \Pi_{d-1}^{s,t}$ and assume $d, s, \alpha \geq 2$, then*

$$\vdash^{\gamma}_{\Sigma_d^{s^\alpha,t}} \Gamma, \bigwedge_{i<\alpha} \bigvee_{j<s} \varphi_{ij} \qquad \Rightarrow \qquad \vdash^{\gamma+\log\alpha+O(d)}_{\Sigma_d^{s^\alpha,t}} \Gamma, \bigvee_{f\in{}^\alpha s} \bigwedge_{i<\alpha} \varphi_{i\,f(i)} \ .$$

Proof. Assume that $\vdash^{\gamma}_{\Sigma_d^{s^\alpha,t}} \Gamma, \bigwedge_{i<\alpha} \bigvee_{j<s} \varphi_{ij}$ and the other assumptions of the Lemma hold. For all $0 \leq a \leq b \leq \alpha$ and $k \leq \lceil \log\alpha \rceil$ it is not hard to show that

$$\vdash^{\gamma+N(k)}_{\Sigma_d^{s^\alpha,t}} \Gamma, \neg \bigvee_{f\in{}^\alpha s} \bigwedge_{a\leq i<b} \varphi_{i\,f(i)}, \bigvee_{f\in{}^\alpha s} \bigwedge_{a\leq i<b+2^k, i<\alpha} \varphi_{i\,f(i)}$$

holds for $N(k) = k + O(d)$. Then the assertion follows for $k = \lceil \log\alpha \rceil$ and $a = b = 0$. □

Finally we can remove the special cut formulas from Lemma 20.

Lemma 22. *Assume $\alpha \geq 2$, $d \geq 1$ and $t \leq s$. Then*

$$\vdash^{\gamma}_{\Sigma_{d+1}^{s,t,\alpha}} \Gamma \qquad \Rightarrow \qquad \vdash^{(\gamma+1)\cdot 2\cdot\log\alpha}_{\Sigma_{d+1}^{s^{\alpha+1},t}} \Gamma \ .$$

The proof of this lemma is postponed to Appendix A.

Proof (of Theorem 19.1). Assume $\vdash^{\gamma}_{\Sigma_d^{s,t}} \Gamma$ for $\Gamma \subset \Pi_{d+1}^{s,t}$ such that $|\Gamma| \leq \log\gamma$ and $d \geq 1$. By Lemma 20 we obtain $\vdash^{O(\log\gamma)}_{\Sigma_{d+1}^{s^\gamma,t,2\cdot\gamma}} \Gamma$. Now Lemma 22 produces $\vdash^{O(\log\gamma)\cdot 2\cdot\log(2\cdot\gamma)}_{\Sigma_{d+1}^{s^{\gamma\cdot(2\cdot\gamma)},t}} \Gamma$. Hence $\vdash^{O((\log\gamma)^2)}_{\Sigma_{d+1}^{s^{\gamma^{O(1)}},t}} \Gamma$. □

Applying Theorem 19, and the Cut Elimination Theorem and the Bounded-Cut Elimination from Section 3 we can draw the following Corollary:

Corollary 23 (Simulation). *Let $(\Gamma_n)_n$ be included in $\Pi_{d+1}^{\mathrm{poly}(n)}$ and the length of Γ_n, $|\Gamma_n|$, be bounded by a constant for all $n \in \mathbb{N}$.*

1. *Assume* $d > 0$ *and* $2_{i+1}((\log^{(j)} n)^{O(1)})$ *grows polylogarithmic in* n*, i.e.* $i+3 \leq j$*. Then*

$$\vdash^{2_{i+1}((\log^{(j)} n)^{O(1)})}_{\Sigma_d^{\mathrm{poly}(n)}} (\Gamma_n)_n \qquad \Leftrightarrow \qquad \vdash^{2_i((\log^{(j)} n)^{O(1)})}_{\Sigma_{d+1}^{\mathrm{poly}(n)}} (\Gamma_n)_n \ .$$

2. *For* $d = 0$ *assume* $2_{i+1}(O(\log^{(j)} n))$ *grows polylogarithmic in* n*, i.e.* $i+2 \leq j$*. Then*

$$\vdash^{2_{i+1}(O(\log^{(j)} n))}_{Var} (\Gamma_n)_n \qquad \Leftrightarrow \qquad \vdash^{2_i(O(\log^{(j)} n))}_{\Sigma_1^{\mathrm{poly}(n)}} (\Gamma_n)_n \ .$$

In particular, for $i = 0$ *and* $j = 2$ *this shows*

$$\vdash^{(\log n)^{O(1)}}_{Var} (\Gamma_n)_n \qquad \Leftrightarrow \qquad \vdash^{O(\log\log n)}_{\Sigma_1^{\mathrm{poly}(n)}} (\Gamma_n)_n \ .$$

□

Final Remarks and Open Problems

We have shown (and represented in Fig. 1) that provability in $S_2^1(\alpha)$ translates to polylogarithmic-height resolution, and provability in $T_2^1(\alpha)$ translates to quasi-polynomial size sequence-like resolution. Is there a system of bounded arithmetic which corresponds to quasi-polynomial size tree-like resolution?

The simulation given by Corollary 23 is unsatisfying in the following aspects: First, it does not hold for super-polylogarithmic height resolution which comes from $\Sigma_i^b(\alpha)$-L^{i}IND for $i \geq 2$. And second, for polylogarithmic-height resolution we have established the simulation only for provability of $\Pi_1^{\mathrm{poly}(n)}$-sequents which does not include $\mathcal{O}\mathrm{Ind}(n)$. This leads to the following questions:
1. What is the "right" propositional proof system corresponding to e.g. $sR_2^2(\alpha)$ (which is the same as $\Sigma_2^b(\alpha)$-L^{2}IND)? Remember that we have the two translations, represented in Fig. 3, that $sR_2^2(\alpha)$-proofs translate on the one side to $2^{(\log\log n)^{O(1)}}$-height resolution, and on the other side to quasi-polynomial-size $R(\log)^*$. Is the "right" system given by combining both proof systems, i.e. by $\vdash^{2^{(\log\log n)^{O(1)}}, 2^{(\log n)^{O(1)}}}_{\Sigma_0^{\mathrm{poly}(n)}}$ which is the same as quasi-polynomial-size $2^{(\log\log n)^{O(1)}}$-height $R(\log)^*$?
2. Can the simulation between $\vdash^{O(\log\log n)}_{\Sigma_1^{\mathrm{poly}(n)}}$ (which corresponds to provability in $S_2^1(\alpha)$) and polylogarithmic-height resolution be extended to formulas of the same kind as $\mathcal{O}\mathrm{Ind}(n)$, e.g. $\Sigma_2^{\mathrm{poly}(n)}$? Or, is there another version of resolution which allows this correspondence?

References

[1] Theodore Baker, John Gill, and Robert Solovay. Relativizations of the $\mathcal{P} =? \mathcal{NP}$ question. *SIAM J. Comput.*, 4:431–442, 1975. 600

[2] Arnold Beckmann. *Seperating fragments of bounded predicative arithmetic.* PhD thesis, Westf. Wilhelms-Univ., Münster, 1996. 600, 601, 603, 604, 605, 606, 607

[3] Arnold Beckmann. Dynamic ordinal analysis. *Arch. Math. Logic*, 2001. accepted for publication. 600, 601, 603, 604, 605, 606, 607

[4] Arnold Beckmann and Wolfram Pohlers. Applications of cut-free infinitary derivations to generalized recursion theory. *Ann. Pure Appl. Logic*, 94:7–19, 1998. 601, 603, 604

[5] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. Near-optimal separation of tree-like and general resolution. ECCC TR00-005, 2000. 600, 607

[6] Samuel R. Buss. *Bounded arithmetic*, volume 3 of *Stud. Proof Theory, Lect. Notes.* Bibliopolis, Naples, 1986. 599, 600

[7] Samuel R. Buss. Relating the bounded arithmetic and the polynomial time hierarchies. *Ann. Pure Appl. Logic*, 75:67–77, 1995. 600

[8] Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *J. Symbolic Logic*, 44:36–50, 1979. 599

[9] Johan Håstad. *Computational Limitations of Small Depth Circuits.* MIT Press, Cambridge, MA, 1987. 600

[10] Jan Krajíček. Fragments of bounded arithmetic and bounded query classes. *Trans. Amer. Math. Soc.*, 338:587–98, 1993. 600

[11] Jan Krajíček. Lower bounds to the size of constant-depth propositional proofs. *J. Symbolic Logic*, 59:73–86, 1994. 602, 605, 606, 608, 609

[12] Jan Krajíček. *Bounded Arithmetic, Propositional Logic, and Complexity Theory.* Cambridge University Press, Heidelberg/New York, 1995. 599, 600, 606

[13] Jan Krajíček. On the weak pigeonhole principle. *Fund. Math.*, 170:197–212, 2001. 600, 605, 607

[14] Jan Krajíček, Pavel Pudlák, and Gaisi Takeuti. Bounded arithmetic and the polynomial hierarchy. *Ann. Pure Appl. Logic*, 52:143–153, 1991. 600

[15] J. Paris and A. Wilkie. Counting problems in bounded arithmetic. In *Methods in mathematical logic (Caracas, 1983)*, pages 317–340. Springer, Berlin, 1985. 600

[16] Chris Pollett. Structure and definability in general bounded arithmetic theories. *Ann. Pure Appl. Logic*, 100:189–245, 1999. 600

[17] Andrew C. Yao. Separating the polynomial-time hierarchy by oracles. *Proc. 26th Ann. IEEE Symp. on Foundations of Computer Science*, pages 1–10, 1985. 600

[18] Domenico Zambella. Notes on polynomially bounded arithmetic. *J. Symbolic Logic*, 61:942–966, 1996. 600

Appendix A. Proofs of Lemma 20 and of Lemma 22

Proof (of Lemma 20). For $\mathcal{A}$ a set of cedents let $\bigvee \mathcal{A}$ be the set of all $\bigvee \Gamma$ for $\Gamma \in \mathcal{A}$. Let $\Gamma \subset \Sigma_d^{s,t} \cup \Pi_d^{s,t}$. We can show

$$\mathcal{A} \vdash^{2^\gamma}_{\Sigma_d^{s,t}} \Gamma \qquad \Rightarrow \qquad \bigvee \mathcal{A} \vdash^{O(\gamma)}_{\Sigma_{d+1}^{s^{2^\gamma},t,2^\gamma+|\Gamma|}} \bigvee \Gamma$$

by induction on γ. Then we obtain 1. by the following argument: Let Γ be the set $\{\bigwedge_{j<s} \varphi_{ij} : i < I\}$ with $\varphi_{ij} \in \Sigma_d^{s,t}$. For $f \in {}^I s$ let Γ_f be the set

$\{\varphi_{if(i)} : i < I\}$ of inversions, then by $(\bigwedge)$-Inversion from Section 3 $\vdash^{\gamma}_{\Sigma^{s,t}_d} \Gamma_f$. From the assertion we obtain $\vdash^{O(\log\gamma)}_{\Sigma^{s^\gamma,t,\gamma+|\Gamma|}_{d+1}} \Gamma_f$ for all $f \in {}^I s$, hence by $|\Gamma|$ many $(\bigwedge)$ inferences $\vdash^{O(\log\gamma)+|\Gamma|}_{\Sigma^{s^\gamma,t,\gamma+|\Gamma|}_{d+1}} \Gamma$.

The idea for proving the induction step of the assertion goes as follows: Given $\mathcal{A} \vdash^{2^{\gamma+1}}_{\Sigma^{s,t}_d} \Gamma$ we can find some set of cedents Γ_i for $i \in I$ such that $\mathcal{A} \vdash^{2^\gamma}_{\Sigma^{s,t}_d} \Gamma_i$ for all $i \in I$ and $\{\Gamma_i : i \in I\} \vdash^{2^\gamma}_{\Sigma^{s,t}_d} \Gamma$. Now we can apply the induction hypothesis to all these derivations, and putting them together suitably yields the assertion. The additional cuts are of the form $\bigwedge_{i\in I} \bigvee \Gamma_i$.

In case of $d = 0$ the same strategy even shows

$$\mathcal{A} \vdash^{2^\gamma}_{Var} \Gamma \quad \text{and} \quad \Gamma \subset Var \qquad \Rightarrow \qquad \bigvee \mathcal{A} \vdash^{O(\gamma)}_{\Sigma_1^{2^{2^\gamma},2^\gamma+|\Gamma|}} \bigvee \Gamma \ .$$

Then we obtain 2. in the following way: Let Γ be the set $\{\bigwedge_{j<s} \bigvee_{k<t} \varphi_{ijk} : i < I\}$ with $\varphi_{ijk} \in Var$. For $f \in {}^I s$ let Γ_f be the set $\bigcup_{i<I}\{\varphi_{i,f(i),0}, \ldots, \varphi_{i,f(i),(t-1)}\}$ of inversions, then by $(\bigwedge)$-Inversion and $(\bigvee)$-Exportation from Section 3 $\vdash^{\gamma}_{Var} \Gamma_f$ for all $f \in {}^I s$. The assertion now shows $\vdash^{O(\log\gamma)}_{\Sigma_1^{2^\gamma,\gamma+t\cdot|\Gamma|}} \bigvee \Gamma_f$. From this we obtain by a direct argument $\vdash^{O(\log\gamma)}_{\Sigma_1^{2^\gamma,\gamma+t\cdot|\Gamma|}} \bigvee_{k<t} \varphi_{0,f(0),k}, \ldots, \bigvee_{k<t} \varphi_{(I-1),f(I-1),k}$ for all $f \in {}^I s$, hence by $|\Gamma|$ many $(\bigwedge)$ inferences $\vdash^{O(\log\gamma)+|\Gamma|}_{\Sigma_1^{2^\gamma,\gamma+t\cdot|\Gamma|}} \Gamma$. □

Proof (of Lemma 22). Again we have to make our assertion a little bit more general. W.l.o.g. let $\varphi \in \Sigma^{s,t,\alpha}_{d+1}$ be of the form

$$\varphi = \bigvee_{i<s} \Big[\bigwedge_{j<\alpha_i} \bigvee_{k<s} \varphi_{ijk} \wedge \bigwedge_{\alpha_i \le j<\alpha} \varphi_{ij} \Big]$$

with $\varphi_{ijk} \in \Pi^{s,t}_{d-1}$ and $\varphi_{ij} \in \Pi^{s,t}_d$. Then let

$$\varphi^* := \bigvee_{i<s} \bigvee_{f\in{}^{(\alpha_i)}s} \Big[\bigwedge_{j<\alpha_i} \varphi_{ijf(j)} \wedge \bigwedge_{\alpha_i \le j<\alpha} \varphi_{ij} \Big] \ .$$

Dually for $\Pi^{s,t,\alpha}_{d+1}$. Observe that $\left(\Sigma^{s,t,\alpha}_{d+1}\right)^* \subset \Sigma^{s^{\alpha+1},t}_{d+1}$ and $\left(\Pi^{s,t,\alpha}_{d+1}\right)^* \subset \Pi^{s^{\alpha+1},t}_{d+1}$

We can prove

$$\vdash^{\gamma}_{\Sigma^{s,t,\alpha}_{d+1}} \Gamma, \Xi \quad \text{and} \quad \Xi \subset \Sigma^{s,t,\alpha}_{d+1} \cup \Pi^{s,t,\alpha}_{d+1} \qquad \Rightarrow \qquad \vdash^{(\gamma+1)\cdot 2\cdot\log\alpha}_{\Sigma^{s^{\alpha+1},t}_{d+1}} \Gamma, \Xi^*$$

by induction on γ which implies the Lemma for $\Xi = \emptyset$. In the induction step we use the previous Lemma 21. □

Author Index

Aehlig, Klaus 59
Akama, Yohji 1
Atserias, Albert 569

Baaz, Matthias 382
Barbanchon, Régis 397
Beauquier, Danièle 306
Beckmann, Arnold 599
Berwanger, Dietmar 352
Böhler, Elmar 412
Bonet, María Luisa 569
Bridges, Douglas 89

Cachat, Thierry 322
Cenciarelli, Pietro 200
Chen, Yifeng 120
Chernov, Alexey V. 74

Duparc, Jacques 322

Ésik, Zoltán 135

Faggian, Claudia 427, 442

Galmiche, Didier 183
Geuvers, Herman 537
Goubault-Larrecq, Jean ... 473, 553
Grädel, Erich 352
Grandjean, Etienne 397

Hasegawa, Masahito 458
Hayashi, Susumu 1
Hemaspaandra, Edith 412
Henzinger, Thomas A. 292
Hodas, Joshua S. 167
Hyland, Martin 442

Ishihara, Hajime 89

Joachimski, Felix 59
Jojgov, Gueorgui I. 537
Jung, Achim 216
Jurdziński, Marcin 292

Kakutani, Yoshihiko 506
Kanovich, Max 44
Kreutzer, Stephan 337
Kučera, Antonín 276
Kupferman, Orna 292

Lasota, Slawomir 553
Leivant, Daniel 367
Leiß, Hans 135
Lenzi, Giacomo 352
Levy, Paul Blain 232
López, Pablo 167

Mairson, Harry G. 151
Marcinkowski, Jerzy 262
McCusker, Guy 247
Méry, Daniel 183
Moser, Georg 382
Moshier, M. Andrew 216

Neven, Frank 2
Nipkow, Tobias 103
Nivelle, Hans de 584
Niwiński, Damian 27
Nowak, David 553

Ogata, Ichiro 490

Pimentel, Ernesto 167
Polakow, Jeffrey 167
Pym, David 183

Rabinovich, Alexander 306
Reith, Steffen 412
Rival, Xavier 151

Schmidt-Schauß, Manfred 522
Schulz, Klaus U. 522
Schuster, Peter 89
Skvortsov, Dmitriy P. 74
Skvortsova, Elena Z. 74
Slissenko, Anatol 306
Stoilova, Lubomira 167
Strejček, Jan 276

Thomas, Wolfgang 322
Truderung, Tomasz 262

Vereshchagin, Nikolai K. 74
Vollmer, Heribert 412